Autonom rekonfigurierbare Workflows

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Inf. Sebastian Richly
geboren am 25.04.1981 in Oschatz

Gutachter:
Prof. Dr. rer. nat. habil. Uwe Aßmann
(Technische Universität Dresden)

Prof. Dr. Rainer Schmidt
(Hochschule Aalen - Technik und Wirtschaft)

Datum der Einreichung: Dresden, 30. September 2011
Datum der Verteidigung: Dresden, 22. Dezember 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Anforderungen an Informationssysteme (IS) haben sich im Zuge der Durchdringung vieler Unternehmensbereiche erweitert. Stand zu Anfang ihre einfache Erfassung und Speicherung verschiedenster Daten im Vordergrund, vergrößerte sich ihr Anforderungsprofil auf die Analyse und Verwertung der Daten. Eine, über die Datenspeicherung hinausgehende, Entwicklung stellt die Integration des Informationssystems in die Unternehmensprozesse dar. Das bedeutet, dass nicht nur die Ergebnisse einer Tätigkeit gespeichert werden, sondern auch die Arbeitsabläufe selbst in das Informationssystem integriert sind. Es entstehen die *Process Aware Information Systems*[1] (PAIS) [53].

Die Integration verschiedener Arbeitsabläufe, seien sie aus der Produktion oder der Verwaltung, in Informationssysteme stellt hohe Anforderungen [138]. Es müssen unterschiedliche, reale Abläufe korrekt abgebildet werden. Zu diesem Zweck wurden *Workflows* eingeführt. Ein Workflow stellt ein vom PAIS ausführbares Ablaufmodell eines realen Arbeitsablaufes dar. Die Ausführung eines Workflows wird *Instanz* genannt. Mit einem Workflow können parallele Arbeitsabläufe ebenso modelliert werden, wie Entscheidungspunkte oder einzelne Aktivitäten. Zudem ist eine Interaktion mit dem Nutzer möglich. Diese muss jedoch effizient und intuitiv gestaltet werden, denn die Verwendung des Workflows in einem PAIS darf den Arbeitsablauf nicht unnötig behindern. Eine zusätzliche Anforderung an ein PAIS ist die Fehlerfreiheit. Eine mögliche Wiederholung verschiedener Tätigkeiten, hervorgerufen durch einen Fehler im PAIS, ist für ein Unternehmen sehr kostenintensiv.

Ein PAIS muss zudem mit sich ändernden Anforderungen umgehen können. Bei der Erstellung der Workflows werden die realen Arbeitsabläufe analysiert, wobei vor allem die verschiedenen Wege, die ein Arbeitsablauf nehmen kann, untersucht werden. Sind diese einmal beschrieben, werden sie in ein Workflowmodell überführt, das vom PAIS ausgeführt werden kann. Doch die Arbeitsabläufe in Unternehmen sind ständigen Änderungen unterworfen, sei es durch die Änderungen in der Produktion oder Gesetze. Aber auch klassische Ausnahmen, beispielsweise ein unausgefülltes Formular, müssen behandelt werden. Dies bedeutet, dass auch der Workflow geändert werden muss.

Essentiell für die Praxistauglichkeit eines PAIS ist es deshalb, dass auf *nicht vorhergesehene Änderungen* reagiert werden kann. Aufgrund dessen werden verschiedene Techniken entwickelt, das Problem der unvorhergesehenen Änderungen zu lösen.

[1] Prozessbewußte Informationssysteme.

1.1 Problemstellung und Zielstellung

Treten unvorhergesehene Änderungen auf, muss der Workflow rekonfiguriert bzw. adaptiert werden. Rekonfiguration ist in [193] wie folgt definiert:

> „Reconfiguration amounts to adding and removing components and connections, and may have to occur without stopping the execution of the system being reconfigured."

Ein Workflow wäre demnach dann dynamisch *rekonfigurierbar*, wenn sein Aufbau zur Laufzeit *geändert* werden kann. Die Behandlung von Änderungen bei Workflows wurde früh [54, 87, 102] ein wichtiges Forschungsgebiet, wobei vier Herausforderungen für Rekonfigurationen identifiziert werden konnten [180]:

- *Dynamische Änderungen*: Treten unvorhergesehene Zustände oder Ausnahmen auf, muss darauf im PAIS reagiert werden. Dabei ist zu entscheiden, ob diese Änderungen nur eine oder alle Instanzen betreffen. Sind alle Instanzen betroffen, müssen diese *migriert* werden, d. h. an die neue Workflowspezifikation angepasst werden. Dies ist jedoch nur unter bestimmten Bedingungen möglich. Wird beispielsweise die Löschung eines bestimmten Workflowfragmentes notwendig, können die Instanzen, die dieses Fragment aktuell ausführen, nicht (sofort) verändert werden. Diese Instanzen müssen unter Umständen abgebrochen oder ohne die Änderungen bis zum Ende ausgeführt werden. Für die Anwendung von Änderungen auf Instanzen müssen deshalb verschiedenste Bedingungen betrachtet werden.

- *Facettenreichtum von Veränderungen*: Veränderungen bei Workflows betreffen nicht nur Anpassungen des Arbeitsablaufes, d. h. die Struktur eines Workflows. Vielmehr müssen alle Perspektiven eines Workflows veränderbar sein. Dies betrifft neben der *Struktur* auch die zu verarbeitenden *Daten*, die Zuweisung von *Ressourcen*, die zur Verfügung stehenden *Dienste* des PAIS und weitere. Die Behandlung dieser unterschiedlichen Perspektiven stellt eine hohe Anforderung an die Handhabung von Rekonfigurationen.

- *Erlaubte Änderungen und Korrektheit*: Ein weiterer wichtiger Gesichtspunkt ist, welche Änderungen gestattet sind und ob diese Änderungen zu einem korrekten Workflow führen. Für jede Änderungsoperation muss sichergestellt sein, dass die resultierende Workflowspezifikation korrekt ist. Dies bezieht sich sowohl auf die syntaktische Korrektheit (Ist dieser Workflow noch ein ausführbarer Workflow[2]?), als auch auf die semantische Korrektheit (Erfüllt der Workflow immer noch die ihm ursprünglich zugeordnete Aufgabe?).

- *Änderungsmanagement*: Neben den Aspekten der Anwendung von Änderungen auf Workflows ist entscheidend, auf welcher Datenbasis der Entschluss zur Änderung gefällt wird. Dazu müssen dem Entscheidungsträger alle notwendigen Informationen über den Zustand aller Instanzen zur Verfügung stehen. Welche Informationen dies sind, hängt vom Entscheidungsträger ab. Ist dies ein Administrator, ist es notwendig, diese Informationen in aggregierter Form bereitzustellen. Dies ist erforderlich, denn bei mehreren gleichzeitig laufenden Instanzen kann die Übersicht durch zu viele Informationen verloren gehen. Anders gestaltet sich dies, wenn das

[2] Er wäre es beispielsweise nicht, wenn der Startpunkt des Workflows gelöscht würde.

PAIS selbst über die Adaptionen entscheiden soll. Hierfür müssen dem PAIS alle Informationen detailliert zur Verfügung stehen, damit eine richtige Entscheidung getroffen werden kann.

Umgesetzt werden die Unternehmensprozesse in PAIS durch *Workflow Management Systeme (WfMS)*, die speziell zur Administration, Erstellung und Verarbeitung von Workflows entwickelt wurden. In jedem WfMS besteht die Möglichkeit, die Workflowspezifikation offline zu ändern und den geänderten Workflow zu starten. Dies gilt allerdings nur, wenn das WfMS nicht noch eine Instanz der Originalspezifikation ausführt, denn dann existieren für den gleichen Workflow zwei unterschiedliche Spezifikationen für die Instanzen. Eine einfache Lösung besteht darin, eine neue Version des Workflows anzulegen, so dass beide Spezifikationen parallel ausgeführt werden können. Dadurch kann eine Änderung allerdings nur für neu gestartete Workflows genutzt werden. Die bereits bestehenden Instanzen bleiben hierbei unverändert, wodurch diese Lösung nur in bestimmten Fällen anwendbar ist.

Aufgrund der Bedeutung von Veränderungen für WfMS wurden verschiedene Forschungsprojekte durchgeführt, die eine Vielzahl an Lösungsansätzen hervorbrachten. In [156, 150] wurden diese Ansätze deshalb in drei große Kategorien eingeteilt:

1. *Rekonfiguration durch Evolution*: Lösungen dieser Kategorie begegnen Änderungen mit einem evolutionären Ansatz. Es wird demnach davon ausgegangen, dass Änderungen selten auftreten und auf Änderungen nicht sofort reagiert wird. Diese Lösungen arbeiten mit dem bereits beschriebenen Versionierungsansatz, wobei spezielle Änderungsvorschriften für Instanzen beschrieben werden. Ungeplante Aktionen werden demnach nicht unterstützt, dahingegen können alle Teilbereiche einer Workflowspezifikation verändert werden.

2. *Rekonfiguration durch Unter- und Überspezifikation*: Lösungen dieser Kategorie konzentrieren sich auf Änderungen, die zeitnah umgesetzt werden müssen, wobei diese Änderungen zeitlich befristet sind und nur eine Instanz betreffen. Dazu existieren sie zwei grundlegende Techniken: Bei der *Unterspezifikation* existiert kein vollständiges Workflowmodell, das geändert werden muss. Hier werden freie Stellen durch Eingliederung einzelner Aktivitäten oder kleinerer Workflows zum Zeitpunkt der Instanziierung (oder zur Laufzeit) befüllt. Es existieren zudem Ansätze, bei denen selbst der Basis-Workflow nicht beschrieben ist und erst zur Laufzeit erzeugt wird. Dadurch werden Änderungen ein essentieller Teil des Workflows selbst. Je nach Ansatz werden beispielsweise ein automatisierter Planer oder Administratoren zur Vervollständigung benötigt. Die Frage der Versionierung stellt sich hierbei nicht, allerdings besteht keine Planungssicherheit und der Aufwand, jeden einzelnen Workflow zu vervollständigen ist hoch. Bei der *Überspezifikation* werden für jeden Workflow bereits beim Entwurf mehrere Varianten für die verschiedenen Aktivitäten festgelegt. Erst während der Ausführung wird sich für eine Variante entschieden. Durch diese Ansätze kann schnell auf Standardprobleme reagiert werden, unvorhergesehene Änderungen können jedoch nur bedingt abgedeckt werden.

3. *Rekonfiguration durch dynamische Änderungen*: Einen Kompromiss zwischen diesen beiden unterschiedlichen Kategorien stellen die *dynamischen Änderungen* dar. Die Ablaufstruktur wird hierbei vollständig beschrieben, es können aber bestimmte Änderungsoperationen zur Laufzeit vorgenommen werden. Dabei sind die Änderungen zumeist befristet und auf einen Workflow beschränkt. Die möglichen Ände-

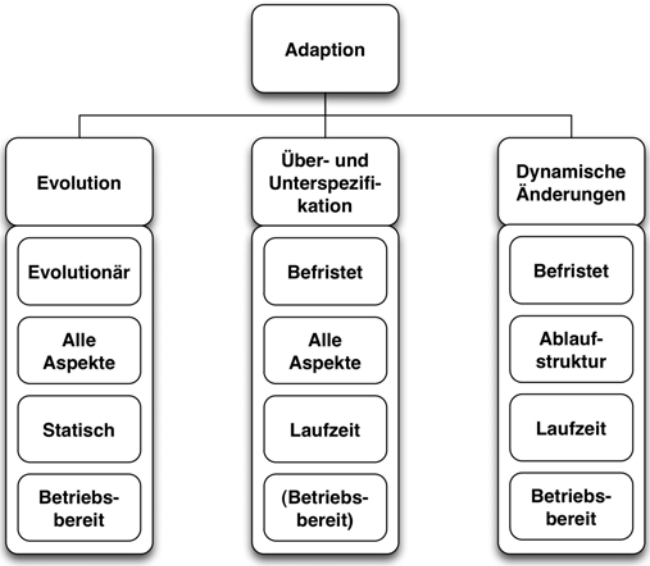


Abbildung 1.1: Adaptionsformen

rungsmuster [191] beschränken sich dabei zumeist nur auf die Anpassung der Struktur des Workflows. Veränderungen, die beispielsweise die Zuweisung eines Nutzers zu einer Aufgabe betreffen, werden nicht betrachtet. In [191] wird zudem gezeigt, dass kein aktuelles System diese beschränkten Änderungsmuster vollständig unterstützt.

Für die Behandlung von Veränderungen ergeben sich demnach drei Hauptmerkmale, in denen sich die Lösungen grundsätzlich unterscheiden:

- *Spezifikationsgrad der Workflows*, d. h. wie strukturiert sind die Workflows beschrieben. Bei evolutionären Lösungen sind sie vollständig beschrieben, bei Unterspezifikation gar nicht oder nur rudimentär.
- *Veränderung aller Perspektiven*: Neben der Perspektive der Ablaufstruktur eines Workflows müssen alle weiteren *kontextbezogenen Perspektiven*[3] unterstützt werden:
  - *Funktionale Perspektive*: Beschreibung der einzelnen Aktivitäten
  - *Informationelle Perspektive*: Beschreibung der Daten und des Datenflusses
  - *Operationale Perspektive*: Beschreibung der internen und externen Dienste
  - *Organisationale Perspektive*: Beschreibung von Ressourcen und Ressourcenzuweisungen zu den Aktivitäten
- *Zeitspanne der Änderung*, d. h. wie lang sind Änderungen gültig sind. Während sie bei den dynamischen Änderungen nur eine befristete Zeit gelten, zumeist solang die konkrete Instanz besteht, sind sie bei den evolutionären Änderungen permanent.

Abbildung 1.1 zeigt die Unterstützung dieser Merkmale bei den verschiedenen Adaptionskategorien im Überblick: Evolutionäre Änderungen sind statisch, während über- oder unterspezifizierte Workflows stark rekonfigurierbar sind, aber nicht ohne nachträgliche

[3] Eine ausführliche Beschreibung folgt in Abschnitt 2.2.2.

Justierung ausgeführt werden können; dynamische Änderungen sind auf Strukturänderungen beschränkt. Ein idealer Ansatz zur Handhabung von Veränderungen in WfMS stellt eine Mischung aus diesen Eigenschaften dar. Änderungen sollten:

- sowohl evolutionär, als auch befristet sein,
- alle Perspektiven betreffen,
- zur Laufzeit durchgeführt werden können und
- sich auf betriebsbereite Workflows beziehen

Zudem ist eine effiziente Steuerung der Änderungsvorgänge notwendig, die sowohl manuelle als auch automatische Eingriffe erlaubt. Das primäre Ziel dieser Arbeit ist die weitestgehend vollständige Abdeckung folgender Kriterien:

- Vollständige Rekonfiguration von Workflows *zur Laufzeit*
  - Entwicklung einer Lösung zur *Anpassung der Ablaufstruktur* von Workflows[4] zur Laufzeit, die die Adaptionsmuster aus [191] vollständig umsetzen kann. Die Lösung soll dabei eine *Kombination* aus *strukturierten Workflows* und dem Ansatz der *Unterspezifikation* sein. Grundidee ist, dass in einem *Basis-Workflow* die grundlegenden Abläufe spezifiziert werden. Die Behandlung von Ausnahmen oder andere Sonderbedingungen werden in einzelnen Workflows beschrieben. Sie stellen somit die *Varianten* des Basis-Workflows dar. Treten Ausnahmen zur Laufzeit auf, sollen die entsprechenden Workflows dynamisch in den Basis-Workflow eingewoben werden.
  - Entwicklung einer Lösung zur *Anpassung* aller *Standardperspektiven*. Dazu ist die Umsetzung einer hochflexiblen Workflow-Beschreibungssprache notwendig, in der nicht nur die bestehenden Perspektiven rekonfiguriert werden können, sondern auch jederzeit neue Perspektiven hinzugefügt werden können.
- Autonome Steuerung der Adaptionen
  - Damit schnell auf unvorhergesehene Veränderungen reagiert werden kann, soll eine *automatische Reaktion sowie Berechnung der Änderungsoperationen* für Workflows möglich sein. Diese Änderungen beziehen sich auf eine konkrete Instanz des Workflows und sind somit befristet.
  - Um *evolutionäre Änderungen* zu unterstützen muss sichergestellt werden, dass die Änderungsvorschriften in der gleichen Situation, bei einem anderen Workflow der gleichen Spezifikation, ebenfalls ausgeführt werden. Die *Adaptionsvorschriften* sollen selbst *veränderbar* sein, um sich selbst etwaigen Veränderungen anzupassen. Dadurch entsteht eine virtuelle Versionierung der Workflowspezifikation, wobei nicht die Workflowspezifikation geändert wird, sondern die Änderungsvorschriften. Die Spezifikation selbst bleibt stabil.
  - Änderungen durch eine autonome Steuerung dürfen in sicherheitskritischen Systemen keine Fehler verursachen[5]. Deshalb ist berechenbare, *stabile* Variante für automatische Berechnung von Adaptionsvorschriften notwendig.

[4] Da dies der primäre Aspekt von Workflows ist, kommt diesem eine besondere Bedeutung zu.
[5] Eine flexible Rechenvorschrift kann zu nicht geplanten Zuständen führen, die in kritischen Systemen ungewollt sind.

  - Die automatisch gesteuerten Adaptionen müssen weiterhin manuelle Adaptionen durch den Nutzer zulassen.
  - Derzeit existieren viele verschiedene WfMS für verschiedene Plattformen. Deshalb ist eine *plattformneutrale* Lösung der *Berechnung* für die Adaptionsvorschriften notwendig.

Die Arbeit konzentriert sich deshalb auf die großen Problemfelder bei dynamischen Änderungen von Workflowinstanzen: der Unterstützung von Änderungsoperationen für alle Workflowperspektiven und der autonomen Änderungssteuerung.

1.2 Ergebnisse der Arbeit

In dieser Arbeit werden Methoden für die Flexibilität im Zusammenhang mit Workflows auf verschiedenen Ebenen vorgestellt. Dies umfasst die Ebene der Adaption von Workflowinstanzen zur Laufzeit, die Ebene der Adaption der Workflowspezifikation und die Ebene der Adaption der Workflow-Engine. Zudem wurde eine neuartige Adaptionssteuerung entwickelt, um auf Veränderungen automatisch reagieren zu können.

Damit stellt die Arbeit einen umfassenden Ansatz für den Umgang der Rekonfiguration von Workflows und WfMS vor. Die wesentlichen Herausstellungsmerkmale der Arbeit sind:

- *Adaption durch Verweben [133]*: Um Adaptionen der Ablaufstruktur zur Laufzeit zu ermöglichen, wurde ein neuartiger Kompositionsmechanismus für Workflowmodelle entwickelt. Zentral ist dabei die Idee des Verwebens von Workflowmodellen bzw. Fragmenten. Ausgangspunkt ist ein *Basis-Workflows*, der nicht wie üblich die Behandlung von Ausnahmen oder alternative Abarbeitungspfade beinhaltet. Diese werden in separaten, kleinen Teil-Workflows beschrieben. Zur Laufzeit werden diese Workflows, je nach Bedarf, in den Basis-Workflow eingewoben. Basis-Workflow und die eingewobenen Workflows bilden einen Variantenraum, d. h. alle möglichen *Ausführungsvarianten*. Dabei unterscheidet sich die Lösung von bisherigen Ansätzen durch den *variablen Rücksprungpunkt*. Bei der Eingliederung von Workflows als Einschub für eine leere Stelle eines Basis-Workflows (Unterspezifikation) wird der Kontrollfluss des Basis-Workflows nicht verändert. Bei der Adaption wird lediglich der eingewobene Workflow ausgeführt und anschließend mit dem vorgesehenen Ablauf des Basis-Workflows fortgefahren. Bei der *Adaption durch Verweben* kann beim Verweben ein zusätzlicher Rücksprungpunkt angegeben werden. Dieser beschreibt, wo mit der Bearbeitung des Basis-Workflows nach der Adaption fortgefahren wird. Der dadurch entstehende *dynamische Kontrollfluss* ermöglicht die Umsetzung *aller*[6] *Adaptionsmuster*.
- *Rekonfigurierbares rollenbasiertes Workflowmodell [134, 67]:* Zur Adaption der kontextbezogenen Perspektiven eines Workflows wurde ein rollenbasiertes Workflowmodell erstellt. Rollen [163, 69] als Weiterentwicklung der Aspektorientierung erlauben es, sowohl die Daten, als auch das Verhalten von Objekten zur Laufzeit zu verändern. Das rollenbasierte Workflowmodell besteht dabei aus einem Kern, der den Aspekt des Kontrollflusses durch ein Petri-Netz abbildet. Dieser Kern kann durch verschiedene Rollen dynamisch erweitert werden, d. h. Rollen können zur

[6] Bis auf das Adaptionsmuster AP12, wie in Abschnitt 3.1.2 beschrieben.

Laufzeit an Objekte gebunden oder von ihnen entfernt werden. Es wurden verschiedene Standardrollen für die kontextbezogenen Perspektiven entwickelt. Diese Rollen verfeinern die Objekte des Petri-Netzes. Eine zentrale Eigenschaft von Rollen ist, dass die Verhaltensänderungen jederzeit zurückgenommen bzw. durch andere ersetzt werden können. Das dadurch entstehende dynamische Workflowmodell ermöglicht es nicht nur, jede Perspektive des Workflows zur Laufzeit zu ändern, sondern hat auch Auswirkungen auf das WfMS:

  - *Rekonfigurierbares WfMS*: Über die Möglichkeit der Laufzeitadaption des Workflowmodells hinaus, wurden rekonfigurierbare WfMS untersucht. Dabei wird ausgenutzt, dass Rollen nicht nur das Verhalten von Objekten zur Laufzeit ändern können. Zudem ist es möglich, neue Rollen zu einem WfMS zur Laufzeit hinzuzufügen oder zu entfernen. Dementsprechend können die vom WfMS bereitgestellten Funktionen, die den Rollen entsprechen, zur Laufzeit rekonfiguriert werden.

  - *Erweiterbarkeit von Workflowsprachen*: Die Rekonfiguration von Funktionen eines WfMS ermöglicht es Workflowsprachen, zugeschnitten auf spezielle Anwendungsgebiete, umzusetzen. Dazu können spezielle Konfigurationen für unterschiedliche Plattformen erstellt werden. Dabei werden die notwendigen Rollen maßgeschneidert zu einem Workflowmodell zusammengestellt. So fehlt beispielsweise bei einem WfMS für eingebettete Systeme die Möglichkeit entfernte Dienste aufzurufen, da diese Plattform die notwendige Kommunikationsschnittstelle nicht zur Verfügung stellt.

- *Universelle Referenzarchitektur zur autonomen Rekonfiguration von Workflows:* Zur Steuerung der Adaption durch Verweben und des rollenbasierten Workflowmodells wurden der *Direktor* und *Koordinator* eingeführt. Beide Komponenten sind Teil eines Regelkreises zur Berechnung und Umsetzung von Adaptionen. Der Direktor berechnet auf Basis der aktuellen Situation des Workflows, zu der auch Ausnahmen zählen, und des WfMS die notwendigen Adaptionen. Diese werden vom Koordinator in Befehle für die Workflow-Engine umgesetzt. Beide Komponenten wurden plattformunabhängig entworfen, d. h. sie wurden nicht für ein spezielles WfMS ausgelegt. Zudem wurde untersucht, welche unterschiedlichen Möglichkeiten für die Umsetzung des Direktors existieren und welche Einsatzmöglichkeiten dadurch entstehen:

  - *BDI-Direktor [133]:* Ein wichtiges Kriterium für die allgemeine Verwendbarkeit des Direktors ist die Unabhängigkeit von einer Domäne, d. h. dass er ohne Anpassungen in verschiedenen Anwendungsbereichen verwendet werden kann. Zur Realisierung dieser Eigenschaft wurde untersucht, inwieweit eine Technik aus der Forschung zu künstlicher Intelligenz zur Berechnung von Adaptionen genutzt werden kann. Diese Technik basiert auf der Idee der Nachahmung menschlicher Entscheidungen, wobei drei essentielle Konzepte eingeführt werden: *Belief*, *Desire* und *Intention*. Die Beliefs bilden das Wissen über den Workflow und seine Umgebung ab. Desires sind Ziele, die eine Workflowinstanz verfolgt, und Intentions repräsentieren die Absichten, die mit den Zielen (eingewobenen Workflow) verfolgt werden. Über den aktuellen Zustand des Workflows (Beliefs) und die aktuellen Ziele lässt sich durch eine definierte Abbildung auf eine Adaption (Intention) schließen. Für die Darstellung und Berechnung der Abbildungen wurde auf Case-based Reasoning

zurückgegriffen.

- *Semantischer BDI-Direktor [140]:* Der BDI-Ansatz kann unabhängig von jeder Domäne eingesetzt werden, da er auf dem numerischen Vergleich aller Zustandsgrößen beruht[7]. Dies bedeutet jedoch auch, dass die konkrete Bedeutung von Zeichenketten oder Zahlen nicht erfasst werden kann. Der BDI-Direktor wurde deshalb um eine semantische Komponente erweitert (SBDI). Dies ermöglichte nicht nur die Optimierung auf eine bestimmte Domäne, sondern erweiterte die Nutzungsmöglichkeiten deutlich. Durch die eingeführte Semantik konnten *kooperative Direktoren* entwickelt werden, die Lösungen austauschen oder voneinander lernen können. Eine weitere Anwendungsmöglichkeit stellt die *Redundanzerkennung bzw. -vermeidung* dar. Durch die bessere Vergleichbarkeit der aktuellen Situationen von Workflows kann erkannt werden, welche Aktivitäten oder Teilworkflows zu einem bestimmten Zeitpunkt redundant ausgeführt werden. In Verbindung mit der Adaption durch Verweben können redundante Ausführungen vermieden werden.
- *Statechart-Direktor*: Der SBDI-Ansatz basiert auf dem Konzept des kontinuierlichen Lernens, d. h. dass die Qualität der Entscheidungen von den gemachten Erfahrungen des Direktors abhängig ist. Dies führt vor allem in der Anfangsphase zu fehlerhaften Entscheidungen, da nicht genügend Erfahrungen gesammelt wurden. Fehler sind bei sicherheitskritischen Abläufen, bspw. in der Produktion, nicht akzeptabel. Es wurde deshalb eine dritte Realisierungsform für den Direktor entwickelt, die diesen Anforderungen Rechnung trägt. Während die verschiedenen Varianten für eine Workflowausführung kontinuierlich neu gelernt werden und in den Case-based Reasoning Datenbanken gespeichert werden, setzt dieser Direktor auf unveränderliche Varianten, die durch ein Statechart beschrieben werden. Die Beschreibung aller Ausführungsvarianten eines Workflows wird somit vom Start an festgelegt und kann dadurch für sicherheitskritische Abläufe eingesetzt werden.

All diese Konzepte stellen grundlegende Lösungskonzepte dar, die unabhängig von jeder Workflow-Beschreibungssprache bzw. WfMS eingesetzt werden können.

Über diese Kernergebnisse hinaus wurden weitere wichtige Resultate erzielt. Als Werkzeugunterstützung wurde für die neu eingeführten Konzepte mit der *Open Service Process Platform* (OSPP) eine Referenzplattform entwickelt. Diese modulare Plattform stellt alle notwendigen Schnittstellen zur Integration unterschiedlicher Direktoren zur Verfügung, um mit der Workflow-Engine zu interagieren und das rollenbasierten Workflowmodell zu verarbeiten. Zudem wurde darauf geachtet, dass unterschiedliche OSPP-Instanzen miteinander kooperieren können. Diese Funktion wurde auch für die Evaluation verwendet, bei die verteilte Ausführung einzelner Workflowfragmente mit den vorgestellten Adaptionsansätzen untersucht wurde. Dies ist beispielsweise bei Genexpressionsprozessen notwendig, denn aufgrund ihrer großen Ressourcenanforderungen ist eine Lastverteilung zwischen verschiedenen Workflow-Engines notwendig. In der Evaluation konnte gezeigt werden, dass mithilfe der Adaption durch Verweben und des rollenbasierten Workflowmodells, Genexpressionsprozesse für *Production und Administrative Workflows*[8] zur Lastverteilung auf verschiedene Workflow-Engines verteilt wer-

[7] siehe Abschnitt 5.2.2.

[8] Es werden die englischen Begriffe verwendet, da die deutsche Übersetzung die Begriffe einschränken würde.

den können. Zugleich konnte der Direktor-Koordinator-Ansatz genutzt werden, um die Steuerung der Adaptionen vorzunehmen.

Mit der Open Service Process Platform wurde zwei Mal erfolgreich am *IEEE Service Computing Contest* teilgenommen. Im Jahre 2008 konnte die Konzeption eines rekonfigurierbaren WfMS auf Basis der BDI-Direktoren und des rollenbasierten Workflowmodells den *Sieg* beim *Service Computing Contest*[9][10] davon tragen [136].

Im Rahmen der Untersuchung zur Vermeidung von redundanten Workflowfragmenten wurde zudem ein neues Adaptionsmuster gefunden: *Merge*. Wird ein Workflowfragment in zwei unterschiedlichen Instanzen gleichzeitig ausgeführt, können die Ergebnisse einer Instanz in die andere übernommen werden[11]. Dazu wird der Kontrollfluss einer Instanz zeitweise durch die andere wahrgenommen.

1.3 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut. Nach einer Einführung in die für das Verständnis der Arbeit notwendigen Grundlagen – Workflow Management Systeme, Workflows und den Umgang mit Veränderungen – in Kapitel 2 werden in Kapitel 3 zwei grundlegende Konzepte vorgestellt. Dies ist zum einen die Adaption durch Verweben, eine neuartige Methode zur Adaption des Kontrollflusses von Workflows. Zum anderen werden in Kapitel 3 der *Direktor* und der *Koordinator* eingeführt. Beide sind Komponenten eines speziellen Regelkreises zur Steuerung der Adaption durch Verweben. In den folgenden Kapiteln 4 bis 6 werden verschiedene Realisierungsformen der *Direktor*-Komponente vorgestellt. Eine Lösung, die sich vor allem für sicherheitskritische Anwendungen eignet, wird in Kapitel 4 vorgestellt. Kapitel 5 verwendet den BDI-Ansatz zur Berechnung der Adaptionen. In Kapitel 6 wird dieser Ansatz um Semantik erweitert, was nicht nur eine Spezialisierung des Direktors ermöglicht, sondern auch die Kommunikation zwischen verschiedenen Direktoren ermöglicht.

Neben dem Aspekt der Veränderung des Kontrollflusses müssen auch alle weiteren Perspektiven veränderlich sein. Dieses Problem wird in Kapitel 7 durch eine rollenbasierte Workflowsprache gelöst. Umgesetzt wurden diese unterschiedlichen Lösungskonzepte in der *Open Service Process Platform*, die in Kapitel 8 eingeführt wird und Basis für die Evaluation dient. Eine Zusammenfassung und Ausblick bietet das Kapitel 9.

[9] http://iscc.servicescomputing.org/2008/index.htm (zuletzt aufgerufen am 25.02.2011).
[10] http://iscc.servicescomputing.org/2008/Results.htm (zuletzt aufgerufen am 25.02.2011).
[11] Vorausgesetzt, die Parameter für die Funktionsaufrufe sind gleich.

Kapitel 2

Adaptivität in Workflow Management Systemen

> „Wer macht was, wann, wie und womit?“ [18]

So einfach die Kernfrage auch ist, eine einheitliche Definition für das Prozess- und Geschäftsprozessmanagement existiert nicht. Allgemein beschäftigt sich das Prozessmanagement mit der Verwaltung und Verarbeitung von Geschäftsprozessen, die häufig auch *Business Processes* genannt werden.

Geschäftsprozesse beschreiben die durchzuführenden Aktivitäten und deren Reihenfolge, Dokumente bzw. Datenstrukturen und Kooperationen zwischen den Teilnehmern. Die Mitarbeiter sind zumeist angehalten, diese standardisierten Arbeitsabläufe umzusetzen, da die Einführung von Geschäftsprozessmanagement eine Steigerung der Qualität, Sicherheit und Effizienz verspricht. Bis auf bestimmte Berufe, in denen es notwendig ist, sich streng an die Vorschriften eines Prozesses zu halten, werden Mitarbeiter aus verschiedenen Gründen von diesen Prozessen abweichen wollen. Die Gründe hierfür sind vielfältig. Wird beispielsweise festgestellt, dass einige Voraussetzungen für eine Aufgabe nicht gegeben sind, werden weitere Tätigkeiten vorangetrieben, bis die gewünschten Voraussetzungen herstellt sind. In anderen Situationen werden die Mitarbeiter zusätzliche Mittel beantragen oder beenden einen Prozess, wenn er komplett zu scheitern droht. Diese Abweichungen innerhalb der Geschäftsprozesse sind nicht geplant und auch nicht vorhersehbar. Das Management erwartet dabei, dass die Mitarbeiter im Rahmen ihrer Zuständigkeit selbst über Adaptionen des Ausgangsprozesses entscheiden.

Der linke Prozess in Abbildung 2.1[1] zeigt einen vereinfachten Geschäftsprozess, wie er vielleicht in einem kleinen Theater angewandt wird. Ein Mitarbeiter des Theaters begrüßt die Gäste und überprüft die Tickets. Bei diesem Prozess gibt es keine Rechnerunterstützung und beide Maßnahmen sind manuell durch den Mitarbeiter durchzuführen. Die Prozessbeschreibung ist dabei nicht exakt, sondern muss interpretiert werden. Aufgaben sind in natürlicher Sprache beschrieben, die Regeln können je nach Situation flexibel ausgelegt werden und Übergänge sind nicht deterministisch. Somit ist es möglich, dass der Besitzer des Theaters für wichtige Gäste spezielle Plätze reserviert. Wenn einige dieser Gäste kommen, werden sie direkt zu den reservierten Plätzen geführt. Die Information, ob jemand eine wichtige Person oder wann das Theater vollständig belegt ist, hängt vom individuellen Eindruck des Mitarbeiters ab.

Durch die zunehmende Durchdringung der geschäftlichen Abläufe mit Informationssystemen, wurde es auch notwendig, die Geschäftsprozesse zu automatisieren. Daraus ergibt

[1] Knoten mit gepunkteten Linien repräsentieren eine manuelle Aktivitäten und die Raute einen Entscheidungsknoten.

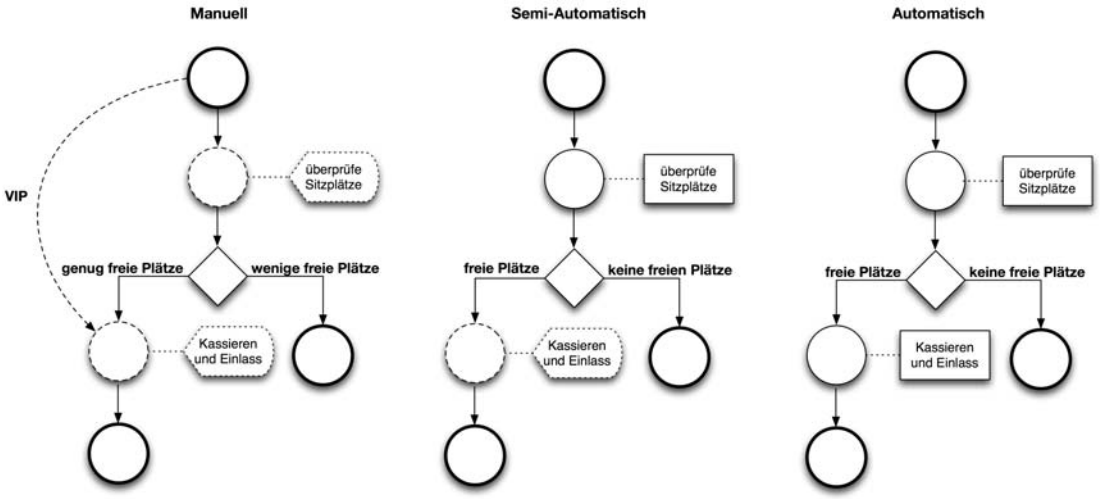


Abbildung 2.1: Automatisierungsformen am Beispiel des Einlasses im Theater

sich, dass die Steuerung, wann welche Aufgabe durchzuführen ist, im Informationssystem hinterlegt wird und dieses sich auch um die korrekte Durchführung Sorge trägt. Die Geschäftsprozesse werden dazu durch *Workflows* beschrieben, welche vom Informationssystem verarbeitet werden. Sie definieren die auszuführenden Aktionen und die Reihenfolge ihrer Ausführung. Allerdings ist es nicht notwendig, dass alle Aktionen durch einen Rechner ausgeführt werden. Sie können auch manuell von einem Mitarbeiter durchgeführt werden. Für ihre Ausführung wird der Workflow angehalten und gewartet, bis der Benutzer ein Signal gibt, dass er die zugehörige Aktion ausgeführt hat. Danach führt das Informationssystem die nächsten Aufgaben aus. Die Ordnung der Aktivitäten ist dabei streng definiert und der Fortschritt im Workflow wird automatisch vorangetrieben. Dies bedeutet, dass das Workflow Management System nicht akzeptieren wird, dass der Arbeitnehmer auf die Reihenfolge der folgenden Aktionen Einfluss nehmen kann. In einem solchen Workflow System besitzt der Mitarbeiter nur die Rolle eines Benutzers und verliert seine Unabhängigkeit.

Im Beispiel in Abbildung 2.1 zeigt der Workflow in der Mitte eine halbautomatische Variante. In diesem Fall hat das Theater eine automatische Kasse gekauft haben. Die anderen Aktionen[2] sind weiterhin informell beschrieben und werden manuell durchgeführt. Die Übergänge sind jedoch durch das Informationssystem gesteuert, die die Anzahl der Gäste speichert und entscheidet, wie viele weitere Plätze angeboten werden können. Diese Teilautomatisierung zwingt das Theater in einen eher formalen Workflow, bei dem die Bedingungen korrekt kontrolliert werden. Ausnahmeregelungen zum Einlass von VIPs müssen bei solchen Systemen aber im Voraus bekannt sein. Nachträgliche Änderungen sind nicht ohne Veränderung des Gesamtsystems möglich.

Eine vollautomatische Version des Beispiels ist auf der rechten Seite der Abbildung 2.1 dargestellt. Angenommen das Theater hat sich einem zentralen Ticketsystem angeschlossen, bei dem der Verkauf über das Internet abgewickelt wird. Der Workflow für die Online-Schnittstelle ist voll automatisiert. Es müssen daher alle manuellen Aktionen durch programmierte Äquivalente ersetzt werden. Die Gäste besitzen die Tickets als formalisierten Sitzplatzanspruch und der Preis der verfügbaren Tickets ist streng durch die

[2] In der Abbildung 2.1 als nicht gestrichelte Knoten dargestellt.

Agentur geregelt. Das Problem derartiger vollständig durchgeplanten Lösungen ist, dass sie nicht auf spontane Änderungswünsche ausgerichtet sind. Dies bedeutet im Fall des Beispiels, dass der Manager des Theaters am Abend der Veranstaltung mehrere Plätze für nicht registrierte VIPs besetzen möchte, obwohl das Reservierungssystem keine Anfragen mehr annimmt. Es können sich aber auch Änderungen im generellen Ablauf der Ticketreservierung ergeben, wenn beispielsweise ein Rabattsystem eingeführt wird.

Eine Anpassung eines Workflows ist jedoch ein komplexer Vorgang. Neben der Änderung der Ablaufspezifikation gilt es weitere Dinge zu beachten, wie beispielsweise die Konsistenz der Daten oder die Zuordnung von Ressourcen für die Aktivitäten. Diese komplexen Eigenschaften stellen ein Geflecht an Beschränkungen für die Reihenfolge der Ausführung von Aktionen dar. Anpassungen können deshalb in vielen Workflow Management Systemen nur unter bestimmten Voraussetzungen vorgenommen werden. Dabei existieren bereits verschiedene Lösungsansätze, die sich sowohl in Anwendbarkeit auf das oben geschilderte Problem, als auch in der praktischen Umsetzbarkeit unterscheiden. Das Spektrum der Lösungen kann in drei Gruppen gegliedert werden: *dynamische* Änderungen, *evolutionäre* Änderungen und Änderungen durch *Unter- und Überspezifikation*, die im Verlauf des Kapitels vorgestellt und bewertet werden. Zuvor werden noch die grundlegenden Begriffe und Konzepte im Bereich der Workflows eingeführt, die für das Verständnis der Arbeit von Bedeutung sind.

2.1 Grundlegende Konzepte von Workflows

Der Begriff des Geschäftsprozesses tritt in der Fachliteratur zumeist in den Hintergrund. Vielmehr wird die Bezeichnung *Workflow* verwendet. Dieser ist nach Ansicht der Workflow Management Coalition (WfMC) ein automatisierter Geschäftsprozess:

> „The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“ [197]

Das DIN-Institut konkretisiert diese Aussage noch weiter. Prozesse seien dann Workflows, wenn sie computergestützt ablaufen [189]. Die Geschäftsprozesse stellen demnach eine computerunabhängige Beschreibungsform dar, die durch weitere Verfeinerungen zu Workflows ausgebaut werden können. Die Verwaltung und Ausführung von computergestützten Prozessen übernehmen die *Workflow Management Systeme* (WfMS). Ziel der Einführung von WfMS in Unternehmen ist es, die vorhandenen Geschäftsprozesse zu automatisieren, bzw. eine Teilautomation durchzuführen und dadurch Kosten zu sparen. Die Grundlage für die Entwicklung von Workflows stellen reale Geschäftsprozesse dar, die analysiert werden, um einen ausführbaren Workflow zu erstellen. Dabei wird entschieden, welche Komponenten des IT-Systems welche Aufgaben übernehmen können und welche weiterhin manuell bearbeitet werden müssen.

Über Jahrzehnte hinweg entwickelte sich ein Sprachlandschaft mit vielen unterschiedlichen Workflow-Beschreibungssprachen. Durch das Aufkommen der SOAP[3] basierten

[3] Simple Object Access Protocol.

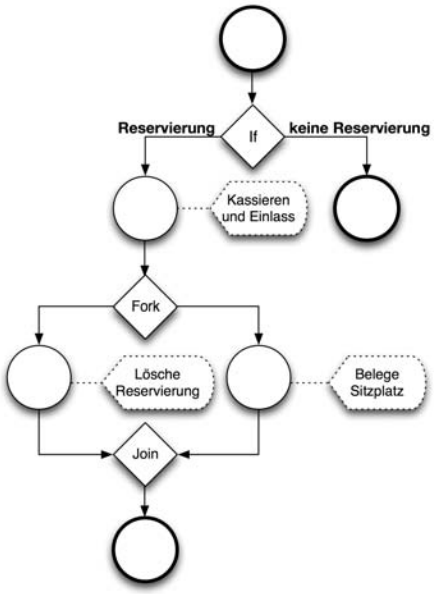


Abbildung 2.2: Erweiterter Theater-Workflow

Dienste [190] stieg ihre Anzahl weiter. Vertreter dieser Web Service Composition Languages[4] sind beispielsweise BPEL, BPMN, ebXML, BPSS, WS-CDL. Klassische Workflow-Beschreibungssprachen, wie z. B. XPDL, jBPM, YAWL, Flower, Staffware oder die bei SAP eingesetzte Sprache ARIS bieten vielmehr die gleichen Sprachkonstrukte, ihr Fokus ist jedoch nicht auf Webservices beschränkt[5].

2.1.1 Gebräuchliche Sprachkonzepte

Anhand des in Abbildung 2.2 erweiterten Theater-Beispiels, sollen die grundlegenden Strukturelemente eines Workflows dargestellt werden. Ein Workflow beginnt immer mit einem *Startknoten*. Die Reihenfolge der verschiedenen Aktivitäten wird durch die Transitionen (Pfeile in der Abbildung) geregelt. Der Entscheidungspunkt (dargestellt als Raute) `Reservierung` folgt als einziger direkt auf den Startknoten, was einer *seriellen* Aneinanderreihung entspricht. Ein Entscheidungsknoten entspricht einem `If` in einer Programmiersprache. Das `If` repräsentiert eine Entscheidung für den zu verfolgenden Pfad, die durch einen logischen Ausdruck beschrieben ist. Besitzen die Gäste keine Reservierung, folgt darauf ein Endknoten, so dass der Workflow beendet ist. Umgekehrt folgt die Aktivität `Kassieren`. Das Konstrukt des `If` wird in anderen Sprachen als *Decision Node* oder als *OR-Fork* bezeichnet.

Auf `Kassieren` folgt ein *AND-Fork*. Dieser spaltet den Kontrollfluss, jedoch werden alle auf den Fork folgenden Pfade *parallel* ausgeführt. In diesem Fall sind dies `Löschen` und `Belegen`. Beide Aktivitäten werden vom System parallel ausgeführt. Diese Nebenläufigkeit wird durch einen *Join* beendet.

Neben der seriellen und parallelen Abarbeitung von Aktivitäten existiert auch die Möglichkeit der Beschreibung von Iterationen. Diese können in anderen Sprachen durch

[4] Webservice Kompositionssprachen.
[5] Webservices können durch diese Sprachen zumeist auch angesprochen werden.

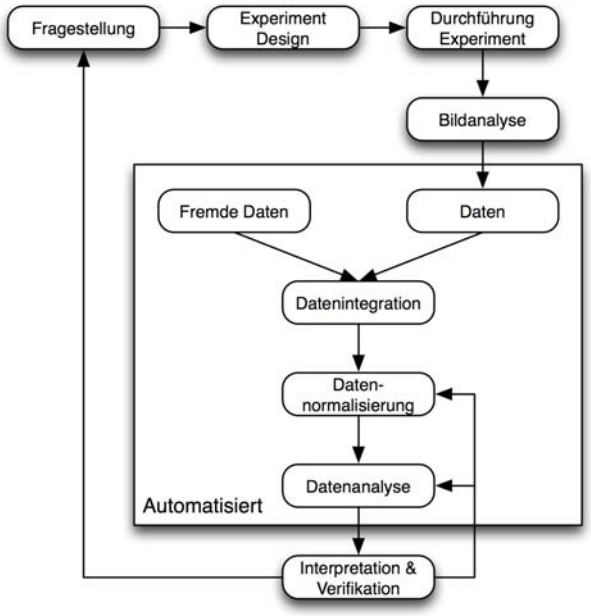


Abbildung 2.3: Ablauf eines Genexpressionsprozesses

For-Each oder *Reapeat-Unit* ausgedrückt werden. Die Semantik der beiden Schleifenkonstrukte entspricht dabei der von Programmiersprachen.

Ein vom WfMS soeben ausgeführter Workflow wird als *Instanz* bzw. Workflowinstanz bezeichnet. Der Abarbeitungszustand der Instanz wird durch ein oder mehrere *Token* und Variablen, die die Arbeitsdaten repräsentieren, beschrieben. Ein Token ist eine Marke, die immer die aktuell ausgeführte Aktion kennzeichnet und gleichzeitig alle Variablen und deren Belegungen speichert. Wird diese abgeschlossen, wandert das Token zu der folgenden Aktivität. Handelt es sich um mehrere Nachfolger, hervorgerufen durch einen Fork, wird die Marke aufgeteilt. Bei einem Join werden die Token wieder zu einem verschmolzen. Der Join darf nur dann ausgeführt werden, wenn alle Token das Ende ihres Teilpfades erreicht haben, d. h. die unmittelbar vor dem Join befindliche Aktivität abgeschlossen haben.

Betritt das Token eine *Aktivität*, auch *Task* genannt, wird eine vorher definierte Funktion ausgeführt. Es gibt dabei festgelegte Strukturfunktionen, die lediglich parametrisiert werden – wie beispielsweise das If, Fork, Join – und freie Aktivitäten. Die Ausgestaltung der freien Aktivitäten ist abhängig vom jeweiligen WfMS. Diese stellen verschiedene Grundfunktionen zur Verfügung und erlauben zusätzlich die Einbindung externer Dienste. Die Bandbreite dieser Funktionen reicht dabei von dem Aufruf einer EJB[6], über das Öffnen einer Word-Datei bis hin zum Einbinden eines Webservices.

2.1.2 Arten von Workflows

Mit diesen Basiselementen können komplexe Abläufe modelliert werden, die für verschiedene Anwendungsbereiche eingesetzt werden können. In [104] wird eine grobe Klassifika-

[6] Enterprise Java Bean

tion für Workflows gegeben – (1) *Production Workflows*, (2) *Administrative Workflows* und (3) *Ad-hoc-Workflows* – die im Folgenden vorgestellt werden.

Production Workflows Die erste Klasse stellen die *Production Workflows* dar. Diese Workflows repräsentieren Prozesse, die immer wiederkehrend gleich durchgeführt werden. Wie der Name erahnen lässt, sind dies vorrangig Prozesse, die in der Produktion genutzt werden, beispielsweise in Fabriken. Zu dieser Klasse gehören ebenfalls wissenschaftliche Prozesse, die vor allem aus komplexen Berechnungen bestehen. Dazu zählen unter anderem die *Grid Workflows*, die für die Verarbeitung von Massendaten verwendet werden [62].

Ein weiteres Beispiel für derartige wissenschaftliche Workflows sind die Genexpressionsprozesse[7] [137, 70]. Sie sind ein Teil der Forschung zur Bestimmung der Funktion eines oder mehrerer Gene der DNA. Eine Möglichkeit für dieses Messungen sind ist die Genexpression auf Basis von *Microarrays* [6, 7, 109]. Diese ermöglichen die Messung der relativen Häufigkeit von Genen, in Bezug auf den Zelltyp, Gewebe und Auswirkungen auf die Umwelt für tausende Gene gleichzeitig.

Die Abbildung 2.3 zeigt einen typischen Genexpressionsprozess. Im ersten Schritt wird die Forschungsfrage definiert und die Gestaltung des Ablaufs des Experimentes geplant. Im Anschluss werden die lokal gewonnen Daten einer Bildanalyse unterzogen. Diese Daten können mit denen anderer Forschungsgruppen kombiniert werden, um die Signifikanz der Ergebnisse zu erhöhen. Danach folgen vier essentielle Analyseschritte:

1. *Datenintegration:* Integration der Genexpressionsdaten in ein gemeinsames Schema.

2. *Datennormalisierung:* Normalisierung der integrierten Daten um Schwankungen zu kompensieren.

3. *Datenanalyse:* Untersuchung auf Beziehungen zwischen den Genen und Proben.

4. *Interpretation und Verifikation:* Wissenschaftliche Interpretation und Visualisierung der Ergebnisse.

Die Schwierigkeit bei derartigen wissenschaftlichen Workflows ist die notwendige Flexibilität. Nach Interpretation und Überprüfung der Analyseergebnisse müssen beispielsweise Normalisierung oder Datenanalyse mit veränderten Parametern oder Methoden wiederholt werden.

Administrative Workflows Diese Workflows bilden Prozesse ab, wie sie in der Verwaltung vorkommen. Diese zeichnen sich zum einen durch eine höhere Interaktion mit dem Nutzer aus, wie beispielsweise das Ausfüllen eines Formulars. Zum anderen sind Änderungen bei dieser Klasse sehr häufig. Hervorgerufen werden diese durch Gesetzes- oder Geschäftsregeländerungen.

[7] Genexpressionsprozesse werden ausführlich im Kapitel 8 vorgestellt.

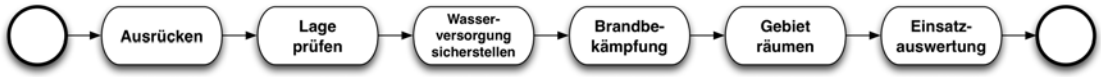


Abbildung 2.4: Ablauf eines Brandeinsatzes bei der Feuerwehr

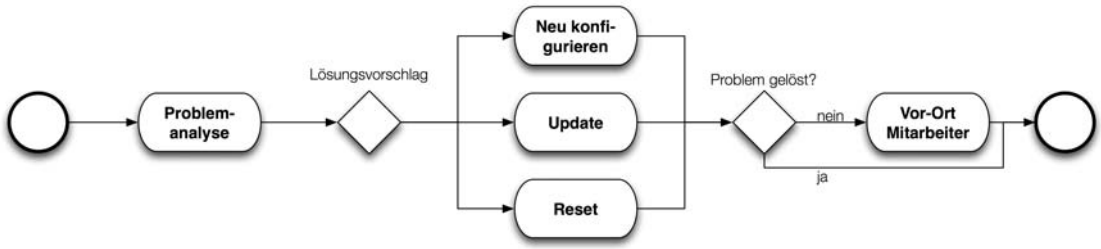


Abbildung 2.5: Ablauf eines Supportgespräches in einem Callcenter

Ein Beispiel für einen solchen *Administrative Workflow* ist in Abbildung 2.4 dargestellt. Sie zeigt den Ablauf eines Brandbekämpfungseinsatzes im Überblick[8]. Nach der Alarmierung durch die Einsatzleitzentrale rückt die Feuerwehr aus und überprüft nach Eintreffen am Einsatzort die Lage, um eine ordnungsgemäße Brandbekämpfung durchzuführen. Sobald die Wasserversorgung sichergestellt wurde, beginnt das Löschen des Brandes. Nach erfolgreichem Abschluss wird das Einsatzgebiet geräumt und die Feuerwehr rückt ab. Der Ablauf der Aktivitäten ist bei jedem Einsatz recht gleich. Jedoch kann es vorkommen, dass bei einem Flashover[9], dass parallel zur normalen Brandbekämpfung, weitere Maßnahmen eingeleitet werden müssen. Das bedeutet, dass vom normalen Ablauf abgewichen werden muss.

Ein weiteres Beispiel ist ein Workflow in einem Callcenter[10] eines Telekommunikationsunternehmens. Ruft ein Kunde wegen eines Problems mit seinem DSL-Router an, versucht der Mitarbeiter des Callcenters, das Problem des Kunden zu erfassen (`Problemanalyse`). Auf Basis dieser Informationen kann der Mitarbeiter einen von drei unterschiedlichen Lösungsvorschlägen – `Neu konfigurieren`, `Firmware-Update` oder `Reset` – unterbreiten. Ist dieser nicht erfolgreich, weißt er einen Vor-Ort Mitarbeiter an, sich dem Problem anzunehmen. An diesem Beispiel werden zwei Probleme deutlich. Zum einen wird ein solcher Workflow schnell komplex, da alle Fehlermöglichkeiten eines Routers in einem Workflow abgebildet werden müssen. Dies führt zwangsläufig zu einer hohen Anzahl an Verzweigungen. Zum anderen muss der Workflow an neue Gegebenheiten angepasst werden, da stetig neue Produkte des Telekommunikationsunternehmens eingepflegt werden müssen.

Ad-hoc-Workflows Die dritte Klasse beinhaltet Prozesse, die sich durch eine hohe Flexibilität auszeichnen. Im Gegensatz zu den beiden vorangegangenen Klassen kann hier

[8] Der Einsatz eines WfMS zur Kontrolle und Steuerung bei der realen Brandbekämpfung ist nicht möglich. Jedoch gibt es Einsatzmöglichkeiten bei der Ausbildung. Durch die permanente Eingabe aktueller Situationsdaten in das WfMS kann es Handlungsvorschläge unterbreiten. Dies kann unerfahrenen Brandbekämpfern bei Übungseinsätzen eine hilfreiche Unterstützung bieten.

[9] Im Deutschen auch Durchzündung genannt, ist eine Phase innerhalb eines Brandereignisses und bezeichnet den schlagartigen Übergang eines Brandes von der Entstehungsphase hin zur Vollbrandphase [52].

[10] Stark vereinfachtes Beispiel.

keine feste Struktur festgelegt werden. Beispiele hierfür sind der vorgestellte Theaterprozess, Vertriebsprozesse oder Personalbetreuung. Durch diese hohe Fluktuation sind ist ein Einsatz von Workflows und Workflow Management Systemen, wie sie im folgenden Abschnitt vorgestellt werden, bei Ad-hoc-Workflows nicht sinnvoll[11].

Eine weitere Unterteilung existiert bezüglich der *Ausführung der Workflows*. So können Workflows auf einem einzelnen WfMS oder auf mehreren WfMS, d. h. verteilt, ausgeführt werden. Diese *Verteilung* wird zumeist aus Gründen der Lastverteilung vorgenommen.

2.2 Workflow Management Systeme

Ausgangspunkt für jedes WfMS sind die *Geschäftsprozesse* eines Unternehmens. Während der *Geschäftsprozessmodellierung* werden diese, zum Zweck einer automatisierten Ausführung, hinreichend detailliert in einem *Workflow* überführt. Der Workflow kann aus *automatisiert* und *manuell* ausführbaren *Aktivitäten* aufgebaut sein.

Die Ausgestaltung der Workflows hängt stark vom WfMS ab, das die Workflow-Beschreibungssprache und die verschiedenen Funktionen vorgibt. Eine einheitliche und vollständige Definition für *Workflow Management Systeme* ist, wie beim Geschäftsprozessmanangement, aufgrund ihrer unterschiedlichen Einsatzgebiete und der daraus resultierenden, teils divergenten, Anforderungen schwer. Die Definition der WfMC ist dabei die am häufigsten zitierte:

> „A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.“ [106]

Die Definition kann nur als eine unter vielen verstanden und nicht als allgemeingültig betrachtet werden, da sie den Fokus auf den Begriff der Workflow-Engine und der Interpretation der Prozessbeschreibung beschränkt. Ähnliche Ansätze, welche auch die Workflow-Engine in den Mittelpunkt eines Workflow Management Systems stellen sind weit verbreitet, stellen dennoch nur eine mögliche Definitionsmöglichkeit dar.

Trotz dieser nicht eingeschränkt allgemeingültigen Definition herrscht für Aufbau und die Funktionsweise von Workflow Management Systemen Einigkeit.

2.2.1 Grundaufbau von Workflow Management Systemen

Auch wenn die Definition für ein Workflow Management System der WfMC nicht allgemeingültig ist, nennt sie doch die notwendigen Grundaufgaben: *Definition* (defines), *Erzeugung* (creates) und *Ausführung* (manages the execution). Diese Aufgabenbereiche beziehen sich auf die Automatisierung von Geschäftsprozessen und stehen somit im Zentrum dieser Arbeit. Des Weiteren zählen laut WfMC die *Überwachung*, *Verwaltung* und *Analyse* der Workflow-Ausführung sowie die *Integration* von Anwendungen zu den weiteren Aufgaben eines Workflow Management Systems. Im Folgenden soll auf den Inhalt der einzelnen Aufgabenbereiche genauer eingegangen werden.

[11] Ansätze für Ad-hoc-Workflows werden in Abschnitt 2.3.3 vorgestellt.

Definition Bevor ein Workflow ausgeführt werden kann, muss er vollständig beschrieben sein. Eine der primären Anforderungen an ein WfMS ist deshalb Werkzeuge zur Verfügung zu stellen, welche die Erstellung und Analyse von Workflow-Beschreibungen ermöglichen.

Erzeugung und Ausführung Die Hauptaufgabe von Workflow Management Systemen liegt in der Erzeugung, Verwaltung und Ausführung von Workflowinstanzen. Bei der Ausführung werden die für die Abarbeitung notwendigen Aktivitäten initiiert. Die Interpretation ersetzt in den meisten WfMS die Übersetzung der Workflowdefinition in Programm-Code. Interpretation heißt, dass das Ablaufschema Schritt für Schritt abgearbeitet wird. Dabei werden benötigte Anwendungsmodule mit entsprechenden Eingabe- und Steuerdaten aufgerufen und ausgeführt. Interpreter zeichnen sich durch eine höhere Laufzeitflexibilität aus, benötigen aber teils mehr Ressourcen (Arbeitsspeicher und Prozessor-Auslastung). In einigen wenigen Bereichen ist es üblich, die Workflowdefinition in eine ausführbare Form zu überführen, da beispielsweise in einem eingebetteten System nur wenige Ressourcen zur Verfügung stehen. Kompilierte Workflow-Beschreibungen mit einer einhergehenden Optimierung auf die jeweilige Plattform sind die Lösung für dieses Problem. Als Beispiele hierfür finden sich *WS-BPEL Process Compiler* [26] oder das Projekt *BPEL to Java* [171].

Integration von Anwendungen Textverarbeitung gehört nicht zu den primären Aufgaben eines WfMS. Für Unternehmen ist die Verarbeitung von Textdokumenten jedoch auch in Workflows eine wichtige Anforderung. Deshalb ist eine Integration von Anwendungen, welche nicht Bestandteil des Workflow Management Systems sind, erforderlich. Diese Integration muss derart gestaltet sein, dass sowohl Daten an externe Anwendungen übergeben, als auch von ihr empfangen werden können. Dabei lässt sich zwischen interaktiven und automatisierten Anwendungen unterscheiden. Interaktive Anwendungen entsprechen einem manuellen *Work Item*. Diese manuell auszuführende Aktivität wird auch *User Task* genannt und reicht vom Ausfüllen einer Tabelle bis hin zu einer Unterschrift. Automatisierte Anwendungen werden vom WfMS ausgeführt.

Verwaltung von Worklists Während des Betriebes eines WfMS werden viele Workflows gleichzeitig ausgeführt. Es ist deshalb nicht ungewöhnlich, dass einem Bearbeiter mehrere manuelle Aktivitäten gleichzeitig zugewiesen werden. Da es nicht möglich ist, diese auch gleichzeitig zu bearbeiten, soll ein WfMS diese Aufgaben in einer Liste – *Worklist* – sammeln. Dadurch ist es möglich, einen Bearbeiter über die von ihm zu erledigenden Work Items zu informieren.

Überwachung, Verwaltung, Analyse Zur Verbesserung der Workflows und des Betriebes des WfMS selbst, ist es erforderlich, dass das WfMS Werkzeuge zur Überwachung, Verwaltung und Analyse von Workflows während ihrer Ausführung zur Verfügung stellt.

Für die Umsetzung hat die WfMC ein Referenzmodell entwickelt, dass die an ein WfMS gestellten Aufgaben auf unterschiedliche Komponenten verteilt. Darüber hinaus wurden Schnittstellen definiert, die die einzelnen Komponenten untereinander verbinden, wie in Abbildung 2.6 dargestellt.

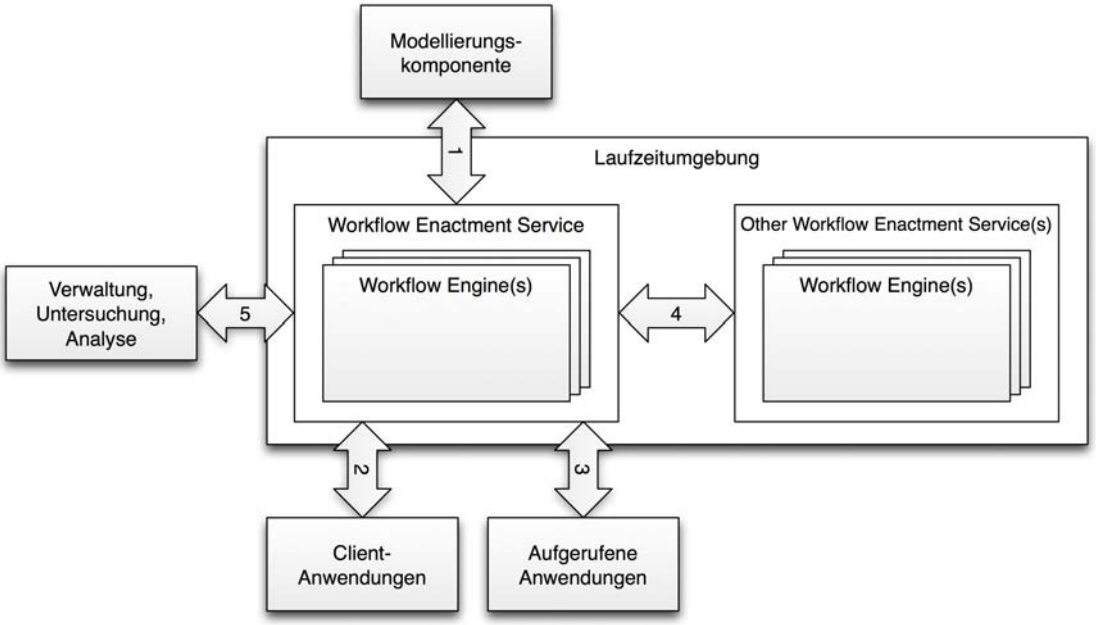


Abbildung 2.6: Workflowreferenzmodell (nach [84])

Die Kernaufgaben werden durch die Komponenten *Modellierung* und die *Laufzeitumgebung* abgebildet. Die Laufzeitumgebung setzt sich dabei aus den Enactment Services zusammen, die die Laufzeitumgebung für die Workflows darstellen und somit dem Konzept der Workflow-Engine entsprechen. Jede dieser Engines kann auf ein bestimmtes Aufgabengebiet spezialisiert sein. Datenaustausch, Synchronisation und die Ablaufkontrolle zwischen einzelne Engines erfolgt mittels der WAPI (Workflow Application Programming Interface and Interchange), einer speziell von der WfMC entworfene Schnittstelle.

2.2.2 Workflow-Perspektiven

Das Referenzmodell stellt aber nur eine Architekturform für den Aufbau des Workflow Management Systems dar. Ein weiterer Aspekt bei der Betrachtung des Geschäftsprozessmanagement ist die Ausgestaltung der Workflow-Beschreibungssprache. Ein entsprechend allgemeines Sprachmodell beschreibt Jablonski in [92, 91, 93]. Aufgrund seines Aufbaus erlaubt es eine einheitliche, systematische und umfassende Betrachtung von WFMS.

Jablonski kategorisiert die elementaren Aspekte einer Workflow-Beschreibungssprache in Perspektiven. Mithilfe dieser Perspektiven können die allgemeinen Anforderungen an Workflows beschrieben werden. In [93] werden fünf essentielle Perspektiven aufgezeigt, die im Folgenden vorgestellt werden.

Funktionale Perspektive Geschäftsprozesse werden ausgeführt, um eine bestimmte Arbeit zu bewältigen oder ein Ziel zu erreichen. Dies erfordert die Abarbeitung einzelner Arbeitsschritte (Aktivitäten), die auch zu größeren Fragmenten (Teilworkflows) zusammengefasst werden können.

Informationsbezogene Perspektive Neben den Aktivitäten eines Workflow sind auch die zu verarbeitenden Daten zu betrachten. Mithilfe der Informationsbezogenen Perspektive lassen sich die zu konsumierenden und zu produzierenden Daten (z. B. Dokumente) sowie der Datenfluss beschrieben. Dafür stehen u. a. die Beschreibungskonzepte: Datenstruktur, Wertebereich, Format, Speicherort, Zugriffsrechte und Datenflusskonstrukte zur Verfügung.

Verhaltensbezogene Perspektive Nachdem sowohl Daten als auch die Aktivitäten des Workflows beschrieben werden können, gilt es eine Ordnung zwischen diesen Elementen zu beschreiben. Die Verhaltensbezogene Perspektive dient der Beschreibung des Kontrollflusses zwischen den Unterprozessen und Aktivitäten eines Workflows und legt somit die Zusammenhänge zwischen den Work Items fest. Kontrollflusskonstrukte, Vor -und Nachbedingungen, Quell- und Zielaktivität, Anfangs- und Endaktivitäten und Übergangsbedingungen sind die von dieser Perspektive bereitgestellten Konzepte, die am Beispiel von BPEL auch in Abschnitt 2.1 vorgestellt wurden.

Operationale Perspektive Geschäftsprozesse weisen oft Arbeitsschritte auf, die nur mithilfe externer Programme durchgeführt werden können. Die Kommunikation zwischen WfMS und der externen Anwendung beschreibt die operationale Perspektive. Dafür werden Konzepte zur Beschreibung der Anwendungen (Speicherort, Anwendungsschnittstelle, Aufrufkonventionen) und für die Zuordnung dieser Anwendungen zu Aktivitäten (Konstrukte für den Anwendungsaufruf) bereitgestellt.

Organisatorische Perspektive Gerade für Unternehmen ist die Integration des WfMS in die bestehende IT-Infrastruktur wichtig. Deshalb ist es notwendig, auch organisationsbezogene Inhalte in der Workflow-Beschreibungssprache zu definieren. Zum einen fällt hierunter die Definition eines Organisationsmodells, zum anderen ist das Rechtemanagement umzusetzen. Letzteres regelt, wer die verschiedenen Operationen eines Workflows ausführen kann beziehungsweise soll. Die organisationsbezogene Perspektive gibt somit Auskunft darüber, wer einen Workflow oder eine Anwendung auszuführen hat. Für die Beschreibung dieser Perspektive werden u. a. Konzepte wie: Aufbauorganisationsmodell, Rollen und Rollenzuordnung, Bearbeiter und Rechte verwendet.

Der Kontrollfluss, d. h. die Verhaltensperspektive, ist der zentrale Aspekt bei der Beschreibung von Workflows[12]. Doch die Beschreibung des Kontrollflusses kann für völlig unterschiedliche Workflows gleich sein. Was sie unterscheidet, ist, die auf den *Kontext* des einzelnen Workflows bezogene, Ausgestaltung der vier – funktional, organisatorisch, operational, informational – weiteren Perspektiven. Diese werden im Folgenden als *kontextbezogene Perspektiven* bezeichnet.

Weitere Perspektiven Über die fünf Basisperspektiven hinaus werden zusätzliche Perspektiven erwähnt. Die *Kausale Perspektive* [93] trägt der Erkenntnis Rechnung, dass

[12] Die datenflussorientierte Sicht, wie sie in der strukturierten Analyse zum Einsatz kommt, wird in dieser Arbeit nicht betrachtet.

die Ausführung eines Workflows zumeist von Unternehmensvorschriften, Business Rules [143], abhängt. Mit ihr lassen sich die Abhängigkeit zwischen Erfüllung der Unternehmensvorschriften und der Ausführung von Workflows beschreiben. In der Praxis oft anzutreffen ist die *Historische Perspektive* [93]. Ihre Aufgabe ist es festzuhalten, welcher Workflow, wann und von wem ausgeführt wurde und welche Daten dabei konsumiert und produziert wurden. Sie entspricht demnach einem vollständigen Logging. Aufgabe der *Transaktionalen Perspektive* ist die Beschreibung von Bedingungen (constraints) zur Gewährleistung der Eigenschaften Atomarizität (A), Konsistenz (C), Isolation (I) und Dauerhaftigkeit (D) zur Sicherung der Datenintegrität (siehe [192, 92]). Bei der *Qualitätsperspektive* [93] steht die Effizienz und Effektivität bei der Ausführung von Workflows im Vordergrund. Dabei werden vor allem die Ausführungszeit und Ausführungskosten betrachtet.

2.2.3 Klassifikation von Rekonfiguration in WfMS

Bevor über verschiedene Formen von Rekonfiguration gesprochen werden kann, ist es wichtig zu verstehen, *wie* und *warum* die Perspektiven bzw. Aspekte von Workflows geändert werden können. Dabei ist zu beachten, dass *Rekonfigurationen* bzw. *Änderungen* grundsätzlich auf zwei Arten betrachtet werden können:

- Die Reaktion auf unerwartete Ereignisse, die zumeist auf eine unvollständige Spezifikation zurückzuführen sind. Dies ist dem Umstand geschuldet, dass ähnlich wie beim Testen, nicht alle Situationen vollständig betrachtet werden können.
- Die Reaktion auf veränderte Bedingungen (z. B. Geschäftsregeln), die eine Abänderung der Spezifikationen notwendig machen.

In [182] zeigen van der Aalst und Jablonski sechs[13] fundamentale Kriterien auf, die eine Änderung in WfMS charakterisieren, die im Folgenden vorgestellt werden sollen.

Grund der Änderungen Änderungen können ihre Ursache *außerhalb* oder *innerhalb* des betreffenden Systems haben. Änderungen von *außerhalb* haben drei Hauptumstände. Zum einen können veränderte oder neue gesetzliche Vorschriften Teile oder ganze Workflows betreffen, die entsprechend geändert werden müssen. Zum anderen kann eine Verlagerung oder Erweiterung der geschäftlichen Aktivitäten Veränderungen verursachen. Dies beinhaltet Änderungswünsche von Kunden oder die Einführung von neuen Produkten. Des Weiteren ist jedes Softwaresystem den Veränderungen der IT-Landschaft ausgesetzt. Workflows oder das gesamte WfMS müssen sich diesen Gegebenheiten auch anpassen. Dies ist z. B. der Fall, wenn Servicepartner ihre Infrastruktur ändern, andere Austauschformate nutzen oder bei Ausfällen von Diensten. *Innere* Änderungen sind vor allem logische Designfehler der Workflows. Diese können in unbrauchbaren Workflows oder gar Deadlocks enden.

Änderungszeitspanne Änderungen in WfMS können *zeitlich befristet* und nur auf bestimmte Elemente oder Workflows beschränkt oder *evolutionär* sein. *Evolutionäre Änderungen* sind zumeist sorgfältig geplant, permanent und gelten für alle neuen Workflowinstanzen. Bestehende Instanzen können ebenfalls betroffen sein, müssen allerdings zur

[13] Das sechste Kriterium – Umgang mit bestehenden Instanzen – wird in Abschnitt 2.3.1 betrachtet.

Laufzeit in den gewünschten Zustand migriert werden, was nicht unter allen Umständen möglich ist (siehe Abschnitt 2.3.1). *Zeitliche befristete Änderungen* werden vor allem bei Einzelfällen eingesetzt, die nicht für andere Fälle wiederverwendet werden können. Als Beispiel für eine derart zeitlich befristete Änderung ist der Einlass von VIP-Gästen und das Einhergehen mit dem Überspringen einer oder mehrerer Aktivitäten. Diese spontanen Änderungen sind nicht vorhersehbar, andernfalls wäre ein entsprechendes Design des Workflows möglich, das diese Varianten abdeckt.

Betroffene Perspektiven Änderungen haben verschiedene Auswirkungen auf die Workflowperspektiven, die in Abschnitt 2.2.2 vorgestellt wurden:

- Die Änderungen die *Verhaltensperspektive* betreffend, sind das Hinzufügen oder Löschen einzelner Aktivitäten, das Auflösen von Parallelität hin zu einer seriellen Abarbeitung.
- In der *Organisatorischen Perspektive* betreffen die Änderungen die Organisationsstruktur, Ressourcen und die Benutzerrollen[14]. Diese können hinzugefügt oder gelöscht werden.
- Die *Informationsbezogene Perspektive* beschreibt die möglichen Datenstrukturen und Typen, die bei Änderungen ebenfalls betroffen sind. Mögliche Änderungen sind hier Typänderung einer Variable, die Modifikation und das Hinzufügen von Datenstrukturen und Typen.
- In der *Operationalen Perspektive* werden neue Dienste (z. B. Webservices) oder Applikationen hinzugefügt, gelöscht oder die Aufrufparameter neu konfiguriert.

Art der Änderung Für alle Perspektiven zusammenfassend lässt sich feststellen, vier Arten von Änderungen möglich sind: Insert, Delete, Replace und Update (auch Re-Link genannt):

- Beim *Insert* werden neue Elemente den jeweiligen Perspektiven hinzugefügt.
- *Delete* löscht verschiedene Elemente der Perspektiven.
- *Replace* stellt die Verbindung beider vorangegangener Änderungen dar und wird oft verwandt, um eine fehlerhafte oder ausgefallene Funktion (z. B. Webservice) auszutauschen.
- Beim *Update* oder *Re-Link* werden beispielsweise Elemente verschiedener Perspektiven neu miteinander verbunden.

Zeitpunkt der Änderung Für die Behandlung von Änderungen gibt es grundsätzlich zwei Möglichkeiten. Die erste – zum Eintrittszeitpunkt (*entry time*) – erlaubt keine Modifikationen an bestehenden Workflowinstanzen, d. h. wird ein Workflow gestartet, ist er für den gesamten Lebenszyklus unveränderbar. Änderungen sind demzufolge statischer Natur und betreffen alleinig das Workflowmodell. Die zweite Möglichkeit besteht in der *On-the-fly* Änderung, d. h. bestehende Instanzen können jederzeit geändert werden.

[14] Role Based Access Control.

In [128, 154] wird eine vergleichbare Taxonomie vorgestellt. Sie führt zu den beschriebenen Kriterien die *Schnelligkeit* (Swiftness) als weiteres Kriterium hinzu. Die Reaktion auf eine Veränderung kann *sofort* (immediate) oder *verzögert* (deferred) stattfinden. Ersteres wird vor allem bei instanzbezogenen Änderungen verwendet, während die verzögerte Reaktion eher bei evolutionären Änderungen Verwendung findet.

2.3 Adaptionsverfahren

Am Beispiel des Ticketsystems für das Theater lassen sich die Probleme der Adaption von Workflows verdeutlichen. Die kurzfristige Entscheidung, VIP Reservierungen zu vergeben, entspricht einer Adaption mit einer kurzen Zeitspanne, die verschiedene Perspektiven einer Workflowinstanz betrifft. Eine Änderung am gesamten Verkaufsprozess, beispielsweise die Integration von Rabatten, ist dahingehend eine langfristige Entscheidung, die alle kommenden Workflowinstanzen beeinflusst. Die Bandbreite von Veränderungen in Workflows ist demnach groß. Die daraus resultierenden, zu beachtenden Konsequenzen sind ebenso vielfältig. Seit langem wurde deshalb an änderbaren Workflows geforscht, was zu einer Vielzahl von Lösungskonzepten führte. Eine implementierungsnahe Klassifikation hierfür findet sich in [156]:

- *Design*: Strategien zur Behandlung von (unerwarteten) Änderungen zur Designzeit.
- *Deviation (Abweichen)*: Strategien zur Behandlung minimaler Veränderungen zur Laufzeit.
- *Underspecification (Unterspezifikation)*: Die Ausführungsstrategie wird zur Laufzeit ausgewählt (auch *Late Binding* genannt).
- *Change (Änderung)*: Veränderungen der Struktur des Workflows (temporär oder permanent).

Eine andere Klassifizierung beschreibt Sadiq in [150]. Er unterteilt den Raum der potentiellen Lösungen in drei, von Implementierungen unabhängige, große Kategorien. Dazu zählen sowohl die *dynamischen* Workflows, die nicht im laufenden Betrieb geändert werden, als auch die *flexiblen* (hier speziell die Ad-hoc) Workflows, deren Ablauf sich für jede Instanz unterscheidet. Eine Zwischenform stellen die *adaptiven Workflows* dar. Ihre Ablaufstruktur kann unter bestimmten Bedingungen zur Laufzeit angepasst werden.

Beide Klassifikationen können aufeinander abgebildet werden:

- Evolutionäre Änderungen: *Design* und *Deviation* ↔ *dynamisch, evolutionäre Workflows*
- Dynamische Änderungen: *Change* ↔ *adaptive Workflows*
- Änderungen durch Unter- und Überspezifikation: *Underspecification* ↔ *flexible Workflows*

Diese drei Klassen stehen im Mittelpunkt der nachfolgenden Betrachtungen und der anschließenden Zielsetzung.

2.3.1 Evolutionäre Adaptionen

Die erste Klasse stellen die *evolutionären* Rekonfigurationen dar. Hierbei wird versucht, sowohl auf kleine Änderungen, als auch komplette Umstellungen im Arbeitsablauf zu reagieren. Die Dynamik besteht hier in einer Evolution der Workflow-Beschreibungen. Die Workflowmodelle werden statisch an die neuen Gegebenheiten angepasst, wobei die Instanzen davon nicht direkt betroffen sind.

Der Modifikation von bestehenden Workflowmodellen wurde in der Forschungslandschaft viel Aufmerksamkeit geschenkt. Sie ist vor allem bekannt unter dem Begriff *Workflow Evolution* [98, 38]. Bei diesem Vorgang wird die Struktur des zentralen Workflowmodells dauerhaft verändert, d. h. es wird eine neue Version des Workflows im Repository (Speicher) abgelegt. Dies ist unkompliziert, solang keine aktive Instanz des originalen Workflowmodells während der Änderungsoperation ausgeführt wird.

Diese Instanzen sind insofern von der Änderung betroffen, da das referenzierte Basismodell sich verändert. Demnach müssen die Workflowinstanzen gesondert betrachtet werden. In [38] werden die theoretischen Grundlagen für die Evolutionsoperationen gelegt. Casati schlägt hierfür drei Handlungsmöglichkeiten vor. Mit der Option *Abort* werden alle laufenden Instanzen des Workflowmodells sofort beendet. Dies löst das Problem der Evolution von Instanzen insofern, dass keine Instanzen migriert werden müssen. Vorsichtiger kann mit der Option *Flush* agiert werden. Hierbei werden alle laufenden Instanzen regulär beendet. Bis dies geschehen ist, können keine neuen Instanzen des Workflows angelegt werden. Dadurch wird vermieden, dass mehrere inkonsistente Versionen des Workflows ausgeführt werden. Die dritte Option – *Progressive* – ist gleichzeitig die schwierigste. Hierbei wird für jede Instanz abhängig vom Ausführungszustand einzeln entschieden, wie die Evolution stattfindet. Dafür stehen mehrere Methoden, von Abbrechen bis hin zur Migration, zur Verfügung. Es ist auch möglich, dass mehrere Versionen des Workflowmodells gleichzeitig ausgeführt werden. Weitere Aspekte dieser *Workflow Evolution* wurden umfangreich untersucht, wobei [38, 98, 147] nur einen Teil der Untersuchungen darstellen.

Nach [182] bestehen fünf Möglichkeiten, wie mit bestehenden Instanzen umzugehen ist:

- Forward Recovery: Die alten Instanzen werden abgebrochen. Hierfür müssen organisatorische Maßnahmen getroffen werden, die diese offenen Fälle abhandeln.
- Backward Recovery: Die alten Instanzen werden abgebrochen und anschießend mit der neuen Workflow-Definition neu gestartet.
- Proceed: Die bestehenden Instanzen werden mit der alten Workflowdefinition fortgeführt und nur neue Instanzen erhalten die neue Definition. Somit gibt es mehrere Versionen des Workflows zur gleichen Zeit.
- Transfer: Die alten Instanzen werden an die neue Workflow-Definition übertragen.
- Detour: Für zeitlich befristete Veränderungen sind *on-the-fly* Änderungen als Umgehung möglich. Auch hier sind, jedoch zeitlich begrenzt, mehrere Versionen des Workflows zur gleichen Zeit im System.

Eine weitere, grundlegende Möglichkeit, evolutionäre Workflows zu definieren, besteht durch die *Entscheidungsknoten* (siehe *If* in Abschnitt 2.1). Damit können Instanzen des gleichen Modells unterschiedliche Wege durch den Workflow vollziehen. Mit solchen Knoten hat der Designer eines Workflows die Möglichkeit Entscheidungspunkte

einzuführen, die eine gewisse Art der Laufzeitflexibilität erlauben. Dies wird dadurch erreicht, dass alle Abläufe in den Workflow eingebracht werden. Allerdings müssen alle Entscheidungspunkte zur Design-Zeit festgelegt werden und können nicht zur Laufzeit geändert werden. Ein spontanes Abweichen in unvorhergesehenen Situationen ist somit nicht möglich. Darüber hinaus müssen alle Varianten in einem Modell beschrieben werden. Wenn die Zahl der Variationen groß ist, kann das Modell auf lange Sicht nicht mehr gewartet werden.

2.3.2 Dynamische Adaptionen

Die evolutionären Workflows eignen sich bei lang geplanten Rekonfigurationen. Soll der Workflow an neue Gegebenheiten schnell angepasst werden, sind instanzbezogene Änderungen besser geeignet. Im Gegensatz zu den vorher betrachteten Evolutionen, werden hier unerwartete Umstände betrachtet, die Veränderungen implizieren. Auf diese muss eine WfMS reagieren können und die betroffenen Instanzen unter Umständen an die neuen Gegebenheiten anpassen. Zur Lösung dieses Problems entwickelten sich zwei Hauptlösungszweige. Im ersten werden die außergewöhnlichen Umstände als *Exceptions* (Ausnahmen) betrachtet, die durch spezielle Aktionen kompensiert werden müssen. Der zweite Lösungsansatz besteht darin, die außergewöhnlichen Umstände als Ereignisse anzusehen, die Veränderungen an der Ablaufstruktur der Workflowmodelle bewirken.

Exception Handling Der Ansatz des Exception Handling [165, 39, 148, 4] geht von der Annahme aus, dass ein Großteil aller außergewöhnlichen Umstände vorausgesehen werden kann. Diese müssen demnach nur geschickt behandelt werden. Das Workflowmodell beschreibt hierbei den Normalfall des Ablaufs. Eine Ausnahme können beispielsweise fehlerhafte Daten oder überschrittene Zeitschranken sein. Wie diese Ausnahmen behandelt werden, hängt stark von ihrem Typ ab. Ist bereits während der Erstellung des Workflows bekannt, dass die Daten an einer bestimmten Stelle inkonsistent sein können, kann durch geschicktes Design diese Ausnahme behandelt werden. Dieser Fall kann beispielsweise auftreten, nachdem ein Mitarbeiter Kundendaten erfasst hat und diese unvollständig sind. Wird dies vorausgesehen, kann die eingebettete (engl. embedded) Fehlerbehandlung [148] zum Einsatz kommen. Hierbei wird eine Fehlerbehandlungsroutine (engl. Compensation) in das Workflowmodell integriert, wie dies bspw. auch in Java durch die `Try-Catch`-Anweisung geschieht. Die Entdeckung des Fehlers erfolgt über einen Entscheidungsknoten, der eine Routine aufruft, wenn Daten fehlen. Problem dieses Ansatzes ist es, dass hierbei nur erwartete Ausnahmen betrachtet werden und das Workflowmodell zudem schnell zu groß wird. Dies ist dadurch zu erklären, dass jede Ausnahme kompensiert werden muss, wobei ihre Anzahl bereits bei kleinen Modellen groß werden kann.

Diese Unübersichtlichkeit des Embedded Exception Handling kann mithilfe des *Separated Exception Handlings* [148] vermieden werden. Hierbei werden die *Compansations* außerhalb des eigentlichen Workflowmodells modelliert. Zusätzlich wird eine Menge von Regeln angelegt, die das Exception Handling steuern. Jede Regel besteht aus einer Abbildung: $(W.P, E) \rightarrow C$. Abhängig von der Position des Tokens im Workflow $W.P$ und der aufgetretenen Ausnahme E wird eine Kompensation C ausgeführt. Dadurch können jedoch euerneut nur bekannte Ausnahmen behandelt werden.

Remove WorkItem	Remove Case	Remove All Cases
Suspend WorkItem	Suspend Case	Suspend All Cases
Continue WorkItem	Continue Case	Continue All Cases
Restart WorkItem	Force Complete WorkItem	Force Fail WorkItem

Tabelle 2.1: Exception Handling Funktionen (nach [4])

Zur Behandlung von Ausnahmen reichen die Basisaktivitäten (siehe Abschnitt 2.1) von Workflow-Beschreibungssprachen nicht aus. Es sind deshalb neue Funktionen notwendig. In [4] wird ein umfassender Überblick gegeben, der in Tabelle 2.1 dargestellt ist. Diese Muster unterstützt z. B. das WfMS YAWL [181]. Die Adaptionen beschränken sich dabei auf die jeweilige Aktivität oder eine Gruppe von Aktivitäten. Dieses kann gelöscht, neu gestartet oder weiterverfolgt werden. YAWL unterstützt drei weitere Muster[15]. Dies ist zum einen *Rollback task*, wobei die Ausführung der Aktion rückgängig gemacht wird, um sie erneut zu starten. Zum anderen aber auch zwei Muster zur Behandlung von Ressourcen. In der organisatorischen Sicht auf Workflows werden sowohl die Nutzer, als auch künstliche Ressourcen beschrieben. Ein Ausfall eines Bearbeiters für eine bestimmte Aufgabe muss ebenso behandelt werden, wie Fehler in den Daten oder Funktionsstörungen. Deshalb existiert in YAWL *Reallocate current work item* und *Reoffer current work item*. Beide Muster behandeln ressourcenbezogene Ausnahmen.

Adaptionsmuster Die Änderungen, die durch das Exception Handling am Workflowablauf vorgenommen werden sind minimal und beschränken sich auf die Wiederholung bzw. auf das Aufrufen alternativer Aktivitäten. In [191] beschreiben Weber et al. 14 essentielle Adaptionsmuster für Workflowinstanzen, die weit darüber hinausgehen. Sie decken dabei alle bekannten Formen der Strukturänderungen für Workflows ab, wie sie in Abschnitt 2.2.3 eingeführt wurden. Sie sind weit komplexer, als die Basisoperationen für das Hinzufügen bzw. Entfernen einer Kante bzw. eines Knoten oder das Verschieben einer Kante. Tabelle 2.2[16] stellt diese, geordnet nach verschiedenen Kategorien dar.

In einer nachgelagerten Untersuchung aus [191] verglichen die Autoren verschiedene WfMS hinsichtlich ihrer Unterstützung der Adaptionsmuster. Dabei betrachteten sie sowohl die Anwendbarkeit auf laufende Instanzen, als auch die Anwendung auf Workflowmodelle. Unter den untersuchten WfMS befanden sich viele akademische: ADEPT2, WIDE, WASA2, HOON, MOVE, CAKE2, YAWL, PoF sowie die kommerziellen Vertreter Flower und Staffware. Während alle (bis auf ADEPT2) die Basisoperationen[17] beherrschten, unterstützt nur ADEPT2[18] eine größere Anzahl von Adaptionen, sowohl für die Workflowmodelle, als auch für einzelne Workflowinstanzen. Ausnahmen stellen die Muster AP4, 5, 10, 14 dar. Die Unterstützung der Adaptionsmuster bei allen anderen Betrachteten ist zumeist nicht gegeben. Lediglich WIDE unterstützt 5 von 14 Mustern bei der evolutionären Adaption. Allerdings ist es nicht nur wichtig, dass eine Instanz adaptiert werden kann. Zugleich ist auch die Steuerung der Adaption von entscheidender Bedeutung. Dazu zählt vor allem die Entscheidung, wann eine Adaption durchgeführt werden soll.

[15] http://www.workflowpatterns.com/patterns/exception/considerations.php (zuletzt aufgerufen am 25.02.2011).
[16] Fragement bedeutet in [191] eine Ablaufstruktur mit mindestens einer Aktivität.
[17] Einfügen und Löschen eines Knotens bzw. einer Kante und das Verschieben einer Kante.
[18] Heute AristaFlow genannt.

Hinzufügen und Entfernen	
AP1: Insert Fragment	Einfügen von Fragmenten
AP2: Delete Fragment	Löschen von Fragmenten
Bewegen von Fragmenten	
AP3: Move Fragment	Versetzen eines Fragments
AP4: Replace Fragment	Ersetzen eines Fragments
AP5: Swap Fragment	Vertauschen von Fragmenten
AP14: Copy Fragment	Kopieren eines Fragments
Unterprozesse	
AP6: Extract Sub Process	Extrahieren in einen Unterprozess
AP7: Inline Sub Process	Einfügen eines Unterprozesses
Komplexe Kontrollflussänderungen	
AP8: Embed Fragment in Loop	Fragment in Schleife einfügen
AP9: Parallelize Fragment	Parallelisierung von Aktivitäten
AP10: Embed Fragment in Conditional Branch	Einfügen in einen bedingten Pfad
AP11: Add Control Dependency	Einfügen einer Abhängigkeit
AP12: Remove Control Dependency	Einfügen einer Abhängigkeit
Bedingungen	
AP13: Update Condition	Verändern einer Bedingung

Tabelle 2.2: Adaptionsmuster nach Weber [191]

Unerwartete Ausnahmen Klassische Exception Handling Strategien können keine unerwarteten Fehler behandeln, sondern bedingen eine explizite Fehlermodellierung. Deshalb gilt es, ein WfMS-System sowie die Workflowmodelle derart zu konstruieren, dass sie auf diese trotzdem reagieren können. Dieses kann nicht während der Designphase geschehen, denn hier sind die Ausnahmen nicht bekannt. Es muss demnach zur Laufzeit reagiert werden. Das WfMS *AgentWork* [47, 111] ist ein solches System, das auf unerwartete Umstände zur Laufzeit mit regelbasierten Workflow-Adaptionen reagiert. Entwickelt wurde diese auf Basis der Workflow-Engine *ADEPT* bzw. *Adeptflex* [131] an der Universität Ulm.

ADEPTflex besitzt durch die Unterstützung vieler Adaptionsmuster eine große Veränderungsfähigkeit. Mit ihnen ist es möglich, Workflowmodelle zur Laufzeit anzupassen, wobei zum einen die Korrektheit garantiert und zum anderen die Konsistenz der Instanz gewahrt werden soll. Dazu besitzt AgentWork spezielle ECA-Regeln – Event-Condition-Action [90, 66, 37]. Diese Aktionsregeln bestehen aus drei Teilen: (1) einem Ereignis, auf das reagiert werden soll; (2) einer Bedingung, die die Voraussetzungen für die Anwendbarkeit der Regel definiert und (3) einer Aktion, die ausgeführt wird, wenn ein Ereignis unter bestimmten Bedingungen auftritt. Im Falle von AgentWork beschreiben diese Regeln, welches Adaptionsmuster ausgeführt werden soll, wenn ein Ereignis auftritt. Dies kann beispielsweise das Löschen oder Hinzufügen einer Aktivität sein.

Im Vergleich zu klassischen ECA-Regeln ist der Aufbau an die Domäne angepasst. Sie besitzen weiterhin die bereits beschriebenen Standardkomponenten – Ereignis, Bedingung, Aktion – jedoch kann damit nicht die zeitliche Gültigkeit der Regel ausgedrückt werden, was für diesen Ansatz wichtig ist. Aus diesem Grund wurde die Regel um zeitliche Komponente erweitert:

```
WHEN   Event
  WITH   Condition
  THEN   Control Action
  VALID-TIME   Time Period
```

Der WHEN/WITH-Block beschreibt das Ereignis und Bedingungen, die eine Adaption notwendig machen. Der THEN-Abschnitt repräsentiert die Adaption (control action genannt), die bei Übereinstimmung von WHEN/WITH ausgeführt wird. Die neue Komponente VALID-TIME legt den Zeitraum fest, in dem die Adaption gültig ist. Im konkreten Beispiel stellt sich dies folgendermaßen dar (aus [111]):

```
WHEN   new finding of patient P
  WITH   leukocyte count < 1000
  THEN   drop drug Etoposid for P
  VALID-TIME   during the next seven days
```

Fällt bei einem Patienten P die Leukozyten-Zahl unter 1000 (WHEN/WITH), wird das Medikament *Etoposid* für Patient P für die nächsten sieben Tage (VALID-TIME) abgesetzt (THEN). Die Umsetzung der Adaption erfolgt über *ActiveTFL* (Active Temporal Frame Logic) [111], die zudem teilweise vorausschauende Adaptionen ermöglicht. Dies ist möglich, da dem System die Gültigkeitsdauer einer jeden Adaption bekannt ist. Dadurch kann berechnet werden, welche anderen Fragmente des Workflows bei diesem Ereignis unter bestimmten Bedingungen betroffen sind.

AgentWork kann durch diesen Ansatz auf unerwartete Ausnahmen reagieren. Die ECA-Regeln bilden, wie beim *Separated Exception Handling*, eine eigene Regelebene, so dass die Adaptionsbeschreibungen unabhängig vom Workflowmodell angegeben werden können. Zusätzlich bieten die ECA-Regeln eine höhere Flexibilität, als die Abbildung des Exception Handlings. Dadurch können neue Ausnahmesituationen beschrieben werden. AgentWork kann dabei nicht auf unerwartete Ausnahmen reagieren. Es ist darauf angewiesen, dass sie nachträglich beschrieben werden. Auch kann AgentWork nicht auf unbekannte oder ähnliche Situationen reagieren und ist somit immer auf einen Administrator angewiesen.

Aspektorientierte Adaption Neben diesen Ansätzen existieren noch Ansätze, die ihren Ursprung bei Programmiersprachen haben. In der Objektorientierung gibt es das Konzept der *Aspekte*, die es ermöglichen, das Verhalten von Objekten zur Laufzeit zu verändern. Hierbei wird ein *Advice* (zumeist ein Codefragment) an einer bestimmten Stelle im Originalcode (*Joinpoint*) eingewoben. Unter welchen Voraussetzungen und in welcher Ausprägung dies geschieht, wird in einem Pointcut festgelegt. Darüber hinaus ist es beispielsweise möglich, bestehende Funktionen zu erweitern oder ggf. zu ersetzen. Diese Eigenschaften ermöglichen einen hohen Grad an Flexibilität, ohne dass der Originalcode angepasst werden müsste. Übertragen auf die Domäne der Workflows bedeutet es ebenfalls einen hohen Grad an Laufzeitflexibilität. Fällt beispielsweise während der Ausführung eines Workflows ein benötigter Webservice aus, ist es möglich, einen adäquaten Ersatz zur Laufzeit einzubinden. Dies geschieht durch das Einweben von

Code, der beispielsweise ein EJB[19] auf einem anderen Server aufruft, welcher gerade verfügbar ist. Dieser eingewobene Code wird anstelle des Originalcodes benutzt. Es wird hierbei zwar ebenfalls der Kontrollfluss umgeleitet, allerdings auf der Ebene einer einzelnen Aktivität. Während bei dem ursprünglichen Ansatz die Struktur des Workflows verändert wurde, betrifft es hier nur die Ausführung einer bestimmten Aufgabe.

Auf Basis dieses Programmierparadigmas wurden verschiedene aspektorientierte Lösungen [40, 45, 99, 155, 153] für dynamische Workflows entwickelt. Hierbei werden vor allem zwei Hauptprobleme von Workflows versucht zu lösen: (1) Fehlende Modularisierung und (2) Veränderungen des Kontrollflusses zur Laufzeit (siehe [40]). Der Lösungsansatz besteht darin dass verschiedene Aktivitäten, die in mehreren Workflows vorkommen (Cross-Cutting Concern), aus den Workflows zu entfernen und nur im Bedarfsfall wieder einzubinden. Dadurch wird zum einen die gewünschte Modularisierung erreicht, zum anderen kann durch das Aspektweben der Kontrollfluss geändert werden. Dabei können nicht nur einzelne Aktivitäten ausgetauscht werden, sondern auch Workflowfragmente miteinander verwoben werden. In [99] können die Fragmente mit *AndSplit, XorSplit, AndJoin* und *XorJoin* untereinander verknüpft werden. Da die verschiedenen Aspektweber Laufzeitadaptionen unterstützen können all diese Änderungen on-the-fly vorgenommen werden.

2.3.3 Adaption durch Unter- und Überspezifikation

Die dritte Lösungskategorie stellen Rekonfigurationen durch Unter- und Überspezifikation dar. Bei der *Überspezifikation* werden für jeden Workflow bereits beim Entwurf mehrere Varianten für die verschiedenen Aktivitäten festgelegt. Erst während der Ausführung wird sich für eine Variante entschieden. Durch diese Ansätze kann schnell auf Standardprobleme reagiert werden, unvorhergesehene Änderungen können nur bedingt abgedeckt werden. Schonenberg beschreibt diese Fähigkeit auch als *flexibility by selection* bzw. *flexibility by configuration* [156].

Die Idee der Unterspezifikation basiert auf dem Konzept von lose gekoppelten bzw. nicht vollständig definierten Aktivitäten. Dies bedeutet, dass entweder die Ablaufstruktur, wie beim *Late Binding*, nur lückenhaft beschrieben wurde oder gar für jede Instanz individuell berechnet wird, wie es bei den *Ad-hoc* und den *deklarativen* Workflows der Fall ist. Dadurch erreichen die Instanzen eine hohe Varianz in ihren Ausprägungen.

Ad-hoc Workflows Die extremste Ausprägung bilden die Ad-hoc Workflows. Während die bisher beschriebenen Workflows eine feste Struktur besitzen, die es anzupassen gilt, besitzen Ad-hoc Workflows keine feste Struktur. Dies ermöglicht ein hohes Maß an Flexibilität. Die Aktionen und ihre Ablaufreihenfolge werden erst beim Start oder gar erst während der Ausführung festgelegt [131]. Es gibt keine vordefinierte Route, die als Basis herangezogen werden kann [64, 49].

Computer Supported Cooperative Work-Systeme (CSCW) [113, 114, 35] sind Vertreter für derartige Ad-hoc-Workflows. Sie bieten eine Umgebung, in der Tätigkeiten einer Organisation entworfen und manuell ausgeführt werden. Allerdings wird keine Struktur für die Bearbeitung vorgegeben, so dass jeder Arbeitsablauf anders gestaltet werden muss. Aktivitäten in CSCW-Systemen werden jedoch nicht automatisch ausgeführt [35, 100],

[19] Enterprise JavaBeans; Dienstkomponente auf Basis von Java.

sondern bedingen immer eine manuelle Aktivierung. Sie stellen deshalb eine Sonderform der Ad-hoc Workflows dar.

Wenn die Auswahl der nächsten Aktionen nicht manuell geschieht, muss auf einer anderen Basis entschieden werden. In [14] wird dazu eine verteilte Regelmenge verwendet. Jede Instanz besitzt einen eigenen Satz von Regeln und wertet diesen anhand der derzeit verfügbaren Laufzeitdaten aus. Da ein solcher Ad-hoc Workflow keine Struktur besitzt, sondern nur Regeln, Daten und Logdaten, wird er auch nicht als Workflow, sondern als *Information Carrier* (INCA) bezeichnet. INCA sind bewegliche Einheiten, die auf verschiedenen Rechnerknoten innerhalb eines Netzwerks ausgeführt werden können. Jeder dieser Rechnerknoten besitzt ebenfalls Daten und Regeln, die bei der Berechnung der Ablaufsteuerung berücksichtigt werden. Es ist deshalb zu keinem Zeitpunkt möglich, Workflowmodelle abzuleiten. Es ist weder bekannt, *was* im Laufe des Workflows durchgeführt wird, noch *wo*[20]. Die dadurch erreichte Flexibilität ermöglicht es, dass Berechnungen auf überlasteten oder gar abgestürzten Rechnerknoten vermieden werden.

Des Weiteren können Ad-hoc-Workflows auch dadurch realisiert werden, indem die Struktur erst zum Zeitpunkt des Ausführungsbeginns festgelegt wird. Der Workflow besitzt anschließend einen definierten Ablaufplan und kann dementsprechend ausgeführt werden. In [100] wird dies als *just in time* bezeichnet, es ist aber auch bekannt unter dem Namen *Late modelling*. Die Ausführung entspricht den Workflows, es gibt jedoch zu Anfang kein geplantes Workflowmodell.

Ein weiterer Weg, Workflowmodelle zu konstruieren, wird in [78] gezeigt. In diesem Papier wird das ADONIS-System beschrieben, ein System, dass ein Workflowmodell aus einer gegebenen Menge von bereits abgeschlossenen Workflowinstanzen erzeugt. Durch die Protokollierung des Verhaltens einer Instanz, können alle Schritte nachvollzogen werden. Das Protokoll beschreibt die erfolgreiche Ausführung einer Instanz, wodurch es sich als positives Beispiel für Trainingsdatensätze eignet. Aus einer großen Menge dieser Datensätze können Rückschlüsse über das beste Verhalten in einer Situation gezogen werden. Dazu werden Techniken aus der künstlichen Intelligenz angewandt, um die Unsicherheit, die in jedem Log zu finden ist, herauszufiltern. Dieses System unterstützt keine klassischen Ad-hoc-Workflows, allerdings kann das System zur kontinuierlichen Erstellung neuer Modelle durch das Lernen aus alten Instanzen genutzt werden.

Deklarative Workflows Eine weiterführende Lösung für Ad-hoc-Workflows werden in [75, 177] *deklarative Workflows* aufgezeigt. Dabei wird versucht, Workflowmodelle auf Basis eines Constraint-Raumes zu synthetisieren. Diese Idee wurde durch das Projekt DECLARE [122] umgesetzt.

Die Beschreibung eines Workflows erfolgt hier nicht über einen Ablaufplan, sondern über ein Netz von Abhängigkeiten und Bedingungen. Dabei werden die Aktivitäten und Bedingungen für die Aktivierung beschrieben. Zusätzlich dazu werden kausale Abhängigkeiten zwischen den einzelnen Workflow-Aktivitäten angegeben. Der Datenfluss kann zudem durch Abhängigkeiten zwischen den Variablen definiert werden. Ein zentrales Konzept für jeden deklarativen Workflow ist das *Ziel*. Dieses gilt es, unter Einhaltung der Abhängigkeiten und Bedingungen zu erreichen. Dies kann z. B. durch einen Planer realisiert werden.

[20] Beispielsweise auf welchem Rechnerknoten.

In [177] werden Bedingungen auf Basis von Temporal-Logik beschrieben. Die Bedingungen sind in *linear-time temporal logic* (LTL) beschrieben. Jede mögliche Ablaufreihenfolge von Aktivitäten, dies beinhaltet auch die Wiederholung einzelner, die eine solche LTL-Formel erfüllt, ist demnach Teil der Workflowinstanz. Jede mögliche Ablaufreihenfolge kann demnach einen Workflow ausmachen, solang diese alle Einschränkungen erfüllt, die dem Ziel der Instanz entsprechen. Dies muss nicht immer zu einem optimalen Plan für die Instanz führen.

Case Handling Eine Mischform stellt das Case Handling [186] dar. Hierbei wird versucht, kleine vordefinierte Workflows flexibel, je nach Bedürfnis zu kombinieren. Ausgangspunkt der Betrachtung ist dabei nicht der Kontrollfluss, wie bei den bisher vorgestellten Lösungen. Im Mittelpunkt steht ein Datensatz, der bearbeitet werden soll. Als Beispiel soll hier eine Supportanfrage dienen. Diese wird zunächst aufgenommen, anschließend zu dem jeweiligen zuständigen Sachbearbeiter weitergeleitet, der diese bearbeitet und ggf. verändert bzw. anreichert. Weitere Sachverständige können im Anschluss daran hinzugezogen werden. Welcher Sachbearbeiter die Anfrage bearbeitet, hängt von ihrem Inhalt ab.

Das Prinzip des Weiterreichens des Datensatzes ist die Grundidee des Case Handlings. Ein *Case* stellt hierbei einen kleinen Workflow dar. Dieser kann aus einer oder mehreren Aktivitäten bestehen. Für jede der Aktivitäten wird dabei festgelegt, welche Daten diese benötigt und für welche Daten sie eingesetzt werden kann. Die *Case Data* stellt den Datensatz dar, der verarbeitet werden soll. Durch einen Matching-Algorithmus, über das Interface der Case Data und den Constraints für eine jede Aktivität, kann festgestellt werden, welche Aktivitäten sich zur Bearbeitung der Case Data eignen. Diese werden in eine Warteschlange eingereiht. Ein Sachbearbeiter, der zumeist hinter einer solchen Aktivität steht, kann diese Aktivität annehmen und ausführen. Im Anschluss wird sie aus der Warteschlange entfernt und es werden neue Aktivitäten für die Case Data gesucht.

Die Zuweisung für die Bearbeitung der Aktivitäten erfolgt durch ein Rollenmodell. Es gibt drei Grundrollen: *Execute* erlaubt es Aktivitäten auszuführen, *Redo* diese zu wiederholen und *Skip* diese zu überspringen. Innerhalb einer Aktivität kann die Case Data über ein der Aktivität zugeordnetes Formular bearbeitet werden.

Auch hier ist zum Startzeitpunkt die Ablaufstruktur des Workflows nicht bekannt. Diese wird erst zur Laufzeit iterativ erarbeitet. Dabei werden auf Basis der Workflowdaten (Case Data) die geeigneten Aktivitäten (Cases) berechnet. Im Zusammenspiel zwischen dieser dynamischen Auswahl und den Bearbeitern, die bestimmte Aktivitäten ausführen, wiederholen oder überspringen können, entsteht eine flexible, sich an verändernde Bedingungen anpassende Workflowinstanz.

Late Binding Eine weitere Möglichkeit stellt das *Late Binding* [137, 149, 5, 150] dar. Hierbei wird ein Workflow-Template zur Laufzeit derart ausgefüllt, dass ein ausführbarer Workflow entsteht. Im Template wird zur Designzeit beispielsweise eine Aktivität nicht vollständig spezifiziert, sondern nur der Typ der Aktivität vorgegeben. Ebenso ist es möglich, Teile des Kontrollflusses nicht vollständig zu spezifizieren. Dies wird auch als *Unterspezifikation* bezeichnet. Dadurch entsteht ein Platzhalter, der zur Laufzeit neu gebunden werden muss.

Die Ansätze unterscheiden sich zumeist in der Art des Auffindens von geeigneten Bindungspartnern. In [188] wird ein Beispiel für Late-Binding Arbeitsabläufe mit *on-the-fly* Veränderungen aufgezeigt. Hier wird eine Lösung vorgestellt, bei der die Aktivitäten im Moment ihrer Ausführung Vorbedingungen erfüllen müssen (vgl. Case Handling.). Wenn eine Vorbedingung nicht erfüllt wird, ist die Aktivität nicht mehr verfügbar und muss durch eine andere ersetzt werden. Mögliche Kandidaten werden anhand einer Ähnlichkeitsfunktion gefunden. Der beste Kandidat wird anschließend ausgewählt und anstatt der ursprünglichen Aktivität ausgeführt. Es ist auch möglich, abstrakte Aktionen zu definieren, die immer dann durch Instanzen mit gleichen Eigenschaften ersetzt werden, wenn der Workflow gestartet wird. Dieser Mechanismus ähnelt der Suche nach Alternativen bei Case-based Reasoning Systemen (CBR System)[21]. Die Ähnlichkeit wird dabei nicht streng geprüft, d. h. die Ähnlichkeit ist kein skalarer Wert, sondern basiert auf der semantischen Distanz in einer Ontologie.

Einen weiteren möglichen Ansatz stellt PreDiCtS[22] [3, 2] dar. Mit diesem ist es möglich, die Bindungsmöglichkeiten auf Basis semantischer Beschreibungen zu finden. Der Ansatz wurde entwickelt, um Nutzern das Erstellen von Workflows zu erleichtern, indem sie nur das Problem, welches sie lösen möchten, beschreiben. Die Software soll zur Laufzeit den geeigneten Service wählen. Dieses Vorgehen entspricht dem des Late Bindings.

Ein Beispiel für Instanz-Komposition in der Domäne der Webservices zeigt [41]. Das *AZTECH* System verwendet eine hierarchische Planung, um Workflows zu komponieren. Die Hauptworkflows werden als Templates beschrieben, die Slots besitzen. Diese können durch Unterprozesse (*fillers* genannt) eingenommen werden. Beide, Templates und Fillers, sind in der Bibliothek zusammen mit einer semantischen Beschreibung gespeichert, die es erlaubt, geeignete Prozesse für bestimmte Anforderungen auszuwählen. Die Komposition der Prozesse wird als *splicing* bezeichnet, ein Hinweis darauf, dass der Ursprung der Idee aus der Aspect Oriented Programming (AOP) Community kommt. [113] und [42] präsentieren ähnliche Lösungen, die ebenfalls Templates nutzen, welche spezielle Punkte für einen Austausch von Funktionen vorsehen.

In [191] werden zudem vier weitere essentielle Adaptionsmuster genannt, die zur Kategorie der *Patterns for Changes to Predefined Regions* gehören. Diese entsprechen grundsätzlich dem Late Binding. Diese Kategorie umfasst die folgenden Muster:

1. *PP1 – Late Selection of Process Fragments*: Hierbei wird erst zur Laufzeit ein bestimmtes Fragment an einen Platzhalter gebunden.
2. *PP2 – Late Modeling of Process Fragments*: Bestimmte Bereiche eines Workflows werden erst zur Laufzeit modelliert.
3. *PP3 – Late Composition of Process Fragments*: Aneinanderreihung verschiedener Fragmente zur Laufzeit
4. *PP4 – Multi-Instance Activity*: Die Häufigkeit der Ausführung einer bestimmten Aktivität wird erst zur Laufzeit festgelegt.

Die Untersuchungen in [191] zeigen auch hier, dass die Muster nur unzureichend von den einzelnen WfMS unterstützt werden. Während PP1 und PP2 vereinzelt unterstützt werden, zeigt [191] auf, dass PP3 und PP4 bisher nicht unterstützt werden. Die Referenz ADEPT2 unterstützt keines der vier Muster.

[21] Vgl. [20] oder [103] für eine Einführung in CBR.
[22] A Personalised Service Discovery and Composition Framework

2.4 Bewertung der Adaptionsverfahren

In Abschnitt 2.2.3 wurde eine Klassifikation der Flexibilität bei Workflows aus [182] vorgestellt, die für die Einordnung und Bewertung der verschiedenen Lösungskategorien verwendet werden kann. Ein ideales Rekonfigurationssystem sollte demnach sowohl evolutionäre, als auch befristete Änderungen unterstützen. Diese sollten alle Perspektiven des Workflows betreffen und alle möglichen Änderungsoptionen unterstützen.

Bei der *evolutionären Adaption* können Änderungen, wie sie beim Theater-Workflow vorgestellt wurden, nur evolutionär vorgenommen werden. Daraus ergibt sich, dass beispielsweise ein Rabattsystem eingeführt werden kann, die Betreuung von VIPs jedoch nicht möglich ist. Ungeplante Aktionen werden demnach nicht unterstützt. Vorteil dieser Methode ist es, dass prinzipiell alle Perspektiven des Workflows in jeglicher Hinsicht verändert werden können.

Das Gegenteil stellen die Änderungen durch *Unter- und Überspezifikation* dar. Die Änderungszeitspanne ist immer befristet und zusätzlich wird jede Instanz isoliert betrachtet. Spontane Änderungen können gut umgesetzt werden. Da zumeist kein zentrales Workflowmodell besteht, sind langfristige Änderungen nicht an einem Workflow festzumachen. Je nach Ansatz müssen beispielsweise der Planer oder die Constraints geändert werden. Die Frage der Migration bestehender Instanzen stellt sich hierbei nicht, da kein zentrales Workflowschema existiert. Nachteilig wirkt sich allerdings die fehlende Grundstruktur bei der Anwendung in Unternehmen aus, die hohe Flexibilität ermöglicht, aber wenig Planungssicherheit garantiert.

Einen Kompromiss zwischen diesen Extremen stellen die *dynamischen Adaptionen* dar. Diese werden weiterhin strukturiert beschrieben, können aber zur Laufzeit verändert werden. Dabei ist die Änderungszeitspanne befristet. Eine gewisse Ausnahme stellt AgentWork, mit einer expliziten Regelebene, dar. Adaptionsregeln (ECA-Regeln), die für eine Instanz beschrieben wurden, können auch für nachfolgende Adaptionen genutzt werden. Eingeschränkt sind hingegen die betroffenen Perspektiven und die Möglichkeiten der Änderungen. Wie in [191] gezeigt wird, unterstützt nur ADEPT2 eine hinreichende Auswahl an Adaptionsmustern. Diese betreffen nur den Aspekt der Verhaltensperspektive. Die informationsbezogene oder gar die organisatorische Perspektive werden von allen bei dynamischen Ansätzen vernachlässigt. Die ersten drei Teilabbildungen in 2.7 zeigen die derzeitige Abdeckung der Änderungsklassifikation der drei großen Rekonfigurationsansätze.

Aufgrund ihrer extremen Anpassungsmöglichkeiten, und der damit einhergehenden Unsicherheit, werden unterspezifizierte Workflows nicht in klassischen Geschäftsprozesssystemen eingesetzt und evolutionäre Anpassungen stellen nur bei langfristigen Änderungen eine Alternative dar. Aufgrund ihrer Brückenfunktion können dynamische Adaptionen in klassischen Geschäftsprozesssystemen eingesetzt werden und bieten gegenüber evolutionären Adaptionen eine höhere Flexibilität. Sie sind zudem theoretisch für alle Änderungsspannen anwendbar. Bisher nicht ausreichend unterstützt sind allerdings die verschiedenen Adaptionsmuster. Zudem existiert kein allgemeingültiger Adaptionsansatz, der in verschiedenen WfMS angewendet werden kann.

Das Ziel dieser Arbeit ist eine vollständige Klassifikationsabdeckung für Rekonfigurationen, wie in 2.7 (d) dargestellt, zu erreichen. Dies beinhaltet eine breite Unterstützung von

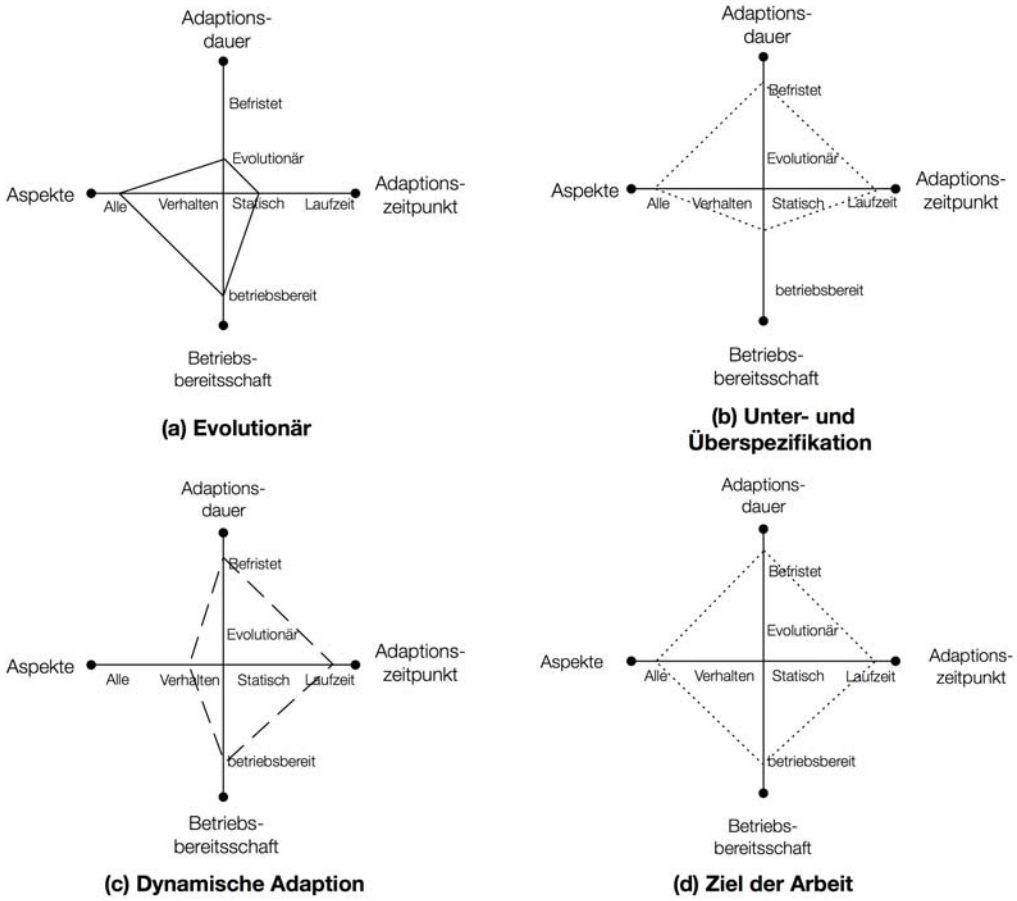


Abbildung 2.7: Ziel der Arbeit

Änderungsoperationen für alle Perspektiven. Dabei können Änderungen sowohl befristetet, als auch evolutionär durchgeführt werden. Des Weiteren ist eine effiziente Steuerung der Adaptionen notwendig. Lösungen wie AgentWork sind nur bedingt einsetzbar. Dort müssen alle Situationen durch je eine ECA-Regel abgebildet werden. Aufgrund der unendlichen Menge von (unerwarteten) Situationen, auf die reagiert werden muss, ist ein solcher Ansatz nicht anwendbar.

In den folgenden Kapiteln wird deshalb ein umfassender Lösungsansatz für die vollständige Rekonfiguration von Workflows zur Laufzeit vorgestellt. In Kapitel 3 wird ein auf Verweben aufbauender Adaptionsmechanismus gezeigt. Dieser wird durch einen lernenden *Direktor* gesteuert, der sowohl evolutionäre als auch befristete Rekonfigurationen ermöglicht. Dieser Direktor kann an verschiedene Bedingungen angepasst werden. Drei verschiedene Varianten in den Kapiteln 4 bis 6 vorgestellt, die in verschiedenen Domänen eingesetzt werden können. In Kapitel 7 wird ein neuartiger Adaptionsmechanismus auf Basis einer rollenbasierte Workflow-Beschreibungssprache präsentiert, die die Rekonfiguration aller kontextbezogenen Perspektiven ermöglicht.

Kapitel 3

Autonom verwebte Workflows

Wie im letzten Kapiteln dargelegt, gibt es bei der Rekonfiguration von Workflows zwei Extreme: zum einen die *evolutionären Adaptionen*, die nur in begrenztem Maße veränderbar sind und zum anderen die Adaptionen durch *Unter- und Überspezifikation* (hier speziell die Ad-hoc), die stark anpassbar aber schlecht planbar sind. Für Unternehmen, die ihre Geschäftsprozesse zwangsläufig immer wieder anpassen müssen, dabei aber nicht auf die Berechenbarkeit verzichten können, sind beide Klassen von Workflows nur bedingt geeignet. Einen Kompromiss zwischen diesen Extremen stellen die *dynamischen Workflows* dar. Diese werden weiterhin strukturiert beschrieben, können aber zur Laufzeit verändert werden.

Werden die dynamischen Workflows genauer betrachtet, ist festzustellen, dass hauptsächlich die Verhaltensperspektive hinsichtlich Veränderungen untersucht wurde. Das bedeutet, dass der Ansatz sich auf die Veränderungen des Kontrollflusses konzentriert. In Abbildung 2.2 werden 14+4 grundlegende Adaptionsmuster vorgestellt, die ein Adaptionsmechanismus unterstützen muss. Diese werden aber bei weitem nicht vollständig unterstützt. Wie die Untersuchung von Weber in [191] zeigt, beschränken sich die Adaptionsmöglichkeiten auf die Anpassung einzelner Schritte im Workflow: das Auslassen einer Aktion oder das Einfügen eines parallelen Pfades. Die einzige Ausnahme stellt hier ADEPT2 [47, 111] dar (siehe Abschnitt 2.3.2).

Ziel der Arbeit ist ein Ansatz für rekonfigurierbare Workflows, der die Adaptionsmuster für die Verhaltensperspektive aus [191] vollständig unterstützt. Um diesen Adaptionsansatz zu realisieren, wird auf das Konzept des *Verwebens* zurückgegriffen. *Workflow-Verwebung* beschreibt die Möglichkeit der Komposition von einzelnen Workflowmodellen zur Laufzeit. Ist eine Abweichung vom normalen Ablauf notwendig, wird ein anderes Workflowmodell in das Ursprüngliche eingewoben. Das Laufzeitverhalten des Adaptionsansatzes entspricht der des Verwebens, woraus sich der Name ergibt: *Adaption durch Verweben*.

Änderungen bei dynamischen Workflows sind normalerweise befristet. Dauerhafte Veränderungen, wie die Einführung eines Rabattsytems in ein Ticketsystem, sind somit nicht abbildbar. Zudem müssen die Adaptionen manuell vorgenommen werden und können dadurch zumeist nicht wiederverwendet werden. Eine explizite Regelebene, wie sie in AgentWork vorgestellt wurde, stellt einen Fortschritt dar. Diese erlaubt evolutionäre Veränderungen, da die Adaptionen über ECA-Regeln gesteuert werden, die permanent aktiv sind. Das System ist jedoch auf ECA-Regeln beschränkt und die Regeln müssen weiterhin manuell erstellt werden. Dies wird mit zunehmender Regelanzahl schwieriger, denn es gilt Seiteneffekte zu verhindern. In diesem Kapitel wird deshalb ein *reflexiver Direktor* vorgestellt, der die Adaption durch Verweben nicht nur regelt und damit befristete und evolutionäre Änderungen unterstützt, sondern auch neue Adaptionen selbstständig

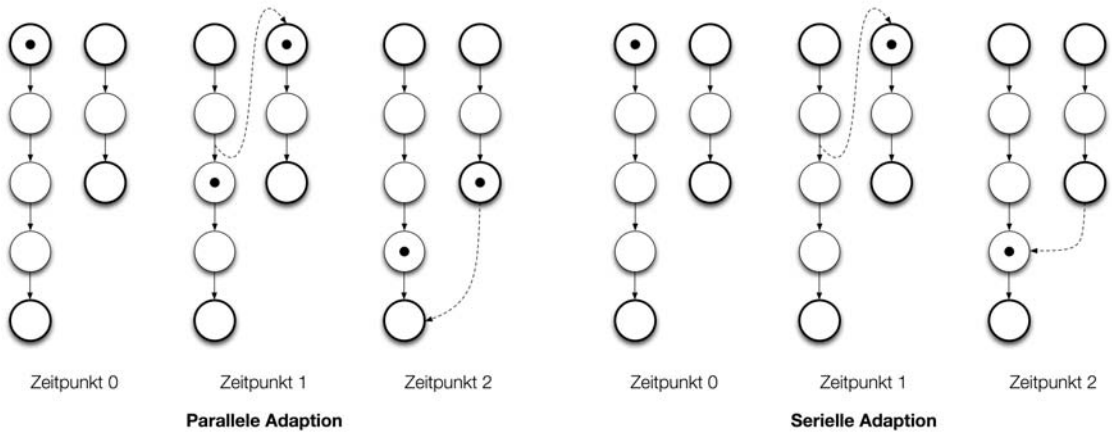


Abbildung 3.1: Parallele und Serielle Adaption

lernen kann. Eine lernende Komponente ist entscheidend für die Anwendbarkeit dynamischer Workflows, denn sie ermöglicht es, sich *selbstständig*, d. h. *autonom*, an neue Umweltbedingungen anzupassen und zudem den Wartungsaufwand für eine komplexe Regelebene deutlich zu senken.

Daraus ergeben sich die folgenden Kriterien für die reflexiv verwebten Workflows, die im Folgenden weiter untersucht werden:

- Ein hohes Maß an *Flexibilität*, d. h. es müssen die Adaptionsmuster (siehe Abschnitt 2.3.2) unterstützt werden.
- Realisierung des Adaptionsansatzes mit einem minimalen, WfMS-unabhängigen, Befehlssatz, damit dieser in verschiedenen WfMS zum Einsatz kommen kann.
- Adaptionen müssen auch (voll-)automatisch vorgenommen und autonom berechnet werden können.

In den folgenden Abschnitten werden zur Erfüllung dieser Kriterien zwei neuartige Konzepte für Laufzeit-Rekonfiguration von Workflows eingeführt: *Adaption durch Verweben* und der *reflexiv autonome Weber*.

3.1 Adaption durch Verweben

Der meistbeachtete Aspekt bei dynamischen Workflows ist die Verhaltensperspektive. An dieser Stelle soll deshalb ein neuer Ansatz vorgestellt werden, der sich dadurch auszeichnet, die Ablaufstruktur nicht direkt zu verändern, sondern das Token der jeweiligen Instanz durch neue Bahnen zu steuern.

Ist eine Abweichung vom normalen Ablauf notwendig, wird ein anderes Workflowmodell in das ursprüngliche *eingewoben*. Einweben bedeutet keine echte Modellkomposition im Sinne einer Assimilation, bei der aus mehreren Modellen ein neues Gesamtmodell einsteht. Es heißt vielmehr, dass in das Modell der Basisinstanz M_1 eine Kontrollflussverzweigung eingeführt wird, die beim Aufruf eine Instanz des zu verwebenden Modells

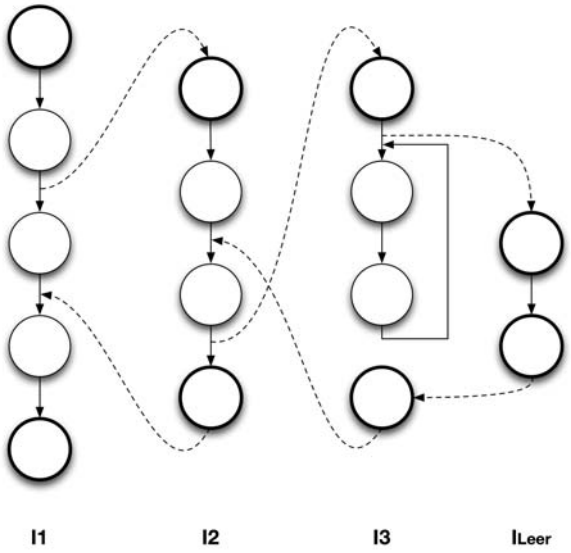


Abbildung 3.2: Vierstufige Adaption

M_2 startet. Dadurch können mehrere Instanzen von verschiedenen Workflowmodellen zusammen als Subprozesse eines zusammengesetzten Workflows agieren. Die Ausführung verläuft nun nicht mehr nur entlang der im Workflow vorgegebenen Bahnen, sondern auch entlang der zur Laufzeit hinzugefügten Kontrollflussverzweigungen.

Bei dieser Art des Verwebens sind zwei verschiedene Betrachtungsweisen möglich: (1) das *parallele* und (2) das *serielle* Verweben. Abbildung 3.1 veranschaulicht beide Arten:

1. *Parallele Verwebung:* Hierbei wird eine Aufspaltung des Kontrollflusses vorgenommen. Dabei wird ein neues Token erzeugt, der den Zustand des originalen Tokens übernimmt. Dieses neue Token wird der neu eingewobenen Instanz zur Ausführung übergeben. Das Ursprungstoken ist von den Veränderungen nicht betroffen und setzt seine Abarbeitung in der Ursprungsinstanz fort (Abbildung 3.1). Dies entspricht grundsätzlich einer zur Laufzeit eingeführten parallelen Abarbeitung.

2. *Serielle Verwebung:* Darüber hinaus ist es möglich, dass kein neues Token erzeugt wird, sondern der bestehende in die neue Instanz überführt wird.

Das Beispiel in Abbildung 3.2 veranschaulicht verschiedene Möglichkeiten der zuvor beschriebenen Webevorgänge. Die erste Adaption ereignet sich nach dem zweiten Knoten von I_1. Eine neue Instanz von I_2 wird erzeugt und in den Workflowverlauf eingewoben. Eine dritte Instanz I_3 wird gestartet, bevor das Token den vierten Knoten von I_2 erreicht. I_3 selbst besteht aus einer Endlosschleife zwischen dem zweiten und dritten Knoten. Das Token führt diese beiden Knoten beliebig oft (die Abbildung zeigt nicht, wie oft) aus, d. h. es wird kein Endknoten erreicht. Wurde die Endlosschleife oft genug durchlaufen (dies kann z. B. der Nutzer festlegen), wird zum Verlassen der Schleife eine Instanz $I_{(Leer)}$ gestartet, deren einziger Zweck es ist, die I_3 bis zu einem gewissen beliebigen Knoten springen zu lassen, hier ein Endknoten. Dies wird durch das Verlassen $I_{(Leer)}$ und die Rückkehr zu dem letzten Knoten bewerkstelligt. I_3 wird beendet und das Token kehrt hinter dem zweiten Knoten von I_2 in die Ursprungsinstanz zurück. Es ist möglich, dass

der Webevorgang vor dem vierten Knoten wiederholt wird (die Skizze kann den zeitlichen Verlauf nicht vollständig darlegen). Das Token erreicht den Endknoten von I_2 und das Token kehrt zurück zum vierten Knoten von I_1. Die mit dem dritten Knoten verbundene Aktion wird nicht ausgeführt, sondern das Token fährt fort mit dem vierten Knoten und beendet schließlich den Workflow.

Die Art des Verwebens ist gleichzusetzen mit Prozeduraufrufen in imperativen Programmiersprachen. Beide gehören zu dem Konzept der symmetrischen Kontrollstrukturen [131]. Das Starten eines Unterprozesses kann mit dem Aufruf von Prozeduren, das Ende von Unterprozessen mit dem Rücksprung in die aufrufende Prozedur verglichen werden. In [36] wird eine ähnliche Kompositionsidee gezeigt. Allerdings wird ein System präsentiert, bei dem Unterprozesse nicht in den Hauptworkflow integriert werden, sondern Hauptworkflow und Unterprozesse werden unabhängig voneinander gestartet. An einem beliebigen Punkt ihrer Ausführung benötigen diese Workflows einen Partnerprozess, der die benötigten Services bietet. Ist ein Partner gefunden, werden beide aneinander gebunden, tauschen gegenseitig ihre Daten aus und werden parallel weiter ausgeführt. Wenn keine Zusammenarbeit mehr notwendig ist, werden die Partner wieder getrennt.

Verweben Die Idee des Ansatzes ist es, die Struktur des originalen Workflowmodells nicht grundlegend zu ändern, sondern lediglich neue Verzweigungen einzubringen. Zudem bezieht sich die Adaption nur auf die jeweilige Instanz und nicht auf alle Instanzen, wie es bei evolutionären Workflows der Fall ist. Wird eine neue Workflowinstanz gestartet, wird das Workflowmodell initialisiert, d. h. es wird eine Laufzeitkopie des Workflowmodells und ein Token mit allen Variablendefinitionen erzeugt und an die Position des Startknotens gesetzt. Während er ausgeführt wird, kann eine Kontrollinstanz bzw. Regelebene von außen beschließen, die Instanz an der aktuellen Position anzuhalten und einen Webevorgang durchzuführen. Es wird nur die zu adaptierende Instanz gestoppt. Die Ausführung aller anderen laufenden Instanzen des Workflowmodells ist davon nicht betroffen.

In dieser Webeanweisung wird angegeben, welcher Workflow eingewoben werden soll und ob es sich um eine serielle oder parallele Adaption handelt. Das Verweben besteht darin, dass nach dem Halt der Hauptinstanz eine neue Instanz des einzuwebenden Workflows erzeugt wird. Bei der seriellen Adaption wird anschließend das Token, mit allen Variablen in die eingewobene Instanz überführt. Damit ist es möglich, Ergebnisse der Hauptinstanz innerhalb der Adaptionsinstanz wiederzuverwenden. Dieses Vorgehen ist nur möglich, wenn das Workflowmodell entsprechend dafür entwickelt wurde, d. h. entsprechende gleichnamige Variablen existieren. Bei einer parallelen Adaption wird ein neues Token erzeugt, der ebenfalls in die neue Instanz überführt wird. Damit auch dieser Token über alle Variablen verfügt, wird eine Kopie der Variablen des Ursprungstoken erzeugt. Dabei kann das einzuwebende Workflowmodell ein beliebiges verfügbares Modell aus einem Repository sein. Es kann sich dabei ebenfalls um eine neue Instanz des gleichen Workflowmodells handeln. Adaptionen können sich auch auf eingewobene Instanzen beziehen, so dass verschachtelte Adaptionen entstehen, wie in Abbildung 3.1 dargestellt.

Eine Adaption kann dabei an jeder beliebigen Aktivität stattfinden, wobei es keine Rolle spielt, ob die Adaptionsentscheidung vor der Aktivität oder danach stattfindet.

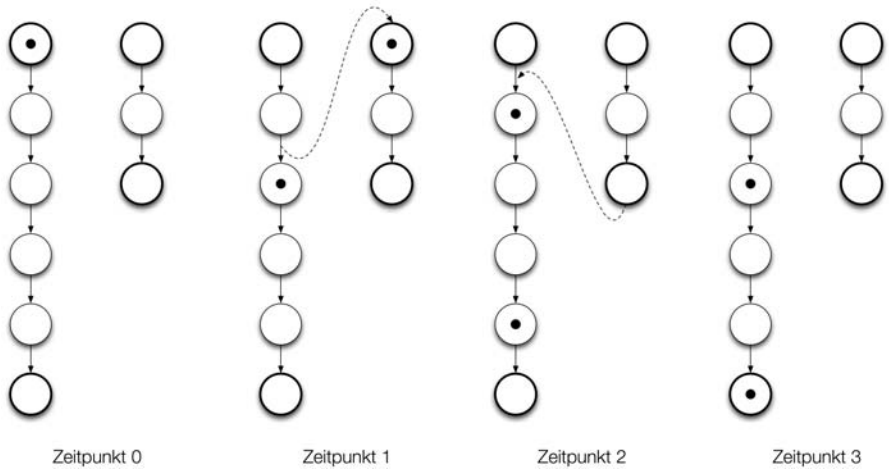


Abbildung 3.3: Mehrdeutigkeit bei paralleler Adaption

3.1.1 Variabler Rücksprungpunkt

Allein das Einweben kann jedoch keinen neuen Adaptionsansatz hervorbringen. Es entspricht dem Adaptionsmuster des Einwebens eines Subprozesses (AP6 und AP7). Dadurch können die weiteren Adaptionsmuster nicht abgebildet werden. Beispielsweise kann kein Workflowfragment ersetzt werden (*Replace Fragment*). Deshalb ist ein weiteres Konzept notwendig: der *variable Rücksprungpunkt*.

Ein Rücksprungpunkt gibt an, an welche Stelle der Hauptinstanz der adaptierte Token zurückspringen soll. Dieser Punkt kann prinzipiell eine beliebigen Stelle im adaptierten Workflow darstellen oder auch nicht gesetzt sein. Dabei müssen einige Einschränkungen beachtet werden, die abhängig von der gewählten Webeart sind.

Wird eine *parallele Verwebung* vorgenommen, kann der Rücksprungpunkt nicht auf eine Stelle gesetzt werden, die bereits ausgeführt wurde. Dieser Fall ist in Abbildung 3.3 dargestellt. Wäre dieser Fall erlaubt, würden zwei Token in einer Instanz existieren, die ursprünglich nur für einen Token ausgelegt ist. Bei einer parallelen Adaption wird das Token in zwei Token, T_1 und T_2, geteilt. T_2 verbleibt in der Ausgangsinstanz, Token T_1 springt in die neue Instanz I_B. Beim Rücksprung des Token T_1 aus der beendeten, eingewobenen Instanz I_B, ist das in der Ausgangsinstanz verbliebene Token T_2 mit seiner Bearbeitung vorangeschritten, denn er wird bei der parallelen Verwebung nicht gelöscht. Durch zwei Token in einer Instanz entsteht eine inkonsistente Situation. Es ist nicht eindeutig, wann der Workflow beendet ist. Für den Nutzer ist nicht mehr eindeutig erkennbar, welchen Status die Instanz derzeit besitzt. Zudem kann es zu verschiedenen Endergebnissen bzw. Zwischenergebnissen führen. Erreicht Token T_2 als erster das Ende[1], erhält der Anwender ein Ergebnis. Jedoch ergibt sich bei der Beendigung der Instanz durch Token T_1 ein anderes Ergebnis. Statt eines einzigen Endes werden hier zwei erreicht. Für den Benutzer ist nun nicht mehr zu entscheiden, welches Ergebnis richtig ist.

[1] Davon kann ausgegangen werden, solang keine weitere Adaption stattfindet.

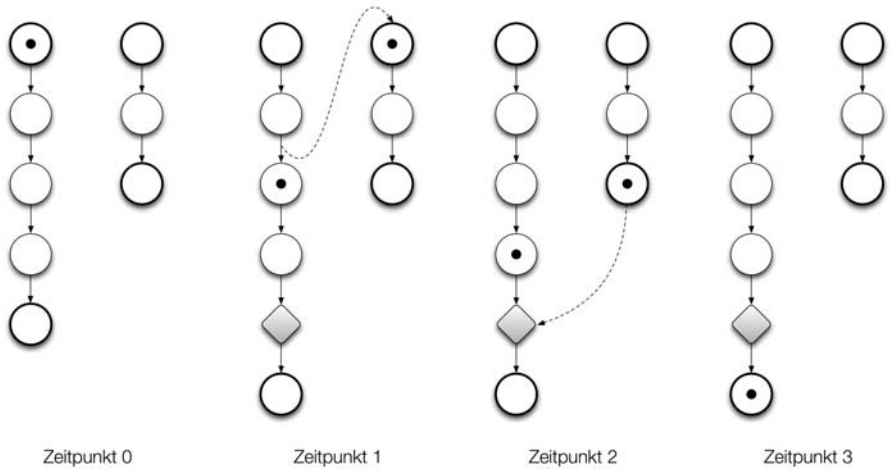


Abbildung 3.4: Virtueller Join

Wenn Token T_1 nicht vor T_2 zurückspringen darf, kann er nur zu einem Punkt springen, der dem Adaptionsentscheidungspunkt entspricht oder hinter diesem liegt. Aber auch hier besteht die Gefahr, dass zwei Token gleichzeitig in einer Instanz vorhanden sind. Um dies zu verhindern, wird an diesem Rücksprungpunkt ein *virtueller And-Join* eingefügt[2], wie in Abbildung 3.4 – Zeitpunkt 1 – dargestellt ist. *Virtueller Join* deshalb, da der Join nicht im Originalmodell enthalten ist und nur für die Zeit der Adaption in der Instanz verbleibt. Das Token T_2 muss auf die Ankunft von T_1 bzw. umgekehrt warten. Sind beide Token an diesem Join angekommen, werden sie zu einem Token verschmolzen, inkl. der Vereinigung beider Variablenmengen[3]. Die Einführung eines Join-Operators bei einem Rücksprungpunkt vor dem Adaptionsentscheidungspunkt ist nicht möglich, da T_2 diesen nicht erreichen wird.

Wird eine *serielle Verwebung* angewendet, existieren weniger Ausnahmen und kein virtueller Join. Bei dieser Adaptionsart verbleibt das Token nicht in der Hauptinstanz. Bei seiner Rückkehr können keine zwei Token in einer Instanz gleichzeitig existieren. Somit ist es möglich, dass der Rücksprungpunkt vor bzw. hinter dem Adaptionsentscheidungspunkt liegt.

Dies ändert sich, wenn die Adaption ein bereits *geteiltes Token* betrifft. Diese geteilten Token treten nach Fork-Anweisungen auf. An einem Fork wird der Kontrollfluss verzweigt, wie in Abbildung 3.5 (a) dargestellt ist. Das Token wird dazu in mehrere Kind-Token aufgeteilt. Diese werden durch einen Join wieder zusammengeführt, d. h. wieder zu einem Token verschmolzen. Es ist aber ebenso möglich, dass die Verzweigungen zu verschiedenen Enden des Workflows führen. Der gleiche Fall existiert bei Entscheidungsknoten, bei denen es sich nicht nur um eine „entweder oder“ Entscheidung handelt, sondern mehrere Aussagen zutreffen. Für jede zutreffende Aussage wird ein Kind-Token erstellt. Wie beim Fork-Knoten können diese Token wieder durch einen Join vereint werden. Wird eine Adaption auf einem dieser geteilten Token ausgeführt, besteht bei der

[2] Raute in Abbildung 3.4.
[3] In der Implementierung (siehe Abschnitt 8.1 wird bei Gleichheit der Variablen, aber unterschiedlichen Werten, nur der aktuellste Wert berücksichtigt.

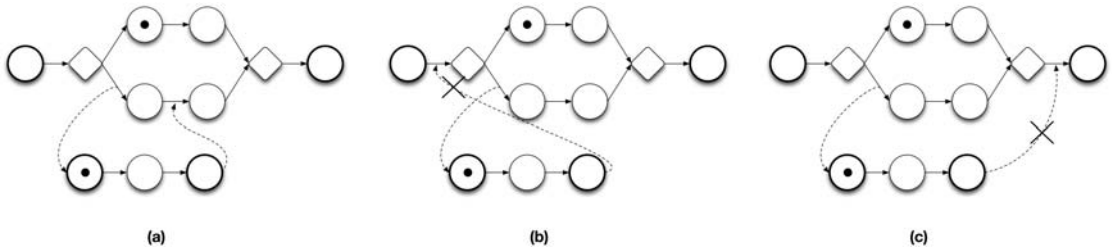


Abbildung 3.5: Adaption bei geteiltem Token

seriellen Adaption die Einschränkung, dass der Rücksprungpunkt gesetzt werden muss, wobei zwei Fälle zu beachten sind:

1. *Der Pfad führt zu einem Join:* Kehrt das Token nicht zurück, kann der Join der Hauptinstanz nicht ausgeführt und der Workflow nicht beendet werden. Es muss demnach ein Rücksprungpunkt gesetzt werden. Es sind auch nur Rücksprungpunkte innerhalb des Pfades erlaubt. Springt das Token vor den Fork-Knoten (Abbildung 3.5 (b)), gibt es die gleichen Inkonsistenzen wie bei der oben beschriebenen parallelen Adaption. Springt er hinter den Join (3.5 (c)), kann dieser ebenfalls nicht richtig ausgeführt und die Instanz nicht korrekt beendet werden.

2. *Der Pfad führt nicht zu einem Join:* In diesem Fall gibt es nur die Einschränkung, nicht vor den Fork zurückzuspringen, denn dies endet in den zuvor beschriebenen Widersprüchen von mehreren Enden eines Workflows. Es ist aber möglich, dass kein Rücksprungpunkt gesetzt wird oder dass ein beliebiger Punkt nach dem Adaptionsentscheidungspunkt gewählt wird, denn es ist kein Join zu beachten.

Die beschriebenen Einschränkungen für Fork-Join Konstrukte gelten ebenfalls für Schleifenkonstrukte.

Ebenfalls beim virtuellen Join sind Sonderfälle zu beachten. Im Falle einer parallelen Adaption verbleibt das Ursprungstoken in der Hauptinstanz. Wird der Rücksprungpunkt des adaptierten Token hinter den Adaptionsentscheidungspunkt gesetzt, wird an diesem Rücksprungpunkt ein Join eingefügt (siehe Abbildung 3.4). Wird jedoch der Ursprungstoken erneut adaptiert, muss der eingefügte Join Operator beachtet werden. Ein Beispiel ist in Abbildung 3.6 dargestellt. Hier wird der Ursprungstoken seriell adaptiert, d. h. er verlässt die Hauptinstanz. Hierbei ist der Rücksprungpunkt für den eingeführten virtuellen Join der von Bedeutung.

Liegt der Rücksprungpunkt der zweiten Adaption, wie in Abbildung 3.6 dargestellt, hinter dem virtuellen Join, muss dieser verschoben werden. Dies ist notwendig, da der Join niemals schalten könnte, da nur das Token der ersten Adaption ihn erreichen würde. Damit beide Token ihn erreichen können, muss er auf den Rücksprungpunkt der zweiten Adaption verschoben werden. Wird der Rücksprungpunkt der zweiten Adaption vor den virtuellen Join gelegt, bleibt der virtuelle Join unverändert. Dies gestaltet sich anders, wenn kein Rücksprungpunkt gesetzt wird. Dies bedeutet, dass das Token nicht in die Hauptinstanz zurückkehren wird. In diesem Fall muss der virtuelle Join gelöscht werden.

Die Adaption durch Verweben entspricht grundlegend den Prozedur-Aufrufen bei imperativen Programmiersprachen, aber im Gegensatz zur imperativen Programmierung ist der Rücksprungpunkt flexibel verwendbar. Er kann sowohl vor als auch hinter dem

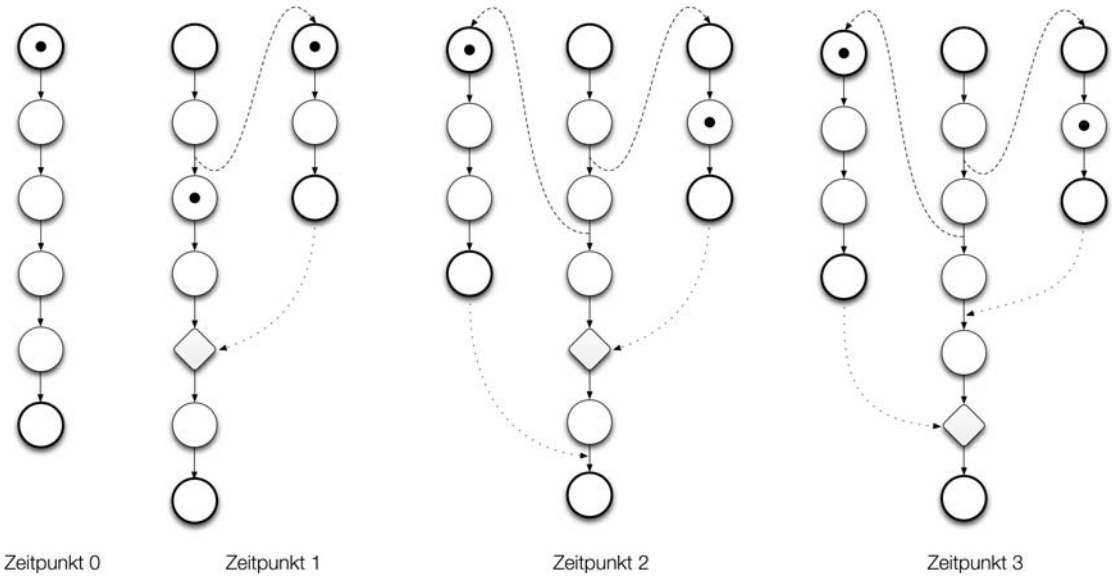


Abbildung 3.6: Aufhebung eines virtuellen Joins

Webepunkt liegen oder auch nicht angegeben werden. Dadurch werden beispielsweise Aktionen, die zwischen dem Webe- und dem vorgelagerten Rücksprungpunkt liegen, nicht ausgeführt oder wiederholt, wenn der Rücksprungpunkt hinter dem Webepunkt liegt. Es ist weiterhin auch möglich, dass durch den Subprozess einige Variablen in ihren Werten verändert wurden. Dadurch kann an Entscheidungspunkten ein anderer Pfad verfolgt werden. Es gibt eine Vielzahl von Möglichkeiten, diesen Adaptionsansatz zu verwenden. Zu beachten ist allerdings immer, dass die Adaptionen auch korrekt sind und keine zusätzlichen Fehler verursachen.

Einschränkungen durch Datenabhängigkeiten Die Einschränkungen für die Rücksprungpunkte bezogen sich bisher nur auf den Kontrollfluss und die Probleme mit mehreren Token in einer Instanz bzw. mit den Schwierigkeiten im Umgang mit Parallelität (Fork-Join Problematik). Dabei wurde ein wichtiger Aspekt von Workflows vernachlässigt: die verarbeiteten Daten. Diese werden in den WfMS durch Variablen abgebildet. Verschiedene Aktivitäten des Workflows bearbeiten diese Daten oder nutzen sie als Parameter für verschiedene Funktionen. Wichtig für die Aktivitäten ist es, dass die Variablen zum Funktionsaufruf mit Werten belegt sind. Ist dies nicht der Fall, kann die Funktion nicht aufgerufen werden und der Workflow wird abgebrochen, bzw. es wird eine bestimmte Fehlerroutine aufgerufen (Exception Handling).

Während der Entwickler beim Workflowdesign darauf achtet, dass die Variable zum richtigen Zeitpunkt mit einem Wert belegt ist, ergibt sich im Falle der variablen Rücksprungpunkte ein Problem. Liegt der Rücksprungpunkt hinter dem Adaptionsentscheidungspunkt, so kann er verschiedene Aktionen überspringen, die den Wert einer Variable setzen. Gelangt er dann zu einer Aktion, bei der diese Variable benötigt wird, so kann diese nicht ausgeführt werden. Deshalb muss sichergestellt werden, dass ein solches Überspringen von essentiellen Aktionen nicht möglich ist. Dazu muss zu jeder Aktion

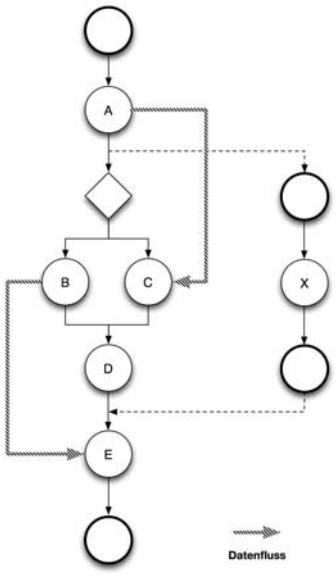


Abbildung 3.7: Datenabhängigkeiten

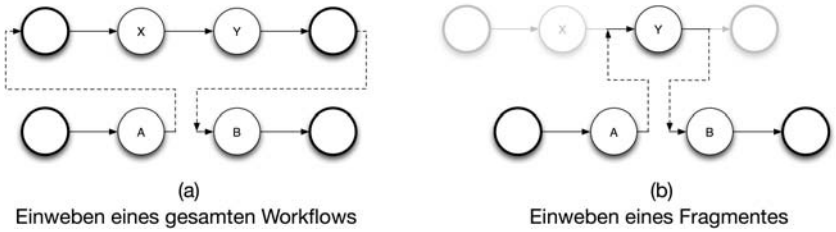


Abbildung 3.8: Selektives Einweben

explizit beschrieben werden, welche Variablen in der auszuführenden Funktion benötigt werden, d. h. die Variable muss einen Wert besitzen[4], und welche Variablen sie setzt.

Abbildung 3.7 zeigt beispielhaft unterschiedliche Fälle für Datenflussabhängigkeiten. Es existieren zwei direkte Abhängigkeiten: (1) $A \rightarrow C$ und (2) $B \rightarrow E$. Soll nun zwischen A und E ein Workflow eingewoben werden um X auszuführen, ist dies nicht möglich. Für die Ausführung von E sind die Daten von B notwendig. X erstellt diese notwendigen Daten nicht. Würde die Datenabhängigkeit nicht bestehen, könnte diese Adaption trotz $A \rightarrow C$ ausgeführt werden, denn C als Konsument der Daten von A wird nicht ausgeführt. Bei jeder Adaptionsentscheidung muss berücksichtigt deshalb werden, dass die Daten-Produzenten wie B, für nachfolgende Aktivitäten nicht übersprungen werden dürfen.

Selektives Einweben Bisher wurde das Verweben nur unter dem Gesichtspunkt betrachtet, dass vollständige Workflows eingewoben werden. Die Datenabhängigkeiten erlauben jedoch ein weit selektiveres Vorgehen. Abbildung 3.8 stellt das *selektive Einweben* in b) dem normalen Einweben eines Workflows in (a) gegenüber. In (a) wird der gesamte Workflow (seriell) eingewoben, wobei alle Aktivitäten ausgeführt werden. In (b) wird nur Y genutzt.

[4]Der Wert der Variable spielt hierbei keine Rolle.

Möglich wird dies durch den Transfer der Variablen bei der Adaption und die expliziten Datenabhängigkeiten. Durch sie ist bekannt, welche Variablen existieren, bzw. belegt sein müssen, um Aktivitäten auszuführen. Es ist demnach für alle Fragmente eines Workflows bekannt, welche Daten benötigt werden, damit dieses Fragment ausgeführt werden kann. Die vorgelagerten Aktivitäten sind dann zu vernachlässigen, denn bei einer Adaption werden alle Variablenbelegungen der Ausgangsinstanz der eingewobenen Instanz übergeben. Bei der Adaptionsberechnung kann demzufolge geprüft werden, ob die für Y benötigten Variablen bereits belegt sind und somit die Datenabhängigkeiten des gewünschten Fragments (hier nur Y) erfüllt sind. Wenn ja, kann ein selektives Einweben vorgenommen werden.

Für die Adaptionsvorschrift bedeutet dies, dass sie folgende Informationen beinhalten muss:

- Einzuwebender Workflow
- Rücksprungpunkt
- Start- und Endknoten des gewünschten Fragmentes

Mit der Adaption können deshalb auch beispielsweise einzelne Aktivitäten eingewoben werden, was einen erheblichen Flexibilitätsgewinn darstellt. Wichtig ist dabei, dass die Adaptionsmuster aus [191] unterstützt werden.

3.1.2 Adaptionsmuster mit der Adaption durch Verweben

Die Adaption durch Verweben beruht auf dem Prinzip der Komposition von Workflowfragmenten, wobei nicht das Workflowmodell direkt geändert wird, sondern neue Kontrollflussverzweigungen eingeführt werden. Die Adaption benötigt vier Basisfunktionen:

1. *Pause:* Das WfMS muss in der Lage sein, eine Instanz zu pausieren, bis eine Entscheidung über eine eventuelle Adaption getroffen wird.
2. *Übertragung:* Instanziierung beliebiger Workflowfragmente und Übertragung des Token, unter Berücksichtigung der Datenabhängigkeiten.
3. *Einfügen:* Einführung von And-Joins in den Basisworkflow
4. *Variablen:* Die Übertragung von Variablen und Variablenbelegungen in eine andere Instanz.

Nur diese vier Operationen sind notwendig, um die Adaption durch Verweben zu realisieren. Dadurch ist auch gewährleistet, dass der Ansatz sich in verschiedene WfMS integrieren lässt. Dies ist weitestgehend heute bereits möglich, wie das Beispiel jBPM[5] zeigt. Diese Workflow-Engine ermöglicht nicht nur das Pausieren von Instanzen, sondern auch einen flexiblen Umgang mit dem Token, der an beliebige Stellen transportiert werden kann. Zudem können die Laufzeitdaten (Variablen) jederzeit ausgelesen und übertragen werden. Die Einführung von And-Joins ist nicht möglich. Diese sind für die parallele Adaption notwendig. Diese Funktionalität ist allerdings durch eigene Erweiterungen nachrüstbar.

[5] Eine Workflow-Beschreibungssprache für das jBPM Workflow-Framework von JBoss.

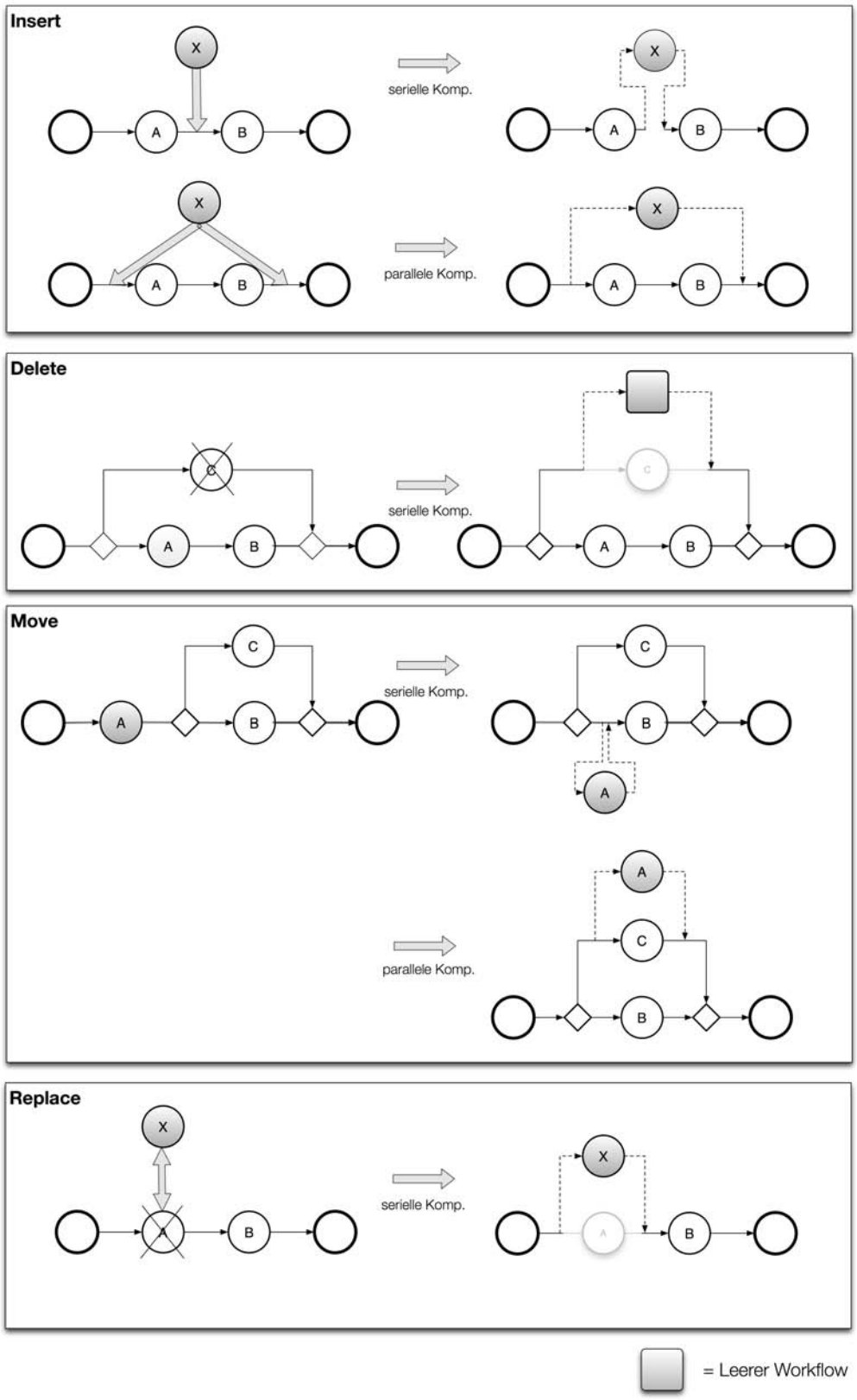


Abbildung 3.9: Adaptionsmuster mit Adaption durch Verweben (1)

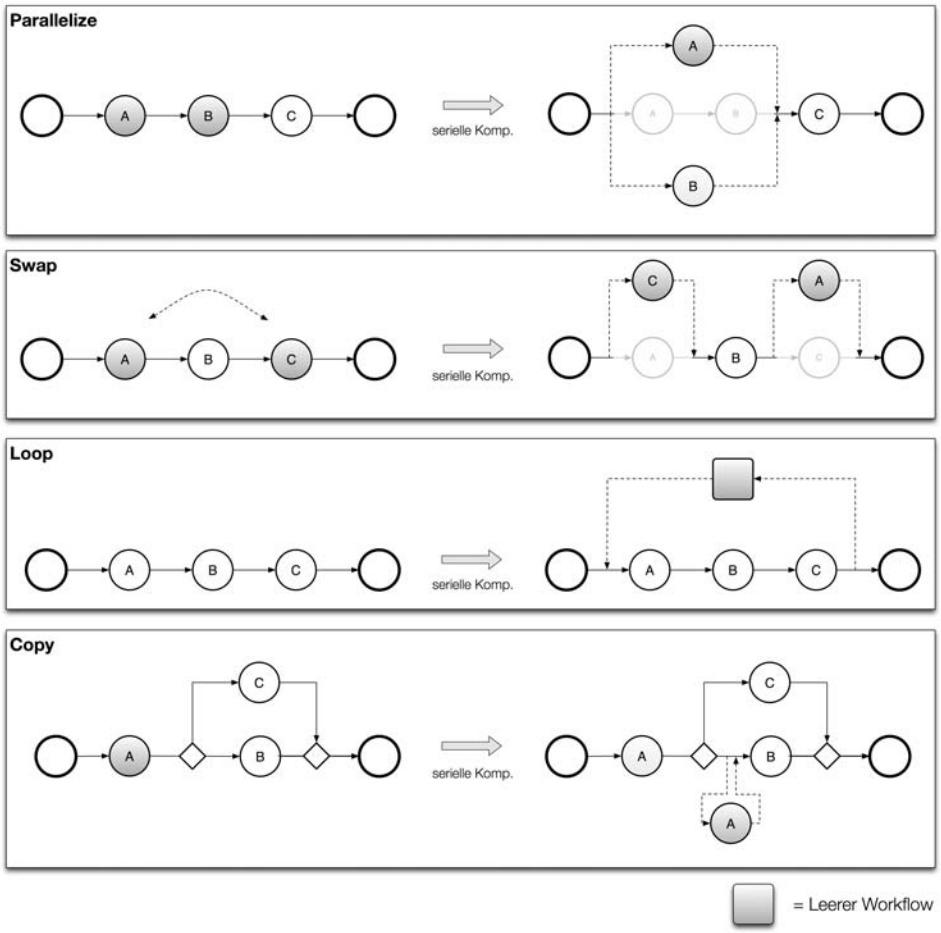


Abbildung 3.10: Adaptionsmuster mit Adaption durch Verweben (2)

Als zweites Kriterium wurden die Adaptionsmuster genannt. Die Abbildungen 3.9 und 3.10 zeigen die Realisierung der Basisoperationen und der erweiterten Adaptionsmuster. Auf der linken Seite ist die durchzuführende Operation dargestellt und der Überführungspfeil in der Mitte definiert die verwendete Webeart. Die rechte Seite zeigt die Adaptionsoperation an. Im Falle des Musters *Insert* wird der Kontrollfluss seriell neu komponiert. Dazu wird in die Instanz I_1 von Workflowmodell M_1 eine Instanz eines anderen Workflowmodells M_2 eingewoben. Bei der zweiten Variante wird eine parallele Verwebung angewandt, wodurch der vorgegebene Kontrollfluss zwischen A und B nicht überbrückt wird. Eine Besonderheit stellt der Leer-Workflow dar (gekennzeichnet durch das Quadrat). Dieser ist ein Hilfskonstrukt zur Realisierung von Sprüngen. Ein Leer-Workflow besteht aus nichts weiter als einem Start- und einem Endknoten. Er vollzieht keine Funktion und seine (theoretische) Verarbeitungszeit ist demnach $0s$. Durch das Verweben mit einem Leer-Workflow wird deshalb nur ein Sprung zum Rücksprungpunkt realisiert (siehe AP8 Loop). Die anderen Adaptionsmuster aus [191] können mit der Adaption durch Verweben wie folgt umgesetzt werden:

- *AP1 bis AP4 (Insert, Remove, Move, Replace):* Können in allen Ausführungsvarianten umgesetzt werden[6]. Die Basisadaptionen – *add node, remove node, add edge,*

[6] Für die in [191] beschriebenen Varianten von AP1 und AP3 – *Conditional Insert* und *Conditional Move* – gilt dies nur indirekt. Diese Adaptionen führen explizite Entscheidungspunkte in das Work-

remove edge, and move edge [191] und sind entsprechend emulierbar.

- *AP5 – Swap:* Für die Operation Swap sind zwei Webevorgänge notwendig. Da es sich um Aktivitäten des gleichen Workflowmodells handelt, werden hier neue Instanzen von M_1 selektiv eingebunden.
- *AP6 und AP7 - SubProcess:* Diese Muster kommen zum Einsatz, wenn ein Workflow zu groß geworden und dadurch nicht mehr zu warten ist. Dieses Vorgehen ist essentieller Bestandteil des Ansatzes, so dass diese Muster ohnehin unterstützt werden.
- *AP8 – Loop:* Ein Loop wird über den Leer-Workflow realisiert. Dieser wird an einer beliebigen Stelle eingebunden, wobei der Rücksprungpunkt vor dieser Stelle liegt. Zu beachten ist dabei, dass die Instanz nicht in eine Endlosschleife (Infinite-Loop) gerät. Um dies zu verhindern, müssen die Variablen beobachtet werden. Verändern diese sich nicht, muss der Loop abgebrochen werden[7].
- *AP9 – Parallelize:* Ist durch parallele Adaptionen realisierbar. Die notwendige Synchronisation wird durch den eingewobenen Join realisiert.
- *AP10 – Embed in Conditional Branch:* Dieses Muster ist ebenfalls integraler Bestandteil des Ansatzes. Die Entscheidungspunkte werden von einer Regelebene eingewoben[8].
- *AP11 – Add Control Dependency:* ist eine der geforderten Basisoperationen und ist damit als gegeben vorausgesetzt.
- *AP12 – Remove Control Dependency:* kann nicht durch die Adaption durch Verweben abgebildet werden. Es setzt voraus, dass in die Kontrollflussbedingungen der unterliegenden Workflow-Engine eingegriffen wird. Dies widerspricht den aufgestellten Kriterien.
- *AP13 – Update Condition:* Die Bedingungen für das Einweben der Regelebene sind nicht explizit vorgegeben und müssen demnach nicht adaptiert werden. Feste Conditions können mit diesem Ansatz nicht verändert werden.
- *AP14 – Copy:* Wird ähnlich dem Move, mit einer selektiven seriellen Verwebung mit einer Instanz des gleichen Modells, realisiert.

Es ist festzustellen, dass fast alle Adaptionsmuster umsetzbar sind. Einzig AP12 und eingeschränkt AP13 sind nicht durch den Ansatz abbildbar. Dies ist eine deutliche Verbesserung zu den existierenden Ansätzen, selbst zu ADEPT2 (siehe Abschnitt 2.3.2).

Die in [191] vorgestellten Adaptionsmuster der Kategorie *Patterns for Changes to Predefined Regions*:

1. *PP1 – Late Selection of Process Fragments*
2. *PP2 – Late Modeling of Process Fragments*
3. *PP3 – Late Composition of Process Fragments*
4. *PP4 – Multi-Instance Activity*

flowmodell ein. Diese werden in diesem Ansatz vom Direktor implizit vorgegeben (Abschnitt 3.2).
[7] Diese Aufgabe muss der Direktor übernehmen (Abschnitt 3.2).
[8] Näheres zu diesem finden sie unter Abschnitt 3.2.

Muster	ADEPT2	YAWL	FLOWER	Adaption durch Verweben
AP1	X	-	-	X
AP2	X	-	X	X
AP3	X	-	-	X
AP4	-	X	-	X
AP5	-	-	-	X
AP6	X	-	-	X
AP7	X	-	-	X
AP8	X	-	-	X
AP9	X	-	-	X
AP10	-	-	-	X
AP11	X	-	-	X
AP12	X	-	-	-
AP13	X	-	-	o
AP14	-	-	-	X
PP1	-	X	-	X
PP2	-	-	-	X
PP3	-	-	-	X
PP4	-	-	-	X

Tabelle 3.1: Unterstützung instanzbasierter Adaptionsmuster (nach [191])

Die Adaption durch Verweben unterstützt jedes der vier. PP1 und PP2 wird als Replace (AP4) abgebildet, wobei bei PP2 das Workflowfragment zur Laufzeit modelliert und im Repository gespeichert wird. Bei beiden wird durch das Replace ein Platzhalter ersetzt. Das Muster PP3 entspricht der Adaption durch Verweben und kann durch serielle oder parallele Adaptionen umgesetzt werden. PP4 wird entsprechend des Musters Parallelize (AP9) umgesetzt.

Tabelle 3.1 zeigt zusammenfassend die Unterstützung der verschiedenen Adaptionsmuster durch die WfMS und die Adaption durch Verweben. Es ist deutlich zu sehen, dass die Adaption durch Verweben deutlich über den Stand der Technik hinausgeht.

3.1.3 Formale Definition der Adaption durch Verweben

Die im letzten Abschnitt beschriebene Möglichkeit der Workflowadaption mithilfe der Integration bestehender Workflowmodelle und einem variablen Rücksprungpunkt (unter Beachtung aller Einschränkungen) wurde bisher nur informell betrachtet. Für die Realisierung eines solchen Systems ist es notwendig, diesen Ansatz formaler zu beschreiben, um ihn später in ein WfMS zu überführen.

Als Basis für diese formale Beschreibung sollen Petri-Netze [132] genutzt werden. Diese Art der Systembeschreibung zeichnet sich durch ein gut untersuchtes, formales Modell aus und kann, wie verschiedene Veröffentlichungen zeigen [121, 181], als formale Basis für fast alle Workflow-Beschreibungssprachen genutzt werden. Abbildung 3.11 zeigt ein Beispiel für ein Petri-Netz. Ein Petri-Netz besteht grundsätzlich aus drei Elementen – Stellen, Transitionen und Kanten – die als Netzelemente bezeichnet werden. Eine Stelle wird durch einen Kreis im Petri-Netz repräsentiert und beschreibt den Zustand

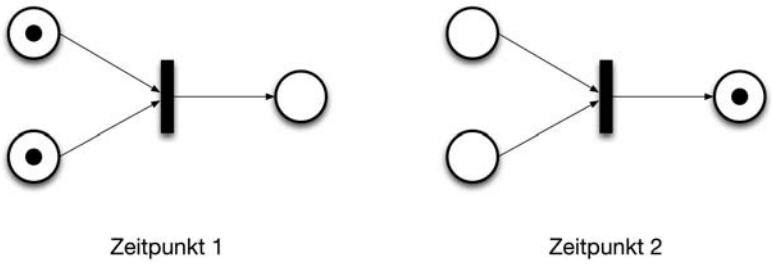


Abbildung 3.11: Petri-Netz

des abgebildeten Systems, aber keinerlei Aktion. Transitionen bilden den Übergang zwischen diesen Systemzuständen ab. Sie werden durch ein Rechteck dargestellt. Die Petri-Netz Transitionen sind vergleichbar mit den Aktionen in verschiedenen Workflow-Beschreibungssprachen. Die Beschreibung des Kontrollflusses erfolgt in Petri-Netzen über Kanten. Eine gerichtete Kante verbindet eine Stelle mit einer Transition und umgekehrt. Transitionen und Stellen werden meist mit dem Begriff Knoten zusammengefasst. Eine Kante kann eine beliebige Anzahl an Vor- bzw. Nachknoten besitzen.

Das Petri-Netz durchlaufen *Marken*, die wir bisher als Token kennengelernt haben. In Abbildung 3.11 werden die Marken durch die schwarzen Kreise dargestellt. Marken warten in Stellen, bis eine Transition schalten kann. In den meisten Petri-Netz-Varianten kann die Transition dann schalten, wenn die benötigte Anzahl an Marken an den Vorstellen anliegt. Die Anzahl kann durch Multiplizitäten an den Kanten festgelegt werden[9]. Dabei zeigt die Zahl an der Kante an, wie viele Marken beim Übergang von den Stellen zur Transition benötigt werden. Schaltet diese, werden die Marken der vorgelagerten Stellen konsumiert und eine festgelegte Anzahl von Marken erzeugt. Die Marken werden an die Nachstellen gesendet. Ihre Anzahl hängt von der Multiplizität der jeweiligen Kante ab.

Durch dieses spezielle Schaltverhalten der Transitionen ist eine Beschreibung von Synchronisationsverhalten möglich, wie Abbildung 3.11 zeigt. Es ist aber auch möglich, dieses spezielle Verhalten zur Komposition von Petri-Netzen zu verwenden, die auch den beiden vorgestellten Webearten entspricht. Es wird bei Petri-Netzen zwischen *Or*- und *And*-Komposition unterschieden. Eine Or-Komposition entspricht der seriellen Adaption. Abbildung 3.12 (a) stellt einen Or-Split dar. Dabei gibt es von einer Stelle zwei ausgehende Kanten. Da für jede Transition zum Schalten eine Marke gebraucht wird, kann nur eine von beiden schalten. Es ist demnach eine Oder-Entscheidung. Dieses Verhalten kann für die Komposition genutzt werden. Die Komposition besteht darin, in das bestehende Petri-Netz eine Kante zu einem fremden Petri-Netz hinzuzufügen. Das Token kann dann nur einen der beiden Pfade nutzen. In Petri-Netzen würde durch Zufall entschieden, welcher Pfad genutzt würde. Es ist aber für den hier vorliegenden Anwendungsfall möglich, die Engine anzuweisen, immer die neu hinzugefügte Kante zu nutzen. Der duale Fall existiert bei der And-Komposition. In Abbildung 3.12 (b) besitzt die Transition zwei ausgehende Kanten. Schaltet sie, entstehen zwei Token, einer pro Kante, d. h. beide Pfade werden beschritten. Die entsprechende Komposition entspricht dann der parallelen Verwebung.

Ein Petri-Netz besteht aus einem Tupel $N = \langle P, T, F, l \rangle$, bei dem gilt:

[9] Wird keine Zahl angegeben, wird eine Marke benötigt.

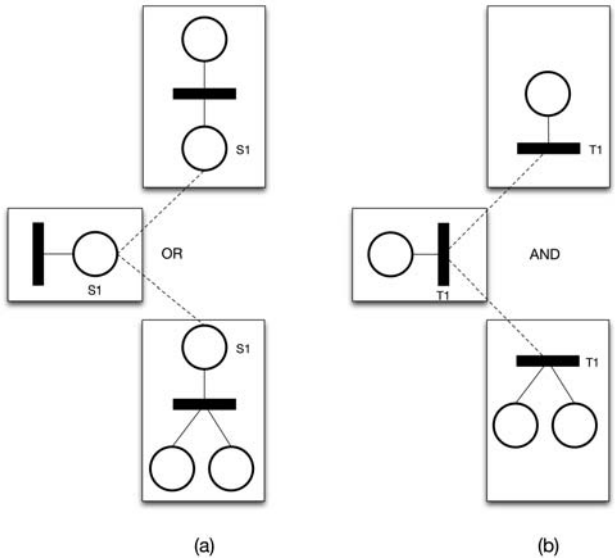


Abbildung 3.12: Komposition von Petri-Netzen

1. P und T sind zwei disjunkte nicht-leere endliche Mengen von Stellen (P) und Transitionen (T). Die Elemente der Vereinigung beider Mengen $P \cup T$ nennt man Knoten von N.
2. $F \subseteq (P \times T) \cup (T \times P)$ beschreibt die Kontrollflussrelation zwischen Stellen und Transitionen und umgekehrt (Kanten).
3. l ist die Markierungsfunktion für Transitionen, die jede Transition $t \in T$ einer Markierung $l(t) \in \sum$ zuweist, wobei $\sum$ eine endliche Menge von Markierungen darstellt.

Wie v. d. Aalst zeigte [187], ist es mit dieser Struktur möglich, die Kontrollflussstrukturen der meisten Workflow-Beschreibungssprachen abzubilden. Sie können somit auch als Basis für verwebte Workflows dienen. Dabei sind folgende Eigenschaften abzubilden:

- Repository für einzuwebende Workflows
- Adaption inkl. der Beschreibung des Rücksprungpunktes
- Einschränkungen des Rücksprungpunktes
- Variablen und Datenabhängigkeiten

Um dies zu erreichen ist es notwendig, die Petri-Netz-Beschreibung zu erweitern. Ein verwebtes Netz besteht aus einem Tupel $N = \langle P, T, F, l, V, B, S, A, M \rangle$, bei dem gilt:

- P, T, F und l sind dabei die Mengen des klassischen Petri-Netzes zur Beschreibung der Stellen, Transitionen, Kanten und der Marken.
- V ist die Menge aller möglichen Variablen.
- $B = \{(x, y) | x \in T \wedge y \in V\}$ beschreibt die benötigten Variablen einer Transition.
- $S = \{(x, y) | x \in T \wedge y \in V\}$ definiert, welche Transition welche Variable setzt.
- A beschreibt das Petri-Netz-Universum; die Menge aller Petri-Netze (jedes davon wird durch ein Tupel beschrieben)

- $M = \langle C, R \rangle$ beschreibt ein zweistelliges Tupel, dass die Adaption darstellt. Die Menge C beschreibt die Verwebungen zwischen Haupt- und Zielworkflow und R den Rücksprungpunkt und bildet dadurch neben F eine zweite Menge an Kanten, die sich ständig ändern kann.

Bei beiden Relationen ist die Adaptionsart für die Vorwärtssprung-Relation C zu beachten:

- Serielle Adaption fügt eine neue Kante zwischen Stelle und Zieltransition hinzu: $C = \{(x, y) | x \in P \wedge y \in T_A \wedge T_A \in A\}$ wobei T_A die Summe aller Mengen T in A darstellt.
- Parallele Adaption fügt eine neue Kante zwischen Transition und Zielstelle hinzu: $C = \{(x, y) | x \in T \wedge y \in P_A \wedge P_A \in A\}$ wobei P_A die Summe aller Mengen P in A darstellt.

Es wird dabei explizit nicht ausgeschlossen, dass der Zielworkflow dem Hauptworkflow entspricht. Zu beachten ist hier, dass es einen Unterschied macht, ob eine parallele oder serielle Adaption durchgeführt wird. Die serielle erfordert eine Or-Komposition, die parallele eine And-Komposition.

Die Rücksprungpunkt-Relation ist demnach die inverse Abbildung:

- Serielle Adaption fügt eine neue Kante zwischen Stelle und Zieltransition hinzu: $R = \{(x, y) | y \in P \wedge x \in T_A \wedge T_A \in A\}$
- Parallele Adaption fügt eine neue Kante zwischen Transition und Zielstelle hinzu: $R = \{(x, y) | y \in T \wedge x \in P_A \wedge P_A \in A\}$

Die Definition von R und C ermöglichen zugleich die selektive Adaption. Das bedeutet, dass nicht bis zum Ende der eingewobenen Instanz gewartet wird, sondern der Rücksprung kann mitten in der laufenden Instanz geschehen.

Für eine wohlgeformte Adaption gelten die folgenden Bedingungen:

- **Datenabhängigkeiten beachten:** (x, y) sei ein möglicher Rücksprungpunkt einer seriellen Adaption und N die Menge der bereits geschalteten Transitionen. Damit das Token von x zur Transition y springen kann, müssen alle Variablen, die y benötigt, gesetzt sein. Dazu wird für jede Variable b, die y benötigt, geprüft, ob eine Transition c in N existiert, die b setzt (d. h. $(c, b) \in S$). Der Rücksprungpunkt ist demnach nur gültig wenn gilt: $\forall (a, b) | (a, b) \in B \wedge a = y \wedge \exists (c, b) | (c, b) \in S \wedge c \in N$. Dieses gilt äquivalent für eine parallele Adaption.
- **Join beachten:** Sei A eine Fork-Transition mit mehreren ausgehenden Kanten, B die Join-Transition mit der gleichen Anzahl an eingehenden Kanten, SA die Menge der Nachfolgerkanten von A und SB die Menge der Vorgängerkanten für B[10], dann muss für einen Rücksprungpunkt y folgendes gelten: $y \in SA \cap SB$. Das bedeutet, dass bei einer Adaption auf einen geteilten Token (nach Fork), der Rücksprungpunkt immer zwischen dem Fork und Join liegen muss.

Mit der Adaption durch Verweben werden die Adaptionsmuster unterstützt. Evolutionäre Änderungen hingegen können mit diesem Ansatz nicht abgebildet werden, da die Adaptionen sich immer auf nur eine Instanz beziehen. Dafür ist eine erweiterte Kontrollinstanz notwendig, ähnlich der in AgentWork vorgestellten Regelebene.

[10] Eine Berechnungsvorschrift für diese beiden Mengen finden sie im Anhang A.

3.2 Autonome Adaptionssteuerung

Das Verweben von Workflowmodellen erfordert, dass das WfMS ein Repository von verschiedenen Workflowmodellen anbietet. Theoretisch ist es damit möglich, jedes Workflowmodell in eine beliebige Workflowinstanz einzuweben. Dies spannt einen Variantenraum über dem Gesamtsystem auf. Im Gegensatz zu den Ad-Hoc Workflows ist es somit möglich, aus einer endlichen Menge von potentiellen Varianten zu wählen.

Die Adaption durch Verweben basiert darauf, dass zum richtigen Zeitpunkt eine richtige Adaptionsentscheidung getroffen wird. Die Beantwortung der Frage, ob vom vorgesehenen Abarbeitungspfad abgewichen werden soll bzw. muss, ist dabei von vielen Faktoren abhängig. Dies beinhaltet den aktuellen Zustand des Token inklusive aller Variablenbelegungen, dem aktuellen Abarbeitungszustand des Workflows, aufgetretenen Ausnahmen und den Umgebungsbedingungen, die durch die Workflow-Engine eingebracht werden. All diese Faktoren beschreiben die aktuelle *Situation* der Workflowinstanz, die Grundlage einer jeden Entscheidungsfindung ist. Zudem ist mit der Erkenntnis, dass eine Adaption stattfinden soll, nur die halbe Arbeit getan. Es ist zudem eine der möglichen Verwebungen aus dem Variantenraum zu wählen. Die Person oder Maschine, die diese Entscheidung triff, muss dazu erst alle potentiell anwendbaren Varianten feststellen, bevor die beste Reaktion auf die aktuelle Situation gewählt wird. Ist für die aktuelle Situation keine Adaption notwendig oder vorgesehen, wird keine Entscheidung getroffen. In diesem Fall fährt die Workflow-Engine mit der Abarbeitung der Instanz wie vorgesehen fort.

In der Praxis sind *on-the-fly*-Modifikationen bisher weniger anzutreffen als evolutionäre Veränderungen bei strukturierten Workflows. Der Hauptgrund dafür ist der höhere Aufwand beim Betrieb. Evolutionäre Veränderungen beziehen sich auf alle Instanzen und versprechen eine dauerhafte Leistungssteigerung. Darüber hinaus ist der Zeitpunkt von vielen Rahmenbedingungen abhängig. Evolutionäre Veränderungen erfolgen zudem nicht während der Laufzeit. Im Gegensatz dazu benötigen die hier vorgeschlagenen instanzbehafteten Laufzeitadaptionen eine Einzelbetreuung. *On-the-fly*-Adaptionen müssen zur Laufzeit überwacht werden und erfordern, im Falle einer kritischen Situation, ein rasches Eingreifen. Werden die Änderungen manuell, von einer verantwortlichen Person durchgeführt, kann dies nicht gewährleistet werden. Es ist möglich ihm die Arbeit zu vereinfachen, indem der Designer die Entscheidung trifft, an welchen Stellen des Workflows Adaptionen möglich sind. Dies würde jedoch die Adaptionsmöglichkeiten reduzieren und eher dem Late Binding (siehe Abschnitt 2.3.3) gleichen.

Besteht der Workflow allerdings weitestgehend aus automatisierten Aktivitäten, ist ein manuelles Eingreifen hinderlich. Ist eine Änderung notwendig, muss das WfMS auf die Eingaben des Überwachers warten, damit dieser eine Modifikation für den aktuellen Zustand angibt. Ist die Entscheidung von sich kurzfristig ändernden Rahmenbedingungen abhängig, ist es kaum möglich, eine Änderung sinnvoll durchzuführen. Je kürzer die Zeitspannen der Ausführung der Aktionen sind, desto kürzer ist die Zeit, die dem Benutzer bleibt, um adäquat zu reagieren. Diese Probleme stehen einer weiten Verbreitung von *on-the-fly*-Adaptionen entgegen. Der einzige vernünftige Weg um *on-the-fly*-Modifikationen zu verwalten, ist es, die Entscheidung für eine bestimmte Änderung und deren Anwendung zu automatisieren.

3.2.1 Reflexion zur Steuerung von Adaptionen

Eine solche Automatisierung bedeutet, dass eine Komponente des WfMS Änderungen selbst berechnet, wählt und anwendet. Dazu ist der aktuelle Zustand (Situation) des Workflows vollständig zu betrachten, ohne den eine Entscheidung nicht sinnvoll getroffen werden kann. In [100] werden diese Änderungen auf Basis von Selbstbeobachtung – Introspection – als *Reflexion* bezeichnet.

Reflexion übertragen auf WfMS, ist die Fähigkeit seine Struktur zu beobachten und auf Basis dieser Informationen sich selbst zu adaptieren [101]. Dies stellt eine Reaktion auf seinen aktuellen Zustand dar, den es zu bewerten gilt. Dazu wird vorausgesetzt, dass ein Workflow ein Ziel besitzt. Erst dieser macht es möglich, Anforderungen an die Ergebnisse der Workflowausführung zu stellen und die aktuelle Situation mit dem Idealzustand zu vergleichen. Ein WfMS muss demnach eine Erwartung über die Ausführung einzelner Workflows besitzen. Weicht eine Instanz von den Erwartungen ab, gilt es Anpassungen vorzunehmen. Abweichungen können durch Ausnahmen auftreten, aber auch durch Veränderungen im WfMS selbst, beispielsweise, dass bestimmte Aufgaben langsamer abgearbeitet werden. Erst Selbstbeobachtung und Selbsteinschätzung erlauben automatische Adaptionen.

Reflexion bietet somit ein erweitertes Verständnis von *dynamischen Workflows*. Diese ändern sich zwar aufgrund schlechter Performance oder Ereignisse, besitzen aber nicht die Möglichkeit zur Selbstanalyse und somit keine Form der Automatisierung. Wie das Beispiel Agentwork zeigt, werden die Bedingungen zur Adaption manuell festgelegt. Die reflexive Steuerung hingegen vereinheitlicht Adaption und Selbstbeobachtung. Adaption bietet die Möglichkeit, auf ungünstige Situationen zu reagieren, während die Introspection die Informationen für die Analyse und Bewertung der Situationen liefert.

3.2.2 Reflexiv autonomer Weber

Die Idee der reflexiv gesteuerten Workflow-Verwebung besteht aus drei zentralen Schritten: (1) Überwachung der Workflowinstanz, (2) Berechnung bzw. Auswahl einer geeigneten Adaption und (3) Anwendung der Adaption auf die Instanz. Als wichtige Rahmenbedingung gilt es zudem zu beachten, dass die *reflexive Adaption durch Verweben* in verschiedene bestehende WfMS integriert werden soll. Dazu ist es notwendig, diese drei Aufgaben losgelöst von der Workflow-Engine (ausführende Komponenten) zu bearbeiten. Um dies zu gewährleisten, sind offene Schnittstellen zur Kommunikation zwischen allen Komponenten notwendig. Eine Grundvoraussetzung für diese Integration ist, dass die Workflow-Engine die grundlegenden Operationen für die Adaption durch Verweben zur Verfügung stellt (siehe Abschnitt 3.1.2). Die reflexiv gesteuerte Verwebung wird durch eine spezielle Architektur – dem autonomen Weber – realisiert.

Ein autonomer Weber muss die oben genannten drei Aufgabenfelder abdecken. Dafür sind zwei unterschiedliche Komponenten vorgesehen: (1) *Koordinator* und (2) *Direktor* (siehe Abbildung 3.14). Beide Komponenten werden lose an die bestehende Workflow-Engine gekoppelt. Der *Koordinator* ist zuständig für die Anwendung der ausgewählten Adaption auf die Workflowinstanz. Über seine Schnittstelle stellt er dabei die notwendigen Funktionen für die Adaption zur Verfügung. Als Eingabe benötigt er folgende Informationen: einzuwebender Workflow, Art des Webevorgangs und Rücksprungpunkt. Diese Daten werden dann in Befehle für die jeweilige Workflow-Engine gewandelt und

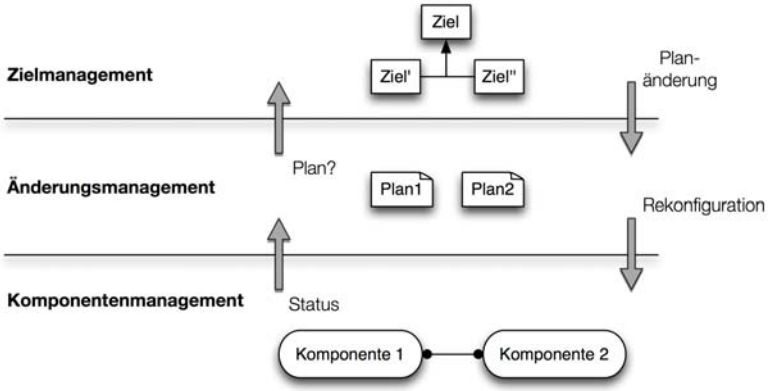


Abbildung 3.13: Drei-Schichten-Architektur für selbstadaptive Systeme (nach [65, 105])

abgesendet. Da es verschiedene Engines mit unterschiedlichen Befehlssätzen gibt, ist es notwendig, dass es verschiedene Koordinatoren für die jeweiligen Engines gibt. Der Koordinator muss demnach dem Entwurfsmuster *Mediator* [63] entsprechen, denn er vermittelt zwischen dem Direktor und der jeweiligen Workflow-Engine, indem der die Adaptionsvorschriften anpasst.

Der *Direktor* stellt die notwendigen Adaptionsinformationen zur Verfügung. Er entspricht demnach der Regelebene, der die laufende Instanz überwacht und die notwendigen Anpassungen berechnet, die er an den Koordinator übermittelt. Die Überwachung der Workflow-Engine erfolgt nicht durch wiederholtes Erfragen des Status der Instanz. Stattdessen informiert die Engine den Direktor über die Fortschritte und Änderungen der Instanz. Dies entspricht dem Entwurfsmuster *Observer* [63]. Zusätzlich besitzt der Direktor Zugriff auf alle bereits modellierten Workflowmodelle im Repository.

Die Aufteilung der Aufgaben zwischen Workflow-Engine, Koordinator und Direktor orientiert sich dabei an der in [65, 105] beschriebenen Drei-Schichten-Architektur für selbstadaptive Systeme, wie sie in Abbildung 3.13 dargestellt ist. Ziel dieser Architekturform ist es, dass ein Softwaresystem, bestehend aus Komponenten, hinsichtlich vorgegebener Ziele verändert werden kann. Dazu werden Pläne berechnet, die zu einer Rekonfiguration der Komponenten führen. Die Basisschicht ist das *Komponentenmanagment*, das für die Ausführung der Komponenten zuständig ist. Übertragen auf den Kontext Workflows entspricht dies einer Workflow-Engine. Der Status der Komponenten wird vom *Änderungsmanagement*, das die Pläne verwaltet, an das *Zielmanagement* weitergeleitet. Ändern sich Stellgrößen, werden die Ziele dahingehend angepasst. Jede Zieländerung bewirkt dabei eine Änderung der auszuführenden Pläne. Diese werden an das Änderungsmanagement weitergeleitet, das diese für das Komponentenmanagement umsetzt. Das Änderungsmanagement wird im Kontext von Workflows durch den Koordinator repräsentiert, der die Adaptionen für die Workflow-Engine umsetzt. Die Adaptionen werden im Zielmanagement berechnet, dessen Aufgaben denen des Direktors gleichen.

Reflexiver Direktor Adaptionsentscheidungen berechnet ein Direktor auf Basis einer Wissensbasis. Dies kann, wie im Falle von AgentWork, eine Menge von ECA-Regeln sein oder, wie die weiteren Kapitel zeigen werden, auch Wissensrepräsentationen aus dem Forschungsbereich der künstlichen Intelligenz. Diese Wissensbasis kann jederzeit neues

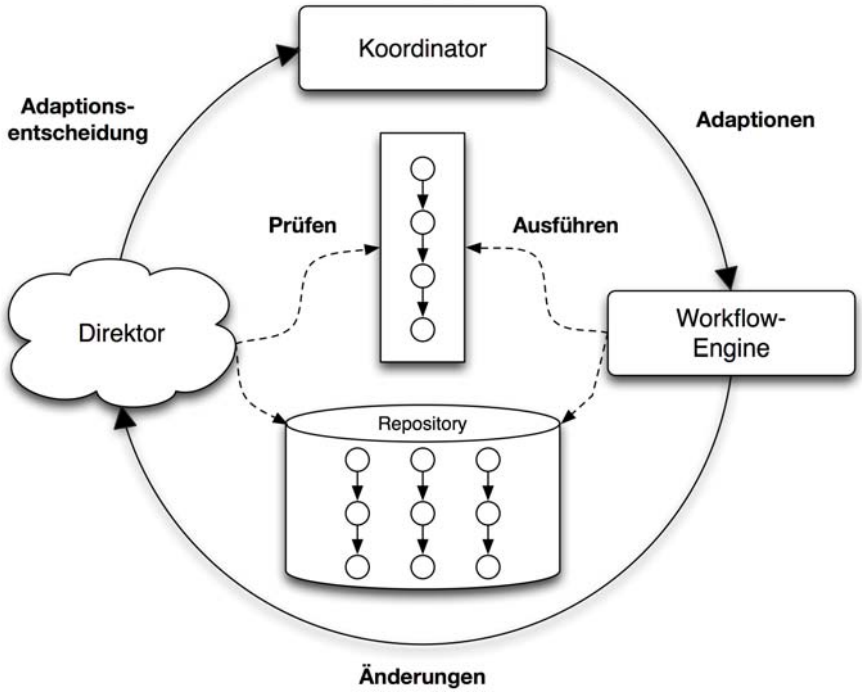


Abbildung 3.14: Autonomer Weber

Wissen aufnehmen und dementsprechend *lernen*. Wenn ein Direktor eine Adaption vorschlägt, speichert er diese und gleichzeitig das Ergebnis, d. h. ob die Adaption positiv oder negativ verlief. Dieses Wissen kann der Direktor bei folgenden Adaptionsentscheidungen wiederverwenden. Ein lernender Direktor kann demnach in einer gleichen oder ähnlichen Situation, ein positives Ergebnis vorausgesetzt, die gleiche Adaption wieder anwenden. Dadurch verstetigen sich erfolgreiche Adaptionsentscheidungen im System und werden quasi permanent. Dies entspricht evolutionären Adaptionen, ohne dass diese explizit im Workflowmodell verankert werden. Der Direktor stellt für jede Instanz sicher, dass die Adaption ausgeführt wird. Erst dadurch können dynamische Workflows auch evolutionäre Adaptionen unterstützen.

Der Direktor muss einen großen Situationsraum verarbeiten. Eine Situation umfasst folgende Daten:

- Eine Referenz auf das von der Instanz ausgeführte originale Workflowmodell. Die verfügbaren Modelle sind in einem Repository gespeichert, wie in Abbildung 3.14 gezeigt, und sind durch eine eindeutige Kennung adressierbar.

- Die aktuelle Position des Token in der Instanz, die Stelle, an der die Workflow-Engine den Direktor nach einem Vorschlag fragt. Dies kann sowohl vor als auch nach einer Aktivitätsausführung sein, wobei abgesehen von Start- und Endknoten jede Position möglich ist.

- Alle Ausnahmen, die bis zu diesem Punkt aufgetreten sind.

- Variablendefinitionen und Variablenbelegungen, die für jede Instanz unterschiedlich sind.

Das Resultat der Berechnung ist eine Adaptionsanweisung für den Koordinator. Diese besitzt folgende Möglichkeiten:

- *Fortfahren*: Dies entspricht der Empfehlung, keine Änderungen vorzunehmen und das bestehende Workflowmodell wie geplant abzuarbeiten. Diese Option ist erforderlich, um Workflows in einer regulären Art und Weise zu beenden.
- *Adaptieren*: Das Einweben eines Subprozesses, wie in Abschnitt 3.1 beschrieben. Dieser Vorschlag enthält folgende Parameter: die ID des Ziel-Workflowmodells, den Rücksprungpunkt und für das selektive Einweben zusätzlich die ID der Start- und der End-Aktivität, zur Eingrenzung des Fragmentes. Des Weiteren ist anzugeben, ob es sich um eine serielle oder parallele Adaption handelt.
- *Abbrechen*: Diese Anweisung kann auf der eingewobenen Instanz angewendet werden und bedeutet den sofortigen Rücksprung zum aufrufenden Workflow. Dies bedeutet auch, dass alle Fortschritte innerhalb der Adaption verloren gehen. Diese Option ist vor allem zur Unterbrechung von endlosen, iterativen Adaptionen oder zum Auflösen von Deadlocks notwendig.

Die Kombination von Adaption durch Verweben und der autonomen Steuerung hat einige Ähnlichkeiten zur Aspektorientierung. Diese wird zur Vermeidung von Cross-Cutting eingesetzt, d. h. es wird versucht, verschiedene Fragmente wiederzuverwenden. Dazu werden *Advices* an *Join Points* durch eine Weaver (Weber) in den Programmcode gewoben. Die Anweisung zum Weben und unter welchen Bedingungen dies geschehen soll wird in einem *Point Cut* beschrieben. Übertragen auf die Adaption durch Verweben ist ein Workflow ein Advice, der durch den Koordinator (Weber) zur Laufzeit in eine Instanz eingewoben wird. Den Punkt (Join Point) und die Bedingungen (Point Cut) dafür werden von Direktor berechnet. Bei der Adaption durch Verweben werden Point Cuts erst zur Laufzeit berechnet und gelten nur für die jeweilige Workflowinstanz. Zudem ist das Konzept des *variablen Rücksprungpunkts* konzeptionell bei der Aspektorientierung nicht vorhanden.

Reflexiver Regelkreis Mit den beiden Komponenten ist es möglich, die Eigenschaften der Reflexion – Introspektion und Adaption – als geschlossenen Kreislauf abzubilden. Abbildung 3.14 zeigt diese Verbindung aller beschriebenen Komponenten. Die Workflow-Engine führt eine Instanz aus und teilt alle Fortschritte dem Direktor mit. Er vergleicht die aktuelle Situation mit dem Sollwert und teilt ggf. notwendige Adaptionen dem Koordinator mit, der die Änderungen für die jeweilige Engine umsetzt.

Diese Form der Interaktion wird in der Literatur als Feedback Loop [51] bezeichnet und ist wie ein Regelkreis aufgebaut, wie Abbildung 3.15 zeigt. Er besteht aus vier Einzelschritten: (1) Informationen sammeln, (2) Analyse, (3) Entscheiden und (4) Agieren. In der hier vorgestellten Lösung übernimmt die Engine Schritt 1, der Direktor 2 bzw. 3, der Koordinator Schritt 4. Die Funktionen der vier Einzelschritte entsprechen dabei denen der hier vorgestellten Komponenten. Die Zusammenlegung von Analyse und Entscheidung in einer Komponente ist hierbei keinerlei Einschränkung im Ansatz.

Im Gegensatz zum Koordinator ist ein Direktor nur für eine Workflowinstanz zuständig. Für jede gestartete Instanz wird dazu ein Direktor initialisiert. Ein Direktor kann demnach als *stateful* bezeichnet werden. Dies ist notwendig, erlaubt es doch, nicht nur die aktuelle Situation zu beurteilen, sondern auch die Historie der Instanz mit in Betracht

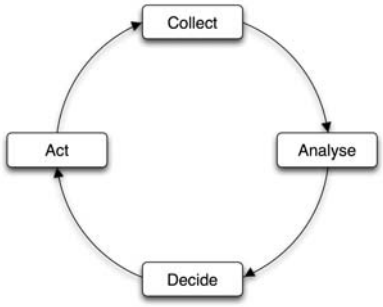


Abbildung 3.15: Feedback Loop (nach [51])

zu ziehen. Diese ist für jede Instanz des gleichen Workflow unterschiedlich und muss demzufolge genau berücksichtigt werden. Die Workflow-Engine benötigt diese Daten zu Abarbeitung der Instanz nicht und zeichnet diese zumeist nicht auf. Die Kopplung von Instanz und Direktor schließt dabei aber nicht aus, dass zwischen den Direktoren auch eine Art gemeinsames Wissen existieren kann[11]. Für den Koordinator ist eine solche Bindung nicht notwendig, setzt er doch nur als Vermittler die Adaptionsbefehle um.

Bisher wurde die Referenzarchitektur nur unter dem Gesichtspunkt der Automatisierung betrachtet. Sie schließt aber keineswegs den Menschen aus, sondern erlaubt es, manuelle Änderungen zu verwirklichen. Um diese zu realisieren, ist der Koordinator geeignet, denn er nimmt berechnete Adaptionen an. Dabei ist völlig unerheblich, wer diese Adaption berechnet hat. Der Benutzer kann z. B. über ein graphisches Interface verschiedene Adaptionen modellieren und anschließend an den Koordinator übermitteln, der diese übersetzt und ausführen lässt. Die dafür notwendige Unterbrechung der Ausführung übernimmt die Workflow-Engine. Dieses Nebeneinander hat außerdem den Vorteil, dass der reflexive Direktor solche Adaptionen in seine Wissensbasis aufnehmen kann. Damit ist es möglich, dass der Direktor ständig neue Informationen zur Verbesserung seiner Entscheidungsfindung bekommt.

3.2.3 Anforderungen an den Direktor

Wie bereits erwähnt, beruht die Adaptionsentscheidung des Direktors auf der aktuellen Situation, d. h. dem Zustand der Instanz. Der Situationsraum, die Menge aller möglichen Situationen, die eine Instanz annehmen ist, allein durch die verschiedenen Variablenbelegungen, unendlich. Ein idealer Direktor sollte jedoch *Vollständigkeit* gewährleisten, d. h. in allen Situationen einen gültigen Vorschlag zu berechnen. Aufgrund des großen Situationsraumes ist es unmöglich, dass der Direktor wirklich jede Situation in einer Wissensbasis speichern kann. Vielmehr ist es sinnvoll, für eine bestimmte Situation Lösungen zu suchen, die für eine ähnliche bereits angewandt wurden. Ein Vorschlag kann dabei nicht die perfekte Entscheidung darstellen, sondern sollte eine valide Ausführung gewährleisten.

Ein idealer Adaptionsvorschlag verbessert das Workflowmodell in Bezug auf die aktuelle Situation. Da die Ausführung des Workflows, in Bezug auf die Ergebnisse von Aktivitäten nicht vollständig vorhersehbar ist, gibt es eine große Unsicherheit, wie sich bestimmte Adaptionen auf das Verhalten auswirken und damit, ob eine Adaption eine

[11] Siehe Kapitel 6.

Leistungssteigerung bewirkt. Daher ist es nicht möglich, einen idealen Direktor zu konstruieren. Es ist ebenfalls zu bedenken, dass der Direktor Vorschläge unterbreitet, die zu keinem gewünschten Ergebnis führen. Um dies langfristig auszuschließen, ist es wichtig, dass der Direktor eine Rückmeldung über seine Entscheidungen bekommt. Über diese Rückkopplung kann er schlechte Entscheidungen für die nächsten Berechnungen vermeiden (Reflexion).

Für den Einsatz in realen Anwendungen ist es zudem notwendig, die automatischen Adaptionen zu überwachen, um ggf. einzugreifen oder Fehler nachzuvollziehen. Dazu muss der Direktor *transparent* sein, d. h. verständliche Informationen über seine Entscheidungen zugänglich machen. Für einen transparenten Direktor ist es wichtig, sein Wissen über die verschiedenen Elemente derart zu speichern, dass Informationen in lesbarer, strukturierter Form vorliegen. Je mehr ein Direktor lernt, desto komplexer oder zumindest größer wird seine Wissensbasis. Es darf aber nicht zu dem Fall kommen, dass nach einiger Zeit die Wissensbasis derart groß wird, dass sie unüberschaubar und nicht zu warten ist.

Eine weitere Notwendigkeit für den produktiven Einsatz ist die Performance der Entscheidungsfindung. Wenn die Workflow-Engine nach einer Adaption fragt, wird die Instanz angehalten, bis der Koordinator eine Antwort erhält. Um eine kurze Ausführungszeit und geringen Ressourcenverbrauch der Workflowausführung zu gewährleisten, ist es notwendig, dass der Direktor innerhalb einer angemessenen Zeit eine Lösung anbietet. Dies wird zum einen dadurch abgemildert, dass der Direktor nicht ständig Lösungen berechnen soll, sondern nur auf Anfrage der Workflow-Engine. Da aber viele Instanzen gleichzeitig ablaufen und diese ebenfalls einen Direktor besitzen, ist jedoch eine ressourcenschonende Lösung wichtig.

3.3 Zusammenfassung

In diesem Kapitel wurde eine neuartige Form für Laufzeit-Rekonfiguration von Workflows eingeführt – *reflexive Adaption durch Verweben*.

Bei der Adaption durch Verweben werden alle Änderungen durch das Einweben von Workflowmodellen realisiert. Die Idee ist, dass bei Bedarf Workflowfragmente in die zu adaptierende Instanz eingewoben werden. Dies geschieht über eine eingeführte Verzweigung, die in den Kontrollfluss eingebracht wird. Diese Verzweigung kann zwei unterschiedliche Adaptionen realisieren (siehe Abbildung 3.1):

- *Parallele Adaption:* Hierbei wird der Kontrollfluss aufgeteilt. Anschließend wird ein Token in die neue Instanz überführt, der andere verbleibt in der Urspungsinstanz und setzt die Arbeit wie vorgesehen fort.
- *Serielle Adaption:* Im Gegensatz zur parallelen Adaption wird das Token nicht geteilt, sondern wird direkt in die neue Instanz überführt.

Die einzuwebenden Workflows bzw. Fragmente werden aus einem Repository gewählt. Um einen vollständig rekonfigurierbaren Kontrollfluss zu erreichen, wurde das Konzept des *variablen Rücksprungpunkts* integriert. Nach Abarbeitung des eingewobenen Workflowfragmentes kann das Token dadurch an einem beliebigen Punkt des Ursprungsworkflows zurückkehren, solang dadurch keine Inkonsistenzen, wie beispielsweise durch Datenabhängigkeiten, entstehen. Die Kombination aus Verweben und variablem Rücksprung-

punkt ermöglicht es, die Vielzahl von Adaptionsmustern, wie sie in [191] beschrieben sind, zu unterstützen. Der Umstand, dass lediglich fertige Workflowfragmente eingewoben werden, bietet zudem den Vorteil, dass sich, im Gegensatz zu Ad-hoc Workflows, die Anzahl der möglichen Adaptionen verringert. Dies stellt eine Form der *Faktorisierung* dar, d. h. dass ein komplexer Workflow in unterschiedliche, abgeschlossene Teile partitioniert wird. Für einen Vorgang wird damit nur ein *goldener Pfad* modelliert. Alle Ausnahmebehandlungen oder selten vorkommenden Varianten werden durch Workflowfragmente je nach Bedarf zur Laufzeit eingewoben. Dies führt zu einer einfacheren, übersichtlicheren Spezifikation. Ein weiterer Vorteil dieses Ansatzes ist es, dass neue oder fremde Arbeitsabläufe in einen bestehenden Workflow einfach integriert werden können, ohne den grundsätzlichen Ablauf des Workflows zu verändern.

Notwendig für die praktische Einführung der Adaption durch Verweben ist eine funktionierende Steuerung. Manuelle Adaptionen sind für viele Anwendungsfälle mit zu hohen Verzögerungszeiten verbunden. Eine Instanz, die automatische Adaptionen berechnet, ist hier von Vorteil. Diese muss zur Selbstbeobachtung und Selbsteinschätzung fähig sein, um sich automatisch anzupassen. Dieses Verhalten wird reflexiv genannt. Zur Umsetzung einer reflexiven Adaptionssteuerung wurden eine neuartige Referenzarchitektur einwickelt: der autonome Weber. Dieser beinhaltet die zwei Hauptkomponenten. Ein *Koordinator*, der die Adaptionen für verschiedene Workflow-Engines umsetzt und ein *Direktor*, der diese Adaptionen berechnet. Dies geschieht aufgrund von Hintergrundwissen, gespeichert in einer Wissensbasis. Durch stetiges Lernen ist es dem Direktor zudem möglich, evolutionäre Adaptionen abzubilden. Es ist allerdings bisher nicht untersucht, wie eine solche Wissensbasis gestaltet sein muss und welche Algorithmen sich zu Adaptionsberechnung eignen.

Dieser Adaptionsansatz macht nicht nur ein anderes Vorgehen im Design von Workflows notwendig, sondern erzwingt dieses beinahe. Durch die Möglichkeit, an jedem Punkt der Instanz einen Subworkflow (automatisch) einzubinden, entfällt zum einen die Notwendigkeit der expliziten Ausnahmebehandlung, zum anderen der expliziten Entscheidungspunkte. Zudem kann der Direktor Verzweigungen beschließen, die den klassischen Entscheidungspunkten entsprechen. Somit repräsentiert der Direktor eine übergeordnete Regelebene, die nicht direkt in den Workflow integriert ist. Dieses wurde ähnlich auch in AgentWork (siehe 2.3.2) vorgesehen. Dadurch müssen nicht alle Entscheidungen in den Workflow integriert werden, sondern befinden sich auf einer leicht zu ändernden Metaebene.

Im folgenden Kapitel soll eine einfache Umsetzungsmöglichkeit für einen Direktor vorgestellt werden. Dieser ist vor allem für die Steuerung der Adaption durch Verweben in sicherheitskritischen Workflows geeignet.

Kapitel 4

Eingeschränkte autonome Rekonfiguration

Wie im vorangegangenen Kapitel beschrieben, soll die Steuerung der Adaptionen der Direktor, in Kombination mit dem Koordinator, autonom vornehmen. Die Autonomität des Direktors macht es erforderlich, dass dieser auf Grundlage einer eigenen Wissensbasis Entscheidungen trifft. Bei der Konzeption dieser Wissensbasis gibt es zwei grundlegende Variationsmöglichkeiten: *Genauigkeit* und *Reflexivität*. Bei der Genauigkeit ist zu entscheiden, wie genau beschrieben wird, unter welchen Bedingungen welche Adaption durchgeführt werden soll. Je genauer die Bedingungen beschrieben werden, desto sicherer ist die Anwendung der Adaption. Das bedeutet jedoch auch, je genauer die Bedingungen, desto weniger kann auf unerwartete Veränderungen reagiert werden. Zudem kann entschieden werden, ob die Wissensbasis zur Laufzeit autonom verändert bzw. erweitert werden kann (Reflexivität) oder ob diese statisch gestaltet ist.

Die ungenaue, reflexive Wissensbasis ist für unerwartete Änderungen geeignet, da sie jederzeit angepasst werden kann. Sie stellt demnach das höchste Maß an Flexibilität für die Direktoren dar. Eine genaue, statische Wissensbasis hingegen ist für sicherheitskritische Workflows geeignet, denn die Bedingungen für die Adaptionen sind vorgegeben und werden sich nicht (ungewollt) ändern. Es entsteht ein *regulärer Variantenraum* im Sinne einer regulären Sprache [85]. Viele Workflows, seien es *Productive* oder *Administrative* Workflows, stellen die Anforderung, möglichst fehlerfrei abzulaufen, da sonst gravierende Probleme für Mensch und Umgebung auftreten. Ein Beispiel dafür ist der Workflow bei einem Brandeinsatz, wie er in Abschnitt 2.1.2, vorgestellt wurde. Eine Situation, bei der eine Adaption notwendig wäre, ist die Folgende. Kommt es während der Aktion der Brandbekämpfung zu einer Situation, dass noch Menschen in einem zu löschenden Gebäude anwesend sind, muss eine Unterbrechung der Brandbekämpfung erfolgen, da sonst diese Menschen aufgrund des hohen Wasserdrucks der Löschvorrichtungen verletzt werden können. Dieser Fall hat zur Folge, dass der Prozess der Menschenrettung gestartet werden muss.

Im Bereich der Workflows stellt dieses Szenario eine Adaption dar, bei der der Direktor, basierend auf der aktuellen Situation, den Menschenrettungsprozess starten und den Prozess des Brandeinsatzes unterbrechen muss. Angenommen, statt des Auffindens von Menschen im Gebäude tritt ein Flashover auf, muss an dieser Stelle eine andere spezifische Vorgehensweise (Flashover-Bekämpfung) zum Einsatz kommen. Dabei darf die normale Brandbekämpfung nicht unterbrochen werden. Muss der Direktor an dieser Stelle über eine Adaption entscheiden und beurteilt diese Situation aufgrund fehlender Erfahrung als ähnlich im Vergleich zum Auffinden von Menschen, stoppt er die Brandbekämpfung und startet mit der Menschenrettung den falschen Prozess. Folglich unterbricht er die Wasserzufuhr, was bei einem Flashover fatal wäre, da Wasser ein essentielles Mittel zur Bekämpfung dessen darstellt. Befinden sich hingegen keine Feuerwehrleute im Haus, ruft

dieses Verhalten keine Probleme hervor. Sobald sich jedoch Menschen während dieser Situation innerhalb des Gebäudes aufhalten, kann es zu lebensbedrohlichen Zuständen kommen, die auf jeden Fall vermieden werden müssen.

In diesem Kapitel soll deshalb eine Lösung zur Steuerung von sicherheitskritischen Workflows vorgestellt werden, die auf einer genauen, statischen Wissensbasis basiert. Die Idee ist, dass der Nutzer verschiedene Situationen für ein Workflowmodell beschreibt, in denen jeweils nur durch ihn definierte Adaptionen durchgeführt werden. Dazu werden Zustandsdiagramme genutzt. Ein Zustandsdiagramm dient im Normalfall dazu, das Verhalten eines Objektes während seines Lebenszyklus zu modellieren. Dieses Konzept wird hier auf die Workflow-Rekonfiguration übertragen. Im Zustandsdiagramm werden alle Adaptionszustände beschrieben, die ein Workflow annehmen kann. Eine Adaption stellt dabei einen Wechsel zwischen zwei Zuständen dar. Durch die Kombination von ECA-Eigenschaften und Zustandsbehaftung bieten die Zustandsdiagramme die Möglichkeit, Workflows auf eine Vielzahl von Situationen reagieren zu lassen, wobei diese jedoch vor Ausführungsbeginn fest definiert sind. Es werden nur die im Zustandsdiagramm vorgesehenen Adaptionen durchgeführt, so dass keine ungewollten Änderungen durchgeführt werden.

4.1 Grundlagen für Zustandsdiagramme

Zustandsdiagramme sind eine Diagrammform aus dem Katalog der *Unified Modeling Language* [117] (UML) und wurden erstmals von Harel beschrieben [74]. Sie dienen in der UML dazu, die verschiedenen Lebensstadien eines Objektes zu beschreiben. Ein Zustandsdiagramm besteht dabei aus zwei Basiselementen – Zuständen und Transitionen. Der Zustand eines Objektes stellt die gegenwärtige Situation dar, die er nur durch neue Ereignisse verlassen kann. Dieser Vorgang wird *Zustandsübergang* genannt. Bei diesem Übergang, der auch an Bedingungen geknüpft sein kann, können Aktionen ausgeführt werden.

Ein Zustand wird im Diagramm als Rechteck mit abgerundeten Ecken, wie in Abbildung 4.1 dargestellt, gezeichnet und beinhaltet verschiedene Eigenschaften [117]:

- Name - eine eindeutige Identifizierung, um einen Zustand von einem anderen zu unterscheiden
- Eingangs/Ausgangs-Aktionen - Aktionen, die beim Betreten bzw. Verlassen des Zustandes ausgeführt werden
- Interne Transitionen - Transitionen die, wenn behandelt, keinen Zustandsübergang hervorrufen
- Unterzustände - Zustände innerhalb eines Zustandes, die entweder nebenläufig oder sequentiell absolviert werden
- Verzögerte Ereignisse - Liste von Ereignissen, die nicht im aktuellen Zustand behandelt, sondern für zukünftige Zustände hinausgezögert werden

Des Weiteren existieren mit dem Anfangs- und Endzustand zwei Pseudozustände. Der Anfangszustand legt die Startposition des Zustandsdiagrammes oder Subzustandes fest, dargestellt als gefüllter Kreis. Der Endzustand, ein gefüllter Kreis umgeben mit einem

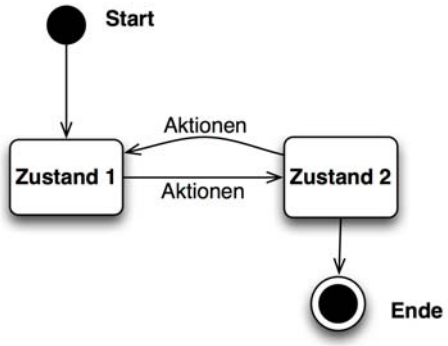


Abbildung 4.1: Einfaches Zustandsdiagramm

Ungefüllten, gibt an, dass die Ausführung des Zustandsdiagrammes oder des Subzustandes beendet ist.

Eine Transition beschreibt den Übergang zwischen zwei Zuständen, hervorgerufen durch das Auftreten eines speziellen Ereignisses. Damit nicht auf jedes beliebige Ereignis sofort reagiert wird, muss eine vorher definierte Bedingung erfüllt sein. Ist mit dem Zustandsübergang eine Handlung verbunden, ist es zudem möglich, dass beim Übergang eine Aktion ausgeführt wird. Diese Elemente entsprechen grundlegend denen der ECA-Regeln, mit dem Unterschied, dass sie explizit abhängig vom aktuellen Zustand sind und in einen anderen Zustand führen. Eine Transition besteht somit aus verschiedenen Teilen:

- Ausgangszustand - Zustand, indem sich das Objekt momentan befindet
- Ereignis - das Auftreten dieses Ereignisses berechtigt die Transition zur Ausführung
- Bedingung - eine logischer Ausdruck, der beim Auftreten des Ereignisses überprüft wird; mit *wahr* bewertet, berechtigt es die Transition zur Ausführung, sonst nicht
- Aktion - eine ausführbare Aktivität, die direkt auf das Objekt des Zustandsdiagrammes einwirkt sowie indirekt auf die Objekte, auf die das aktuelle Objekt zugreifen kann.
- Zielzustand - Zustand, der aktiviert wird, nachdem die Transition erfolgreich ausgeführt wurde

Zustandsdiagramme gehören zudem zu der Klasse der reduzierbaren Graphen [8], d.h. durch das Hinzufügen von Oberzuständen besteht die Möglichkeit ein Zustandsdiagramm innerhalb eines anderen Zustandsdiagrammes zu definieren.

4.2 Zustandsgesteuerte Adaptionen

Wie bereits dargelegt, ist es das Ziel, mittels Zustandsdiagrammen den Variantenraum der möglichen Adaptionen derart einzuschränken, dass der Direktor für sicherheitskritische Workflows genutzt werden kann. Sie reagieren auf Ereignisse unter Berücksichtigung des aktuellen Adaptionszustandes und weiteren Bedingungen und führen erst dann die jeweiligen Aktionen aus. Dabei ist die korrekte Ausführungsreihenfolge von Bedeutung.

Der Entwickler beschreibt während der Modellierung eines Workflows unter Zuhilfenahme eines Zustandsdiagrammes den Variantenraum, der abhängig von der jeweiligen Situation des Workflows, mögliche Adaptionen definiert. Ausgehende Transitionen stellen Anpassungen dar, bei denen der Nutzer Ereignisse, Bedingungen und die notwendigen Adaptionen festlegt. Die Situation besteht dadurch weiterhin aus den Variablen und Ausnahmen sowie dem aktuellen Adaptionszustand der Instanz. Der Adaptionszustand ist hierbei der Zustand, in dem sich die aktive Zustandsmaschine befindet. Durch ihn lässt sich beschreiben und gleichzeitig erkennen, welche Anpassungen bereits durchgeführt wurden bzw. welche in Zukunft ausgeführt werden können. Das hat den Vorteil, dass der Nutzer an jeder Stelle des Workflows weiß, was als nächstes passiert. Somit kann das Fehlverhalten bezüglich des Flashovers, einen falschen Workflow zu starten, nicht eintreten, da für *dieses* Ereignis keine Transition existiert, die als Aktion den Workflow Menschenrettung vorsieht.

Regelbasierte Lösungen, wie AgentWork [111] (oder XChange [32]), könnten hier ebenfalls zum Einsatz kommen. Sie besitzen allerdings einige Nachteile. Endliche Zustandsautomaten, wie es Zustandsdiagramme sind, sind im Gegensatz zu ECA-Regeln entscheidbar. Zudem lassen temporallogische Anfragen [55, 23] an die Adaptionen stellen. Dies zeigt beispielhaft das Projekt *Labelled Transition System Analyser* (LTSA) [175]. ECA-Regeln besitzen zudem mit ihrer eigenen Grammatik und Syntax eine hohe Einstiegshürde für den Anwender. Eine domänenspezifische Sprache oder eine visuelle Darstellung könnten hier helfen. Des Weiteren verliert der Nutzer bei vielen Regeln schnell den Überblick, denn es ist nicht einfach zu überschauen, welche ECA-Regel in welcher Situation aktiv sein wird. Ein weiterer Nachteil ist, dass sie nicht von sich heraus zustandsbasiert sind, wie es bei den Direktoren der reflexiven Workflows der Fall ist. Diese beziehen für die Berechnung einer Adaption immer die aktuelle Adaption mit ein. ECA-Regeln reagieren nur auf ein bestimmtes Ereignis und nutzen nicht explizit den aktuellen Adaptionszustand aus. Dadurch ist es nicht möglich, abhängige Adaptionen zu erstellen.

Deshalb soll in den folgende Abschnitten beschreiben werden, wie ein Zusammenspiel zwischen Workflow und Zustandsdiagramm funktionieren kann, ohne die Nachteile von AgentWork zu bsitzen.

Zusammenspiel von Workflows und Zustandsdiagrammen Übertragen auf Rekonfiguration von Workflows beschreibt ein Adaptions-Zustandsdiagramm demnach alle zu erwarteten Ereignisse und die daraus folgenden Adaptionen, auf die eine Instanz zur Laufzeit stoßen kann.

Ein Adaptions-Zustandsdiagramm besitzt einen Startzustand, der den Zustand des Workflows beschreibt, indem noch keine Adaption durchgeführt wurde. Des Weiteren existiert mindestens ein Endzustand (oder mehrere), nach dessen Erreichen keine weiteren Adaptionen möglich sind. Dadurch ist es möglich, ab einem bestimmten Punkt weitere Adaptionen zu unterdrücken. Erreicht die Workflowinstanz ihr (geplantes) Ende, bevor die Adaptions-Zustandsmaschine ein korrektes Ende erreicht, wird sie ebenfalls beendet. Die Adaptions-Zustandsmaschine muss nicht zwingend beendet werden, da sie zur Unterstützung der eigentlichen Workflowausführung dient. Eine unvollständige Ausführung bedeutet in diesem Kontext, dass der Workflow ohne größere Vorkommnisse ausgeführt wurde.

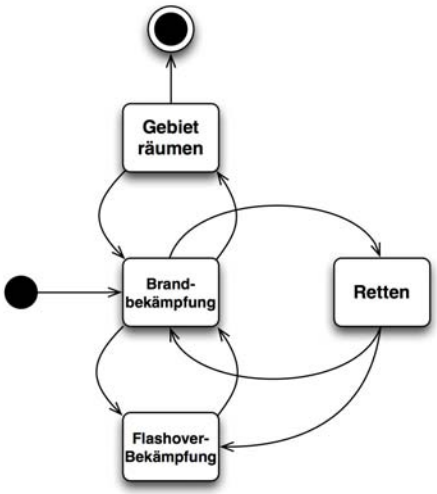


Abbildung 4.2: Zustandsdiagramm für die Brandbekämpfung

Der Zustand einer Adaptions-Zustandsmaschine stellt die gegenwärtige Adaptionssituation der Workflowinstanz dar. Die Adaptionssituation bezeichnet die zu einem bestimmten Zeitpunkt bereits absolvierten und noch möglichen Adaptionen einer Workflowinstanz. Die noch möglichen Adaptionen ergeben sich aus den ausgehenden Transitionen des Zustandes. Treten Ereignisse auf, die keiner Transition zugeordnet sind, werden diese ignoriert, was auch der ursprünglichen Semantik von Zustandsdiagrammen entspricht. In UML ist es bei Zustandsdiagrammen möglich, für jeden Zustand Eingangs- und Ausgangsaktionen zu definieren, die unabhängig von Event und Bedingungen ausgeführt werden. Diese werden nicht in die Adaptions-Zustandsdiagramme übernommen, da dieses Verhalten ohne Ereignisbezug nicht gewünscht ist. Aktionen sind hier gleichzeitig Adaptionen, so dass es entscheidend ist, ob eine Aktion passgenau angewendet wird.

Ebenso wie Transaktionen und Zustände müssen die Ereignisse und Bedingungen auf die Rekonfiguration von Workflows übertragen werden. Für die Ereignisse stehen dabei alle instanzrelevanten Ereignisse zur Verfügung, wie z. B. Änderungen von Variablen, Fortschritte im Prozessablauf (z. B. Beendigung einer Aktivität), usw. Die Bedingungen, die die Schaltbedingung der Adaptions-Zustandsmaschine darstellen und vor allem bei mehreren Transitionen mit gleichem Ereignis notwendig sind, beziehen sich auf die Instanzvariablen. Der Workflowdesigner kann somit festlegen, welche passende Adaption ausgeführt werden soll.

Abbildung 4.2 zeigt das Zustandsdiagramm, welches den kompletten Variantenraum für mögliche Adaptionen innerhalb der jeweiligen Zustände darstellt. Zu Beginn des Einsatzes befindet sich das System im Zustand *Brandbekämpfung*, welches dem Standardzustand eines Brandeinsatzes entspricht, solang das Feuer nicht gelöscht und kein besonderes Vorkommnis geschehen ist. Drei Ereignisse lösen an dieser Stelle einen Zustandsübergang aus. Wird das Feuer gelöscht, erfolgt ein Übergang zu *Gebiet räumen*, durch das Auftreten eines Flashovers ein Übergang in den gleichnamigen Zustand sowie beim Auffinden von Verletzten in den Zustand *Retten*. Bei näherer Betrachtung fällt auf, dass ein Zustandsübergang von *Retten* zu *Flashover bekämpfen* existiert, jedoch nicht umgekehrt. Diese Modellierungsweise soll die Anforderung beschreiben, dass während eines Flashovers, aufgrund eines zu großen Risikos, keine Menschenrettung erfolgen darf. Sämtliche Kapazitäten der Feuerwehr müssen sich zu diesem Zeitpunkt auf

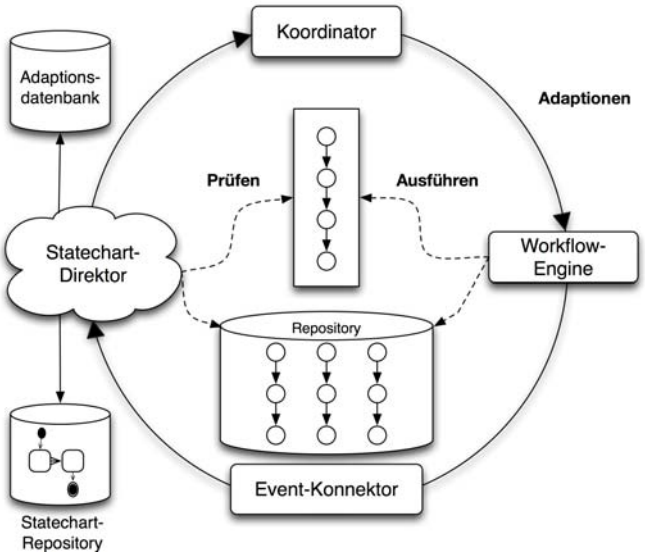


Abbildung 4.3: Konzept des Statechart-Direktors

die Bekämpfung dessen konzentrieren. Sobald der Flashover unter Kontrolle erscheint, wird ein neues Ereignis ausgelöst und das System wechselt wieder in den Ursprungszustand *Fire Combat*, von dem aus nun eine Menschenrettung möglich ist.

4.3 Statechart-Direktor

Die Adaptions-Zustandsdiagramme benötigen für eine korrekte Umsetzung drei zu beschreibende Faktoren: (1) die Ereignisse von der Workflow-Engine, (2) die Bedingungen und (3) die Aktionen, d. h. Adaptionen, die ausgeführt werden sollen.

Diese Anforderungen bedingen Konkretisierungen an der Referenzarchitektur. Primär ist ein Repository zur Speicherung der Adaptions-Statecharts notwendig. Hierfür wurde das *Statechart-Repository* eingeführt, auf das der Direktor Zugriff besitzt. Zusätzlich dazu ist der Zugriff des Direktors auf die Ereignisse der Workflow-Engine notwendig. Dazu ist eine Abbildung von WfMS-eigenen *Ereignistypen* auf die des Direktors notwendig. Hierfür kommt ein *Event-Konnektor*, der für die korrekte Verwaltung der Ereignisse (Events) zuständig ist, zum Einsatz. Darüber hinaus wird zur Verwaltung der Aktionen eine *Adaptionsdatenbank* eingeführt. Abbildung 4.3 zeigt einen Überblick zum Zusammenspiel der Komponenten.

Nach der Initialisierung der Instanz durch das WfMS startet der Direktor die Adaptionszustandsmaschine. Falls Änderungen der Variablen vorliegen, erzeugt die Engine ein neues Ereignis und übergibt es dem *Event-Konnektor* hinzu. Vor Ausführung der darauffolgenden Aktion eines Tokens überprüft der Direktor den Event-Konnektor auf neue Ereignisse, die mit abgehenden Transitionen des aktuellen Zustandes übereinstimmen. Bei einer negativen Auswertung erfolgt eine Fortsetzung der Instanz ohne Adaption. Wird eine Transition mit dem entsprechenden Ereignis gefunden, wertet der Direktor die Bedingung aus, extrahiert bei positiver Übereinstimmung die Adaptionsvorschrift und gibt diese an den Koordinator weiter. Daraufhin entfernt der Direktor die Ereignisse aus

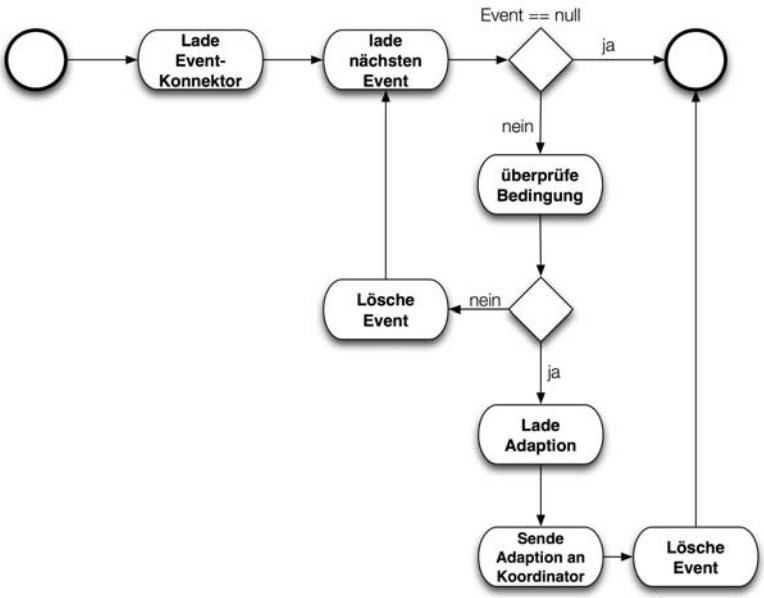


Abbildung 4.4: Ablauf der Adaptionsentscheidung

dem Event-Konnektor und löst einen Zustandsübergang aus, der den Zustandsautomaten in den nächsten Zustand führt. In diesem angekommen, wartet er auf die Ausführung eines Knotens und der Prozess beginnt von neuem. Der Direktor bleibt solang aktiv, bis entweder die Instanz oder der Zustandsautomat beendet wird. Abbildung 4.4 zeigt den Ablauf zusätzlich in Form eines Aktivitätsdiagrammes.

4.3.1 Integration in die Referenzarchitektur

Damit dies möglich ist, sind für die Realisierung eines *Statechart-Direktors*, einige Erweiterungen an Zustandsdiagrammen und am Direktor-Ansatz aus Kapitel 3 vorzunehmen.

Der Statechart-Direktor benötigt Zugang zu allen Variablen, Ausnahmen und Laufzeitdaten und allen Ereignissen des WfMS. Während der Ausführung einer Instanz kann eine Vielzahl von unterschiedlichen Ereignissen auftreten. In [176] wird eine Ontologie beschrieben, die die möglichen Ereignisse, wie in Abbildung 4.5, katalogisiert. Dabei unterscheiden sich die Ereignistypen (auch Event Types genannt) zwischen den Workflow-Engines, da sich einige Ereignisse auf Sprachspezifika beziehen[1]. Es ist demnach notwendig, eine abstrakte Abbildungsschicht zwischen Direktor und WfMS einzuführen, ähnlich wie es beim Koordinator vorgenommen wurde.

4.3.2 Event-Konnektor

Adaptionen können nur nach der Ausführung einer Aktivität vorgenommen werden. Das Abbrechen der aktuellen Aktivitäten ist nicht vorgesehen. Während der Ausführung einer Aktivität treten allerdings Ereignisse auf, die zu Adaptionen führen können. Wenn

[1] Beispielsweise unterstützt eine BPEL 1.0 Engine keinen *for-each*-Event.

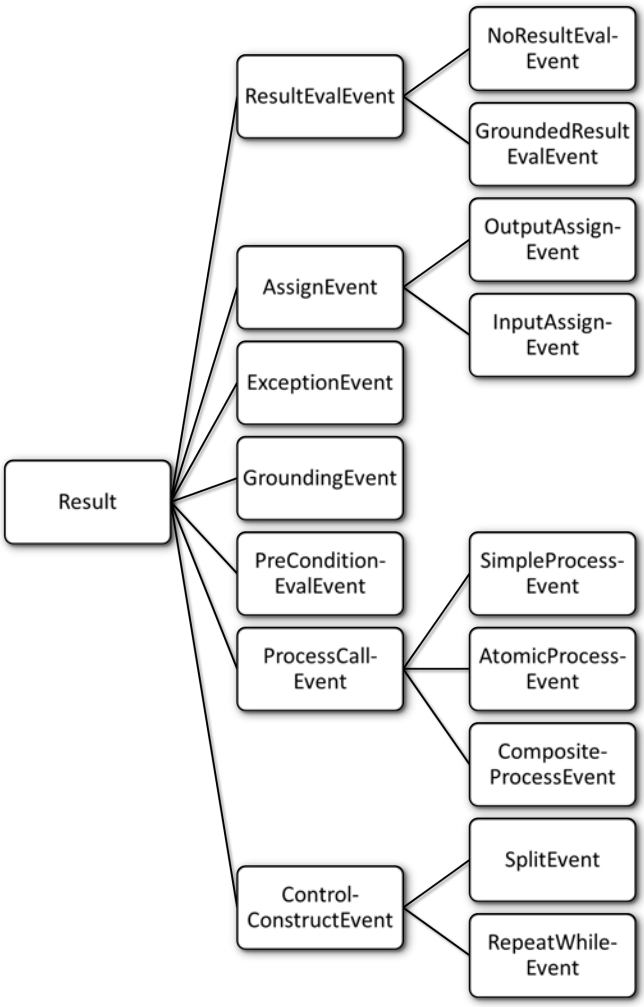


Abbildung 4.5: Event-Klassifikation (nach [176])

nach der Beendigung einer Aktivität die Überprüfung für eine mögliche Adaption durchgeführt wird, sind diese Ereignisse wichtig und dürfen somit nicht verloren gehen. Deshalb wird für jeden aktiven Token einer Instanz ein *Event-Konnektor* geführt, der alle Ereignisse chronologisch seit der letzten Adaptionsüberprüfung puffert. Nach jeder Prüfung wird die Liste für den jeweiligen Token geleert. Dieser Event-Konnektor ist eine eigenständige Komponente, so dass keine Änderungen an der Workflow-Engine vorgenommen werden müssen. Der Vorteil liegt darin, dass das WfMS unbeeinflusst von der Implementierung der Zustandssteuerung bleibt.

Eine Alternative stellt ein globaler Event-Konnektor dar. Dieser beinhaltet sämtliche Ereignisse, die durch Ausführung von Workflows im System entstanden. Damit besteht die Möglichkeit, auch auf Ereignisse zu reagieren, die in verteilten Workflows auftreten. Dadurch wird die die Funktionalität der Zustandskomponente nicht auf eine lokale Maschine begrenzt.

Die in [176] und in Abbildung 4.5 aufgezeigten Eventtypen lassen sich auf zwei Grundtypen verallgemeinern: Daten- und Ablaufereignisse. Erstere treten auf, sobald eine Variable während der Ausführung manipuliert wird. Zur Bestimmung einer Veränderung existieren zwei Methoden. Zum einen kann die Workflow-Engine nach jedem ausgeführten

Schritt überprüfen, ob Änderungen einzelner Variablen auftraten, so dass bei positiver Auswertung ein neues Datenereignis in den Event-Konnektor eingefügt wird. Zum anderen ist es möglich, dass die Variablen selbst das Ereignis auslösen, wenn beispielsweise in den set-Methoden der Variable dieses vorgesehen ist. Die zweite Art von Ereignissen tritt auf, sobald bestimmte Konstrukte im Workflow gestartet oder beendet wurden, wie z. B. das Split-Event aus Abbildung 4.5. An dieser Stelle fügt die Workflow-Engine dem Event-Konnektor ein Ablaufereignis hinzu.

Beide Ereignistypen stellen *atomare* Ereignisse dar, wie sie in [33] beschrieben sind. Ein atomares Ereignis tritt dabei zu einem bestimmten Zeitpunkt, was an dieser Stelle dem Hinzufügen des Ereignisses in den Event-Konnektor entspricht. Bei der Beschreibung der ECA-Regeln muss allerdings nicht immer nur ein Event angegeben werden. Es ist auch eine Kombination von Daten- und Ablaufereignissen möglich. Diese Verknüpfung entspricht dabei einer Transaktion, die statt nur einem, mehrere Ereignisse enthält. In der Literatur wird diese Form von Ereignissen als *Verbundereignis* oder auch *Composite Event* bezeichnet [33]. Ein Verbundereignis wird nur ausgelöst, sobald die erforderliche Kombination der enthaltenen atomaren Ereignisse eintritt. Die Ereignisse können durch logische Ausdrücke (and, or, not) miteinander verbunden werden. Bei der Modellierung des Adaptionszustandsdiagrammes besteht zum einen die Möglichkeit, dass der Entwickler keine zeitliche Befristung festlegt. Das bedeutet, dass zwischen den notwendigerweise auftretenden Ereignissen auch ein längerer Zeitraum liegen kann[2]. Zum anderen kann er dem Verbundereignis eine Aktivierungsdauer hinzufügen, welche nur Ereignisse in einem bestimmten Zeitrahmen betrachtet.

4.3.3 Adaptionsdatenbank

Die Verarbeitung der Ereignisse und Bedingungen können durch den Konnektor unabhängig von der Workflow-Engine verarbeitet werden. Zu einer vollständigen Unterstützung von Adaptions-Zustandsmaschinen fehlen die Aktionen, die durch Adaptionen repräsentiert werden. Dazu wird eine Adaptionsdatenbank eingeführt. Für eine Adaption müssen folgenden angegeben werde:

- Webeart
- Einzuwebender Workflow
- Selektives Einweben
  - Eintrittspunkt in den einzuwebenden Workflow
  - Austrittspunkt aus dem einzuwebenden Workflow
- Rücksprungpunkt

Die Aufgabe des Designers besteht darin, die Adaptionen, die bei Aktivierung des Zustandsüberganges durchgeführt werden sollen, korrekt zu definieren. Aufgrund der festen Definition der durchzuführenden Adaption entsteht die *eingeschränkte Adaptionsfreiheit*. Der Nutzer kann durch das Anlegen verschiedenster Transitionen eine Vielzahl von Adaptionen definieren, wobei das System immer gleich verhalten wird.

[2] Nicht vergleichbar mit der *VALID-TIME* bei AgentWork.

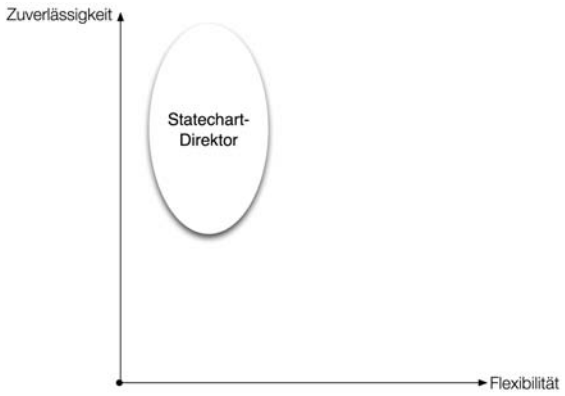


Abbildung 4.6: Vergleich von Zuverlässigkeit bei einem Statechart-Direktor

4.4 Zusammenfassung

In diesem Kapitel wurde eine auf sicherheitskritische Systeme spezialisierte Form zur Steuerung der Adaption durch Verweben vorgestellt. Sie verwendet eine *genaue* und zudem *statische* Wissensbasis, die sich für sicherheitskritische Workflows eignet, denn die Bedingungen für die Adaptionen sind vorgegeben und werden sich nicht ungewollt ändern.

Bisherige Ansätze zur Steuerung setzen vor allem auf Event-Condition-Action Regeln (ECA), wie zum Beispiel AgentWork [111]. Diese Regeln beschreiben, unter welchen Bedingungen eine Adaption durchgeführt werden soll. Dazu werden zwei Indikatoren genutzt: (1) wird immer auf ein Ereignis (Event) reagiert, (2) nur unter bestimmten Bedingungen (Condition). Dadurch lässt sich beschreiben, wann eine Adaption durchzuführen ist. Dieser Ansatz besitzt den Nachteil, dass die Falldatenbank sehr schnell unübersichtlich wird und grundsätzlich ein Adaptionszustand fehlt. Dieser Adaptionszustand ist wichtig, denn durch ihn kann festgestellt werden, welche Adaptionen bisher durchgeführt wurden. In Abhängigkeit dieser Informationen kann über neue Adaptionen entschieden werden.

Der Statechart-Direktor setzt deshalb auf unveränderliche Adaptionsentscheidungen, die durch ein Statechart beschrieben werden. In diesem wird festgelegt, unter welchen Bedingungen eine Adaption erfolgen soll. Jede Adaption führt in einen neuen Zustand, so dass Adaptionen andere bedingen können. Diese Zustandsorientierung ermöglicht zudem einen einfachen Überblick, in welchem Adaptionszustand sich eine Instanz befindet. Für die Umsetzung der wurde die Referenzarchitektur erweitertet. So besitzt der Direktor Zugriff auf ein Statechart-Repository, in dem die Adaptions-Zustandsdiagramme für jeden Workflow gespeichert werden. Für die Verwaltung und Verarbeitung der Ereignisse wurde der Event-Konnektor eingeführt. Die Verwaltung der Adaptionen wird von einer Adaptionsdatenbank übernommen.

Der hier vorgestellte Direktor stellt allerdings nur eine mögliche Umsetzungsmöglichkeit dar, die zudem nur eingeschränkt reflexiv ist. So kann der Direktor die Rekonfigurationen an den Workflowinstanzen durchführen, er selbst ist jedoch aufgrund der statischen Wissensbasis nicht reflexiv. Er kann aufgrund der statischen Regeln zudem nicht auf unbekannte Situationen reagieren. Wie in Abbildung 4.6 dargestellt, deckt der Statechart-Direktor damit nur einen kleinen Bereich des Spannungsfeldes zwischen Zuverlässigkeit

und Flexibilität ab. Es gilt demnach zu untersuchen, welche Techniken dazu genutzt werden können, um vollständig reflexive Direktoren umzusetzen, die auf unerwartete Bedingungen reagieren können. Die folgenden zwei Kapitel 5 und 6 werden sich intensiv mit der Ausgestaltung eines solchen Direktors befassen. Kapitel 5 stellt eine Umsetzung für einen domänenunabhängigen BDI-Direktor vor, während in Kapitel 6 spezialisierte und kooperative Direktoren im Vordergrund stehen.

Kapitel 5

Universelle autonome Rekonfiguration

Im vorangegangenen Kapitel wurde ein Direktor vorgestellt, der mithilfe der Adaption durch Verweben den Kontrollfluss eines Workflows vollständig ändern kann, seine Wissensbasis selbst nicht. Er ist demnach nicht vollständig reflexiv. Er basiert auf einer genauen Wissensbasis, umgesetzt mit Statecharts. An dieser Stelle soll nun eine Umsetzungsmöglichkeit vorgestellt werden, die zum einen vollständig reflexiv und *universell einsetzbar* ist, zum anderen auf einer ungenauen Wissensbasis beruht. Somit könnte er auf unerwartete Situationen reagieren.

Reflexion bedeutet, dass auf Erfahrungen zurückgegriffen werden kann. Ein Direktor muss deshalb die Fähigkeit zum Lernen besitzen. Dazu muss der Direktor über die Zeit Verbindungen zwischen Situationen und den angemessenen Adaptionen lernen. Dabei ist es wichtig, dass der Direktor die beste Adaption aus verschiedenen Möglichkeiten findet. Für die Erstellung dieses Wissens und die geeignete Auswahl der Adaptionen existieren verschiedene Ansätze.

Universell einsetzbar bedeutet zum einen, dass der Direktor mit verschiedenen WfMS sowie den Workflow-Metamodellen umgehen kann und die Ausdrucksmächtigkeit des Workflow-Metamodells nicht eingeschränkt, wie z. B. durch das Verbot verschiedener Aktivitätstypen. Zum anderen bedeutet es, dass ein Direktor nicht an eine spezielle Domäne oder eine bestimmte Menge von Workflow-Metamodellen angepasst werden soll, d. h. er muss *domänenunabhängig* sein. Diese Unabhängigkeit ist unerlässlich für die Wiederverwendung des Direktors in verschiedenen WfMS. Ein solcher Direktor muss mit der daraus resultierenden Ungewissheit umgehen können. Wird auf domänenspezifische Anpassung verzichtet, bedeutet dies, dass der Direktor keinerlei *semantische Informationen* über seine Umgebung (Workflows) besitzt. Damit ist jede Aktivität nur durch ihre System-ID identifizierbar, ihre spezifische Aufgabe (beispielsweise ein Webservice Aufruf) bleibt verborgen. Es gibt auch keinerlei Informationen über den Zweck oder die Bedeutung der Workflowmodelle, sondern der Direktor kann nur über die syntaktische Struktur, d. h. Variablendefinitionen, Ablaufreihenfolge, etc. entscheiden.

Für diese Voraussetzungen gilt es sowohl eine Wissensrepräsentation, als auch eine Adaptionssynthese zu finden. Planer, wie sie bei Ad-hoc-Workflows Einsatz finden, können in diesem Szenario nicht eingesetzt werden. Sie setzen ein vollständiges Wissen über die Umgebung voraus. Unvorhersehbare Änderungen können deshalb nur unter abgegrenzten Bedingungen behandelt werden. Eine mögliche Lösung stellen Expertensysteme dar. Sie basieren auf einer erweiterbaren Wissensbasis, in der zumeist für eine bestimmte Problemstellung Lösungen angeboten werden. Neue Situationen und Lösungen können fortwährend hinzugefügt werden. Zudem können Lösungen für verschiedene Situationen verwendet werden. Im Gegensatz zum Statechart-Direktor, bei dem zu jeder Situation die Aktion (Lösung) vorgegeben war, ist die Wissensbasis *ungenau*.

Eine Lösung auf Basis einer erweiterbaren, flexiblen und ungenauen Wissensbasis ist für die Umsetzung eines vollständig reflexiven Direktors deshalb sinnvoll. In den folgenden Abschnitten werden deshalb zuerst Techniken untersucht, wie eine solche Wissensbasis gestaltet sein muss, um im Direktor Verwendung zu finden.

5.1 Grundlegende Techniken zum Lernen und Entscheidungsfindung

Künstliche Intelligenz ist seit vielen Jahren ein Forschungsschwerpunkt, bei dem die an den Direktor gestellten Anforderungen eine große Rolle spielen:

- Wissensrepräsentation
- Lernen
- Lösungsfindung in unvollständig beschriebenen Umgebungen

Dazu wurde eine Vielzahl von Ansätzen für die verschiedensten Anwendungsfälle entwickelt. Aufgrund dieser Vielfalt und Komplexität kann in dieser Arbeit aber nicht die gesamte Bandbreite der Forschung (siehe [145]) dargelegt werden. Stattdessen sollen drei Ansätze ausführlicher betrachtetet werden, die für den Direktor in Frage kommen können. Wenn von künstlicher Intelligenz gesprochen wird, ist häufig auch von Agenten die Rede. Ein Agent ist ein abstraktes Subjekt. Es kann ein Roboter, ein Mensch oder eine Softwarekomponente sein. [95] definiert einen Agenten dabei wie folgt:

> „. . . an agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives. . . “

Ein Beispiel für Agenten-Systeme ist ein Expertensystem, d. h. ein Assistent, der bei Entscheidungen mithilfe von Expertenwissen berät. Agenten besitzen zumeist eine geschichtete Architektur. Jeder Agent stellt dabei eine Ansammlung von ausführbaren Aufgaben dar. Jede Aufgabe beschreibt ausführbares Verhalten. Dabei wird ein kontinuierlicher Strom von Eingaben auf Ausgaben abgebildet. Abhängig von der aktuellen Situation wird eine Aktion automatisch (autonom) gewählt. Die klassische Architektur sieht ein Drei-Schichtenmodell vor, wie es Abbildung 5.1 darstellt ist.

Zu den zentralen Konzepten von Agenten zählen Autonomie, Flexibilität und Situationsbewußtsein. Zur Verarbeitung der aktuellen Situation benötigt er Zugriff auf verschiedene Sensoren, die ihm Kontextinformationen übermitteln. Für die Flexibilität werden sowohl Reaktivität als auch Proaktivität vorausgesetzt. In [145] werden verschiedene Arten von Agenten vorgestellt:

- *Einfache Reflex-Agenten* agieren immer nach festgelegten Regeln, vergleichbar dem regelbasierten Schließen.
- *Umwelt-Agenten* verfügen über ein internes Modell ihrer Umwelt. Darin wird beschrieben, wie sich diese Umwelt entwickelt und welche Aktionen Auswirkungen auf diese besitzen.
- *Zielorientierte Agenten* versuchen ein Ziel zu erreichen. Dazu wird zumeist ein Plan von Aktionen berechnet, der den Agenten dem Ziel näher bringt.

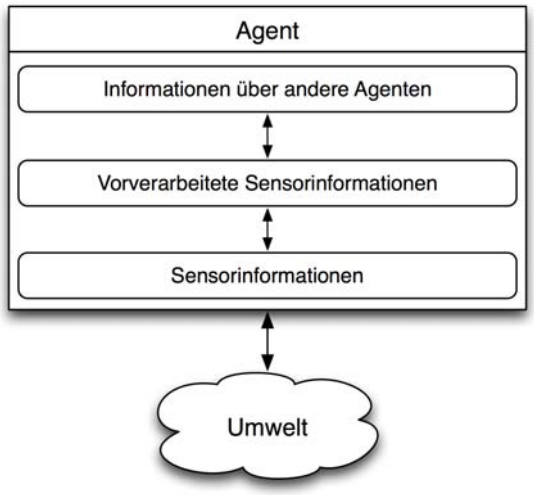


Abbildung 5.1: Geschichteter Aufbau von Agenten

- *Nutzenorientierte Agenten* versuchen aus eine Menge gleichzeitig aktiver Ziele, die *Nutzenfunktion* zu maximieren.

Im Folgenden stehen die zielorientierten Agenten im Mittelpunkt, denn diese sind im Verhalten einem Direktor ähnlich. Zielorientierte Agenten besitzen immer ein *Ziel*. Existiert dies nicht, kann nicht entschieden werden, was als nächstes zu tun ist. Wie auch der Direktor, benötigt ein Agent Wissen über den Situationsraum und die möglichen Aktivitäten. Ein Agent kann neues Wissen auch durch Probieren sammeln. Versuch und Irrtum kann unter Umständen nach einer gewissen Zeit zum Ziel führen. Der Vorteil bei diesem Vorgehen ist, dass der Agent ohne Vorwissen Wissen über seine Umgebung und Aktionsmöglichkeiten sammeln kann. Der Vorgang des Informationssammelns wird als *Lernen* bezeichnet. Das Lernen kann dabei zu unterschiedlichen Zeiten durchgeführt werden. Es wird zwischen *On-line* und *Off-line* Lernen unterschieden [9]. On-line Lernen bezeichnet das inkrementelle Berechnen neuer Hypothesen, sobald ein neues Beispiel verfügbar ist. Werden diese Hypothesen zu festgelegten Zeitpunkten generiert, wird von Off-line Lernen gesprochen. Die Menge an Wissen wird bei der Verfolgung von Zielen als *Problemlösungsraum* beschrieben. Es gibt zum einen den diskreten, beobachtbaren, deterministischen Raum, der Laborbedingungen entspricht und zum anderen den kontinuierlichen, indeterministischen Raum. Dieser repräsentiert reale Bedingungen.

Das Prinzip der *Entscheidungsfindung* basiert auf dem *Schließen* über dem verfügbaren Wissen, um daraus Neues abzuleiten. Eine grundlegende Eigenschaft, die sich die Forschung dabei zu Nutze macht, ist die vorhandene Korrektheit eines logischen Schlusses, wenn die zugrunde liegenden Informationen korrekt sind. Für die Schlussfolgerung gibt es eine Vielzahl an Möglichkeiten, wie die Aussagenlogik oder die Prädikatenlogik erster Stufe. Das Hintergrundwissen für das Schließen wird in der zentralen Komponente eines Agenten gespeichert, in einer Wissensbasis (engl. knowledge base). Die Wissensbasis eines Agenten besteht aus (Daten-) Sätzen, in einer definierten Sprache. Dafür kann je nach Konzept eine Sprache basierend auf Aussagenlogik, Prädikatenlogik oder einem anderen abstrakten Konzept beschrieben sein. Eine einfache Repräsentation der Wissensbasis ist eine Liste von Sätzen, ähnlich einer Prolog-Faktenbasis.

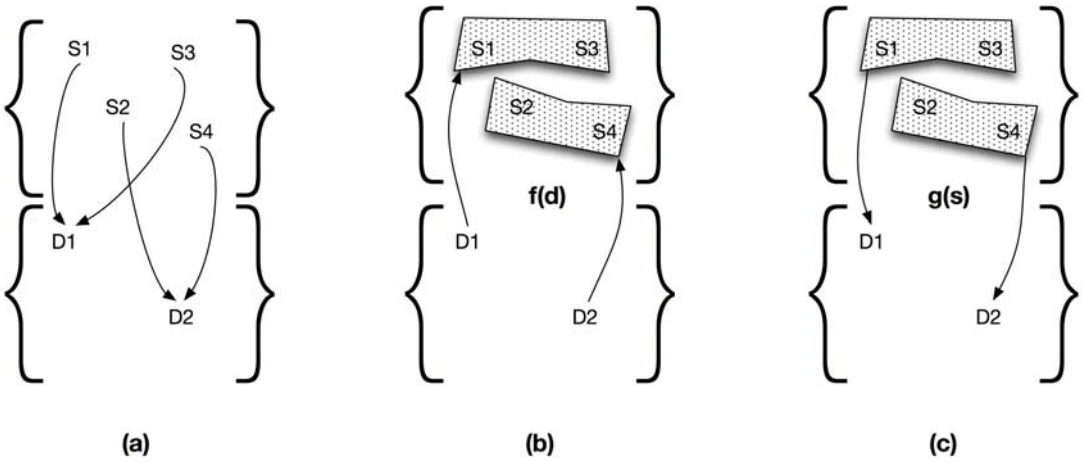


Abbildung 5.2: Abbildung einer Menge von Situation S_i auf Lösungsvorschläge D_i.

Durch das selbstständige Aufbauen einer Wissensbasis wird dem Agenten ermöglicht, autonom zu agieren. Einfache und allgemeingültige Schließ-Verfahren sind das Regelbasierte und das Fall-basierte Schließen [145]. Daneben existieren datengesteuertes, zielorientiertes, räumliches, psychologisches und unsicheres Schließen.

Im Bereich der Workflows bestehen für den Direktor jedoch einige Einschränkungen. Zum einen kann nicht davon ausgegangen werden, dass alle notwendigen Informationen über die Workflow-Engine und den jeweiligen Workflow zur Verfügung stehen. Für verschiedene Ansätze des Schließens, wie z. B. das Planen ist es zwingend erforderlich, dass alle Informationen über Aktionen im Workflow formalisiert vorliegen. Daraus ergibt sich, dass auch formal bekannt sein muss, welche Operation ein Webservice auf bestimmten Daten vornimmt und nicht nur, in welcher Form die Input- und Output-Daten vorliegen müssen. Dies ist in der Domäne der Workflows nur schwer umzusetzen, denn dieses Wissen wurde bisher nicht formal beschrieben. Da der Direktor und Koordinator zudem ohne große Änderungen am WfMS umzusetzen sind, eignen sich diese Arten des Schließens für den Direktor nicht. Verfahren zum Schließen auf zugänglichen Daten oder unvollständigen Daten sind demnach von Vorteil. Dazu zählt das regelbasierte Schließen, das auf Entscheidungsbäumen mit abstrahierten Regeln beruhen kann und das fallbasierte Schließen, das auf (unvollständigen) Situationen aufbaut. Diese werden im Folgenden näher vorgestellt.

5.1.1 Regelbasiertes Schließen

Das zu lösende Grundproblem bei derartigen Systemen ist, wie von der aktuellen Situation auf eine Adaption geschlossen werden kann. Eine einfache Lösung für dieses Problem ist eine direkte 1-zu-1 Abbildung von Situation zur Lösung, wie es in Abbildung 5.2(a) dargestellt ist. In einem System, wie hier bei WfMS, ist die Situationsmenge zu groß, so dass eine 1-zu-1 Abbildung unmöglich ist. Eine weit bessere Lösung ist die Aufteilung des Situationsraumes in Cluster. Wie in 5.2(b) dargestellt ist. Zur Einteilung, welche Situation in welchen Cluster gehört, können verschiedene Berechnungsformeln verwendet werden (im Beispiel $f(d)$). Anschließend wird jeder Lösung aus d eine Situationscluster zugewiesen. Diese Lösung ist vor allem dann einzusetzen, wenn die Anzahl der Lösungen

gering ist, da dadurch wenige Abbildungen anzugeben sind. Ist dies nicht der Fall, sollte die umgekehrte Abbildung eingesetzt werden. Diese beschreibt, welche Cluster die gleichen Lösungen erfordern, wie in 5.2(c) dargestellt. Ist im Situationsraum die Clustermenge zu hoch, besteht auch hier das gleiche Problem, dass zu viele Zuordnungen notwendig sind.

Die Lösung für dieses Problem besteht darin, die Komplexität zu verringern, um die richtige Adaption für eine bestimmte Situation in der großen Menge von Clustern und Lösungsmöglichkeiten zu finden. Dazu werden Regeln eingesetzt. Eine *Regel* ist eine Abbildung von einer bestimmten Lösung zu einer Menge von Situationen, für die die Entscheidung eine geeignete Lösungsmöglichkeit darstellt. Dabei muss die Menge der Situationen nicht vollständig beschrieben sein, d. h. eine Regel beschreibt nicht alle Situationen, für die diese Entscheidung geeignet ist. Eine Regel ist deshalb keine vollständige Äquivalenzrelation, sondern beschreibt indirekt, dass wenn eine bestimmte Situation in dieser Menge existiert, dann diese Lösung die beste ist. Die Situationsmenge wird dabei durch eine Bedingung beschrieben. Diese wird für jede Situation berechnet. Diese Regeln werden in einer *Regelbasis* (engl. rule-based) gespeichert. Der Prozess der Integration neuer Regeln in die Regelbasis wird als Lernen bezeichnet. Verschiedene Implementierungen für Regelbasen variieren in der Art, wie die Bedingungen geordnet werden und wie diese Ordnung erlernt werden kann [145]. Eine der bekanntesten ist der *Entscheidungsbaum*.

Damit eine Situation in einen Entscheidungsbaum integriert werden kann, muss die Situationsbeschreibung in ihre Elemente/Attribute zerlegt werden. Ein Attribut besitzt, wie eine Workflowvariable, einen bestimmten Typ, der einen bestimmten Wertebereich beschreibt. Der Situationsraum ist das Kreuzprodukt der Wertemengen der Attribute a_i, das eine Situation vollständig beschreibt. Im Falle von Workflows wären dies die gesamten Mengen von Variablen einer Workflowinstanz. Die Regelbedingungen sind Boolsche-Funktionen über die Werte der Attribute einer Situation X. Ein Entscheidungsbaum erfordert Bedingungen in der Form:

$$C^r(X) = \bigwedge a_i(X) \in c_i^r,$$

wobei $C^r(X)$ die Bedingung der Regel r darstellt. Dabei müssen alle Attributwerte $a_i(X)$, in der Situation X, in der Menge der zulässigen Attributwerte c_i^r für die Regel r liegen.

Jeder Knoten des Entscheidungsbaums ist mit einem der Attribute versehen. Die Übergänge zu den Kind-Knoten sind gekennzeichnet mit dem Wert des jeweiligen Attributes. Die Blattknoten sind die Vorschläge für die jeweilige Situation. Ein Pfad zu einem Blattknoten stellt eine Bedingung dar, die auch als aussagenlogische Formel ausdrückbar ist. Der Pfad und der dazugehörige Blattknoten beschreiben ein Lösungstupel. Die Reihenfolge der Attribute ist hierbei vernachlässigbar. Ein Beispiel ist in Abbildung 5.3 dargestellt.

Im Falle von Workflows sind die Attribute im Regelbaum die Elemente der Situationsbeschreibung. Ein regelbasierter Direktor wird bei der Entscheidungsfindung die Hierarchiestufen des Baums von der Wurzel an ablaufen. Dabei folgt er den Kanten, die den Variablenbelegungen entsprechen. Ein ähnliches Vorgehen kann auch beim Einfügen einer neuen Regel in den Baum verwendet werden. Der Baum wird traversiert, bis zu dem Attribut, bei dem es für den aktuellen Variablenwert keinen Übergang zum nächsten

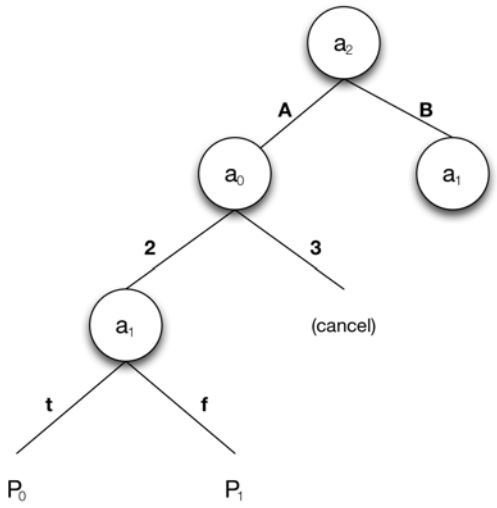


Abbildung 5.3: Entscheidungsbaum mit drei Attributen – a_0, a_1 und a_2

Attribut gibt. Die fehlende Kante wird eingefügt und mit einem Lösungsvorschlag verknüpft. Attribute, die während der Traversierung des Baumes nicht untersucht wurden, werden nicht in den Baum aufgenommen. Somit kann der Lösungsvorschlag auch in anderen Situationen erreicht werden, wenn zumindest alle Attribute passen, die auf dem Pfad liegen. Dieses Vorgehen vermindert die Komplexität bei großen Attributmengen. Auch vereint der Baum auf diese Weise verschiedene Bedingungen von unterschiedlichen Regeln in Teilpfaden. Ist der Baum ordnungsgemäß erstellt worden und haben alle Pfade, die gleiche oder zumindest eine ähnliche Länge, beträgt die durchschnittliche Länge des Pfads $log_2(n)$, wobei n die Anzahl der Knoten ist. Verglichen mit dem oben geschilderten Lösung von ungeordneten Regelabbildungen reduzieren Entscheidungsbäume, die Komplexität von n auf $log_2(n)$. Dies ermöglicht eine effiziente Lösungsfindung.

Der Vorteil eines, mittels regelbasiertem Schließen konzipierten, Direktors ist, dass Adaptionsentscheidungen immer auf einer nachvollziehbaren Grundlage getroffen werden. Der Direktor traversiert den Baum und kommt anhand der Übergänge zu einem Ergebnis. Dies entspricht annähernd einer verschachtelten *Switch*-Klausel[1], die auch kompiliert werden kann. Der Regelbaum wird jedoch bei vielen Bedingungen bzw. Regeln, wie beim Switch, schnell groß. Daraus ergibt sich, dass die Anzahl von Regeln den Situationsraum füllt, so dass es kaum möglich ist, von einer Situationsbeschreibung zu abstrahieren[2]. Nur noch das vollständige Auswerten einer Situation führt zur Lösung. Praktische Erfahrungen zeigen zudem, dass wenn zu viele Situationen erlernt werden, die Qualität der Lösungsvorschläge abnimmt [43]. Ein Entscheidungsbaum kann nur bis auf ein optimales Niveau angelernt werden. Um das Problem der Überanpassung zu vermeiden, dürfen die Trainingsdaten nur eine gewisse Anzahl an repräsentativen Proben beinhalten, d. h. Regeln, die typische und häufig vorkommende Fälle beschreiben. Diese Proben können manuell von einem Experten erstellt werden. Abhängig von der Komplexität der Domäne bedeutet dies eine große Anstrengung für den Experten, der darauf zu achten hat, wichtige Fälle nicht zu vergessen und trotzdem keine Überanpassung zu produzie-

[1] Beispiel für Java: http://java.sun.com/docs/books/jls/secondedition/html/statements.doc.html (zuletzt aufgerufen am 25.02.2011).

[2] Abstraktion in diesem Fall bedeutet, dass nicht alle Attribute für das Auffinden einer Lösung notwendig sind.

ren. Zu bedenken ist, dass der Wertebereich der Attribute zumeist unendlich ist. Bereits bei einer kleinen Menge von Variablen wird die Anzahl der Kanten im Baum zu groß. Die Übergänge sind mit *einzelnen* Werten für das Attribut belegt. Besitzt ein Attribut einen großen Wertebereich, muss dieser in einzelne Cluster zerlegt werden, d. h. die exakten Werte müssen durch Teilmengen repräsentiert werden. Diese Teilwertebereiche ermöglichen es, eine endlich (kleine) Menge an Kanten anzugeben.

Zusammenfassend kann gesagt werden, dass ein Direktor mit Entscheidungsbaum, die getroffenen Adaptionsentscheidungen gut darlegen kann, aber die Regelmenge ist bereits bei einfachen Szenarien schwer zu verwalten. Zudem hängt die Qualität der Aussagen stark von den vorgegebenen und gelernten Situationen ab.

5.1.2 Fallbasiertes Schließen

Die Einschränkungen und Probleme bei Entscheidungsbäumen beruhen zu aller erst auf der starren Integration der Situationsbeschreibung. Die Struktur des Baumes hat dabei einen starken Einfluss darauf, wie Situationen in sinnvolle Regeln umgesetzt werden. Deshalb gehen andere Ansätze den Weg, die Situationsbeschreibungen nicht als Regelabbildungen zu speichern, sondern Situation und Lösungsvorschlag als unabhängiges Paar anzusehen. Die Synthese von Regeln wird dadurch umgangen. Dieses Paar wird als *Case* (Fall) bezeichnet. Er vereint in sich die direkte Abbildung von *Situation* und *Lösung* und ist zentrales Konzept des *Case-based Reasoning* (CBR) [145, 103].

Die Case-Beschreibung der Situation ist eine Menge von Attribut-Wert-Paaren, ähnlich den Kanten im Regelbaum. Alle Attribute werden mit der gleichen Priorität behandelt. Es ist nicht notwendig, ein Hauptattribut zur Unterscheidung von Situationen zu wählen. Im Falle des Regelbaumes wurden nur positive Regeln in den Regelbaum aufgenommen, d. h. die Pfade hatten immer eine sinnvolle Adaption zur Folge. Regelbasiertes Schließen ermöglicht es nicht, negative Fälle aufzunehmen. Wurde bei einer bestimmten Situation eine Adaption vorgeschlagen, deren Auswirkung eine Verschlechterung der Ausführung des Workflows ist, kann diese Information nicht gespeichert werden. Sie kann demnach immer wieder auftreten. Anders beim Case-based Reasoning. Hier können schädliche Fälle in die Wissensbasis einbezogen werden, um damit eine erneute Fehlentscheidung zu verhindern. Zur Unterscheidung zwischen positiven und negativen Lösungen muss es ein Datum geben, dass die Lösung als solches charakterisiert. Deshalb beschreiben *Situation*, *Lösung* und *Ergebnis* gemeinsam einen *Case*. Damit stellt ein Fall die Erfahrung dar, die während einer Workflowausführung getätigt wurde. Das Ergebnis beschreibt die Auswertung der Adaption und stuft den Nutzen des Cases ein. Oft wird das Ergebnis bestimmt, nachdem der Workflow beendet wurde und einen Vergleich mit anderen ähnlichen Fällen durchgeführt wurde. Es gibt jedoch keinen Standardalgorithmus für die Ermittlung eines Ergebnisses. Wieder ist ein Sachverständiger bzw. Experte notwendig, der die Qualität des Falles bestimmt. Im Gegensatz zum Entscheidungsbaum lehnt der Experte einen Fall nicht ab, wenn er ein schlechtes Ergebnis erzielte. Er dient in der Falldatenbank als Abschreckung.

Ablauf des Schließens Wenn ein fallbasierter Direktor um einen Vorschlag gebeten wird, muss er eine Reihe von Operationen auf den Falldaten ausführen: *Retrieval*, *Reuse* und *Retaining* [1]. Zuerst lädt (retrieve) er alle Fälle, die für die Entscheidungsfindung

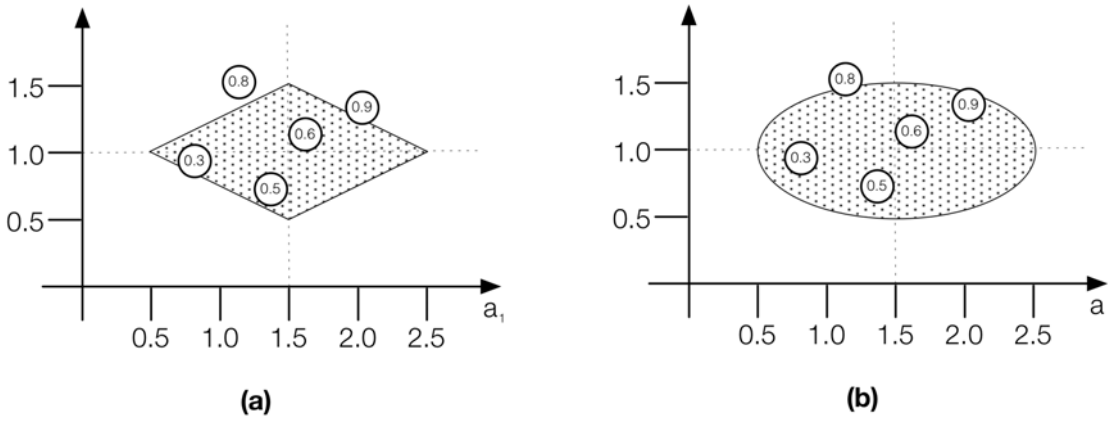


Abbildung 5.4: Zweidimensionaler Situationsraum mit zwei Attributen – a_1 und a_2.

notwendig sind. Dabei werden alle Fälle, deren Situationsbeschreibung *ähnlich* der aktuellen sind, aus der Falldatenbank geladen. Dabei ist die Lösung und das Ergebnis der Anfrage nicht von Bedeutung, sondern alleinig die Ähnlichkeit zur Situation. Die Ähnlichkeit der beiden Situationen ist ein quantitativer Wert von $0,0$ bis $1,0$. Eine Ähnlichkeit von $0,0$ bedeutet, dass die beiden Fälle keine übereinstimmenden Attribute besitzen; umgekehrt entsprechen sich beide Situationen bei einem Wert von $1,0$. Standardmäßig werden beim Retrieval alle Fälle, deren Ähnlichkeit über einem bestimmten Schwellwert liegen, zurückgeliefert. Die Ähnlichkeit hängt von mehreren unabhängigen Werten ab und berechnet sich aus der gewichteten Summe der Gemeinsamkeiten zwischen diesen Werten. In Abbildung 5.4 (a) bestehen die Fälle aus zwei Attributen a_1 und a_2, deren Wertebereich von $0 - 2,5$ reicht. Die zusammengesetzte Ähnlichkeit der beiden Fälle c und c' ist die gewichtete Summe mit einer Wichtung von: $w_1 = 2, w_2 = 1$. Daraus ergibt sich die in Abbildung 5.4 (a) schraffiert Fläche. Dies ist allerdings nicht optimal. Zum Beispiel ist der Fall mit der Ergebnis (Outcome) von 0,9 nicht im Bereich der gewichteten Ähnlichkeit. Die Ähnlichkeit ist dabei höher als die des Falls, der mit 0,3 Outcome markiert ist, der jedoch im betrachteten Situationsraum liegt. In diesem Szenario wird die bessere Lösung nicht betrachtet, obwohl sie nur ein wenig von der geforderten Ähnlichkeit abweicht. Die Güte der gefundenen Ergebnisse hängt demnach auch von der Ähnlichkeitsfunktion ab. Eine bessere Funktion ist in Abbildung 5.4 (b) dargestellt. Sie nutzt die pythagoreische Entfernung[3] aller Attributdimensionen.

Für die Ermittlung der Ähnlichkeiten von Variablen existieren verschiedene Berechnungsvorschriften, die eine Berechnung des allgemeinen Abstands für bestimmte Wertebereiche ermöglichen. Ein Beispiel dafür ist die Ähnlichkeitsfunktion für positive reelle Zahlen, die in [20] zu finden ist:

$$sim(x, y) = 1 - \frac{|x - y|}{x + y}$$

Jedoch gibt es nicht für alle Datentypen eine solche Ähnlichkeitsfunktion. Ist der Datentyp eines Attributs ein Aufzählungstyp (Enumeration)[4], kann eine Ähnlichkeit von Werten für dieses Attribut nicht festgelegt werden, mit Ausnahme des trivialen Falls, dass

[3] $Abstand = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$
[4] Der Wertebereich ist eine Reihe von einzigartigen, aber unabhängigen Elementen.

beide Attribute identische Werte aufweisen. Ein weiterer Ansatz ist die statistische Analyse der Werteverteilung aus einer Menge an Trainingsdaten. Mit Clustering-Algorithmen kann allerdings der statistische Zusammenhang zwischen den Werten ermittelt werden. Die Ähnlichkeit wird somit über Erwartungen und Wahrscheinlichkeiten ausgedrückt, die unter Umständen auch fehlinterpretiert werden können.

Hat der Direktor nun über die Ähnlichkeitsfunktion passende Fälle gefunden, muss er versuchen, dass mit ihnen verbundene Adaptionswissen wiederzuverwenden. Die Annahme dabei ist, dass die Anwendung der Lösung eines bestimmten Falles zu einem ähnlichen Ergebnis bei der derzeitigen Situation führt. Selbst von einer Lösung, die aus den Ergebnissen von mehreren Fällen gebildet wurde, wird erwartet, dass sie ein ähnliches oder besseres Ergebnis zeigt. Der Direktor muss dabei versuchen, Fälle mit einem guten Ergebnis zu nutzen und solche mit schlechten zu ignorieren. Aber er kann sich nicht über das zukünftige Ergebnis völlig sicher sein, denn: (1) selbst wenn ein Fall der Situationsbeschreibung exakt gleicht, gibt es einige Umwelteinflüsse, die nicht von der Situationsbeschreibung erfasst werden können und (2) es ist nicht unwahrscheinlich, dass Fälle mit ähnlichen Situationen und gleicher Lösung zu unterschiedlichen Ergebnissen führen. Der Direktor steht dabei vor dem Risiko, wenn sich die Ergebnisse derart unterscheiden. Er kann sich für eine schlechte Alternative entscheiden, die zu einer ähnlichen Situation gehört, aber in 50% der Fälle versagt, während es viele ähnliche Situationen mit einer unterschiedlichen Lösung gibt, die zu 90% funktionieren.

Der Direktor könnte derart entwickelt werden, dass er mehrere Fälle wiederverwendet, anstatt eines einzelnen. Es wäre zudem möglich, Mittelwerte für bestimmte Teile der numerischen Berechnung von Ergebnissen zu nutzen oder er wählt die am häufigsten (in dieser Situation) ausgeführten aus. Es gibt eine Menge an Möglichkeiten, wie ein Ergebnis aus einer Reihe von ähnlichen Fällen zu gewinnen ist. Dies ist die Stärke des Case-based Reasoning. Es synthetisiert neue Ergebnisse aus bereits gegebenen.

Das Hinzufügen von Fällen in die Falldatenbank ist unkompliziert [1]. Fälle, die aus Trainingsdatensätzen stammen oder im laufenden Betrieb gelernt wurden, werden in die Datenbank eingefügt. Dabei ist es nicht notwendig, die vorhandenen Fälle anzupassen oder ggf. ähnliche bzw. gleiche zu löschen. Auch die Reihenfolge, in der die Fälle eingefügt werden, hat keine Auswirkungen auf die Qualität der Fallbasis. Bei großen Datenbanken ist der Einsatz von Indizes notwendig, damit die Zeit für das Querying nicht zu groß wird. Ein Index bildet Gruppen von Fällen. Dafür kann ein gemeinsamer Wert ein Kriterium sein, d. h. es werden Cluster von Fällen nach der Ähnlichkeit des einen oder mehrerer Attribute gebildet. Die Auswahl der Kriterien hängt von der Struktur der Situationsbeschreibungen ab, wobei die Wahl der richtigen Indexattribute ist dabei nicht immer einfach und immer von der jeweiligen Domäne abhängig. Eine andere Möglichkeit (unabhängig von der Indizierung) ist die Strukturierung der Fallbasis selbst. Fälle werden dann nicht mehr in Mengen gespeichert, sondern in baumartigen Datenstrukturen, die es ermöglichen, die Situationsbeschreibungen in mehrdimensionalen Attributräumen zu ordnen. Fälle mit ähnlichen Situationen besitzen einen kleineren Abstand im Baum, als solche mit einer geringeren Ähnlichkeit. Die Ordnung bezieht sich dabei auf einzelne Attribute und nicht auf mögliche Kombinationen (größer Aggregate), wie sie für Indizes verwendet werden. Das einzige Maß, um die Situationen zu vergleichen, sind die Ähnlichkeitsfunktionen, die bereits für das Retrieval benutzt wurden. Im Gegensatz zu Indizes, erfordert diese Ordnung keine weiteren semantischen Informationen.

Nachdem ein Fall gefunden und angewendet wurde, ist es möglich, das Ergebnis für den

Case in die Falldatenbank einzutragen, d. h. wie gut oder schlecht die Lösung war. Dieser Wert kann nicht berechnet werden. Ein Sachverständiger, der die Log-Aufzeichnungen nach Ausführung auswertet, kann jedoch eine Bewertung der neuen Fälle vornehmen. Es ist auch möglich, dass der Direktor das Ergebnis durch Überwachung des Workflows erfährt, geeignete Sensoren vorausgesetzt. Dies würde es dem Direktor ermöglichen, mit einer geringen Anzahl an Trainings-Cases zu starten und dann im Produktivbetrieb immer weiter selbstständig dazu zu lernen.

Einsatzmöglichkeiten Ein auf Case-based Reasoning basierender Direktor kann Adaption auf Grundlage von *Erfahrung* vorschlagen. Der Direktor hat Zugang zu einer Basis an bewerteten und bereits angewandten Adaptionen und versucht, erfolgreiche Verhaltensweisen zu imitieren. Er weiß aus Erfahrungen auch, wie bestimmte Vorschläge die Leistung eines Workflow in der Vergangenheit beeinflusst haben und interpoliert diese Kohärenz in der unmittelbaren Zukunft. Mit Case-based Reasoning kann der Direktor ähnliche Fälle für die aktuelle Situation finden und sich durch das dynamische Hinzufügen von Fällen sich selbst anpassen. Er wäre somit vollständig reflexiv. Im Gegensatz zu den regelbasierten Systemen ist die quantitative Ähnlichkeit genauer und vorhersehbarer und hängt nicht von der Dichte der gespeicherten Fälle ab. Auch das Problem der Überanpassung kann nicht auftreten und es gibt keine Grenze für die Komplexität der Situationsbeschreibung.

Zudem kann die Fallbasis auch widersprüchliche Cases speichern, d. h. Fälle, die die gleiche oder eine ähnliche Situation als Grundlage besitzen, aber unterschiedliche Lösungen und unterschiedliche Ergebnisse haben. Der Direktor kann dann dynamisch zwischen diesen Fällen entscheiden. Auch kann die Datenbank redundante Fälle beinhalten, die jedoch mit unterschiedlichen Ergebnissen ausgestattet sind. Der Direktor kann von diesen erkennen, wie hoch das Risiko bei der jeweiligen Lösung ist und kann je nach Situation die Alternativen beurteilen.

Eine besondere Stärke des Case-based Reasoning ist, dass der Retrieval-Algorithmus Fälle mit einer unvollständigen Situationsbeschreibung toleriert. Dies tritt vor allem dann auf, wenn verschiedene Variablen nicht gesetzt sind, wofür Verzweigungen verantwortlich sein können. Die Attribute, die nur in einer Situation existieren, können dann nicht verglichen werden und werden mit dem Wert 0 geschätzt. Fehlt bei Regelbäumen ein Attribut, kann der Baum nicht vollständig traversiert und keine Adaption vorgeschlagen werden. Zudem unterstützen die Ähnlichkeitsfunktionen verschiedene Arten von Eingabewerten. Verschiedene Ähnlichkeitsfunktionen für unterschiedliche Datentypen ermöglichen es, Werteähnlichkeit von Attributen zu gewinnen. Für die in Workflows selbstdefinierte Typen, wie Enumerationen oder komplexe Datentypen, müssen angepasste Ähnlichkeitsfunktionen angegeben werden.

Ein generelles Problem bei allen Anwendungen von Case-based Reasoning ist, dass eine große Menge an Probenfällen erforderlich ist, um akzeptable Ergebnisse zu erzielen. Die Falldatenbank muss mit genügend Situationsproben gefüllt werden, so dass beim Retrieval überhaupt Lösungen gefunden werden, denn bei Case-based Systemen ist es nicht möglich, dass einzelne Situationen einen großen Situationsraum beschreiben. Regeln hingegen müssen nicht alle Attribute beschreiben und können trotzdem viele Situationen abdecken.

Vergleich In den zwei vorangegangenen Abschnitten 5.1.1 und 5.1.2 wurden mögliche Ansätze diskutiert. Zur Prüfung ihrer Chancen und Risiken werden zwei unterschiedliche Kriterien angewandt: (1) die Erwartungen an die Leistung und ob die jeweiligen Lösungen bereits erfolgreich umgesetzt wurden und (2) die Probleme und Risiken für die Umsetzung der jeweiligen Lösung.

Die beiden Ansätze können aus (Stich-)Proben lernen, welches Verhalten in bestimmten Situationen angebracht ist. Zur Laufzeit versuchen sie ihr Wissen abzurufen und bestehende Lösungen auf die aktuelle Situation anzuwenden. Dabei leiten sie keine neuen Erkenntnisse aus dem Wissen ab, sondern können lediglich neues Wissen hinzulernen. Die Unterschiede liegen in der Art, wie das Wissen gespeichert und gelernt wird sowie wie die gelehrten Entscheidungen in einer bestimmten Situation angewendet werden. Der regelbasierte Ansatz kann dabei nur auf positive Erfahrungen zurückgreifen, während beim Case-based Reasoning sowohl gute als auch schlechte Erfahrungen akzeptiert werden. Damit ist es möglich negatives Verhalten zu erlernen, damit es in den betreffenden Situationen nicht erneut zur Anwendung kommt. Beide besitzen die Fähigkeit, Entscheidungen, die nicht exakt auf die aktuelle Situation passen, zu finden, d. h. sie können vergleichbare bzw. ähnliche Fälle aufspüren. Die Ähnlichkeitsfunktion des Case-based Reasoning ist dabei wesentlich umfangreicher als die Verallgemeinerung bei Entscheidungsbäumen. Das Hauptproblem der Entscheidungsbäume bzw. regelbasierter Systeme ist die begrenzte Wissensbasis. Der Entscheidungsbaum hat eine bestimmte Kapazität und neigt mit steigender Anzahl zu einer Überangepasstheit. Beim Case-based Reasoning besteht dieses Problem nicht, erfordert aber auf der anderen Seite eine gute Situationsraumabdeckung, um sinnvolle Ergebnisse zu liefern. Durch die Ähnlichkeitsfunktion ist zudem eine *ungenaue* Wissensbasis realisierbar, da die Situationen nicht zu 100% übereinstimmen müssen.

5.1.3 Belief-Desire-Intention

Die zuvor beschriebenen Möglichkeiten zur Umsetzung eines reflexiven Direktors basieren auf gemachten Erfahrungen oder Expertenwissen. Diese Informationen sind auch für Menschen wichtig, um Lösungen für aufkommende Probleme zu finden. Dabei ist immer das aktuelle Ziel zu beachten. Ein *Ziel* beschreibt eine erwünschte Situation bzw. Bedingungen. Es beschreibt allerdings keine Aktionen, die zu der Situation führen. Es ist lediglich möglich zu ermitteln, inwieweit die aktuelle von der Wunschsituation abweicht. Ein zielbasierter Direktor versucht demnach, für eine Instanz einen bestimmten Zustand zu erreichen. Dazu muss er beurteilen können, wie weit weg die aktuelle Situation von dem Ziel ist, bzw. inwieweit eine Adaption erfolgreich war.

Der wesentliche Vorteil der *verwebten* Workflows ist die Kombination aus Komposition von Subprozessen und variablem Rücksprungpunkt. Werden diese Webevorgänge mit dem Direktor automatisiert durchgeführt, muss dieser passende Workflowmodelle finden. Es gibt eine Vielzahl von Kriterien, die zum Vergleich herangezogen werden können. Neben der aktuellen Situation ist das Ziel bzw. der Zweck der Instanz von Bedeutung. Ein Ziel kann die Entscheidung für eine Adaption entscheidend beeinflussen. Dabei wird nicht jede Instanz mit dem gleichen Ziel gestartet. Es ist deshalb nicht sinnvoll, das Ziel an das Workflowmodell zu binden, sondern an jede Instanz. Der Direktor versucht während der Ausführung alle Ziele der Instanz zu erfüllen. Jedoch existiert keine einheitliche Sprache zur Beschreibung von Zielen und muss demnach entwickelt werden.

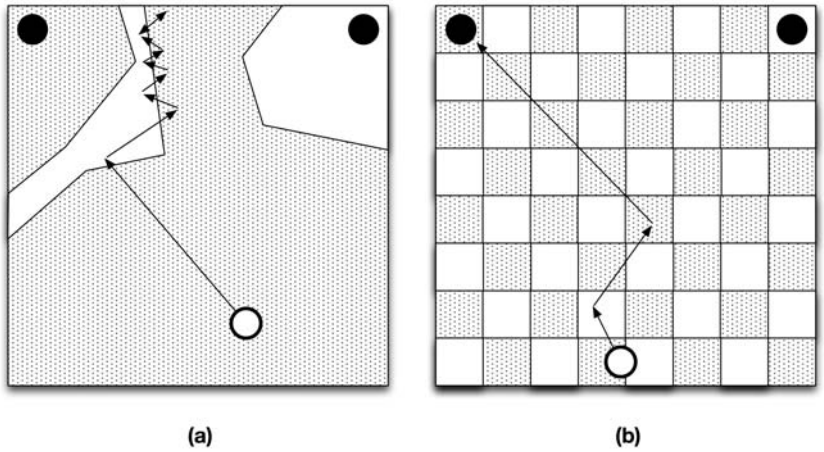


Abbildung 5.5: Verhalten von Agenten

Während es für das regelbasierte und fallbasierte Schließen bereits Implementierungen existieren, können Ziele nur nachgeahmt werden. Ein Ansatz dafür ist Belief-Desire-Intention[5] (BDI), der bereits in verschiedenen Arbeiten Verwendung fand [89, 123, 86, 126]. Die Grundlagen für den BDI-Ansatz wurden in [27] beschrieben. Mit BDI – einer einfachen Form des Planens – wird versucht, das menschliche Verhalten durch die Analyse der Motivation zu erklären. Die meisten psychologischen Modelle, warum Menschen handeln wie sie handeln, konzentrieren sich auf die Quellen der menschlichen Motivation. Der Hauptgrund ist, dass der Prozess der menschlichen Entscheidungsfindung zu komplex ist und nur schwer durch empirische Studien untersucht werden kann. Zu viele Einflussfaktoren sind zu beachten. In [27] wird deshalb ein theoretisches Modell vorgestellt, das nicht versucht, alle Aspekte der Entscheidungsfindung einzubeziehen, aber immer noch näher an die menschliche Natur heranreichen soll, als alle auf Logik basierenden Modelle. Das ursprüngliche Verständnis war, dass Menschen (1) ihre aktuelle Situation zunächst analysieren, (2) mögliche Handlungsoptionen identifizieren, (3) diese bewerten und (4) schlussendlich eine Option wählen. Dieser Ansatz lässt dabei bewusst außer Acht, dass das menschliche Handeln oft irrational und willkürlich zu sein scheint.

Ein zweites Problem beim Nachahmen menschlichen Verhaltens wurde erkannt, als sich Forscher mit der Entwicklung von künstlicher Intelligenz beschäftigten. Es wurde mittels Agenten versucht, individuelles und vernünftiges Verhalten zu (re-)produzieren. Dabei konnte irrationales Verhalten von Anfang an ausgeschlossen werden, da nicht ein menschliches Gehirn, sondern ein Computer benutzt wurde. Solang der Agent nur eine Aufgabe bekommt, erfüllt er die Erwartungen der Entwickler in vielen Fällen. Allerdings haben die Agenten bei mehreren, sich möglicherweise widersprechenden, Aufgaben, das Problem, sich auf eine dieser Aufgaben zu konzentrieren. Sie neigen dazu, zwischen den Problemen hin- und her zu springen. Durch das abwechselnde schnelle Umschalten zwischen den verschiedenen Aufgaben, können die Agenten keines der Probleme lösen. Die Menge an verfügbaren Informationen über alle möglichen Situationen lässt die Anzahl der Möglichkeiten stark steigen, so dass der Agent bei der Entscheidung nach dem aktuell zu lösenden Problem, die Problemlösung vergisst. Abbildung 5.5 zeigt zwei Beispiele. Beide enthalten zwei Zielsituationen[6] (Zielzustände), die es nacheinander zu erreichen

[5] Glaube-Wunsch-Absicht.
[6] Gefüllte schwarze Kreise in Abbildung 5.5.

gilt, um die beiden entsprechenden Aufgaben zu erfüllen. Die Situationen sind weit entfernt, d. h. die Belegung einiger Attribute ist unterschiedlich. Bei Zustand Z_1 muss eine Attribut den Wert 50 annehmen, während er in Z_2 -50 sein muss. Bewegt sich der Agent in Richtung Z_1, nähert sich a dem Wert 50. Gleichzeitig bemerkt der Agent, dass er sich damit mehr vom Zustand Z_2 entfernt. Eine vernünftige Lösung wäre, eines der beiden Ziele zuerst zu erreichen und anschließend das Zweite anzusteuern. Der Agent aber kann sein Vorgehen nicht planen. Nach jedem Schritt wird neu berechnet, welches Ziel es zu erreichen gilt. Das Muster in Abbildung 5.5 zeigt die Ergebnisse dieser Berechnung. In (a) wählt der Agent das Ziel, das ähnlich zu seiner derzeitigen Situation ist. Dies ist Z_1, so dass sich der Agent in Richtung Z_1 orientiert. Auf seinem Weg ändert sich allerdings die Ähnlichkeit. Die Ähnlichkeit[7] um Z_1 herum entspricht der von Z_2, wodurch der Agent versucht Z_2 zu erreichen. Auf diesem Weg ändert sich allerdings wiederum die Ähnlichkeit. Schließlich wird sich der Agent an der Grenze zwischen den beiden Bereichen entlang bewegen, wie durch die Pfeile in der Abbildung dargestellt ist. Ein Ziel erreicht er nie. In (b) sind die Gebiete kleiner und der Agent erreicht schließlich eines der Ziele. Allerdings ist die Wahl der Aufgabe des Agenten völlig willkürlich und es ist nicht klar, ob das andere Ziel wird erreicht.

Das Problem ergibt sich dabei allerdings nicht aus ungenauen Berechnungen. Nachteilig wirkt sich die ständige Neuorientierung aus, denn der Agent berechnet nach jedem Schritt sein Ziel neu. Er verliert ein Ziel aus den Augen. Dieses Problem ist bekannt als das *Vergesslichkeitsproblem* [61]. Verpflichtet sich der Agent zu einem bestimmten Ziel und vergisst seine aktuelle Aufgabe nicht, kann dieses Problem überwunden werden. Aufbauend auf diesen Forschungsergebnissen entwickelte Bratman einen anderen Ansatz. Im Gegensatz zu früheren Ansätzen, basiert BDI nicht nur auf Berechnungen über die aktuelle Situation, sondern bezieht eine „geistige“ Haltung in die Berechnung mit ein. Diese entspricht einem übergeordneten Ziel als Richtschnur für Entscheidungen.

Nach Bratman [27] zeichnet einen vernünftigen Agenten aus, dass er die besten Entscheidungen auf Basis seiner eigenen *Fähigkeiten* und allen zugänglichen Informationen findet. Die Entscheidungen, die aus diesem Wissen abgeleitet werden, können fehlerhaft sein, sind allerdings die Besten, die vom Agenten erwartet werden können. Das Wort *Belief* beschreibt diese Idee. Der Agent hat eine Vorstellung[8] über die ihn umgebende Welt, die sein ganzes Wissen repräsentiert[9]. Zu diesem Wissen gehören laut Bratman atomare Aussagen, Prädikate und logische Ausdrücke.

In einem BDI-System gründet sich die Auswahl der Handlungen auf der Vorgabe von Zielen. In BDI werden Ziele *Desires* (Wunsch) genannt. Die Ziele werden von einer externen Quelle vorgegeben und stellen den inneren Antrieb des BDI-Agenten dar. In [126] werden die Desires als motivgesteuerte Ausrichtung (engl. motivational state) eines BDI-Agenten bezeichnet. Desires sind dabei keine Aufgaben, die ein Agent abzuarbeiten hat, sondern stellen ein separates Konzept dar. Zudem sind nur wenige der vielen möglichen Desires zu einem Zeitpunkt aktiv.

Ein BDI-Agent kann sich durch das Mittel der Desires an veränderte Umweltbedingungen anpassen und somit reflexiv handeln, indem er neue Desires den aktuellen hinzufügt oder bestehende aktive Desires nicht mehr anwendet, d. h. inaktiviert. Veränderungen an der Menge der Desires sind durch Bedingungen geregelt, die explizit angeben werden müssen.

[7] Als schraffierte Flächen dargestellt.
[8] Die wörtliche Übersetzung von Belief wäre Überzeugung oder Glaube.
[9] Dieses Wissen muss weder vollständig noch richtig sein.

Aus der Menge aller möglichen Desires entnimmt der Agent neue Desires, wenn eine entsprechende *Create*-Condition für den aktuellen Zustand zutrifft. Das Desire dauert an, bis eine entsprechende *Drop*-Condition wahr wird. Ein Desire wird deaktiviert, wenn seine Drop-Condition mindestens einen Schritt lang gilt.

Praktische Anwendungen der BDI-Architektur zeigen, dass es mehrere Arten von Desires gibt. In [123] werden vier Arten beschrieben: Achieve (Erreichen), Query (Anfordern), Maintain (Aufrechterhalten) und Perform (Durchführen). Am häufigsten wird das Desire *Achieve* genutzt. Es definiert eine Situation, die der Agent zum Erfüllen des Desires erreichen muss. Sobald der Agent die gewünschten Situationen erreicht hat, kann er das Desire aufgeben. Das *Query*-Desire (in [125] auch als Test-Desire bezeichnet) bedeutet, dass der Agent neue Informationen in die Beliefs eintragen soll. Zur Erfüllung eines solchen Desires muss der Agent Informationen durch gezielte Anfragen an seine Umgebung sammeln. Dabei hängt die formale Darstellung von der Art der Beliefs ab. *Maintain*-Desires sind ähnlich zu Achieve-Desires, nur dass hier die gewünschten Situationen beibehalten werden sollen. Der Agent kann allerdings innerhalb einer gewissen Toleranzschwelle Änderungen vornehmen. *Perform*-Desires beschreiben im Gegensatz zu den anderen keine gewünschten Situationen, sondern legen nur eine auszuführende Funktion fest, wobei der dadurch erreichte Zustand für das Desire nicht relevant ist und deshalb auch nicht beschrieben werden muss.

Wie in Abschnitt 3.2 beschrieben, soll der Direktor *on-the-fly*, d. h. zur Laufzeit, Änderungen durchführen. Traditionelle zielorientierte Systeme arbeiten allerdings nicht auf diese Weise. Hier muss der Gesamtplan vor der Durchführung feststehen. Jedoch werden alle erforderlichen Aktionen, um ein Ziel zu erreichen, zur Laufzeit ermittelt und geplant, bevor der Agent zu handeln beginnt[10]. Die BDI-Architektur nimmt diesen Gedanken auf und führt den Begriff der *Intention* (Absicht) ein. Eine Intention ist ein Plan, der zur Erfüllung eines Desires beitragen soll. Der Agent erzeugt dabei nicht neue Intentions, sondern wählt eine bereits vordefinierte Lösung aus. Wenn der Agent eine neue Intention benötigt, bewertet er den Nutzen möglicher Lösungen hinsichtlich der aktiven Desires. Die Auswahl einer geeigneten Intention wird als *means-ends reasoning* (Mittel-Zweck-Überlegung) bezeichnet [196].

Sobald der Agent eine Intention gewählt hat, ist er auf diese festgelegt [126]. Er folgt dann diesem Plan (Anweisungen) und wird diesen nicht ablegen. Das bedeutet, dass der Agent ein hohes Vertrauen in die aktuelle Intention legt. Wenn jedoch die derzeitige Intention fehlzuschlagen scheint, ist der Agent in der Lage, die Planausführung zu stoppen und eine neue, wahrscheinlich geeignetere Intention zu wählen [196]. Das BDI-Modell legt dabei aber nicht fest, wann ein Agent einen fehlgeschlagenen Plan aufzugeben hat. In [126] wird zwischen drei Arten von Agenten bzgl. der Abbruchbedingungen unterschieden. Der *blinde* Agent gibt nie eine Intention auf, der *zielstrebige* Agent entscheidet nur auf Basis der Desires und der *aufgeschlossene* Agent berücksichtigt Beliefs und Desires gemeinsam.

Beliefs, Desires und Intentions sind als zentrale Elemente der Ausführung bei jedem Ausführungsschritt Änderungen unterworfen. Als erstes verarbeitet der Agent alle eingehenden Informationen (Input) und aktualisiert die Beliefs. Dies ermöglicht die Über-

[10] Fortgeschrittene Planer sind allerdings in der Lage zu agieren, selbst wenn es einen noch nicht vollständig spezifizierten Plan zur Zielerreichung gibt, d. h. sie führen Teilpläne aus. Sie stützen ihr Handeln aber weiterhin auf eine Reihe vorgeplanter Aktionen. [145] gibt eine gute Einführung in das Thema „Automatische Planung“.

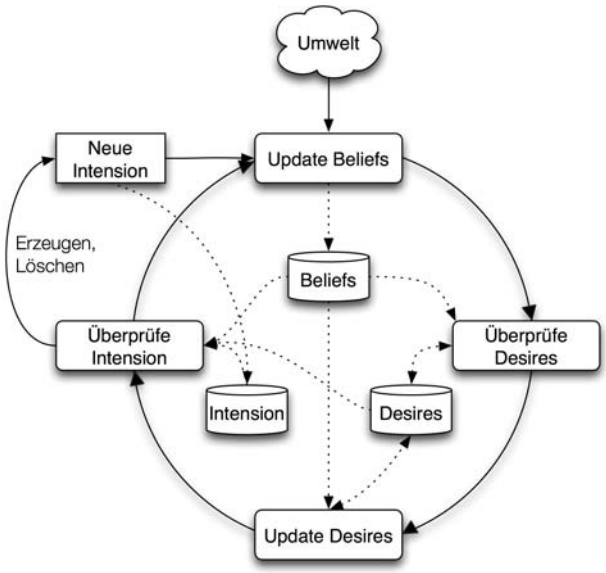


Abbildung 5.6: Phasen des BDI-Zyklus

prüfung, ob einige der aktiven Desires mit dem letzten Schritt erreicht wurden. Abhängig von diesen Änderungen werden alle Create- und Drop-Bedingungen für die Desires geprüft. Zum Schluss wird der Fortschritt der aktuellen Intention überprüft, um z. B. festzustellen, ob diese abgebrochen werden muss. Wurde die aktuelle Intention beendet, muss entsprechend nach einer neuen gesucht werden. Damit entsteht ein Bearbeitungszyklus, der in Abbildung 5.6 dargestellt ist. In der Mitte der Abbildung sind die drei passiven Elemente dargestellt: Belief, Desire und Intention. Um sie herum sind die verschiedenen Phasen des BDI-Zyklus angeordnet.

5.2 BDI-Direktor zur autonomen Rekonfiguration von Workflows

Die BDI-Architektur eignet sich demnach gut zur Imitation von menschlichem Verhalten, weshalb sie als Basis für die Realisierung der Steuerung für verwebte Workflows gewählt wurde. Dazu muss jedoch der Ansatz an die Workflow-Domäne angepasst werden, d. h. er muss die Workflowinstanzen überwachen, Laufzeitdaten verarbeiten und Adaptionen vorschlagen. Außerdem muss er in der Lage sein, fehlendes semantisches Wissen auszugleichen, eine Funktion, die nicht Teil der ursprünglichen BDI-Architektur ist. Das Ziel ist es aber nicht, einen bestehenden BDI-Reasoner zu nutzen und diesen mit Input- bzw. Output-Adaptern zu versehen, um den BDI-Ansatz in ein WfMS zu integrieren. Vielmehr soll der Direktor selbst als BDI-System agieren und dabei alle Workflowkonzepte als klassische BDI-Elemente verarbeiten. Abbildung 5.7 veranschaulicht die Beziehungen in einem konzeptionellen UML Klassendiagramm.

Belief Variablen, Ausnahmen und Laufzeiten repräsentieren die Beliefs des BDI-Direktors, wobei es sich hier um eine 1-zu-1 Beziehung handelt. Variablen repräsentieren das Primärwissen über die gerade ausgeführte Workflowinstanz, da der Direktor jeweils nur

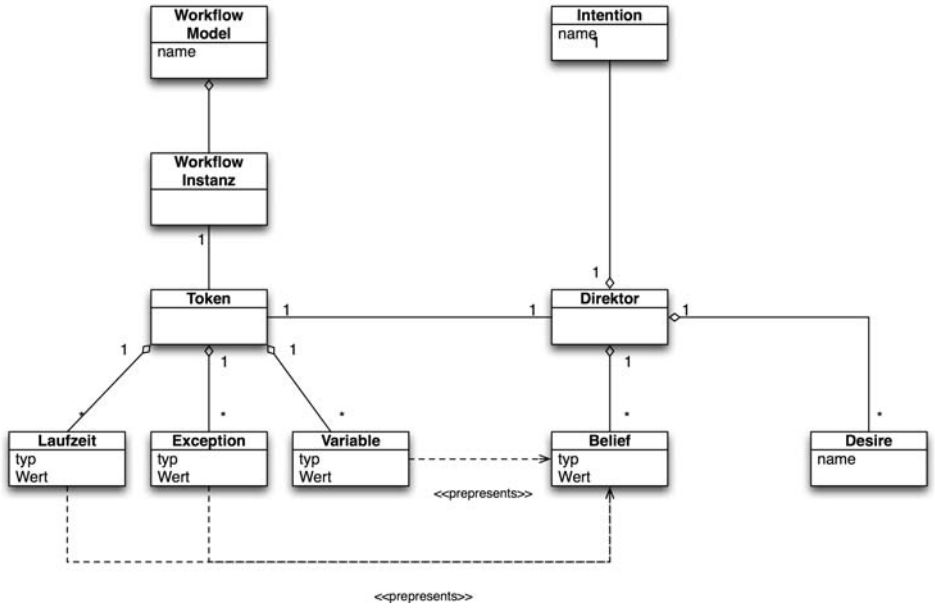


Abbildung 5.7: Abbildung des BDI-Ansatzes auf Workflowelemente

eine Instanz überwacht. Zudem stehen weitere Informationen über die Workflowinstanz zur Verfügung. Dazu zählen die aufgetretenen Ausnahmenbedingungen und die Laufzeiten der einzelnen Aktionen. Beide Informationsquellen sind wichtige Indikatoren für den Zustand einer Instanz. Es gilt zu reagieren, wenn Ausnahmen auftreten oder Funktionsaufrufe, wie z. B. ein Webservice Aufruf, zu lang dauern. Die Laufzeiten können als Variablen aufgefasst werden. Durch eine geschickte Kodierung eines Variablennamens, beispielsweise durch eine Kombination aus dem Namen des Workflows und TaskID, lassen sich diese Werte wie Variablen verarbeiten, die zudem automatisch gesammelt werden.

Desire Während Beliefs gut mit den Variablen in Workflows korrespondieren, gibt es kein Äquivalent für *Desires*. Desires können demnach nicht im WfMS selbst gespeichert werden, da es dem Ziel, den Direktor ohne Änderungen in das WfMS zu integrieren widersprechen würde. Die Desires müssen demnach im Direktor selbst gespeichert werden. Desires sind dabei keine komplizierten Gebilde, denn sie repräsentieren Ziele, die ein abstraktes Konzept darstellen. Deshalb wird eine Desire mit einem Namen, einer eindeutigen Nummer (ID) und einem Zustand (aktiv oder passiv) beschrieben.

Im BDI-Ansatz wird zwischen vier Arten von Desires unterschieden, die bereits in Abschnitt 5.1.3 beschrieben wurden. Für die Verwendung des BDI-Ansatzes in Workflows kommt nur das Maintain-Desire in Frage[11]. Zum Start der Instanz wird ihr eine Menge an Desires übergegeben, die ihre Ziele charakterisieren. Der Direktor überwacht anschließend die Ausführung und stellt mit seinen Adaptionsvorschlägen sicher, dass diese Ziele

[11] Query- und Perform-Desires müssen nicht unterstützt werden, denn es muss kein neues Wissen akquiriert und keine explizite Aktion durchgeführt werden. Das Achieve-Desire gibt Situationen/Zustände vor, die erreicht werden sollen. Ein Workflow Direktor müsste demnach Aktionen durchführen, um einen bestimmten Workflowzustand zu erreichen. Dieser gewünschte Zustand würde der Workflowinstanz von außen oktroyiert, was wiederum dem Konzept des Direktors als Beobachter widerspricht.

unter allen Umständen erreicht werden. Das bedeutet auch, dass Desires nicht deaktiviert werden, wenn sie erreicht wurden oder eine beliebige Aktion ausgeführt wurde. Die Bedingungen, wann ein Desire erstellt oder gelöscht wird, können nur durch das Lernen gewonnen werden. Dies bedeutet, dass Desires unter allen Umständen befolgt und nur dann durch andere abgelöst oder ersetzt werden, wenn der Direktor gelernt hat, dass es zur Erreichung des Ausgangsziels besser ist.

Intention Das dritte umzusetzende Teilkonzept ist die *Intention*. Die Adaption durch Verweben integriert ein fremdes Workflowmodell in eine laufende Workflowinstanz. Dieser fremde Workflow stellt dabei die Lösung für ein Problem dar. In BDI ist die Intention eine Lösung, um ein Desire zu erreichen. Rao und Georgeff beschreiben in [126] eine Intention als eine *Sequenz von Aktionen*, die ein Agent aus einem *Store* auswählen kann. Es existieren demnach Parallelen zwischen diesen beiden Ansätzen, so dass beide Welten – Intention und Lösungs-Workflowmodell – miteinander vereint werden können. Die Ähnlichkeiten zwischen beiden Ansätzen (Intentions/Adaption) sind groß:

- Beide Ansätze beschreiben Pläne, die eine bestimmte Anzahl von Aktionen mit einer Ordnung beschreiben.
- In beiden Ansätzen ist nur ein Plan gleichzeitig aktiv.
- Intentions werden aus einem definierten Satz an vorgegebenen Intentions gewählt, Workflows aus einem Repository.

Für den BDI-Direktor steht folglich eine Menge von Workflowmodellen als *Intention* aus dem Repository zur Verfügung. Jedes Modell stellt einen Maßnahmenplan dar, den der Direktor in seiner Rolle als BDI-Agent annehmen kann. Auch der BDI-Zyklus mit seinen unterschiedlichen Aktivitäten entspricht den Anforderungen an den Direktor. Im Falle einer Situationsänderung wird der Direktor benachrichtigt und führt die BDI-Phasen aus. Abbildung 5.6 zeigt den BDI-Zyklus von den Updates der Beliefs bis zum möglichen Abbruch einer Intention. Dem Direktor stehen nach *Check Intention* drei Möglichkeiten zur Verfügung:

1. *Adaptieren:* Erzeugen einer neuen Intention und somit Einweben eines Workflows
2. *Abbrechen:* Abbrechen der aktuellen Intention bzw. Abbrechen der Adaption
3. *Fortfahren:* Fortfahren mit der aktuellen Intention, d. h. *Continue* vorschlagen

Es ist demnach möglich, die Konzepte des BDI-Ansatzes auf den Direktor-Ansatz zu übertragen. Für das Funktionieren eines BDI-Direktors ist es aber notwendig zu wissen, welche Desires zur Verfügung stehen und wann und unter welchen Umständen Desires und Intentions geändert werden. Da der Direktor kein a-priori-Wissen über die Abläufe des Workflows besitzt, muss er diese Vorgänge sukzessiv lernen. Das bedeutet, wenn B^S die Menge der Beliefs in einer bestimmten Situation S, D^S die Menge der Desires in dieser Situation, und I^S die aktuelle Absicht ist, dann müssen folgende Zuordnungen

$$B^S \times I^S \rightarrow D^{S^+} \text{ und } B^S \times D^{S^+} \rightarrow I^{S^+}$$

erlernt werden. wobei S^+ die neue, nachfolgende, Situation darstellt. Die Abbildungen werden im Folgenden *Desire- bzw. Intention-Abbildung* genannt. Die Desire-Abbildung berechnet aus den aktuellen Beliefs und der zu diesem Zeitpunkt geltenden Intention

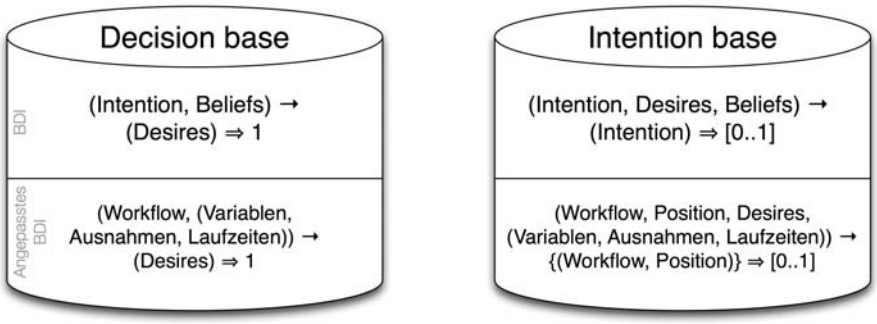


Abbildung 5.8: Spezifikation der Falldatenbanken

die neu gültigen Desires (D^{S^+}). Diese werden, in Verbindung mit den aktuellen Beliefs, in der Intention-Abbildung dazu verwendet, um die neue Intention zu berechnen. Die Abbildungen werden demnach aufeinander aufbauend verwendet.

5.2.1 Lernen von Adaptionen im BDI-Direktor

Die Herausforderung bei der Gestaltung des BDI-Direktors ist demnach die Definition bzw. das Lernen der beiden BDI-Abbildungen. Verschiedene Lernverfahren können dazu verwendet werden, die bereits in Abschnitt 5.1 vorgestellt wurden.

Die beste Möglichkeit der Umsetzung dieses zu lernenden Wissens, ist in Form von Cases, wie sie beim Case-based Reasoning verwendet werden. Die Grundkonzepte von CBR [145, 103] wurden in Abschnitt 5.1.2 vorgestellt. Die Idee, Case-based Reasoning zur Umsetzung des BDI-Ansatzes zu verwenden, ist nicht neu und wurde auch in anderen Publikationen vorgenommen. Dabei wurde aber oft nur eine Abbildung verwendet bzw. gelernt, wie z. B. in [118] und [44]. Wie in vielen anderen Publikationen ist der Grund dafür, dass die Desires nicht als explizite Elemente in den Abbildungen vertreten waren. Sie werden oft zusammen mit den Beliefs betrachtet. Dieser Ansatz kommt vor allem aus dem Umfeld des automatischen Planens, das eng mit BDI verbunden ist. Hierbei ist ein Desire einer Zielsituation, die der Planer zu erreichen versucht, identisch. In dieser Arbeit werden jedoch beide Konzepte getrennt betrachtet. Beliefs werden hier durch die Workflowvariablen repräsentiert. Würden Desires Beliefs zugeordnet, würde dies bedeuten, dass bestimmte Variablenbelegungen ein Ziel darstellen und wäre somit ein Teil der Workflowdefinition. Dies entspricht jedoch einer Vermischung von Konzepten zur Adaptions- und der Workflow-Beschreibung. Der Direktor wäre somit nicht mehr unabhängig vom WfMS.

Um diese zwei unterschiedlichen Abbildungen zu speichern und zu lernen, sind zwei verschiedenen Falldatenbanken und Fallspezifikationen notwendig, die in Abbildung 5.8 dargestellt sind. Im oberen Bereich ist die BDI-Abbildung dargestellt; im unteren Bereich das Äquivalent für den Direktor. Ein Fall besteht in beiden Datenbanken aus drei Komponenten: *Situation*, *Lösung*, *Ergebnis*. Grundsätzlich besteht eine Situation aus einer Reihe von Daten, die den Zustand des WfMS zu einem bestimmten Zeitpunkt beschreiben. Kolodner bezeichnet dies als eine *Mapping von Dimensionen zu Werten* [103], d. h. die Werte aller zur Verfügung stehenden Attribute. Welche dies sind, unterscheidet sich bei beiden Falldatenbanken. Die Lösung eines Falles beschreibt die Änderungen der

Desires oder einer Intention, während das Ergebnis die Güte der Veränderung charakterisiert. Dies geschieht über die Angabe eines Wertes zwischen 0 und 1.

Die Ausprägungen der Cases sind allerdings für jede Abbildung unterschiedlich. Die beiden Abbildungen sind nach ihren Datenbanken benannt: *Desire-Cases* und *Intention Cases*. Die Situationsbeschreibung der *Desire-Cases* enthält die Intention (Adaption), die der Direktor gerade ausführt (bzw. ausführen lässt) und eine Menge von Beliefs. Der Case enthält auch eine Lösung in Form eines Desires und der Information, ob ein Desire erstellt (aktiviert) oder gelöscht (deaktiviert) wurde. Daraus ergibt sich, dass dieser Case nur eine Desire-Änderung beschreiben kann. Wenn mehrere Desires in einem Schritt geändert werden sollen, müssen mehrere Fälle angegeben werden.

Intention-Cases beschreiben die Abbildung der Beliefs und Desires auf Intentions, d. h. Adaptionen. Beliefs werden hier wieder auf die Menge der Variablen, Ausnahmen und Laufzeitdaten abgebildet. Die Desires ergeben sich aus den aktiven Desires des Direktors. Darüber hinaus enthält die Situationsbeschreibung die aktuelle Intention und wann diese Adaption vorgenommen wurde. Dies lässt Rückschlüsse zu, welche und wie häufig bestimmte Workflows zur Erreichung eines Ziels verwendet wurden. Die Menge der Informationen könnte weiter gesteigert werden, wenn alle Kontextinformationen des WfMS mit in Betracht gezogen werden, um damit einen umfassenden Einblick in die Situation zu erlangen. Doch die vielen Informationen würden den Lösungsraum vergrößern, so dass das Problem des Findens ähnlicher Fällen erschwert wird. Stattdessen wird zur besseren Orientierung die Position des Tokens im Workflow festgehalten. Die Positionsangabe ist entscheidend für eine Adaptionsentscheidung. Für den Intention-Case gibt es drei Lösungstypen: Fortfahren, Adaptieren und Abbrechen.

Das Ergebnis ist ein Wert zwischen 0 (Misserfolg) und 1 (Erfolg). Diese Bewertung kann durch äußere Experten oder durch den Direktor selbst (Sensoren und Evaluatoren vorausgesetzt) erfolgen. Des Weiteren wird die Nutzungshäufigkeit (UseCount) des Falles gespeichert. Dieser Wert muss vom Direktor bei jeder Nutzung erhöht werden. Der Direktor kann diesen Wert anschließend als Entscheidungskriterium für oder gegen bestimmte Cases nutzen, indem er bei Pattsituationen den Fall mit der höheren Nutzungshäufigkeit bevorzugt.

5.2.2 Berechnung von Adaptionen

Wie bereits beschrieben, wird Case-based Reasoning angewandt, um die BDI-Direktoren zu realisieren. Dabei sind die CBR-Algorithmen nicht zu komplex, müssen aber an die Domäne angepasst werden [103, 145]. Dies führt dazu, dass keine standardisierte CBR-Engine genutzt werden kann. Es gibt einige wenige offene Bibliotheken, die CBR-Funktionalitäten zur Wiederverwendung in anderen Projekten bereitstellen, wie z.B. das *CBR-Toolkit* der Universität Edinburgh[12] oder das Open Source Projekt *Selection Engine*[13]. Diese sind allerdings ungeeignet zur Verarbeitung komplexer Situationen mit Attributen vieler verschiedener Datentypen. CBR-Tools beschreiben die Fälle mit deklarativen oder logischen Sprachen. Ein großer Vorteil dabei ist, dass diese Sprachen durch Grammatiken formalisiert sind und eine bessere Wiederverwendung der Fallbeschreibungen bieten. Auch kann die Basis leicht zu einer Datei oder Datenbank ausgelagert werden. Eine Umsetzung für Workflows stellt aber erweiterte Anforderungen. Es müssen

[12] `http://www.aiai.ed.ac.uk/project/cbr/` (zuletzt aufgerufen am 25.02.2011).
[13] `http://sourceforge.net/projects/selectionengine/` (zuletzt aufgerufen am 25.02.2011)

zusätzlich komplexe und native Datentypen unterstützt werden. Bei der Verwendung von Webservices in einem Workflow werden neue Datentypen zur Laufzeit hinzugefügt, die dem Request- und Response-Typen des Webservices entsprechen, d. h. die Typen können nicht statisch festgelegt werden. Dies können bestehende Implementierungen nicht abdecken.

Die zentrale Funktion im Case-based Reasoning ist die Ähnlichkeitsfunktion. Die Formel zur Berechnung der Ähnlichkeit hängt dabei von den Attributen der Situationsbeschreibung ab und ist somit ein zusammengesetzter Wert. Der gewichtete Mittelwert der einzelnen Attributähnlichkeiten ist eine weit verbreitete Lösung. Dabei müssen die Variablen, die gleichzeitig die Laufzeitdaten der Funktionen beinhalten, und die Ausnahmen verglichen werden. Die Ähnlichkeitsfunktionen für native Datentypen können dabei einfach implementiert werden. Die Ähnlichkeitsfunktion für double-Variablen ist eine leicht modifizierte Variante der im Abschnitt 5.1.2 beschriebenen Formel[14]:

$$sim(x, y) = 1 - \frac{|x - y|}{|x| + |y|}$$

Für Zeichenketten kann entweder ein einfacher Test auf Gleichheit oder ein Maß für die Editierdistanz herangezogen werden. Dieser Vergleich ist hierbei auf die syntaktische Ebene beschränkt. Synonyme können mit diesen Methoden nicht erkannt werden. Ein semantikfreier Direktor ist demnach vor allem bei WfMS einsetzbar, die Workflows mit numerischen Berechnungen ausführen[15].

Komplexe Datentypen müssen die Ähnlichkeitsfunktion selbst implementieren[16]. Gleiches gilt für die ebenfalls in den Beliefs gespeicherten Ausnahmen. Die Ähnlichkeitsfunktion berechnet Werte zwischen $0,0$ – keine Gleichheit – und $1,0$ für vollständige Gleichheit. Die beiden Mengen der Instanzvariablen werden in zwei Phasen verglichen: Zuerst werden die beiden Mengen von Variablen verglichen und anschließend die Werte der Variablen. Beispielsweise wird die Ähnlichkeit der Desire-Sets durch folgende Funktion berechnet:

$$\mathrm{sim}(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|}$$

Diese berechnet das Maß, wie viele Desires die beiden Mengen gemeinsam haben.

Wenn der Direktor nach einer Adaption gefragt wird, aktualisiert er die Desires mithilfe der Desire-Abbildung und wählt anschließend eine Intention auf Basis der Intention-Base. Für die Aktualisierung der Desires sucht der Direktor alle Cases, die zumindest ähnlich zur derzeitigen Intention- und Belief-Situation sind, wobei die Ähnlichkeit einen gewissen Schwellwert überschreiten muss. Für jeden gefundenen Case wird das darin gespeicherte Desire dem Direktor hinzugefügt oder entfernt.

Da nur eine Intention als Adaptionsentscheidung an den Koordinator zurückgeliefert werden kann, gilt es aus der Menge der ähnlichen Cases den Case zu finden, der die beste und sicherste Adaption darstellt. Die gefundenen ähnlichen Cases sind die Basis für die Priorisierung der Ergebnisliste. Diese enthält alle Fälle, die über dem Schwellwert liegen und somit die besten Ergebnisse darstellen.

[14] Kann ähnlich für alle numerischen Datentypen angewandt werden.
[15] Wie beispielsweise die Genexpressionsprozesse, die in Abschnitt 8.2 beschrieben sind.
[16] z. B. wäre das bei Java über die Compare-Funktion möglich.

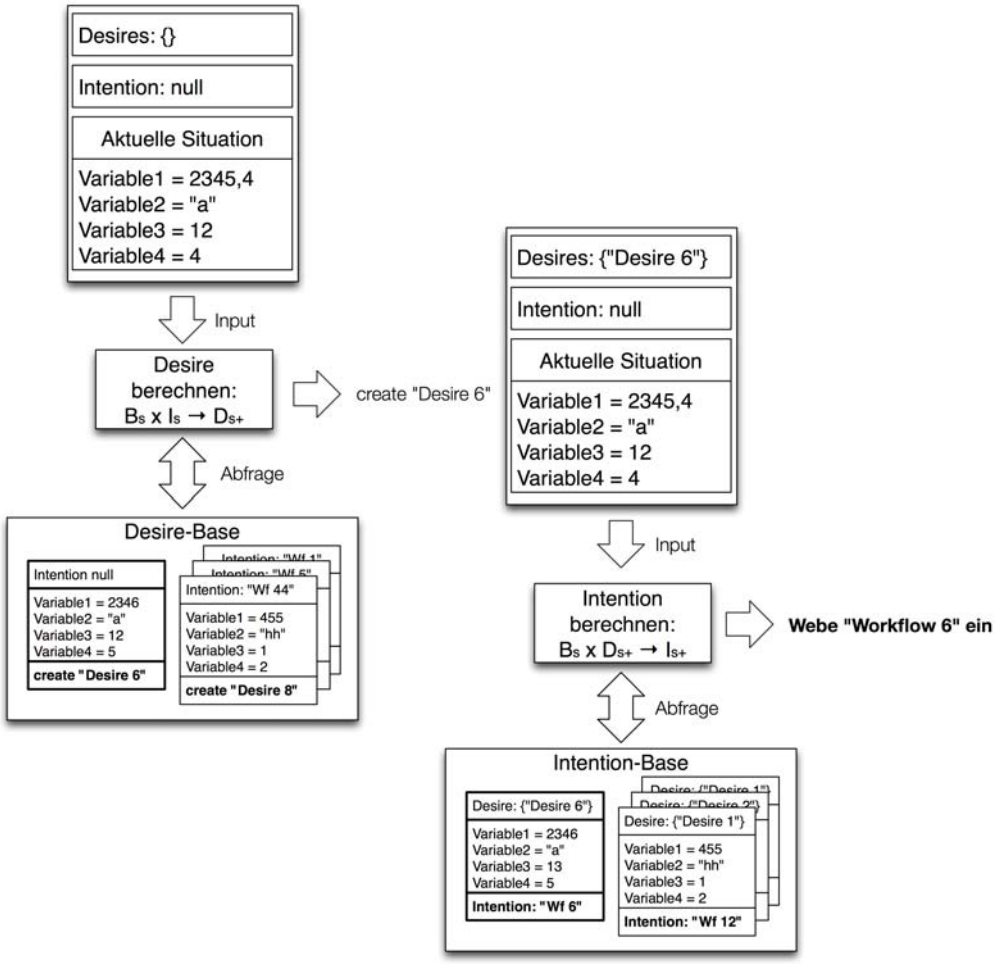


Abbildung 5.9: Adaptionsberechnung bei einem BDI-Direktor

Gibt es mehrere gleich gute Cases, wird durch eine Zufallsfunktion einer ausgewählt. Eine willkürliche Auswahl erscheint für diese Domäne nicht geeignet, spiegelt allerdings die Eigenschaften von BDI und CBR-Techniken wieder. BDI versucht das menschliche Verhalten, das ebenso willkürlich und unberechenbar Formen der Entscheidungen annehmen kann, zu imitieren. Ein Algorithmus, der immer die gleiche Antwort findet, wäre zu präzise. Des Weiteren sind die Retrieval-Ergebnisse von CBR nicht unbedingt optimal. Abhängig von der Qualität der Falldatenbanken, ist es unwahrscheinlich, dass in jeder Situation das optimale Ergebnis mit dem höchsten Outcome gewählt wird.

Beispielhafter Ablauf der Adaptionsberechnung Der vollständige Ablauf der Adaptionsfindung stellt sich wie in Abbildung 5.9[17] dargestellt dar. Im ersten Schritt wird die Gesamtsituation, bestehend aus Desires, Intention und der aktuellen Variablensituation, zur Berechnung der neuen Desires herangezogen. Dazu werden Cases in der Desire-Base gesucht, deren Variablensituation und die Intention der aktuellen Gesamtsituation (ausgenommen die Desires) ähnlich sind. Dies ist in Abbildung 5.9 der linke Case. Der untere, dritte Abschnitt des Desire-Cases ist die Anweisung, ob Desires erzeugt oder verworfen

[17] Die Abbildung stellt nur die essentiellen Fakten dar. Auf Details wurde aufgrund der Übersichtlichkeit verzichtet.

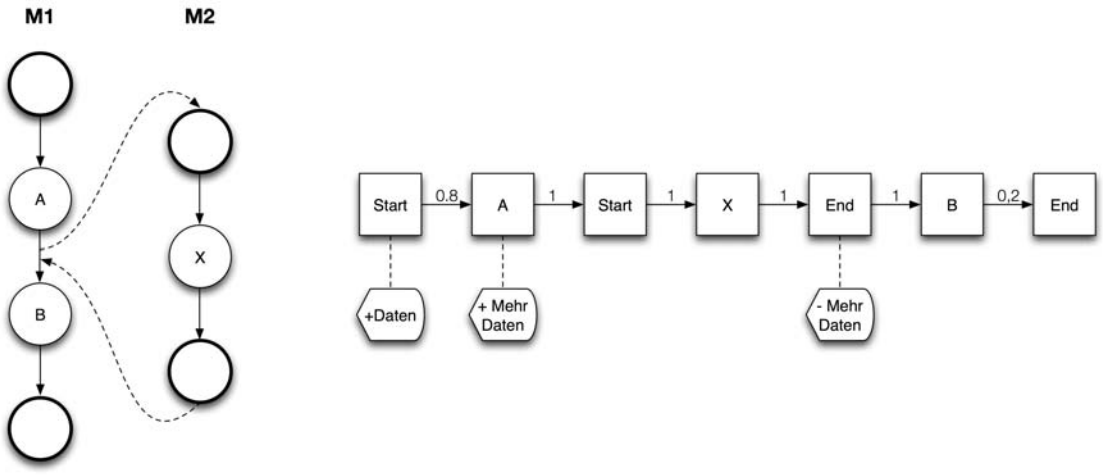


Abbildung 5.10: Trace-Modell einer Adaption

werden sollen. Im Beispiel wird das `Desire 6` dem Direktor hinzugefügt. Im zweiten Schritt wird mit der Ähnlichkeitssuche auf der Intention-Base (Ähnlichkeitsvergleich Desires und Variablensituation) die neue Intention ermittelt[18]. Am Ende der Gesamtberechnung steht das Ergebnis des Einwebens von `Workflow 6`.

5.2.3 Einsatz von Expertenwissen

Bevor die beiden Falldatenbanken genutzt werden können, muss eine angemessene Zahl an Fällen bereitstehen. Wie bereits in Abschnitt 5.1.2 diskutiert, ist die Menge der Fälle wichtig, so dass für jede vorkommende Situation zumindest ein ähnlicher Fall abgerufen werden kann. Die Qualität des Retrieval verbessert sich zunehmend, wenn mehr als ein adäquater ähnlicher Fall vorhanden ist. Es gilt zu vermeiden, dass der Direktor in eine Situation kommt, in der er keinen Vorschlag geben kann.

Eine einfache Möglichkeit für den Direktor neue Fälle zu erzeugen ist das Try&Error-Prinzip. Hierbei generiert der Direktor zufällig Adaptionen und trägt diese anschließend zuzüglich des Ergebnisses. Dieses Verfahren führt jedoch zwangsläufig zu erheblich mehr negativen als positiven Ergebnissen und ist deshalb keine sinnvolle Lösung.

Um diese großen Mengen adäquater Fällen in effizienter Weise zu erhalten, bietet es sich an, sie durch Traces zu gewinnen. *Traces* (Spuren) können als spezialisierte Log-Daten verstanden werden. Sie sind ein Protokoll der Ausführung eines Workflows, der von einem Domänenexperten durchgeführt wurde und manuell in Ausnahmesituationen adaptiert wurde. Es zeigt eine lineare Ordnung der Aktionen, die zu einem oder mehreren Workflows gehören und speichert alle Daten für die Situationsbeschreibung, die Desires und die Adaptionen. Abbildung 5.10 zeigt ein Beispiel[19] eines Traces von Workflow M_1 in den M_2 eingewoben wurde. Die Schritte der Traces repräsentieren entweder Aktionen, die stattgefunden haben oder zeigen, wann der Workflows explizit adaptiert wurde.

[18] Auch hier ist der linke Case in Abbildung 5.9 der ähnlichste.

[19] Der Experte bewertet jeden Übergang einzeln. In Abbildung 5.10 wird bspw. die Adaption mit 1, während das reguläre Ende (Übergang von B zu End) mit $0,2$ bewertet wird. Zudem kann der Experte die Veränderungen von Desires annotieren ($+Daten$ oder $+MehrDaten$).

Jeder einzelne Schritt eines solchen Traces kann als eine Entscheidung der Experten interpretiert werden. Wenn ein anderer Workflow eingewoben wurde handelt es sich um eine Adaption, wenn nicht, wird die Ausführung des Workflows in der vorgegebenen Weise fortgesetzt (Continue). Zudem kann ein Abort abgebildet werden. Durch die Analyse des Traces kann die Reihenfolge der Maßnahmen abgelesen werden. Das Ziel ist, dass ein unabhängiger Direktor diese Adaptionen in der gleichen Situation durchführt und zur selben Ablaufreihenfolge kommt[20].

Obwohl diese Daten direkt während der Ausführung aufgezeichnet werden, fehlen wichtige Informationen, die bei einer Ausführung nicht mit aufgezeichnet werden können. Dies betrifft vor allem die Desires, die kein natürliches Konzept von WfMS sind. Es ist demnach nicht möglich, die Entwicklung der Desires zu verfolgen. Sie müssen über einen anderen Weg in den Trace gelangen, als nur über die Ausführung eines Workflows. Eine Möglichkeit sind Anmerkungen von Experten. Für jeden Knoten des Workflows kommentiert der Experte den Trace, wenn ein Desire aktiviert oder deaktiviert wurde. Dies kann sowohl während der Ausführung als auch im Nachhinein geschehen. Dies bezieht sich auch auf die Bewertung der Adaption, die durch den Experten dem Trace hinzugefügt werden muss. Sie ist eine individuelle Entscheidung und wird durch den Wert von 0 bis 1 ausgedrückt. Das Ergebnis drückt dabei den Nutzen einer Intention aus. Dies ist dabei eine höchst subjektive Einschätzung, die eine entscheidende Bedeutung für den gesamten Prozess hat. Das Expertenwissen ist aber in diesem Fall die einzige Quelle, die ein Ergebnis bestimmen kann.

5.3 Zusammenfassung

In diesem Kapitel wurde ein neuartiger autonomer Steuerungsmechanismus für die Adaption durch Verweben vorgestellt. Dafür wurde auf Erkenntnisse aus der Forschung zu künstlicher Intelligenz zurückgegriffen. Um eine Entscheidung für eine Adaption zu treffen, wird erstmalig der BDI-Ansatz [126, 125] verwendet. Er nutzt zur Nachahmung menschlicher Entscheidungen drei Konzepte: Belief, Desire und Intention. Die Beliefs bilden das Wissen über den Workflow und seine Umgebung ab. Dieses Wissen repräsentieren die Variablen des Workflows. Desires sind Ziele, welche die Workflowinstanz verfolgt. Intentions repräsentieren die Absichten, die mit den Zielen verfolgt werden. Über den aktuellen Zustand des Workflows (Beliefs) und die aktuellen Ziele lässt sich durch eine definierte Abbildung auf eine Adaption (Intention) schließen. Die Menge der Beliefs B^{S} enthält alle Variablen, Ausnahmen und Laufzeitinformationen der aktuellen Situation S. Die Desires D^{S} enthält alle aktuellen Ziele dieser Situation und der aktuelle Plan ist in Form des Workflowmodells in der Menge der Intentions I^{S} hinterlegt. Der BDI-Direktor kann mithilfe dieser Konzepte Adaptionen berechnen.

Dies erfolgt auf Grundlage folgender Abbildungen: B^{S} x $I^{S} \Rightarrow D^{S+}$ und B^{S} x $D^{S^{+}} \Rightarrow I^{\mathrm{S}+}$. Die erste Abbildung berechnet auf Basis der Beliefs und der aktuellen Intention die neuen Ziele für den Workflow. Diese und die aktuellen Beliefs werden anschließend zur Berechnung der neuen Intention (Adaption) benutzt. Beide Abbildungen werden in einer separaten Datenbank, basierend auf *Case-based Reasoning*, gespeichert. Dadurch ist es möglich, dass der Situationsraum nicht vollständig beschrieben werden muss, wie es beispielsweise beim Statechart-Direktor der Fall ist. Der BDI-Direktor kann durch

[20] Vorausgesetzt das Ergebnis der Adaption war positiv.

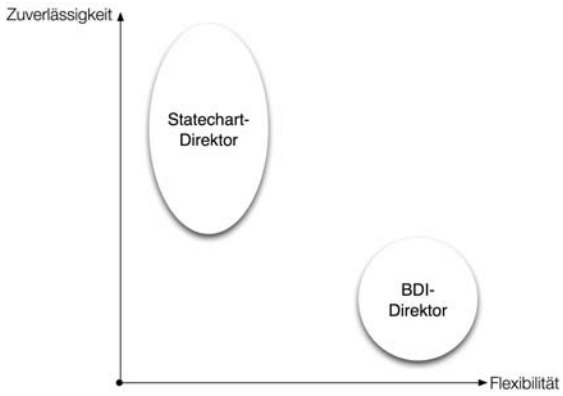


Abbildung 5.11: Vergleich von Zuverlässigkeit von Statechart- und BDI-Direktor

diese *ungenaue Wissensbasis* besser auf unerwartete Situationen reagieren. Das Wissen für diese Datenbanken kann durch Experten zusammengetragen oder aus den Daten von Traces extrahiert werden.

Wird der Direktor nach einer Adaption gefragt, vergleicht dieser die aktuelle Workflowsituation mit den bereits gelernten Situationen in den Datenbanken. Der Vergleich wird auf Basis einer Ähnlichkeitsfunktion durchgeführt. Essentiell für die Selbsteinschätzung ist die Rückkopplung, ob eine Adaption erfolgreich war. Diese erfolgt in Form eines Werts im Bereich [0,1], der angibt, ob die Adaption einen Erfolg oder Misserfolg darstellt. Dieser wird in den Datenbanken gespeichert, um in späteren Berechnungen diese Erfahrung zu berücksichtigen.

Der semantikfreie BDI-Direktor bietet, im Gegensatz zum Statechart-Direktor, eine hohe Flexibilität. Diese betrifft nicht nur die Adaption von Workflows, sondern auch die Veränderung der Wissensbasis selbst. Die Ähnlichkeitsfunktion ermöglicht es, dass Adaptionen effizient in einem großen Lösungsraum gefunden werden. Jedoch kann durch die Ähnlichkeitsfunktion nicht sichergestellt werden, dass die beste Lösung gefunden werden. Zudem kann der BDI-Direktor aufgrund der unzureichenden Ähnlichkeitsfunktion für Zeichenketten nur für *Production Workflows*. Im Vergleich zum Statechart-Direktor deckt der BDI-Direktor einen anderen Bereich des Spannungsfeldes zwischen Zuverlässigkeit und Flexibilität, wie in Abbildung 5.11 dargestellt, ab.

Kapitel 6

Kooperative autonome Rekonfiguration

Der im letzten Kapitel vorgestellte BDI-Direktor ermöglicht eine autonome domänenunabhängige Steuerung der Adaption durch Verweben. Der BDI-Ansatz wird durch Case-based Reasoning umgesetzt, d. h. Entscheidungen beruhen auf Grundlage ähnlicher Erfahrungen. Damit der Direktor in verschiedenen Domänen eingesetzt werden kann, muss auf explizite Semantik verzichtet werden. Deshalb basiert das Case-based Reasoning auf Ähnlichkeitsfunktionen für die Attribute einer Situation. Diese lässt jedoch keine Aussage über die tatsächliche inhaltliche Bedeutung einzelner Entscheidungselemente zu. Der BDI-Direktor weiß dadurch z. B. nichts über die Bedeutung einer bestimmten Zeichenkette. Der BDI-Direktor wird deshalb bei Synonymen keine Ähnlichkeit feststellen.

Der Verzicht auf explizite Semantik birgt allerdings einige Probleme. In einem Abrechnungsprozess für einen Bestellvorgang wird beispielsweise die Kreditkartennummer in der Variable *creditNumber* gespeichert. In einem einzuwebenden Workflow ist die entsprechende Variable als *number* bezeichnet. Beim Verweben werden die Variablen in die adaptierte Instanz übertragen. Die Übertragung von einer Variable auf die andere kann in diesem Fall nicht funktionieren. Ein Vergleich auf Basis der Editierdistanz schlägt fehl. Ohne Semantik ist nicht festzustellen, dass diese beiden Variablen aufeinander abgebildet werden können bzw. das gleiche bedeuten. Beim selektiven Weben führt dies dazu, dass Datenabhänigkeiten in diesem Fall nicht erfüllt werden können.

Um diese Probleme zu lösen, soll der BDI-Direktor um explizite Semantik erweitert werden. Dadurch die Erkennung von Synonymen kann der BDI-Ansatz nicht nur *Production Workflows* sondern auch für die „textlastigen“ *Administrative Workflows* umgesetzt werden. Zudem erlaubt diese Erweiterung zwei entscheidende Neuerungen: (1) die *Kooperation* in *verteilten Umgebungen* und (2) die *Redundanzvermeidung* als Anwendung der Kooperation.

Kooperation in verteilten Umgebungen Die fehlende Semantik des BDI-Direktors verhindert zudem, dass einzelne Einträge der Falldatenbank von verschiedenen, verteilten Direktoren verglichen werden können. Dies ist vor allem in großen Unternehmen kritisch. Um die Skalierbarkeit eines Workflow Management Systems in großen Unternehmen sicher zu stellen, müssen verschiedene WfMS-Instanzen auf verschiedenen Rechnerknoten ausgeführt werden. Dies führt dazu, dass die gleichen Workflowmodelle auf verschiedene Server verteilt werden. Dies betrifft auch die Direktoren, die ihre Falldatenbanken ebenfalls verteilt erstellen. Die verteilten Direktoren lernen im Verlauf ihrer Lebenszeit unterschiedliche Erfahrungen. Dies führt in Summe zu unterschiedlichen Falldatenbanken. Durch die fehlende Semantik wird verhindert, dass die einzelnen Fälle zwischen verschiedenen Direktoren vergleichbar sind. Eine *Kooperation* der Direktoren, um das gesammelte Wissen zu teilen, ist somit nicht möglich. Mit der Einführung von Semantik

in den BDI-Direktor können Direktoren nicht nur ihr Wissen gegenseitig austauschen, sondern auch anderen bei Adaptionsanfragen helfen. Wenn ein Direktor keine Adaption vorschlagen kann und *Continue* empfehlen würde, kann er bei anderen nachfragen, ob sie für die aktuelle Situation eine geeignete Adaption kennen. Wird diese gefunden, kann sie angewandt werden. Diese Kooperation beschränkt sich dabei nicht nur auf die Direktoren einer Organisation, sondern kann auch zwischen verschiedenen Organisationen stattfinden.

Vermeidung von Redundanzen durch Kooperation In Workflow Management Systemen wird nicht nur eine Instanz eines Workflows zur gleichen Zeit ausgeführt, sondern es existieren viele Instanzen der unterschiedlichsten Workflowmodelle. Es kann demnach auch vorkommen, dass zur gleichen Zeit zwei Instanzen des gleichen Workflows ausgeführt werden. Sind die Instanzen mit gleichen Parametern gestartet worden, werden sie *redundant* ausgeführt. Das bedeutet, dass zum gleichen Zeitpunkt oftmals die gleichen Aktivitäten in verschiedenen Instanzen ausgeführt werden. Dies ist beispielsweise bei wissenschaftlichen Workflows der Fall. So werden von verschiedenen Wissenschaftlern unterschiedliche Workflows für Genexpressionsprozesse mit gleichen Datensätze ausgeführt. Dies führt zu unnötigen mehrfachen Berechnungen, die vor allem bei Genexpressionsprozessen sehr viele Ressourcen benötigen. Würden diese redundanten Berechnungen nur einmal durchgeführt und die Ergebnisse mit den anderen Instanzen geteilt, könnte Zeit und Ressourcen gespart werden.

Das bedeutet, dass auch kleine Abschnitte eines Workflows redundant ausgeführt werden. Diese Redundanz könnte vermieden werden, wenn ein Direktor alle Ausführungen überwacht, Gleichheiten bei Instanzen feststellt und eine spezielle Adaption anweisen kann. Bei den bisher vorgestellten Adaptionen wurde das Token in eine neue Instanz des einzuwebenden Workflowmodells migriert. Es ist aber ebenso denkbar, dass es in eine bestehende Instanz integriert wird. Dies geschieht mithilfe einer selektiven seriellen Adaption. Dabei wird das Token T_1 der Instanz I_1 auf die Position von T_2 in I_2 adaptiert. Die Datenabhängigkeiten können hierbei vernachlässigt werden, denn T_2 hat alle erforderlichen Variablen für die Position, auf der er sich befindet.

Von diesen Daten kann T_1 profitieren und diese direkt übernehmen. Dadurch wird eine erneute Berechnung von Daten vermieden, die bereits vorhanden sind. In Abschnitt 3.1.1 wird beschrieben, dass bei einer beendeten parallelen Adaption die Variablen beider Token zusammengeführt werden. Im hier umgekehrten Fall, der Migration in eine bestehende Instanz, können die Variablen des migrierten Tokens übernommen werden. Eine semantische Beschreibung der einzelnen Aktivitäten und Variablen hilft dabei, dass auch nicht gleichnamige Variablen aufeinander abgebildet werden können. Ist eine Abbildung der über Datenabhängigkeiten notwendigen Variablen möglich, kann eine *Migration* erfolgen. Das Token verlässt die Instanz am geplanten Rücksprungpunkt mitsamt den gewonnen Daten oder beendet den Workflow in seiner neuen Instanz, je nachdem, wie die Adaption definiert wurde. Für die Workflow-Engine bedeutet das, dass zur gleichen Zeit zwei Token in einer Instanz vorhanden sind. Standard ist die Unterstützung eines einzelnen Token pro Instanz. Die explizite Ausführung zweier Token ist gleichwohl nicht notwendig. Die Migration T_1 wird zur Vermeidung von Redundanz durchgeführt, d. h. es soll von den Ergebnissen in I_2 profitieren. Dementsprechend kann die Migration auch umgesetzt werden. T_1 wird für die Zeit der Migration ein Teiltoken von T_2, so dass lediglich ein Token durch den Workflow steuert. T_1 löst dementsprechend auch in diesem

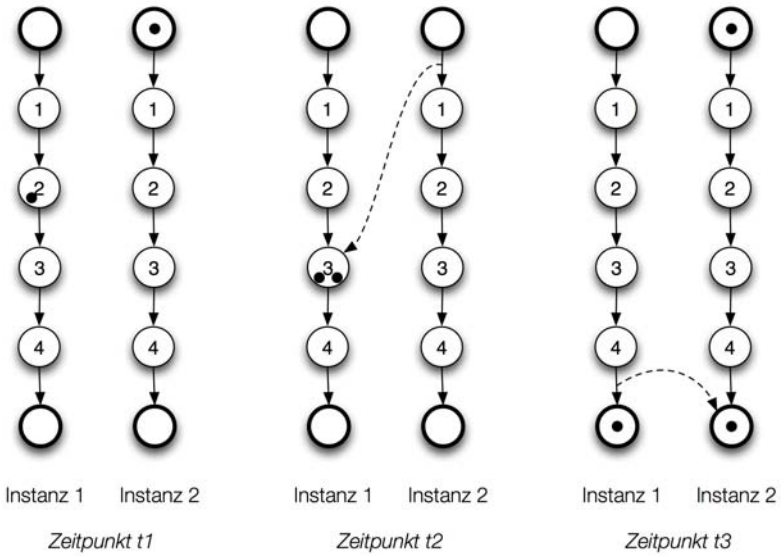


Abbildung 6.1: Kommunikation von zwei Instanzen

Zeitraum keine Aktionen aus. Würde er dies tun, würde die Aktion wiederum doppelt ausgeführt werden. Dieses Vorgehen ermöglicht eine einfache Umsetzung.

Abbildung 6.1 stellt ein einfaches Szenario dar. Hierbei werden zwei Instanzen des gleichen Modells mit gleichen Parametern – redundant – ausgeführt[1]. Workflow Instanz I_1 wurde einige Zeit vor I_2 gestartet (Zeitpunkt t_1). In diesem Fall kann das Token von Instanz I_2, wie bei Zeitpunkt t_2 dargestellt, in I_1 überführt werden. Diese (temporäre) Vereinigung von Workflows wird *Migration* oder *Merge* genannt. Nach Beendigung der Aktivität 4, wird zum Zeitpunkt t_3, das Token wieder in die Ausgangsinstanz überführt. Es wird ein *Split* ausgeführt. Durch diese Adaption, wurde verhindert, dass die Aktivitäten 1 bis 4 doppelt ausgeführt wurden.

In einem verteilten Szenario kann nicht immer davon ausgegangen werden, dass die Workflowmodelle sich vollständig gleichen. Die Einführung von Semantik kann hierbei helfen, die Gleichheit von Aktivitäten oder einer Menge von Aktivitäten festzustellen, auch wenn die Struktur der Workflowmodelle grundsätzlich unterschiedlich ist. In Abbildung 6.2[2] sind zwei Workflowmodelle dargestellt, die unterschiedliche Aufgaben repräsentieren. Die Abfolge der Aktivitäten von 1 und 3 sind nicht nur strukturell gleich, sondern auch semantisch äquivalent. Eine Kooperation ist für diese beiden unterschiedlichen Modelle demnach in Teilen möglich.

In den folgenden Abschnitten wird gezeigt, wie der Ansatz des BDI-Direktors durch die Einführung von Semantik erweitert werden kann. Dieser soll auf Grundlage der semantischen Beschreibung der Workflowmodelle die Kooperation zwischen Direktoren ermöglichen und durch die Erkennung von Redundanzen die Performance des WfMS erhöhen. Dazu werden im Folgenden grundlegende Technologien und verwandte Arbeiten vorgestellt, die für die Kooperation und Redundanzerkennung notwendig sind. Anschließend wird ein Kommunikationsprotokoll für Direktoren vorgestellt.

[1] Es wird von deterministischen Berechnungen ausgegangen.
[2] Die unterschiedlichen Formen der Aktivitäten symbolisieren ihre unterschiedliche Semantik.

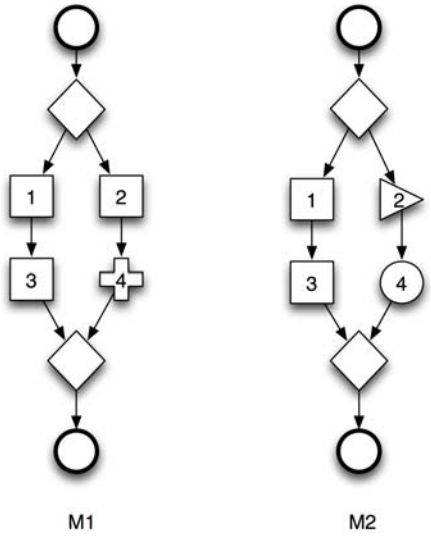


Abbildung 6.2: Partielle semantische Äquivalenz

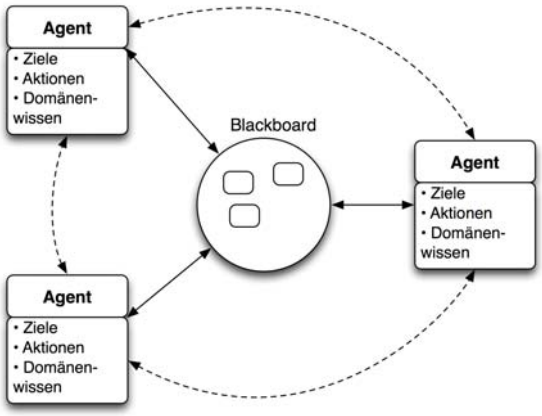


Abbildung 6.3: Kooperative Agentensysteme (nach [164])

6.1 Grundlagen der Kooperation

Wie bereits in Kapitel 5 beschrieben, gibt es viele Übereinstimmungen zwischen der Steuerung der Adaption durch Verweben und Agentensystemen. Das BDI Modell ist dementsprechend auf den Direktor übertragbar. Sollen nun die Direktoren miteinander kooperieren, können ebenfalls Anleihen bei Agenten gefunden werden. Eine Spezialform der Agenten-Systeme stellen die Multi-Agenten-Systeme (MAS) [58] dar, d. h. sie verbinden mehrere (kooperative) Agenten. Dies wurden speziell dazu entwickelt, um kooperativ zu lernen, d. h. dass sich diese Agenten in verschiedenen Situationen gegenseitig unterstützen können. Sie können, wie in Abbildung 6.3 dargestellt, direkt oder über eine Blackboard-Architektur untereinander interagieren.

Ein einzelner Agent (siehe Abschnitt 5.1) ist eine autonom agierende Softwarekomponente, die auf Änderungen reagieren kann. Ein Multi-Agenten-System erweitert diesen

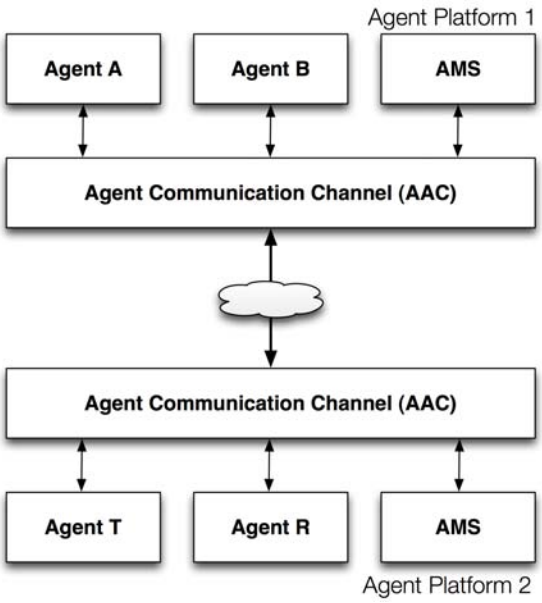


Abbildung 6.4: FIPA Referenzmodell

Ansatz um mehrere untereinander agierende Agenten. Multi-Agenten-Systeme können in verschiedenen Anwendungsgebieten eingesetzt werden. Sie können, wie in [38] beschrieben, zur Koordination und Automatisierung von Arbeitsaufgaben verwendet werden. [96] sieht ein Multi-Agenten-System als ein Kollektiv von Agenten, die auf zweierlei Wegen umgesetzt werden können: zum einen durch einen auf Logik basierenden Ansatz, wie er beispielsweise in [79, 88] vorgestellt wird, zum anderen durch eine Schichtenarchitektur, wie in Abbildung 5.1 dargestellt. Wie bei Single-Agent-Systemen kann auch, wie in [34] beschrieben, in MAS der BDI-Ansatz verwendet werden. Der Ansatz, nach dem Pläne durch Ereignisse ausgelöst werden, wird hier durch ein Situation-based BDI-Modell, SBBDI-Modell, abgelöst. Die Agenten besitzen ein Gedächtnis, so dass sie mehrere Ereignisse erfassen und korrelieren können. Dadurch wird eine Analyse des operationalen Kontextes der Umwelt ermöglicht. In [159, 158] wird dagegen das BDI-Modell indirekt erweitert, durch Nutzung anderer Algorithmen zur Entscheidungsfindung bzw. durch die Einbeziehung der Kommunikation. Beide Ansätze stellen ein theoretisch-mathematisches Modell dar.

Bei MAS verlagert sich nach [57] die Komplexität, weg von großen, schwer wartbaren Agenten, hin zu kleinen Agenten, deren Kommunikation komplexer wird. Durch die Kooperation ist es möglich, Agenten nur mit einem begrenzten Umfang an Funktionalität auszustatten. Fehlende Funktionen können durch Kooperation mit Agenten ausgeglichen werden, die über die entsprechende Ausstattung verfügen. Dazu müssen Multi-Agenten-Systeme drei essentielle Fähigkeiten unterstützen: *Koordination*, *Kooperation* und *gemeinsames Lernen*. Für alle drei Problemkomplexe wurden bei den MAS Lösungen entwickelt, die im Folgenden vorgestellt werden.

6.1.1 Kommunikation und Kooperation

Die Basis von Multi-Agenten-Systeme stellt die direkte oder indirekte *Kommunikation* dar. Ohne diese ist eine *Kooperation* nicht möglich. Dazu müssen die Agenten sowohl ihre

Umwelt erfassen können, als auch eine einheitliche Beschreibungssprache besitzen.

Die Organisation FIPA – Foundations of Intelligent Physical Agents – erarbeitete 2002 einen Standard [59] für die Kommunikation zwischen verschiedenen Agenten. Dabei gingen die Autoren von einer Peer-to-Peer-Kommunikation aus. Die Referenzarchitektur, dargestellt in Abbildung 6.4, sieht verschiedene *Agentenplattformen* vor, die untereinander kommunizieren. Diese Plattformen stellen die Laufzeitumgebung – *Agent Management System* (AMS) – für die Agenten dar. Die Kommunikation wird durch den *Agent Communication Channel* (ACC) durchgeführt, welcher für den Austausch von Nachrichten zuständig ist.

Der FIPA-Standard beschreibt zudem den Aufbau der Nachrichten in der *Agent Communication Language* (ACL) [60]. Die wichtigsten Bestandteile einer ACL-Nachricht sind in Listing 6.1 aufgezeigt. Durch den Aufbau einer ACL-Nachricht wird festgelegt, welches Anliegen welcher Agent mit einem anderen austauschen will. Wie für Peer-to-Peer-Kommunikation notwendig, ist es durch die IDs möglich, asynchronen Nachrichtentransfer durchzuführen.

Listing 6.1: Aufbau einer ACL-Nachricht

```
1  sender          Absendender Agent (ID)
2  receiver        Zielagent der Nachricht
3  content         Nachrichteninhalt
4  performative    Zweck der Nachricht
5    ASK           Anfrage eines Agenten
6    PROPOSE       Angebot eines Agenten
7    REFUSE        Agent kann kein Angebot unterbreiten
8    ACCEPT        Agent nimmt Angebot an
9    INFORM:       Informationsaustausch ohne Rückkanal
10   FAILURE:      Anfrage kann nicht bedient werden.
```

Die unterschiedlichen Typen für `performative` sind dabei nicht expliziter Bestandteil der Spezifikation und können in jeder Implementierung anders definiert werden. Des Weiteren sieht die FIPA-Spezifikation eine Kommunikation auf Basis von Peer-to-Peer-Nachrichten vor. Es ist ebenfalls möglich, die Kommunikation über ein Blackboard-System abzuwickeln. Dieses schwarze Brett ist dabei die zentrale Austauschplattform für Nachrichten. Dazu erhalten die Agenten das Recht, Nachrichten zu lesen und zu schreiben. Sie kommunizieren nicht mehr direkt miteinander, sondern müssen immer auf Veränderungen achten. Der Aufbau der Nachrichten kann bei dieser Architekturform übernommen werden.

Die *Koordination* stellt einen zweiten, wesentlichen Teil der Kooperation in MAS dar. Diese kann wie die Kommunikation *zentral* oder *dezentral* erfolgen. Bei der zentralen Koordination sammelt ein Agent alle Informationen und Angebote bzw. Nachfrage und berechnet gleichzeitig die Lösung. Das Konzept der autonomen Agenten wird bei der dezentralen Koordination benötigt. Hierbei gibt es Kommunikationsbeziehungen zwischen zwei Agenten, wobei ein Agent mehrere bilaterale Beziehungen führen kann. Beide Koordinationsformen können sowohl über die Peer-to-Peer als auch über die Blackboard-Kommunikation realisiert werden.

6.1.2 Gemeinsames Lernen

Ein weiterer zentraler Aspekt bei der Realisierung von Kooperation in MAS ist das *gemeinsame Lernen*. Agenten in MAS können dabei auf drei verschiedene Arten lernen.

Zum einen kann der Agent zur Laufzeit vom Wissen anderer Agenten partizipieren (siehe *INFORM* in Listing 6.1) oder verschiedene Agenten versuchen kooperativ eine Lösung zu finden, die entsprechend in die Wissensbasis aufgenommen wird. Die dritte Möglichkeit besteht darin, sich wie ein einzelner Agent zu verhalten, so dass das Lernen isoliert stattfindet. Stone beschreibt in [164] verschiedene Algorithmen, die das Lernen zwischen Agenten ermöglichen. Es ist zu beachten, dass verschiedene Realisierungen auch negative Eigenschaften aufweisen. Vor allem darf durch die Kooperation und das gemeinsame Lernen das Grundverhalten eines Agenten nicht nachhaltig beeinflusst werden.

Dabei gibt es zwei Aspekte des Lernens, die unterschieden werden müssen. Zum einen ist der Zeitpunkt entscheidend, wann gelernt wird. Wie in Abschnitt 5.1 beschrieben wird in [9] zwischen *Online* und *Offline Lernen* unterschieden. Nach [9] ist für Multi-Agenten-Systeme das Online Lernen geeigneter, denn neue Eindrücke werden unverzüglich verarbeitet. Eine Möglichkeit, über den Zeitpunkt der Hypothesengenerierung beim Offline Lernen zu entscheiden, bietet eine Schranke. Waren beispielsweise 50% aller Entscheidungen falsch, wird eine Neugenerierung gestartet. Neben dem Zeitpunkt der Hypothesengenerierung kann auch die grundlegende Art des Lernens unterschieden werden. In der Literatur finden sich dafür zwei Arten, *logik-basiertes* und *Feedback* bzw. *Reinforcement Learning* [166]. Hierbei wird durch die Interaktion mit anderen Agenten gelernt. Dabei werden zielführende Aktionen belohnt, andere bestraft. Basis bilden numerische Funktionen, wobei zwischen einer Belohnungsfunktion und einer Nutzenfunktion unterschieden wird.

Bei kooperativen Agenten ergeben sich unabhängig von der Lernmethode verschiedene Kooperationsszenarien, die berücksichtigt werden müssen:

1. *Propagierung eines Plans:* Agent A besitzt keinen Plan zur Erreichung seines Zieles, weshalb er andere Agenten um Hilfe bittet[3]. Ein Agent B besitzt einen Plan für dieses Ziel, nach dem A fragt. Dieser Plan muss an A übermittelt werden.
2. *Erstellung eines Plans:* Agent A hat keinen Plan und muss eine Lösungshypothese aufstellen, wobei er auch das Hintergrundwissen von Agent B nutzen kann.
3. *Erzeugung von Hintergrundwissen:* Agent A besitzt keinen Plan und fragt Agent B um neues Hintergrundwissen zu akquirieren. Daraus kann anschließend eine Lösungshypothese generiert werden.
4. *Austausch von Hintergrundwissen:* Beide Agenten tauschen ihr Hintergrundwissen aus.

Im [88] werden zwei weitere Szenarien genannt, bei denen eine Kooperation notwendig ist:

1. Ein Agent kann den Lernprozess nicht starten, da er nicht genug Beispiele besitzt.
2. Ein Agent kann nicht erschließen, warum ein Plan fehlgeschlagen ist. Er fragt bei anderen Agenten nach Gründen.

Um über verschiedene Anfragen und Lösungen Aussagen zu treffen, ist es notwendig, auf einer einheitlichen Wissensbeschreibung sowie einem gleichen Konzeptverständnis zu arbeiten. In [107] wird ein Konzept vorgestellt, bei dem Agenten gemeinsam eine Ontologie lernen. Jeder Agent besitzt hierbei Wissen in Form von Experience Cases, denen jeweils eine Liste mit Wörtern zugeordnet wird, ähnlich Keywords oder Tags. Ein

[3] Dabei ist es irrelevant, ob dies über ein Peer-to-Peer-Netzwerk oder über ein Blackboard geschieht.

Kombination der Lernphasen	Erläuterung
verteilte Datenakquise aber individuelle Problemlösung und Lernen	Kooperation bei der Datenakquise ermöglicht eine umfassenderen Sicht auf die Umwelt
individuelles Lernen und verteilte Problemlösung	Verteilung der Problemlösung auf verschiedene Agenten und somit Wissensbasen
individuelles Lernen und individuelle Problemlösung und Wissensintegration	Austausch des Wissens nach individuellem Lernen
verteiltes Lernen und individuelle Problemlösung	Lernen für verschiedene Wissensbasen. Jeder Agent versucht das Problem anschließend separat zu lösen

Tabelle 6.1: Kooperationsszenarien (nach [28])

Experience Case, d. h. die Keywords, wird als numerischer Vektor repräsentiert. Dadurch ist es möglich, semantische Regeln numerisch zu definieren und zu vergleichen.

Diese Art des Lernens wird als Conceptual Learning bezeichnet. Die Konzeptnamen können sich zwischen den Agenten unterscheiden, während sich die zugehörigen Beschreibungsvektoren gleichen. Dadurch ist eine Abbildung zwischen den Wissensbasen mehrerer Agenten möglich. Beim Conceptual Learning wird der Lernprozess in drei unterschiedliche Phasen eingeteilt: die Datensammlung, eine Lern- und Problemlösungsphase. Alle drei Phasen können sowohl individuell als auch gemeinsam durchgeführt werden. In Tabelle 6.1 werden mögliche Szenarien der Kooperation und die daraus folgenden Implikationen aufgezeigt.

Für kooperative Direktoren ist es deshalb notwendig, drei essentielle Konzepte umzusetzen: Kommunikation, koordiniertes Lernen und eine gemeinsame Wissensrepräsentation inkl. der Möglichkeit, Konzepte aufeinander abzubilden.

6.1.3 Grundlagen der Redundanzerkennung

Die Redundanzerkennung ist ein wichtiger Baustein für den kooperativen Direktor. Werden viele Instanzen in einem WfMS ausgeführt, kommt es häufig vor, dass Instanzen die gleichen Aktionen bzw. Aktionsketten ausführen. Diese redundante Ausführung ist keineswegs effizient. Durch die Kooperation zwischen den Direktoren ist es möglich, diese redundanten Situationen zu erkennen. Dies bedeutet, dass die Kommunikation zwischen den Direktoren sich nicht nur um den Austausch von Wissen, bzw. das gemeinsame Lernen drehen kann, sondern auch dem Austausch über Laufzeitdaten dient. Mit diesen Informationen ist es ihnen möglich, geeignete Adaptionen zu berechnen, die die verschiedenen Instanzen virtuell miteinander verschmelzen. Dazu ist es notwendig, Redundanzen zu erkennen und geeignete (Rück-)Sprungpunkte zu bestimmen.

Eine Lösung zur Elimination von Redundanzen in Workflows findet sich in [151]. Die Autoren stellen einen Algorithmus vor zur Berechnung der *Longest Common Subsequence*[4] (LCS) zwischen zwei gerichteten azyklischen Graphen (DAG). Die LCS ist ein Problem,

[4] Längste gleiche Teilsequenz.

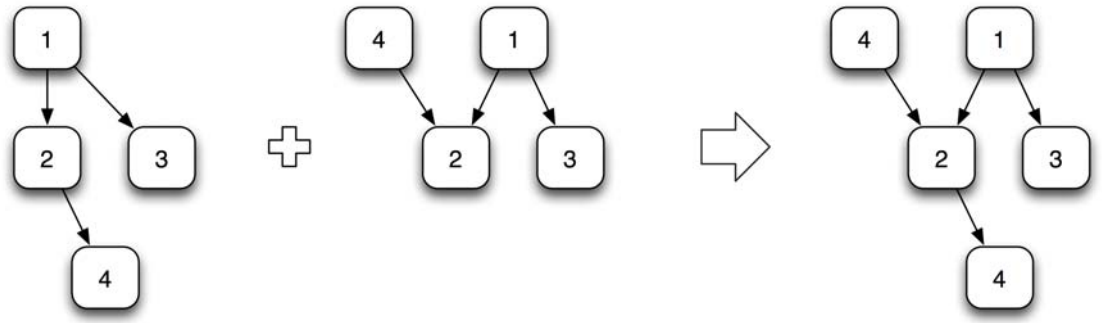


Abbildung 6.5: Vereinigung von DAGs (nach [151])

bei dem die längste gemeinsame Sequenz zwischen von zwei Strukturen gesucht wird. Beispielsweise ist „CBD“ die LCS von „ABCBAD“ und „CBDE“. Die Autoren überführen das Problem auf DAGs und zeigen, wie sie zwei DAGs zusammenführen können, um Redundanzen zu vermeiden, wie dies Abbildung 6.5 zeigt. Bei dieser Vereinigung wird ebenfalls die LCS berechnet. Diese Berechnung kann für Berechnung von redundanten Fragmenten zwischen zwei Workflows genutzt werden, wenn Zyklen (d. h. Schleifen) nicht betrachtet werden.

Weitere Ansätze zur Vermeidung von Redundanzen finden sich vor allem in der Programmcodeoptimierung beim Compilerbau [174]. Ziel der Optimierungen ist das Erkennen und Entfernen von redundanten Codefragmenten, Speicherzugriffen oder Prozeduraufrufen. Dies wurde im Laufe der Jahre immer notwendiger, denn es wurde festgestellt, dass mit steigendem Abstraktionsniveau einer (Programmier-)Sprache die Anzahl der unnötigen oder redundanten Codefragmente zunimmt. Dies trifft insbesondere auf objektorientierte Sprachen zu. Ein Ansatz zur Vermeidung von Redundanzen ist die statische Optimierung während der Übersetzung. Des Weiteren existiert die Möglichkeit der dynamischen Optimierung zur Laufzeit, welche allerdings einen hohen Ressourcenverbrauch aufweist. Für dynamische Workflows wäre diese Optimierung ideal. Zur Beschleunigung der Ausführung ist es möglich, die Modelle direkt vor der Ausführung zu analysieren, wie dies bei *Just in time* Compilern üblich ist oder es wird jeweils nur das aktuelle Teilstück eines Programms betrachtet. Im hier vorliegenden Fall sind es die Vorgänger bzw. Nachfolger des Token, wobei deren Anzahl variabel festgelegt werden kann. Wird Redundanz festgestellt, muss diese behoben werden. Die Optimierung umfasst dabei die Analyse selbst, das Auflösen der Redundanz und die Transformation in die jeweilige Zielsprache. Übertragen auf die vorliegende Domäne der Workflows bedeutet dies die Analyse der aktuellen Instanzen und als Lösung Adaptionsanweisungen an den Koordinator. Diese Anweisungen müssen sowohl für die Migration als auch für die Trennung erfolgen.

Der Fokus bei der Redundanzvermeidung liegt nicht in der Verminderung von redundanten Speicherzugriffen und Prozeduraufrufen, sondern darin, redundante Instanzfragmente nicht auszuführen. Im Gegensatz zu objektorientierten Programmen gibt es bei Workflowmodellen oft eine partielle Ordnung, d. h. die Modelle sind in gerichteten Graphen darstellbar, was sowohl die Analyse als auch die Optimierung vereinfachen kann. Tabelle 6.2 zeigt dazu einen Ausschnitt aus [174]. Dargestellt sind sowohl eine Auswahl von Compiler-Optimierungen als auch die Entsprechungen für Workflowinstanzen. Deutlich werden die Gemeinsamkeiten bei der Vermeidung von partiellen Redundanzen. Ein Compiler versucht dabei mehrfach vorkommende, identische Berechnungen zu finden.

Optimierung	benötigte Information	Entsprechnung im Workflowumfeld
statische Auswertung oder Vereinfachung von Ausdrücken	Abschätzung über die Werte der Operanden	keine, da die einzelnen Aktivitäten von Workflowmodellen als atomar angesehen werden
Entfernen toten oder nicht erreichbaren Codes	Kenntnis über Verwendung von Ausdrücken und über die Realisierbarkeit von Kontrollflusspfaden	keine, da die Workflowmodelle ungetastet bleiben
Vermeidung partieller Redundanzen	Menge der Variablen, die durch eine Anweisung gelesen oder geschrieben werden	Merge von Instanzen, um mehrfache Auführung in unterschiedlichen Instanzen zu vermeiden
Reduktion dynamischer Objekte	Lebenszeiten von Objekten	keine, da alle Instanzen Berechnungen durchführen, Ziel ist nur die Einsparung von Rechenzeit

Tabelle 6.2: Compileroptimierungen und Entsprechungen bei Workflows (Erweiterter Auszug aus [174])

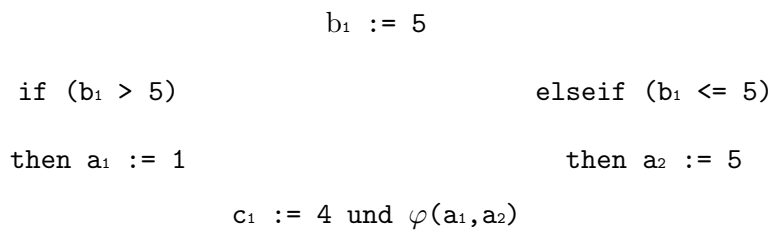
Dabei wird jedes Vorkommen durch einen Verweis auf eine zentrale Implementierung der Funktion ersetzt. Andere Optimierungsarten sind nicht zur Erkennung bzw. Vermeidung von Redundanzen ausgelegt.

Für die Redundanzerkennung ist eine geeignete Repräsentation erforderlich. Repräsentationen für Quellcode sind Graphen wie der Kontrollflussgraph [13] oder der Grundblockgraph [13]. Auch Workflowmodelle sind gerichtete Graphen[5], demzufolge ähnlich der Kontrollflussgraphen. Damit ist es prinzipiell möglich, bekannte Techniken aus dem Compilerbau zu adaptieren, wie z. B. das Static Single Assignment (SSA) [46]. SSA stellt eine spezielle Zwischenrepräsentation von Programmcode dar, wobei jede Variable nur einmal zugewiesen werden darf. Das bedeutet, dass eine Variable X nur einmal auf der linken Seite einer Zuweisung auftreten darf. Jede weitere Zuweisung bedeutet in SSA eine Versionierung der Variable, dargestellt durch einen kleinen Index-Zusatz. Ein kleines Beispiel soll dies verdeutlichen:

```
a := 1                          a₁ := 1
a := 2      wird transformiert zu   a₂ := 2
b := a                          b₁ := a₂
```

Dadurch entsteht zwischen den einzelnen Befehlen ein Netz von Datenabhängigkeiten, dass zur Optimierung der Programme genutzt werden kann. Die Versionierung kann aber auch zu Unentscheidbarkeiten führen. Angenommen, bei einer *If*-Verzweigung wird eine Variable a sowohl im *If*-Zweig gesetzt, als auch im *else*-Zweig. SSA wandelt die Variable a in zwei unterschiedliche Instanzen um: a_1 und a_2, wie im Folgenden dargestellt:

[5] Ausgehend von Petri-Netzen.



Nach Beendigung des *If* ist bei statischer Analyse nicht entscheidbar, welche Variante von a mit 4 addiert auf c übertragen wird. Zur Lösung dieser Probleme wurde die Φ-Funktion eingeführt. Sie bildet alle möglichen Versionen einer Variablen auf die entsprechende Zuweisung oder Nutzung ab.

Die Φ-Funktion wird immer dann gebraucht, wenn Dominanzgrenzen erreicht werden, wie im oberen Beispiel das bei der If-Verzweigung. Ein Knoten A kann B nur dann streng dominieren, wenn Knoten B im Kontrollflussgraphen nur über A erreicht werden kann. A dominiert demnach B, wenn entweder A B streng dominiert oder $A = B$ ist. Eine Dominanzgrenze wird dann erreicht, wenn A B nicht mehr streng dominiert, d. h. es gibt einen anderen Pfad, über den B erreicht werden kann. Zur Berechnung der Dominanzgrenzen gibt es verschiedene Algorithmen. Zudem variiert, abhängig vom Ziel der Optimierung, die Anzahl der Φ-Funktionen. Beispiele hierfür sind: Pruned SSA und Semi-Pruned SSA [30] sowie Location Forced SSA [174].

Eine weitere Eigenschaft der SSA-Darstellung ist die Semantikäquivalenz bei syntaktischer Gleichheit. Liegen zwei Programme in SSA-Darstellung vor, können sie verglichen werden, wobei die Versionierungen der Variablen eine gute Vergleichsbasis darstellen. Dies lässt sich auch auf Workflowmodelle übertragen. Um Gleichheit zweier Workflowmodell zu bestimmen können im Zuge der Initialisierung der Instanzen die vorhandenen Modelle in eine SSA-konforme Repräsentation transformiert werden. Ein Direktor kann für eine beliebige Situation bzw. Adaption andere Direktoren fragen, ob diese von ihnen gerade ausgeführt werden, wobei die beiden SSA-Darstellungen verglichen werden. Wird ein gleiches Workflowfragment in einer Instanz gefunden, die der Situation entspricht, kann ein *Merge* der Instanzen erfolgen. Es muss zudem der Rücksprungpunkt angegeben werden, der dem Ende des Workflowfragmentes entspricht.

6.1.4 Statische Semantik für Workflowmodelle

Workflowmodelle können sich aber auch unterscheiden, obwohl die beiden die gleichen Aufgaben beinhalten. Ein Beispiel: In Workflow A wird in der Aktivität A_2 ein Webservice aufgerufen, der sowohl die Kreditwürdigkeit einer Person prüft, als auch seine aktuelle Adresse aus der Datenbank lädt. Workflow B erfüllt die gleiche Funktion, allerdings wird in B_2 die Kreditwürdigkeit geprüft und in B_3 die Adresse geladen. Strukturell sind beide Modelle unterschiedlich. Es gilt jedoch: $A_2 \equiv B_2 + B_3$. Die Äquivalenz der Modelle kann nur durch Typisierung (statische Semantik) der Aktivitäten festgestellt werden.

Die Einführung von Semantik ist zudem für die Kommunikation zwischen den Direktoren von Vorteil. Unterschiedliche Wissenrepräsentationen machen es unmöglich, über ein gemeinsames Thema zu kommunizieren. Eine Abbildung zwischen verschiedenen Konzepten ist notwendig. Die Lösung dieses Problems ist Forschungsschwerpunkt des Semantic

Web. Schwerpunkt hierbei ist die formale Beschreibung und Speicherung von Informationen über Objekte und deren Assoziationen untereinander. Dazu wurden drei grundlegende Beschreibungssprachen entwickelt: *RDF* (Resource Description Framework), das erweiterte Resource Description Framework Schema *RDFS* und die Ontologiebeschreiungssprache *OWL* (Web Ontology Language) [82].

RDF Ziel des Resource Description Framework ist es, Objekte und Konzepte in Relation zueinander zu setzen. Dazu stellt das Datenmodell drei Basiskonzepte für eine Realtion bereit: *Subjekt*, *Prädikat* und *Objekt*. Ein Subjekt kann selbst wiederum ein Objekt in einer weiteren Relation sein, wodurch hierarchische Strukturen entstehen können. Für die textuelle Repräsentation existieren verschiedene Notationsformen, wie beispielsweise N3 [24] oder Turtle [19].

RDFS Für die Redundanzerkennung ist RDF allerdings nicht geeignet. Es fehlt ein Konzept zur Beschreibung von Klassen, Äquivalenzen und Synonymen. Dieses Problem löst das Resource Description Framework Schema (RDFS). In RDFS ist es möglich, Generalisierungsbeziehungen zwischen Ressourcen zu beschreiben. Das RDFS stellt dazu Elemente wie *rdfs:Resource*, *rdfs:Class*, *rdfs:Property* und *rdfs:range* zur Verfügung[6]. Aber auch RDFS ist zur Erkennung aller Redundanzen einsetzbar. Es ist nicht möglich, die Gleichheit festzustellen, wenn Entitäten eine identische Struktur besitzen, aber unterschiedliche Namen tragen.

OWL und OWL-S Eine Lösung für diesen Umstand bietet OWL [17]. Durch die Web Ontology Language (OWL) werden domänenspezifische Ontologien definiert. Die Web Ontology Language basiert auf RDFS. Die Web Ontology Language wurde entwickelt, um Ontologien in einer formalen Sprache zu beschreiben und zu verteilen. OWL gliedert sich in drei Sprachen, die unterschiedlichen Graden der Einschränkung unterliegen. Die größten Einschränkungen besitzt OWL-Lite. Im Gegenzug wird eine schnelle Berechenbarkeit garantiert. OWL-DL garantiert eine Berechenbarkeit nach den Regeln der Aussagenlogik und besitzt weniger Einschränkungen zur Beschreibung von Ontologien. Keinen Einschränkungen unterliegt schließlich OWL-Full und geht dabei weit über die Ausdrucksmächtigkeit des RDF-Schemas hinaus. Dies beinhaltet auch Sprachkonstrukten ähnlich der Prädikatenlogik. Der Preis dieser Freiheit ist das Fehlen jeglicher Garantien bzgl. der Berechenbarkeit. Abbildung 6.6 stellt diesen Überblick grafisch dar.

Der Vorteil einer Beschreibung von Prozessen und Diensten mithilfe einer Ontologie liegt in der Beschreibung und Erkennung von semantischer Äquivalenz. Es entsteht eine einheitliche Begriffs- und Beschreibungswelt, durch die alle Elemente charakterisiert werden. Darüber hinaus existieren bereits in OWL-Lite Konstrukte wie *equivalentClass*, *equivalentProperty*, *sameAs*, *differentFrom* und *allDifferent* um die Äquivalenz zwischen verschiedenen Konstrukten, auch anderer Ontologien, zu beschreiben. Beschreiben zwei Workflowelemente das gleiche Konzept, ist dies dadurch zu erkennen. Durch die Verwendung von Ontologien als Beschreibungsmittel, in oder für Workflows, ist eine Äquivalenzerkennung auch bei Variablen weit besser möglich, als nur über die Namengleichheit. Dies wirkt sich zusätzlich positiv auf die Redundanzerkennung aus. Möglich ist damit

[6] Das Präfix *rdfs* ist die gebräuchliche Abkürzung für den URI http://www.w3.org/200/01/rdf-schema#

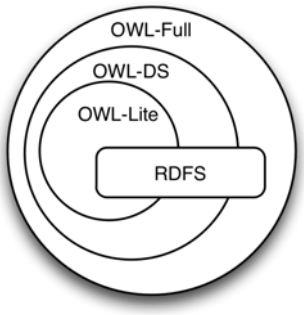


Abbildung 6.6: Überblick über die OWL-Teilsprachen und die Integration des RDFS.

auch die Erkennung von Redundanzen, wenn zwei Aktivitäten unterschiedliche Dienste nutzen, aber über die Ontologie festgestellt werden kann, dass diese Dienste gleich sind.

Wie eine vollständige semantische Workflowbeschreibung in OWL gestaltet sein kann zeigt OWL-S (Web Ontology Language for Services) [173, 110, 31]. Der Fokus liegt dabei auf der Beschreibung der Fähigkeiten und des Verhaltens von Diensten. Dazu werden drei Arten von Wissenrepräsentationen verwendet: das *Serviceprofil*, ein *Prozessmodell* und ein *Servicefundament*. Das Serviceprofil beschreibt welche Parameter der Dienst voraussetzt – Input- und Output-Parameter – und welche Dienste geboten werden. Das Prozessmodell beschreibt das Verhalten des Dienstes, insbesondere wie sich der Dienst im Inneren aufbaut. Wie der Dienst angesteuert werden kann, wird im Servicefundament beschrieben. Um OWL-S zu nutzen, muss zu jedem Dienst eine Servicebeschreibung existieren, die auf Serviceprofil, Prozessmodell und Servicefundament verweist. Dies bedeutet im Gegensatz zu klassischen Workflow-Beschreibungssprachen, dass zu jeder Aktivität vier Beschreibungen erstellt werden müssen. Unabhängig von den Tasks ist dann auch noch eine Beschreibung der internen Prozesse nötig. Das Prozessmodell, dargestellt in Abbildung 6.7, ist in OWL-S ein zusammengesetzter Prozess, der wiederum aus zusammengesetzten Prozessen oder atomaren Aktivitäten besteht. Eine atomare Aktivität kann nicht weiter zerlegt werden und führt in einem Schritt zum Ergebnis. Zusammengesetzte Prozesse können mithilfe von Standardkonstrukten beschrieben werden:

- *sequenz*, d. h. sequentielle Ausführung
- *if-then-else*, d. h. bedingte Ausführung
- *choice*, d. h. nicht-deterministische Ausführung
- *split*, d. h. parallele Ausführung
- *split + join*, d. h. parallele Ausführung mit Synchronisation
- *any-order* jede Ordnung (ungeordnete sequenzielle Ausführung)
- *repeat-while* und *repeat-until*, d. h. iterative Ausführung

Mit OWL-S ist es demnach möglich, sowohl Prozesse als auch alle darin verwendeten Dienste semantisch zu beschreiben. Die OWL-S Beschreibung wird für Workflows in der Praxis nicht häufig eingesetzt, da die Beschreibung aller notwendigen Modelle aufwändig ist, jedoch können die Grundkonzepte in vereinfachter Form wiederverwendet werden.

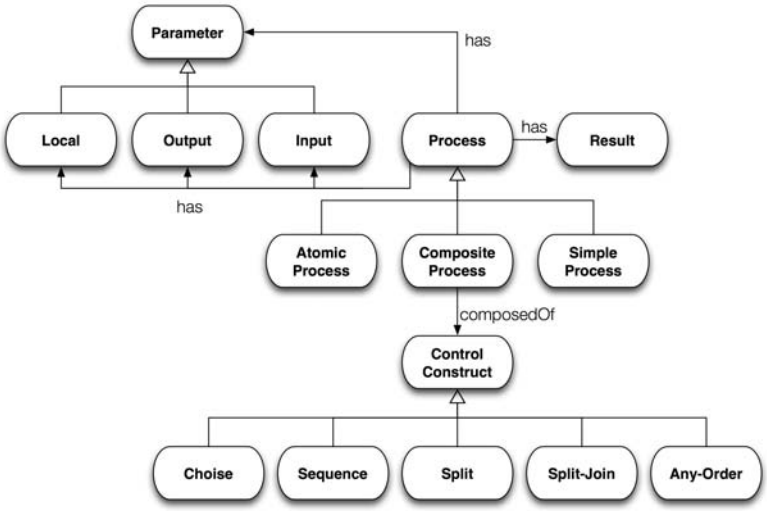


Abbildung 6.7: OWL-S Process Model (nach [173])

Einen weiteren Ansatz für die semantische Beschreibung eines einheitlichen Prozessvokabulars ist die *Web Service Modeling Ontology* [56], wie sie bei Webservices zum Einsatz kommt. Diese Sprache dient dazu, einen Gesamtdienst zu beschreiben, wobei eine Verfeinerung der Dienste in einem Prozessmodell wie bei OWL-S unterbleibt. Dies bedeutet, dass letztlich ein Interface für einen Gesamtprozess beschrieben werden kann, nicht aber alle Teilschritte, wodurch der Ansatz nicht zur Redundanzerkennung geeignet ist.

6.2 Kooperative BDI-basierte autonome Adaption

Ein Vorteil von kooperativen Direktoren besteht im Austausch von Lösungen. Dadurch können unterschiedliche Erfahrungswelten miteinander zu einer gemeinsamen Wissensbasis verknüpft werden. Zudem können sich kooperative Direktoren über den Zustand ihrer überwachten Instanzen austauschen und somit Redundanz vermeiden. Wie in Abschnitt 6.1 dargelegt, sind für eine funktionierende Kooperation drei Dinge notwendig:

- *Kommunikation*, d. h. ein standardisiertes Austauschformat für Nachrichten
- *Koordination*, d. h. ein Protokoll zur Kommunikation und zum Wissensaustausch zwischen den Direktoren
- *Gemeinsames Lernen*, d. h. vergleichbares Hintergrundwissen

Querschneidend zu diesen drei Aspekten der Kooperation ist die Nutzung von semantischen Beschreibungen über Umwelt notwendig. Die Umwelt für Direktoren stellen WfMS dar, deren Workflows es gilt, semantisch zu beschreiben.

6.2.1 Integration von Semantik in BDI

Um die Koordination und das gemeinsame Lernen bei Direktoren zu ermöglichen, ist es notwendig, dass alle Direktoren ihre Umgebung einheitlich beschreiben können. Bei Multi-Agenten-Systemen gibt es grundsätzlich zwei Möglichkeiten für eine gemeinsame Wissensrepräsentation: (1) Alle Agenten benutzen dieselben Begriffskonzepte oder (2)

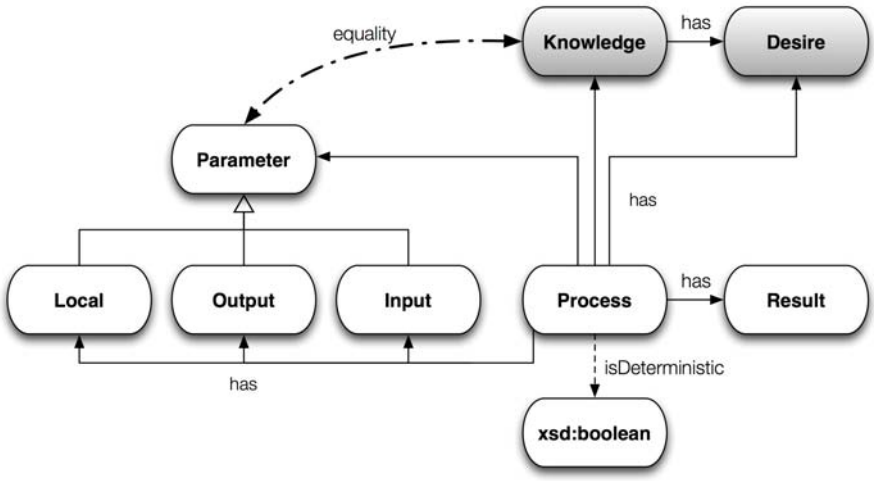


Abbildung 6.8: Erweitertes OWL-S Process Modell

es existieren unterschiedliche Wissensrepräsentationen, aber es existiert eine Abbildung zwischen den unterschiedlichen Konzepten.

Ein Beispiel für (2) wird in [107] vorgestellt. Hierbei lernen Agenten eine gemeinsame Ontologie. In diesem Framework besitzt jeder Agent ein eigenes Vokabular und zusätzlich eine Übersetzungstabelle (engl. translation table). Diese repräsentiert die Abbildung zwischen den unterschiedlichen Vokabularen der Agenten. Es wird allerdings auf die Verwendung von OWL oder RDF verzichtet.

Eine weitere Möglichkeit ist es, das Wissen der einzelnen Direktoren über die Workflows in verschiedenen OWL-Ontologien zu beschreiben und über Äquivalenzrelationen (siehe Abschnitt 6.1.4) die Abbildung zu realisieren. Eine semantische Beschreibung eines Workflows stellt das OWL-S Process Model dar (siehe Abschnitt 6.1.4)[7]. Das OWL-S Process Model ist jedoch nicht für die Verwendung im Kontext von BDI konzipiert worden. Es sind deshalb verschiedene Änderungen vorzunehmen. Dies betrifft primär die Beschreibung der Beliefs, d. h. der Wissensbasis des Direktors und die Beschreibung Ziele (Desires).

Die Abbildung des Ziels für einen Workflow ist in OWL-S nicht vorgesehen und muss deshalb explizit eingeführt werden. Dies ist in Abbildung 6.8 dargestellt. Dabei wird *Desire* dem *Process* zugewiesen. Damit besitzen automatisch auch *Atomic Process*, *Simple Process* bzw. *Composite Process* das Desire als Attribut, denn sie sind Subklassen von Process. Der *Process* wurde zusätzlich um das Attribut *deterministic* erweitert. Dadurch ist es möglich auszudrücken, ob sich eine Aktivität deterministisch verhält, d. h. dass sie bei gleichen Eingabeparametern das gleiche Ergebnis berechnet. Ist eine Aktivität nicht deterministisch, kann der Merge, als eine Möglichkeit der Kooperation, nicht ausgeführt werden. Ein Merge basiert auf der Annahme, dass durch die Migration eines Token in eine bestehende Instanz die Abarbeitung von redundanten Aktivitäten verhindert wird. Eine nicht deterministische Aktivität kann jedoch nicht redundant ausgeführt werden, denn sie führt auch bei gleichen Eingangsparametern zu einem unterschiedlichen Ergebnis.

[7] Auf die Beschreibung der Schnittstellenbeschreibung der Dienste kann hier verzichtet werden, da sie für die Kommunikation nicht zwischen den Direktoren nicht notwendig ist. Die verringert den Aufwand für die semantische Beschreibung.

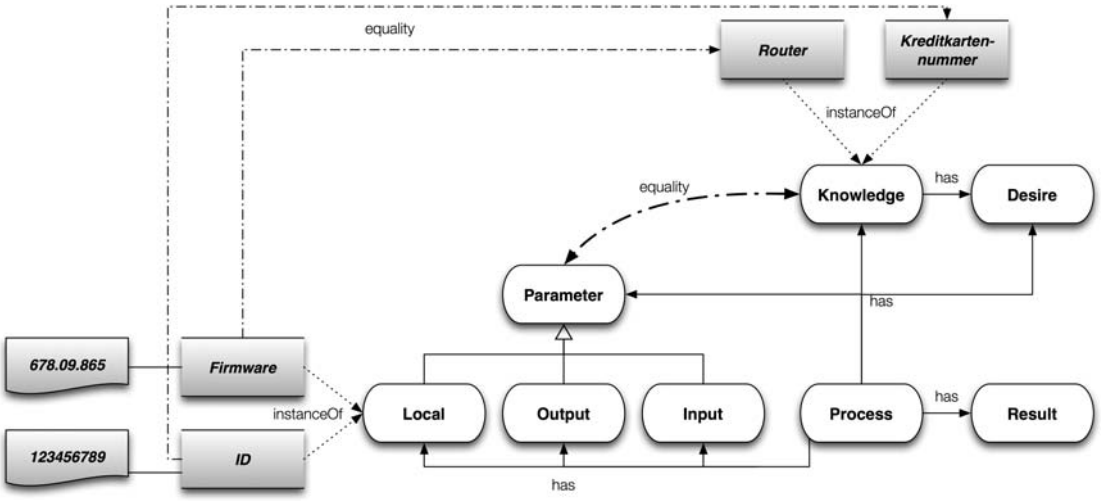


Abbildung 6.9: Äquivalenzen zwischen Knowledge und lokalen Parametern

Die neue Ontologie enthält des Weiteren die gelernten Konzepte der Direktoren. Diese werden als Subklassen zu der Oberklasse *Knowledge* angelegt. Jeder Direktor verwaltet seine eigene Ontologie, denn jeder soll eigene Erfahrungen sammeln können. Die Konzepte stellen zudem eine abstrakte Repräsentation der *Parameter* (siehe Abschnitt 6.1.4) dar, die durch eine Äquivalenzbeziehung aufeinander abgebildet werden können. Somit kann der Parameter ID vom Typ $xsd : int$ für einen Dienst auf das Konzept der *Kreditkartennummer* abgebildet werden. Dadurch wird zudem die Vergleichbarkeit der verschiedenen Variablen in unterschiedlichen Workflowmodellen sichergestellt. Es ist damit auch möglich, eigenen Datentypen zu modellieren. In Abbildung 6.9 sind beispielsweise die Klassen *Device* und *Firmware* definiert, um einen Router mit einer speziellen Firmware abzubilden. Die explizite Trennung von Basis- und erweiterten Konzepten unterstützt die dabei Wiederverwendung von Workflows.

Zwischen *Knowledge* und *Desire* besteht zudem eine explizite Verbindung. Diese Verbindung ist notwendig, um aus der aktuellen Situation auf ein zu wählendes Desire zu schließen. Wird der Direktor zu einem bestimmten Zeitpunkt der Ausführung nach möglichen Adaptionen gefragt, analysiert er zuerst seine Umgebung. Dies betrifft, neben anderen Aspekten, vor allem die Workflowvariablen. Ist eine Variable mit einem Konzept vom Typ *Knowledge* assoziiert, kann der Direktor das damit verbundene Desire aktivieren. Desires wiederum können ebenfalls mit Workflowmodellen oder Fragmenten assoziiert werden. Ein Schließen besteht darin, dass von einer Variable im Workflow auf eine Klasse in der Ontologie geschlossen wird. Eine mögliche Adaption kann berechnet werden, indem alle Workflows gesucht werden, die das Desire der Variable erfüllen. Das beschriebene Schließen stellt sich wie folgt dar:

$$Instance\ A \Rightarrow_{hasA} Knowledge\ B \Rightarrow_{hasA} Desire\ {}_{hasA}\Leftarrow Workflowmodell\ C$$

wobei A für eine Instanz steht, die einer bestimmten Variablenbelegung entspricht.

In Abschnitt 2.1.2 wurde ein vereinfachter Workflow in einem Callcenter vorgestellt. Die in Abbildung 6.10 gezeigte Abwandlung soll die vorgestellten Zusammenhänge darlegen. In diesem Beispiel können Kunden weiterhin ihre Probleme schildern, wobei das Ziel des

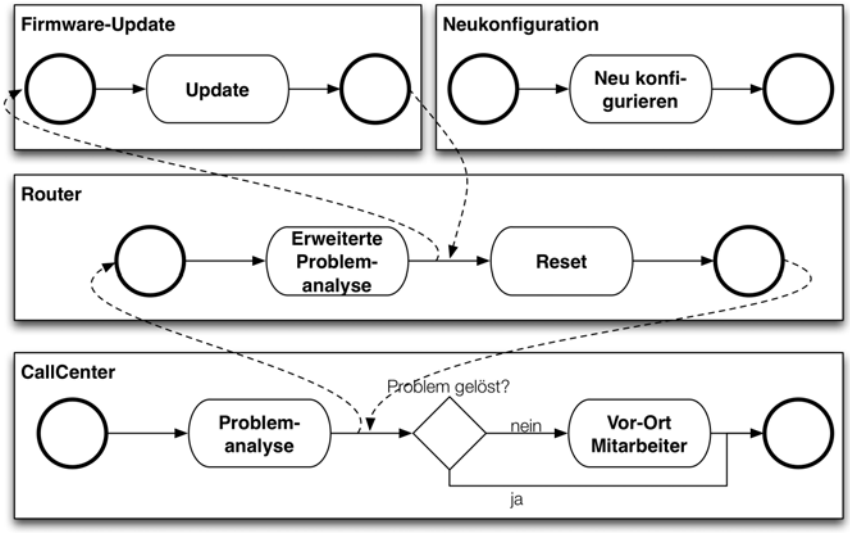


Abbildung 6.10: Adaption eines Callcenter-Workflows

Workflows die *Problemlösung* für den Kunden sei. Da die Firma eine breite Produktpalette besitzt, kann der Workflow für den Callcenter-Mitarbeiter nicht vollständig vorgegeben werden. Deshalb nimmt der Mitarbeiter erst alle notwendigen Daten zu dem Problem des Kunden in der Aktivität *Problemanalyse* auf. Im Anschluss hält die Workflow-Engine die Instanz an und der zugehörige Direktor wird nach einer Adaption gefragt. In diesem Beispiel wird das Problem auf eine FritzBox 7210 eingegrenzt. Aus der Variablenbelegung erkennt der Direktor, dass es sich um einen Router handelt, der auch dem Desire *Repair Router* zugeordnet ist. Der Direktor besitzt nun zwei Desires: Problemlösung und Router. Er sucht deshalb nach Workflows, die dieser Desire-Kombination entsprechen. Deshalb schlägt er vor, den Workflow *Router* einzuweben. Dieser startet ebenfalls mit einer Dateneingabe. Dabei werden die Seriennummer und die Firmware-Version des Routers abgefragt. Sind für diese keine Desires vorhanden, wird lediglich ein *Reset* empfohlen. Für die Firmware-Version 678.09.865 ist allerdings bekannt, dass diese Fehler verursacht und ein Update notwendig ist. Dementsprechend ist diesem Fall das Desire *Update* zugeordnet. Als Konsequenz wird der Workflow *Firmware-Update Router* eingebunden.

Durch das zusätzliche semantische Wissen ergeben sich Änderungen an den BDI-Abbildungen. Beim BDI-Direktor gelten folgende Abbildungen: B^S x $I^S \Rightarrow D^{S+}$ und B^S x $D^S \Rightarrow I^{S+}$. Durch die Einführung der expliziten Semantik und der damit verbundenen neuen Abbildung von Desires ergibt sich eine neue Berechnungsvorschrift für die neuen Desires (D^{S+}). Dabei bleibt die Abbildung B^S x $I^S \Rightarrow D^{S+}$ bestehen, wird allerdings ergänzt durch eine zweite Abbildung, die aus den Beliefs und dem semantischen Wissen (SK^S) ebenfalls die Desires berechnet. Die Ergebnisse der beiden Abbildungen (D^U und D^V) werden in einem zweiten Schritt zu einer Gesamtmenge vereinigt:

$$D^{S+} := D^U \bigcup D^V \text{ wobei gilt: } B^S \times SK^S \Rightarrow D^U \text{ und } B^S \times I^S \Rightarrow D^V$$

SK bezeichnet hier das semantische Wissen (engl. Semantic Knowledge). Die Abbildung $B^S \times SK^S \Rightarrow D^U$ muss nicht in einer Case-Datenbank gespeichert werden, sondern kann direkt über die Ontologie abgeleitet werden. Die Abbildung $B^S \times D^{S+} \Rightarrow I^{S+}$ kann, wie bereits beim semantikfreien BDI-Direktor beschrieben werden, verwendet werden.

6.2.2 Kooperation von Direktoren

Die semantische Beschreibung ist die Basis für die Kommunikation und das gemeinsame Lernen von Direktoren. Wie in Abschnitt 6.1.1 beschrieben, gibt es für die Koordination zwischen Agenten zwei grundlegende Ansätze. Bei dem *Blackboard*-Ansatz gibt es eine zentrale Instanz, die die Kommunikation zwischen allen Beteiligten regelt. Die Agenten kennen dementsprechend auch nur die Zentrale als direkten Kommunikationspartner. Beim *Peer-to-Peer*-Verfahren tauschen sich die Akteure untereinander aus.

Wie in Tabelle 6.1 beschrieben, gibt es vier verschiedene mögliche Kooperationsabläufe. Dabei ist die Kombination aus „individuellen Lernen und individueller Problemlösung + Wissensintegration“ für die Direktoren am besten geeignet. Diese ermöglicht es, dass ein WfMS weiterhin nur eine Instanz eines Direktor betrieben muss, eröffnet aber durch die Wissensintegration einen Weg, sich das Wissen der möglichen anderen Direktoren anzueignen, wenn in der eigenen Wissensbasis keine Lösung gefunden werden kann.

Während der Ausführung einer Workflowinstanz überprüft der Direktor, ob die aktuelle Situation einer anderen in der Falldatenbank gleicht oder ähnelt. Gleicht sie einer bekannten, kann die bereits gelernte Lösung dem Koordinator übergeben werden. Kennt der Direktor die Situation allerdings nicht, müsste er, wie in Abschnitt 5.1.3 beschrieben, ein *Fortfahren* zurückgeben, da er keinen besseren Vorschlag anbieten kann. In einem kooperativen Szenario ist dies nicht mehr zwingend nötig. Dazu besitzt jeder Direktor Zugriff auf ein *Blackboard*, auf dem er Nachrichten publizieren und lesen kann[8]. Die Nachrichten entsprechen abgewandelten ACL-Nachrichten des FIPA-Standards. Dabei wurde auf *Receiver* verzichtet, da über das Blackboard jeder Direktor angesprochen werden soll.

Im dem Fall, dass der Direktor keine ähnliche Situation kennt oder nur unzureichend ähnliche Situationen vorfindet, publiziert der Direktor sein Problem, in der Hoffnung, dass andere Direktoren diese Situation kennen. Nach dem Publizieren wartet der Direktor auf eine Antwort. Die Wartezeit W_f kann hierbei aber nicht beliebig sein, denn der Workflow kann nicht unendlich aufgehalten werden[9]. Die anderen Direktoren haben dabei zwei Möglichkeiten, um an die Fragestellung zu gelangen. Sie können in regelmäßigen Abständen das Blackboard abrufen (Pull-Prinzip), ob es neue Fragestellungen gibt. Wenn ja, prüfen sie, ob sie die gegebene Situation kennen. Ebenfalls möglich ist ein Broadcast (Push-Prinzip), der die Fragestellung an alle bekannten Direktoren weiterreicht. Diese Alternative besitzt den Vorteil, dass beim selbstständigen kontinuierlichen Überprüfen des Blackboards (Pull-Prinzip) und einem zu lang eingestellten Warteintervall[10] der fragende Direktor übermäßig nicht lang warten muss. Zudem kann die gestellte Frage wieder vom Blackboard gelöscht sein, da die Wartezeit des Anfragers abgelaufen ist. Im Zeitraum W_f können verschiedene Direktoren dem Anfragenden antworten, die festgestellt haben, dass sie bereits die gleiche oder eine ähnliche Situation bearbeitet haben und eine Lösung dazu kennen. Der Vergleich von Situationen basiert dabei vornehmlich auf den Variablennamen bzw. den ontologischen Typen und den Variableninhalten. Wenn alle Variablennamen von Situation S_1 in Situation S_2 enthalten

[8] Auf die Verwendung des Peer-to-Peer-Ansatzes wird an dieser Stelle verzichtet, damit nicht jeder Direktor eine Liste der anderen verfügbaren Direktoren verwalten und sich zusätzlich um deren Aktualität kümmern muss.

[9] Die Wartezeit muss in einer Implementierung über eine frei wählbare Konstante festgelegt werden.

[10] Das Warteintervall der gefragten Direktoren.

sind bzw. äquivalent sind, dann ist S_1 ähnlich[11] S_2.

Die Antworten publizieren die Direktoren ebenfalls auf dem Blackboard, wobei die ursprüngliche Nachricht referenziert wird. Somit kann die Antwort der Frage zugeordnet werden. Dazu wurde in die ACL-Nachricht das Attribut *reference* eingeführt:

Listing 6.2: Aufbau einer BDI-Nachricht

```
sender          Absendender Direktor
reference       Bezugsnachricht
content         Situationsbeschreibung bzw. Case
performative    Zweck der Nachricht
  ASK           Anfrage mit unbekannter Situation
  REDUNDANT     Anfrage, ob ein Workflow bereits ausgeführt wird
  COOPERATE     Gemeinsames Lernen
  LEARN         Angebot mit passendem Case
  THANKS        Direktor lehnt Angebot ab
```

Nach dem Ablauf der Wartezeit ruft der anfragende Direktor die Antworten ab. Erhält ein fragender Direktor keine Antworten, liefert er dem Koordinator ein *Continue* als Vorschlag zurück. Erhält er Antworten, wird im Anschluss daran aus dieser Antwortmenge die beste Antwort ausgewählt.

6.2.3 Formen der Kooperation

Wie das Listing 6.2 zeigt, gibt es verschiedene Antworttypen eines Direktors, die im Folgenden vorgestellt werden sollen:

Gemeinsames Lernen Die Antwort *Learn* stößt den Prozess des gemeinsamen Lernens an, d. h. es werden im besten Falle gegenseitig die bereits gelernten Lösungen zu der gegebenen Situation ausgetauscht. Besitzt der fragende Direktor keine Lösung, übernimmt er lediglich die Lösung des Antwortenden (einseitiges Lernen). Abbildung 6.11 stellt den Ablauf des gegenseitigen Lernens dar[12]. Hierbei übernimmt der Fragende die neuen Situationen[13] in seine Intention-Base und antwortet direkt auf diese Nachricht. Diese Antwort enthält die ihm bekannten ähnlichen Cases zur fraglichen Situation.

Kooperation bei gleichem oder ähnlichem Problem Die zweite Antwortmöglichkeit stellt die Kooperation – *COOPERATE*[14] – dar. Beschäftigt sich der antwortende Direktor soeben mit der gleichen Situation, unterbricht er seine betreute Workflowinstanz und antwortet mit einem *COOPERATE*. Dadurch signalisiert er seine Bereitschaft zu kooperieren und verhindert gleichzeitig die weitere Ausführung der zugeordneten Instanz. Dadurch wird gewährleistet, dass der Anfragende die eingetroffenen Antworten auswerten kann, ohne dass die Kooperationsgrundlage zerstört wird. In der Zeit bis zur Bestätigung könnte die Instanz weiter fortgeschritten sein, wodurch die Entscheidung für einen Merge nicht mehr gültig ist. Eine Kooperation wird vom Anfragenden ebenfalls mit einem COOPERATE beantwortet. Erhält er eine Nachricht vom Typ *THANKS*, signalisiert dies, dass er sich wieder seinen Aufgaben widmen kann.

[11] Ohne eine quantitative Aussage, wie ähnlich sie sind.
[12] Eine vollständige Darstellung der Kommunikation zeigt die Abbildung 8.10.
[13] Eine Antwort kann eine oder mehrere Situationen beinhalten.
[14] Das Protokoll ist ausführlich in Abbildung 8.10 dargestellt.

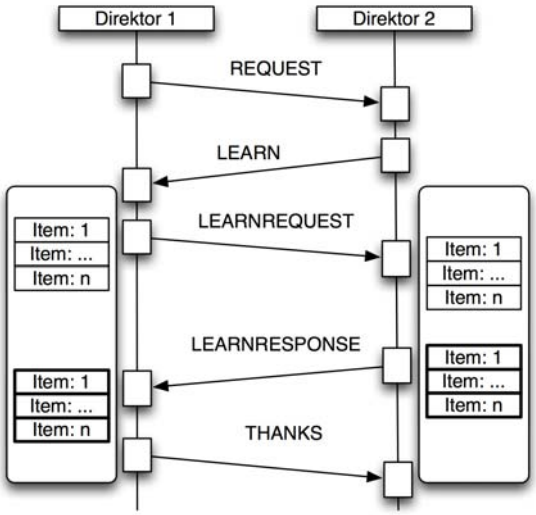


Abbildung 6.11: Kommunikation zwischen zwei Direktoren

Bei der Kooperation (*COOPERATE*) zwischen zwei Direktoren existieren viele unterschiedliche Konstellationen. Diese sind in Abbildung 6.12 dargestellt. Eine triviale Situation ist, dass der Direktor keinen Vorschlag bieten kann, da er kein Vorwissen zur vorliegenden Situation besitzt. Sein Verhalten in dieser Situation ist abhängig von seiner Implementierung und kann in zwei Arten unterschieden werden. Entweder ist er naiv und antwortet mit einem *Continue* oder er lernt von anderen Direktoren.

Merge von Workflowinstanzen Eine weitere denkbare Situation für eine Kooperation – *Merge* – ist in Abbildung 6.12(a) dargestellt. Diese Kooperation kann auf zwei verschiedene Arten angestoßen werden. Zum einen, wenn ein Direktor keine Antwort auf eine Situation kennt und nach ähnlichen Lösungen fragt. Ein anderer Direktor kann darauf mit COOPERATE antworten und somit den Merge einleiten. Zum anderen kann ein Direktor direkt über das Blackboard alle anderen fragen, ob der Workflow – mit den gleichen Variablenbelegungen –momentan bereits ausführen. Dies geschieht über den BDI-Nachrichten-Typ *REDUNDANT*. Empfängt ein Direktor diese Anfrage und kann sie positiv beantworten, muss er als erstes seine Workflowinstanz pausieren, um zu verhindern, dass die angefragte Adaption beendet wird, bevor die Migration der Ausführung erfolgt ist. Diese Situation wurde bereits im vorhergehenden Abschnitt angesprochen. Auch in dieser Situation wird vom Gefragten ein *COOPERATE* zurückgegeben. Nun kann eine Adaption, d. h. der Merge mehrerer Workflowinstanzen durch die Übergabe des eigenen Token – zur Bearbeitung in einer anderen Workflowinstanz – erfolgen.

In Abbildung 6.12(b) und (c) sind Situationen dargestellt, die sich mit Teiläquivalenz beschäftigen. Ein Direktor stellt vor der nächsten Aktion eine Anfrage über das Blackboard. Betreut ein anderer Direktor einen Workflow mit einer äquivalenten Aktion, kann im Idealfall auch in dieser Situation eine Delegation der Ausführung erfolgen, wenn die Variablenbelegungen stimmen[15]. Dieser Fall ist in Abbildung 6.12(b) dargestellt. Hierbei wird ein Token in die andere Instanz migriert. Abbildung 6.12(c) zeigt eine ähnliche Situation, in der ein Abschnitt einer Workflowinstanz identisch ist mit einem Abschnitt

[15]Der Gefragte antwortet in diesem Fall mit *COOPERATE*.

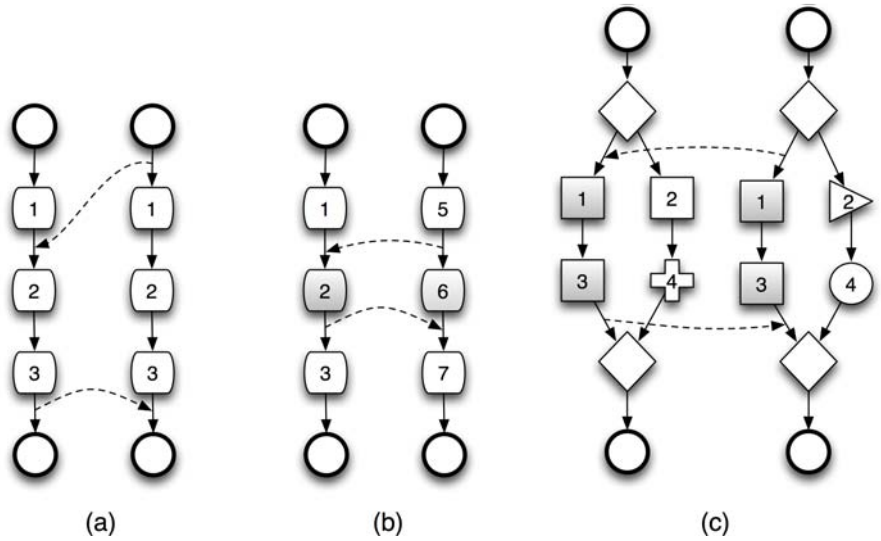


Abbildung 6.12: Kooperationssituationen

einer anderen Workflowinstanz. Hier zeigt sich, dass es sinnvoll ist, nur den aktuell folgenden Ausführungspfad eines verwendeten Modells zu betrachten. Die Konsequenz der Antwort auf die Frage des Direktors ist auch hier eine COOPERATE-Nachricht.

Umsetzung des Merge Wie in den vorherigen Abschnitten aufgezeigt, gibt es Situationen, in denen eine Migration sinnvoll ist. Dabei wurden jedoch keine Aussagen über den Vorgang selbst getroffen. Dieser weicht vom bisherigen Konzept ab, denn bei einem Merge befinden sich zu einem Zeitpunkt mehrere Token in einer Instanz: der originale und mindestens ein fremdes Token. Dieses ist in Petri-Netzen nicht ungewöhnlich, in herkömmlichen WfMS allerdings nicht gebräuchlich. Dieser Abschnitt zeigt, wie ein Merge erfolgt, wann er erfolgen kann und wie er wieder beendet wird.

Dabei ist zu beachten, dass die zusätzlichen Token nur aus einem Grund in einen anderen Workflow überführt werden: zur Vermeidung von Redundanz. Dies bedeutet, dass ein Workflow in bestimmten Teilbereichen von den Ergebnissen eines anderen Workflows profitieren möchte, d. h. ein Merge dient primär dem Datenaustausch. Für die Realisierung des Merges kann dabei auf die bestehenden Mechanismen der beschriebenen Adaption zurückgegriffen werden. Wie bei der normalen Adaption wird die Variablenbelegung beim Rücksprung in die Ausgangsinstanz übernommen. Damit der eingeführte Token allerdings keine Aktionen auslöst, besitzt das Token der ausführende Instanz einen Tokenpuffer, für eventuelle Teil-Token. Token in diesem Puffer werden von der Workflow Engine ignoriert.

Bevor auf die Bedingungen eingegangen wird, wann ein Merge erfolgen kann, muss betrachtet werden, wie entschieden wird, ob eine Delegation des eigenen Token erfolgt[16]. Ein Merge erfolgt nur im Rahmen der Kooperation der vorhandenen Direktoren, wenn der Gefragte eine Prozess- oder Semantikäquivalenz feststellt und eine Kooperation vorschlägt. Ein Merge wird demnach nur vorgeschlagen, wenn ein Direktor in einer gegebenen Situation keine Idee hat und nach einer Lösung fragt. Empfängt ein Direk-

[16] Die Delegation umfasst die Entscheidung, ob ein Merge erfolgen soll, die Berechnung der Splitposition und das Übergeben des Token an die entsprechende Workflowinstanz.

tor eine Nachricht, erfolgt seinerseits eine Prüfung, ob ein Merge möglich ist. Damit der gefragte Direktor einen Merge für sinnvoll erachtet, muss der Vergleich der eigenen Variablenbeschreibungen mit denen der entfernten Instanz erfolgreich verlaufen, damit die Daten entsprechend ausgetauscht werden können. Ein Vergleich der Variablen kann dabei auf Basis der Ontologie sicher geschehen. Weiterhin müssen die Instanzen syntaktisch oder semantisch gleich sein. Um unnötige Berechnungen zu vermeiden, wird dazu geprüft, ob es sich um das gleiche Workflowmodel handelt. Dazu kann man die Modell-UID überprüfen. Schlägt dieser einfache Vergleich fehl, ist es möglich, über einen SSA-basierenden Vergleich die beiden zugehörigen aktuellen Instanzen auf syntaktischer Ebene zu prüfen. Dabei werden die SSA-Repräsentationen der beiden Workflows miteinander verglichen Wenn auch dieser Vergleich fehl schlägt werden beide Instanzen auf semantischer Ebene verglichen. Hierbei wird ebenfalls auf nicht deterministische Aktivitäten geprüft. Diese werden als semantisch unterschiedlich behandelt, so dass ein Merge für diese Aktivitäten nicht berechnet wird. Dadurch kann trotz syntaktischer Gleichheit ein Unterschied festgestellt werden.

Kann der gefragte Direktor einen Merge vorschlagen, antwortet er auf die Frage mit einer Nachricht vom Typ COOPERATE, wie es in Abschnitt 6.2 beschrieben wurde. Anschließend veranlasst er seine Instanz dazu, für eine definierte Zeitspanne zu warten. Dadurch wird verhindert, dass die betreffende Instanz beendet wird, bevor der Merge erfolgt ist. Der Merge wird vom Direktor, wie auch vom Koordinator, wie eine Adaption behandelt. Durch das Vorhandensein von mehreren Token entsteht die Notwendigkeit zu kontrollieren, ob einer das Token fertig bearbeitet wurde und in seinen Ursprungsworkflow zurückkehren kann. Eine Möglichkeit, dies zu erreichen, bietet sich dadurch, dass jedes Token seine Rücksprungposition enthält. Die erste der beiden Forderungen wird erfüllt durch die Berechnungen, die vor der Delegation durchgeführt werden. Letztere Forderung kann erfüllt werden durch eine Funktion, die nach jeder Aktivitätsausführung den Token auswertet und im Falle eines notwendigen Splits, den Token in die Ursprungsinstanz verschiebt. Damit ist ein Merge der Instanzen erfolgt.

Eine Kooperation kann wie folgt zusammengefasst werden:

1. Wenn ein Direktor mit einer unbekannten Situation konfrontiert wird oder nach redundanten Ausführungen fragen möchte, publiziert er seine Frage auf einem Blackboard und wartet auf n Antworten. N und die Wartezeit werden durch Konstanten definiert.
2. Erhält ein fragender Direktor keine Antwort, muss er selbst lernen (Try and Error) oder ein *Continue* zurückgeben.
3. Die Direktoren überprüfen das Blackboard regelmäßig auf Anfragen und begutachten diese.
   a) Ist die aktuelle betreute Workflowinstanz syntaktisch oder semantisch gleich, wird eine Kooperation vorgeschlagen ($COOPERATE$).
   b) Existiert in der Intention-Base ein Fall, der hinreichend ähnlich zur angefragten Situation ist, wird mit Lernen ($LEARN$) geantwortet.
4. Egal, ob ein Direktor nach einer gewissen Wartezeit W_f eine Antwort auf seine Frage erhalten hat oder nicht, wird diese vom Blackboard gelöscht.
5. Kann der Fragende mit einer Antwort direkt etwas anfangen, nutzt er die Antwort und löscht die Anfrage.

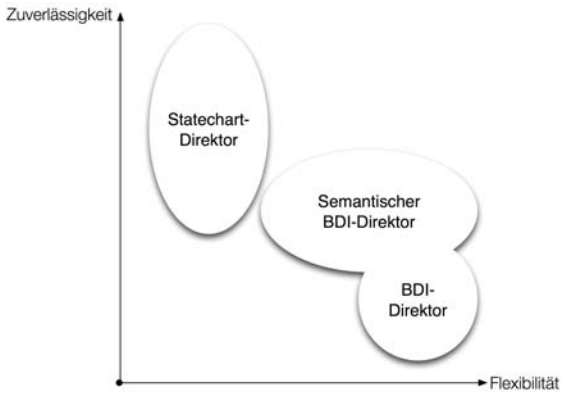


Abbildung 6.13: Vergleich von Zuverlässigkeit aller Direktoren

6.3 Zusammenfassung

In diesem Kapitel wurde der in Kapitel 5 vorgestellte BDI-Direktor um explizite Semantik erweitert. Dabei werden sowohl der gesamte Workflow als auch alle einzelnen Aktivitäten oder Variablen semantisch beschrieben. Durch die eingeführte Semantik konnten *kooperative Direktoren* entwickelt werden. Mit der Einführung einer Ontologie und der dadurch entstandenen expliziten Vergleichbarkeit des Wissens der Direktoren können diese untereinander Lösungen austauschen oder von anderen lernen. Zusätzlich konnte dadurch der Vergleich von Variablen und damit die Ähnlichkeitsberechnung für Situationen verbessert werden. Dadurch können mit dem BDI-Ansatz nicht nur *Production Workflows* sondern auch *Administrative Workflows* umgesetzt werden. Durch die verbesserte Ähnlichkeitsfunktion ergibt sich zudem eine andere Einordnung in das Spannungsfeld zwischen Zuverlässigkeit und Flexibilität als es beim herkömmlichen BDI-Direktor der Fall ist. Zudem trägt die Kooperation dazu bei, dass die Direktoren auf eine größere Wissensbasis zugreifen können, so dass die Zuverlässigkeit steigt. Dies wird in Abbildung 6.13 dargestellt.

Eine weitere Anwendungsmöglichkeit stellt die *Redundanzerkennung bzw. -vermeidung* dar. Ausgehend von den semantischen Beschreibungen der Workflows, können Workflows oder Workflowfragmente nicht nur strukturell, sondern auch bzgl. ihrer funktionalen Gleichheit verglichen werden. Dadurch ist es möglich festzustellen, ob eine andere Instanz die gleichen Aufgaben durchführt bzw. kurz zuvor durchgeführt hat. Dies wird nur dadurch möglich, dass die Direktoren untereinander Informationen austauschen und ihre Instanzen, inkl. aller Variablen, sicher vergleichen können. Wurde einer Gleichheit festgestellt, kann eine Kooperation beginnen. Dabei wird Token der einen Instanz in eine andere Instanz migriert. Dadurch kann er die bisherigen Ergebnisse (Variableninhalte) übernehmen und so lange in der Instanz verweilen, wie diese mit seiner Ausgangsinstanz gleich ist. Ist sie es nicht mehr, wird die Kooperation beendet und das Token verlässt die Instanz. Dieses parasitäre Verhalten ist dabei nicht negativ. Es vermeidet die redundante Ausführung von Aktionen. In großen WfMS ist dies ein Vorteil, denn es werden Ressourcen gespart. Der *Merge* stellt ein neues Adaptionsmuster, zu den bisher von Weber et. al [191] beschriebenen, dar.

Die Kooperation besitzt nicht nur den Vorteil der Vermeidung von Redundanz. Direktoren können auch Informationen über gelernte Adaptionen austauschen und erreichen dadurch viel schneller ein hohes Wissensniveau.

Kapitel 7

Vollständige Rekonfiguration kontextbezogener Workflow-Perspektiven

Die in Kapitel 3 vorgestellte Adaption durch Verweben bietet eine umfassende Unterstützung der Adaptionsmuster aus [191]. Das Konzept des Direktors zur Steuerung der Rekonfiguration von Workflows kann auf verschiedene Arten umgesetzt werden. Techniken aus dem Bereich der künstlichen Intelligenz, wie z. B. der vorgestellte BDI-Ansatz, sind dabei ebenso möglich, wie Ansätze mit Zustandsdiagrammen als Adaptionssteuerung. Diese Ansätze besitzen in verschiedenen Situationen ihre Vor- und Nachteile. Alle drei hier exemplarisch vorgestellten Direktoren setzen dabei auf der gleichen Adaptionstechnik auf. Adaptionen durch Verweben sind Manipulationen des Kontrollflusses, die während der Laufzeit eines Workflows durchgeführt werden. Als Basis dienen eine Vielzahl von vormodellierten Workflows, die je nach Situation in die aktuelle Instanz eingewoben werden, um bestimmten, nicht explizit im Workflow vorgesehenen, Situationen begegnen zu können. Dieser Adaptionsansatz beschränkt sich demnach auf Änderungen der Verhaltensperspektive.

Der Ansatz der reflexiven Workflows ist jedoch nicht auf diese Art der Adaption beschränkt. Der Regelkreis für rekonfigurierbare Workflows wird durch drei Hauptkomponenten charakterisiert:

1. *Workflow-Engine*: Zuständig für die Ausführung einer Workflow-Instanz.
2. *Direktor*: Zuständig für Adaptionsentscheidungen, die er auf Grund der aktuellen Ausführungssituation trifft.
3. *Koordinator*: Zuständig für die Umsetzung der Adaptionsentscheidung durch Anweisungen an die Workflow-Engine.

Rekonfigurierbare Workflows sind unabhängig von der technischen Umsetzung der Adaption. Hierfür entscheidend ist der Koordinator, der die Adaptionsentscheidungen für die jeweilige Workflow-Engine, mit ihrer spezifischen Änderungsmechanik, umsetzt. Auch die Direktoren selbst sind nicht an die Adaption durch Verweben gebunden. In ihren Fall- bzw. Adaptionsdatenbanken werden die Adaptionsanweisungen getrennt von den Situatonsbeschreibungen gespeichert. Es ist demnach einfach, die Adaptionsanweisung zu verändern. Die Adaptionstechnik durch Verweben mit variablem Rücksprungpunkt ist deshalb nicht die einzige Möglichkeit, Adaptionen zu realisieren.

Zusätzlich zu den klassisch strukturändernden Adaptionsmustern müssen jedoch ebenfalls die des Exception Handling betrachtet werden. Abbildung 7.1[1] zeigt ein Beispiel für

[1] nach [4] und http://www.workflowpatterns.com/patterns/exception/considerations.php (zuletzt aufgerufen am 25.02.2011).

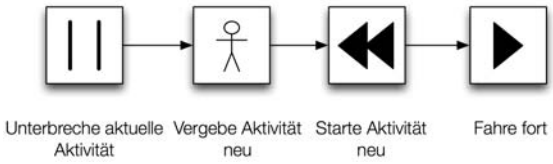


Abbildung 7.1: Exception Handling mit vier Schritten

eine komplexe Exception Handling-Strategie. Dargestellt ist eine Kompensation, bei der ein *User Task* nicht ausgeführt werden kann, da der zugewiesene Bearbeiter nicht zur Verfügung steht. Damit die Instanz nicht abgebrochen werden muss, ist für diesen Fall eine Adaption vorgesehen, die aus zwei unterschiedlichen Teilen besteht: Der Zuweisung eines neuen Bearbeiters und die wiederholte Ausführung der Aktivität. Die Kontrollflussadaption stößt hierbei an ihre Grenzen. Sie ermöglicht die Wiederausführung der Aktivität, allerdings ist es nur schwer möglich, einen neuen Bearbeiter zuzuweisen. Dies ist grundsätzlich durch das Einweben eines User Tasks, bei dem ein anderer Bearbeiter zugewiesen ist, möglich. Dies bedeutet auch, dass für jede mögliche Kombination von User Task und Bearbeiter eine extra Aktivität physisch im Repository angelegt werden müsste. Der Aufwand für diese Lösung übersteigt dabei den Nutzen. Eine Adaption der kontextbezogenen Eigenschaften ist in diesem Zusammenhang wesentlich geeigneter.

Bisher wurde die Rekonfiguration der *kontextbezogenen Perspektiven* nicht betrachtet. Weder die operationale, noch die informationsbezogene oder die organisatorische Perspektive können beeinflusst werden. Ein Adaptionsansatz für diese Eigenschaftsänderungen fehlt demnach.

Diese Eigenschaftsadaption ist nicht mit der Adaption der Verhaltensperspektive vergleichbar, denn sie verändert den Workflow auf einer anderen Ebene. Sie kann deshalb auch nicht mit den klassischen Adaptionsverfahren, wie sie in Abschnitt 2.3 vorgestellt wurden, umgesetzt werden, denn hierbei werden die Eigenschaften und das Verhalten der Aktivität verändert. In Abschnitt 2.3.2 wurden die Adaption mittels aspektorientierter Ansätze vorgestellt. Mit ihnen ist es möglich, einzelne Eigenschaften eines Workflows zu verändern.

In [153] wird eine aspektorientierte Workflow-Beschreibung gezeigt, die ebenfalls kontextbezogene Eigenschaften ändern kann. Dabei wird das Workflowmodell in *Repräsentationselemente*, *Verbindungspunkte* und *Verbinder* aufgeteilt. Die Verbindungspunkte an den Repräsentationselementen werden durch die Verbinder miteinander verknüpft. Die Repräsentationselemente sind durch Aspekte repräsentierte Artefakte des Workflowmodells. Zur Verknüpfung dieser Aspekte wurden zwei unterschiedliche Verbinder eingeführt. Durch den temporalen Verbinder können Kontrollflussabhängigkeiten abgebildet werden, der nicht-temporale entspricht herkömmlichen Assoziationen. Die aspektelementorientierte Workflowrepräsentation ermöglicht somit strukturelle und kontextbezogene Änderungen sowie Modellerweiterungen.

Die Aspektorientierung besitzt jedoch trotz jahrelanger Entwicklung, keine einheitliche Semantik und ihre Verwendung ist deshalb auch stark von der jeweiligen Realisierung abhängig. Es ist beispielsweise nicht geklärt, ob Aspekte auch einen Zustand besitzen oder ob und wie Pointcuts abhängig vom Kontext beschrieben werden können. Beide Eigenschaften sind vor allem bei komplexen Problemen hilfreich. Aspekte unterscheiden

sich daher auch in ihrer Repräsentationsform auf den verschiedenen Abstraktionsebenen der Softwareentwicklung. Diese ungeklärten Fragen erschweren die Verwendung von Aspekten in vielen Bereichen.

Einen fortschrittlicheren Ansatz stellt die *Rollenorientierung* [163, 69] dar. Dieser erweitert die klassische Objektorientierung sowohl in der Modellierung, als auch in der Programmierung. Der Begriff *Rolle* (oder Objektrolle) ist nicht vergleichbar mit dem Rollenbegriff in der Workflowdomäne, der mit dem Role-Based Access Control Modell[2] (RBAC) verwandt ist. Die Rollenorientierung ist vielmehr mit dem Begriff der Rolle in UML-Klassendiagrammen oder Entity-Relationship-Diagrammen vergleichbar. In der Rollenorientierung existieren zwei Haupttypen [163, 69]: der Basistyp (engl. core type) und Rollen (engl. roles). Ein Basistyp, der die aus der Objektorientierung bekannten klassischen Objekte repräsentiert, ist mit einem Rollentyp über eine *can-play-a* Assoziation verbunden. Das bedeutet, dass ein Objekt eine Rolle spielen kann, was dem umgangssprachlichen Begriff der Rolle entspricht. Angewandt auf das obige Beispiel kann das Verhalten einer Aktivität geändert werden. Ist der Webservice nicht verfügbar, kann die Rolle *WebServiceCall* von der Aktivität abgelegt und gleichzeitig die Rolle *EJB Call* angenommen werden. Dies geschieht ausschließlich zur Laufzeit, ohne dabei das Originalmodell verändern zu müssen.

Wird dieser Rollenansatz konsequent angewandt, ist es möglich, dass ein System nur aus rudimentären Basistypen besteht, die erst durch Rollen vervollständigt werden. Die Basistypen verfügen nur über wenige Daten und Funktionen. Zusätzlich existieren Rollen, die je nach Bedarf Daten und Funktionalität beisteuern. Das führt zu einem schlanken und gleichzeitig flexiblen System, da nur wenige Basistypen existieren, die durch Rollen erweitert werden. Ein System mit derartigen Eigenschaften ist gut für reflexive Workflows geeignet, denn alle benötigten Laufzeiteigenschaften auf Aktivitätsebene können geändert werden. Die Bindungen von Basistypen und Rollen müssen allerdings gesteuert werden. Dabei ist es wichtig, dass die Bindungen zum richtigen Zeitpunkt aufgebaut werden und diese zielgerichtet eingesetzt werden. Für diese neue Ebene der Adaption kann ebenfalls der Direktor-Ansatz genutzt werden. Hierbei beschreibt die Adaptionsvorschrift nicht den einzuwebenden Workflow und den Rücksprungpunkt, sondern die aufzubauenden bzw. zu lösenden Rollenbindungen.

Als Basis für ein Workflowmodell mit Rollen wird ein nicht spezialisiertes Ablaufmodell benötigt, das durch Rollen erweitert werden kann. Dafür eignen sich *Workflow-Nets* [178, 185], ein Derivat von Petri-Netzen. In [121, 181] präsentieren die Autoren eine Workflow-Engine auf Basis von Workflow-Nets und zeigen dabei, dass diese verschiedene Vorteile, z. B. gegenüber gerichteten Graphen besitzen: (1) eine fundierte formale Semantik, (2) eine zustandsbasierte Struktur und (3) vorhandene Analysetechniken (Check für Deadlocks, Liveness und Fairness).

Werden Workflow-Nets als Basis für eine Workflow-Beschreibungssprache verwendet, die durch Rollen verfeinert wird, entsteht eine hochflexible Sprache, mit speziellen Eigenschaften:

1. *Kompatibel* zu allen, auf Petri-Netzen basierenden, Workflow-Beschreibungssprachen.

2. *Adaptives* WfMS, d. h. Hinzufügen neuer Aktivitätstypen selbst zur Laufzeit.

[2] Rollenbasierte Zugriffskontrolle.

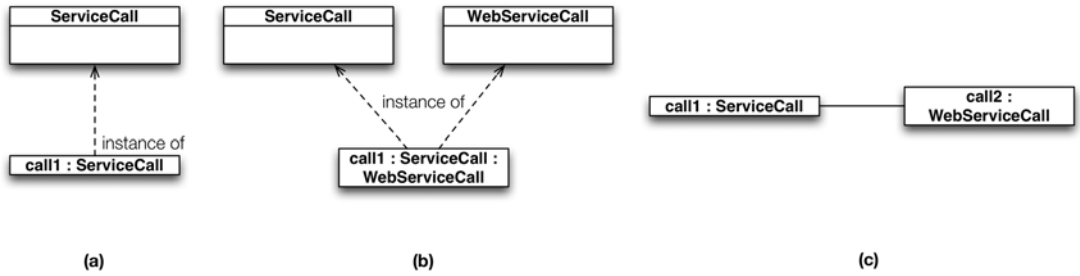


Abbildung 7.2: Typenbindung in statischen Typsystemen

3. *Flexible* Ausführungssemantik, d. h. die Rollen einzelner Aktivitäten können durch den Direktor jederzeit geändert werden.

4. *Domänenspezifische* und *domänenübergreifende* Workflow-Beschreibungssprachen.

In den folgenden Abschnitten wird der Entwurf eines rollenbasierten Workflowmodells zur Rekonfiguration der *kontextbezogenen Perspektiven* gezeigt. Des Weiteren wird am Ende des Kapitels aufgezeigt, wie mithilfe der beiden Rekonfigurationsmethoden komplexe Adaptionen umgesetzt werden können.

7.1 Grundlagen der rollenorientierten Modellierung und Programmierung

Eine laufzeitadaptive Workflow-Beschreibungssprache ist mit klassischen objektorientierten Techniken nur bedingt zu realisieren. In [179] wird eine adaptive Workflow-Beschreibungssprache vorgestellt, die auf Vererbung beruht. Wie bei evolutionären Workflows werden Änderungen am Kontrollfluss durch Versionierung umgesetzt. Diese Versionen stehen aber nicht separat für sich, sondern sind in eine Vererbungshierarchie des Workflowmodells eingebunden. Dabei wird das veränderte Workflowfragment als neue Objektstruktur angelegt. Der unveränderte Teil des Workflowmodells wird von der Ursprungsvariante geerbt. Dieser Ansatz beschränkt sich auf Kontrollflussänderungen. Es ist aber ebenfalls denkbar, dass die *kontextbezogenen Perspektiven* durch Vererbung versioniert werden. Dies führt jedoch nur bedingt zu einem vollständig adaptiven Workflowmodell. In einem statischen Typsystem kann eine Aktivität ihren Typ nicht ändern, sondern ist an diesen gebunden. Die Aktivität `Call1` in Abbildung 7.2(a) ist eine Instanz der Klasse `ServiceCall`. Soll diese Aktivität zur Laufzeit statt eines Webservices ein EJB über RMI aufrufen, ist dies nicht möglich. Die klassische Lösung in der Objektorientierung besteht im Einsatz von *Mehrfachvererbung*, wie die Abbildung 7.2(b) dargestellt. Das Objekt instanziiert nun ein Konglomerat aus Service- und WebService-Call. Dadurch ist es möglich, zur Laufzeit zwischen den Funktionalitäten zu wechseln. Jedoch ist diese Lösung auf diese zwei Möglichkeiten beschränkt. Für weitere Flexibilität muss das Modell erweitert werden, d. h. es können nur vorhersehbare Kombinationen abgebildet werden. Neue Funktionen erfordern ständige Anpassungen, die nicht zur Laufzeit durchgeführt werden können.

Eine weitere Lösung für dieses Problem sind *Delegations*. Die Delegation arbeitet auf dem Level von Objekten und bedeutet, dass ein Objekt o_1 ein anderes Objekt o_2 referenziert und die Möglichkeit hat, Methoden von o_2 aufzurufen (siehe Abbildung 7.2(c)).

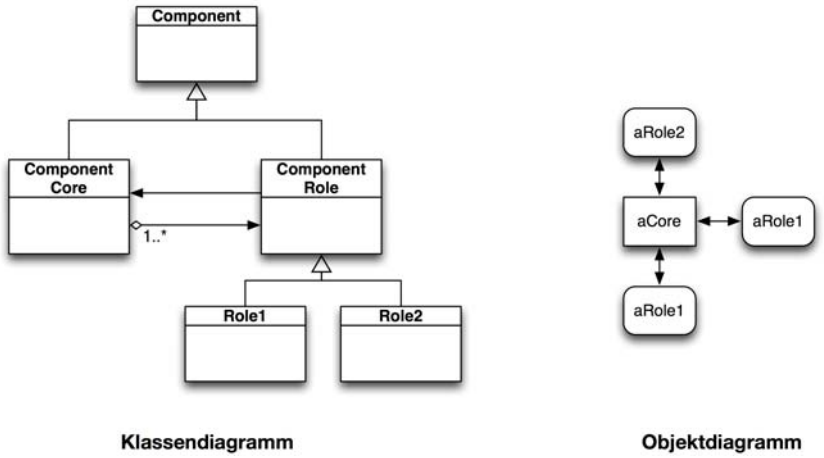


Abbildung 7.3: Role Object Pattern

Dadurch ist es möglich, das oben genannte Szenario zu realisieren, jedoch skaliert diese Lösung nicht. Jede neue Funktion muss mit dem Objekt verbunden werden. Gleichzeitig müssen die Managementmethoden angepasst werden, die zwischen den verschiedenen Funktionalitäten vermitteln.

Eine erweiterte Lösung für dieses Problem stellt das Role Object Pattern [16] dar, welches konzeptionell Rollen in die Objektorientierung einführt. Das Entwurfsmuster führt dabei keine neuen Konstrukte ein, sondern löst das Problem durch geschickte Strukturierung, wie in Abbildung 7.3 dargestellt. Der Managementcode ist dabei in den `ComponentCore` (Kern) ausgelagert. Die `ComponentRole` (Rolle) stellt die einzelnen Funktionalitäten dar, zwischen denen der Kern vermittelt. Beide – Kern und Rolle – erben vom gleichen Interface, so dass sie nach außen hin polymorph adressiert werden können. Die Klasse `Role` kann weiter verfeinert werden, z. B. durch den EJB- und WebServiceCall. Das Muster stellt demnach eine Mischung aus den beiden vorherigen Lösungen dar, die das Problem der Skalierung durch Strukturierung lösen. Das Role Object Pattern besitzt aber weiterhin die Grundprobleme der beiden oberen Lösungen.

7.1.1 Theoretische Grundlagen

Klassische objektorientierte Lösungen besitzen, durch eher starre Konstrukte, in dynamischen Szenarien viele Nachteile. Es fehlt eine Möglichkeit, dynamische Kollaborationen zwischen verschiedenen Objekten zu beschreiben. Dieser Nachteil wurde bereits 1987 von Rumbaugh beschrieben:

> „class-based object-oriented implementations of object collaborations hide the semantic information of collaborations but expose their implementation details“ [144].

Das Konzept der Rollen bietet diese Möglichkeit und grenzt sich deutlich von dem klassischen Begriff der Rolle in WfMS ab, der bisher im Zugriffsrechtemanagment – Role-Based Access Control model (RBAC) – genutzt wurde. Der hier verwendete Begriff der *Rolle* [163, 69] stammt vielmehr aus dem Kontext des Theaters. Während eines Stückes spielen die Schauspieler unterschiedliche Rollen, wie z. B. in dem Theaterstück über „Dr. Jekyll und Mr. Hide“, in dem der Schauspieler zwei Rollen spielt. Im wirklichen Leben,

spielen wir ebenfalls viele Rollen gleichzeitig. Zum Beispiel kann eine Person die Rolle eines Lehrers, eines Vater und eines Autobesitzers annehmen. Ein Auto kann zunächst die Rolle einer Ware oder später eines Sammlerstücks annehmen.

Ein Rollenmodell enthält grundsätzlich zwei Arten: *Kern-Typen* – die den klassischen Objekten entsprechen – und eine Reihe von *Rollen*. Ein Kern kann eine oder mehrere Rollen spielen, d. h. der Kern und die Rolle sind durch die *can-play-a*-Beziehung verknüpft. Die Fähigkeit, parallel mit mehreren Rollen zu agieren, erlaubt es, den Kern in den unterschiedlichsten Situationen anzupassen, je nachdem, welche Rollen er spielt. Dieses Spielen ist dabei dynamisch. Es ist dem Kern möglich, die Rollen jederzeit, d. h. zur Laufzeit, hinzuzufügen oder zu entfernen. Rollen können demnach *aktiv* oder *inaktiv* sein. Auch kann ein Kern eine Rolle temporär ablegen und zu einem späteren Zeitpunkt wieder annehmen. Dadurch erreicht dieser Ansatz ein hohes Maß an Flexibilität, Erweiterbarkeit und Anpassbarkeit.

Wie Klassen eine Reihe von Objekten definieren, die die gleichen Eigenschaften besitzen, definieren Rollenklassen Rollenobjekte mit den gleichen Eigenschaften. Klassen und Rollen sind unterschiedliche Konzepte. Sie unterscheiden sich in zwei entscheidenden Eigenschaften: der *Rigidity*[3] und *Foundation*[4] [69].

Mit *Rigidity* wird die Eigenschaft eines Typs bezeichnet, aus eigener Kraft zu existieren. Beispielsweise können Instanzen vom Typ `Buch` allein existieren. Instanzen vom Typ `Leser` können dies nicht, denn es muss immer eine Person existieren, die der Leser ist. Typen wie der `Leser` werden als *non-rigid* bezeichnet. Rollentypen sind demnach ebenfalls non-rigid, im Gegensatz zu Klassen. Die Eigenschaft *non-rigid* ist eine Notwendigkeit für das Konzept der Rolle, denn sie werden immer gespielt. Es kann keine Rolle ohne ihre Spieler existieren.

Die zweite Eigenschaft der Rolle ist die *Foundation*. Typen gelten als *founded*, wenn ihre Instanzen immer an einer Kollaboration teilnehmen. Der `Leser` ist *founded*, weil es keinen Leser ohne ein zu lesendes Buch geben kann. Founded Typen können demnach ebenfalls nicht allein existieren, sondern bedingen immer mindestens ein Gegenüber. Das bedeutet, dass es auch mehrere assoziierte Typen gibt. Ein founded Typ kann vorschreiben, dass seine Instanzen mit mehreren (1..∗) anderen founded Types in Verbindung stehen. In Bezug auf Rollen bedeutet dies, dass diese generell founded sind und sie immer in einem *Kontext* gespielt werden. Dieser Kontext beschreibt die Kommunikationsmöglichkeiten zwischen den Rollen.

Wie im wahren Leben können die Rollen nicht beliebig miteinander kommunizieren oder wahllos an einen Kern gebunden werden. Mithilfe von *Role-Constraints* lassen sich die Kombinationsmöglichkeiten einschränken. In [141] beschreibt Riehle die wichtigsten von ihnen. Um beispielsweise auszudrücken, dass zwei Rollen immer parallel gespielt werden müssen, verwendet Riehle die *role-equivalence*. Eine schwächere Form ist *role-implication* zwischen zwei Rollen. Sie bedeutet, dass, wenn die erste Rolle gespielt wird, die zweite durch den gleichen Spieler gespielt werden muss, aber nicht umgekehrt. Des Weiteren existiert die *role-prohibition* um auszudrücken, dass zwei Rollen nie durch den gleichen Spieler parallel gespielt werden dürfen.

Neben der bisher beschriebenen dynamischen Typisierung und erweiterten Kollaboration gibt es noch weitergehende Rollenkonzepte. So können Rollen beispielsweise in [80, 10]

[3] Stabilität.
[4] Fundament oder kontextbezogen.

eigene Daten besitzen und das Grundverhalten des Kerns verändern. Dies geschieht in den verschiedenen Ansätzen zumeist durch Überladen von Methoden oder durch Techniken aus der Aspektorientierung. Trotzdem sind beide Ansätze nur schwer vergleichbar, denn die Rollenorientierung besitzt nicht diese Mehrdeutigkeit der Aspekte.

7.1.2 Realisierungen

Die Rollenorientierung ist kein neues Paradigma, sondern tritt in verschiedenen Abstufungen bereits seit über zwanzig Jahren in Erscheinung. Dies geschah durch erste Ansätze in der Modellierung bis hin zu heutigen rollenorientierten Programmiersprachen.

Im Jahr 1989 führte Halpin [72] das Object-Role Modeling (ORM) ein. Ziel des Ansatzes ist die Überwindung der bereits beschriebenen Einschränkungen der Objektorientierung und der Entity-Relationship-Modellierung. Der Ansatz konzentriert sich ausschließlich auf die Modellierung der Daten, da Halpin im Design von Informationssystemen, wie z. B. ERP-Systemen, große Potentiale sah. Der wesentliche Unterschied zur objektorientierten (Daten)-Modellierung besteht darin, dass nicht nur Objekte modelliert werden, sondern auch die Rollen, die sie spielen. Halpin sieht Objekte als Entitäten oder Werte an, die strukturierte (Unternehmens-) Daten repräsentieren. Beziehungen (Relationships) stellen eine Korrelation zwischen diesen Entitäten her. Der zweite wichtige Begriff stellen die Rollen dar. Halpin sieht Rollen als essentiellen Teil der Relationships. Diese werden an den Enden der Beziehungen als Andockpunkte für die jeweiligen Spieler in der Beziehung beschrieben, wie sie später auch in den UML-Klassendiagrammen Verwendung fanden.

Auf Basis dieser Gedanken folgten weitere Arbeiten, die die Relationships in den Fokus stellten und dabei auf die Einführung der Rolle als Konzept verzichteten. Gleichwohl hatten sie die gleichen Ziele. Biermann und Wren [25] sowie Balzer et al. [12] untersuchten die Beziehungen in der objektorientierten Programmierung. Wie Halpin verstehen sie Beziehungen als Kooperationen von Objekten untereinander und behaupten, dass objektorientierten Sprachen diese nicht unterstützen. Daher fordern sie, die Beziehungen als erstrangige Konstrukte einzuführen, wie dies bei Klassen der Fall ist. Damit ist es möglich, Kollaborationen und Einschränkungen von Kooperationen deklarativ zu beschreiben. Aus diesen Gründen wurde RelJ [25] eingeführt, eine neue Sprache mit Beziehungen als eigenständiges Konstrukt.

Ein ähnlicher Ansatz findet sich bei Smolander: das OPRR[5] Meta-Metamodel [161]. Es besitzt vier grundlegende Konzepte: Objekte, Eigenschaften, Beziehungen und Rollen. OPRR ist dabei eine Methode zur Beschreibung von Information Systems (IS) und zur Datenmodellierung, d. h. auch hier fehlt die verhaltensändernde Eigenschaft von Rollen, wie es bereits bei den Relationship-Ansätzen der Fall ist. Die Begriffe Objekt und Eigenschaft entsprechen denen in der Objektorientierung. Eine Eigenschaft besitzt dabei einen Typ und einen Wert. Beziehungen verbinden Objekte miteinander und beschreiben zudem, wie sie verbunden sind. Neben Objekten, Eigenschaften und Beziehungen enthält OPRR auch Rollen als explizites Konzept. Diese Rollen befinden sich ebenfalls am Ende der Relationen und besitzen hier auch eigene Eigenschaften. Somit ist es möglich, dass die Informationen streng zwischen Objekt und Rolle getrennt werden können, wie z. B. das Gehalt eines Angestellten. Dieses gehört nicht direkt zur Person, sondern immer zu einer bestimmten Kollaboration mit einem Unternehmen.

[5] Object-Property-Role-Relationship.

Einen weiteren Ansatz für Rollen stellen Reenskaug und Anderson mit der OOram[6] Software Engineering-Methode [127] vor. Ihre Hauptmotivation für die Einführung der Rollen ist die Modellierung von Aktivitäten. Eine Aktivität beschreibt eine Menge von Objekten, die miteinander, zur Erreichung eines bestimmten Ziels, interagieren. In der Objektorientierung würden sich Fragmente einer Aktivität in allen verbundenen Klassen wiederfinden, was auch als *Crosscutting* bezeichnet wird[7]. Als Lösung für dieses Problem werden Rollen eingeführt. Ein Rollenmodell beschreibt dabei eine Aktivität auf Basis verschiedener Rollen. Verbunden werden diese durch Rollenpfade, welche eine Menge von Methoden festlegen, welche in dieser Aktivität genutzt werden können. Ein jedes Objekt kann eine oder mehrere Rollen gleichzeitig spielen. Dieser Ansatz fokussiert demnach das Verhalten und nicht die Modellierung der Daten. Rollen in OOram besitzen zudem keinen eigenen Zustand, d. h. keine[8] Daten.

Es existieren verschiedene (konkurrierende) Ansätze zur Rollenmodellierung. Gleichzeitig entstanden auch Programmiersprachen, die das Konzept der Rollen unterstützen. Dazu gehören beispielsweise: powerJava, Rava, DOOR, EpsilonJ und ObjectTeams.

Boella et al. [11] bieten mit *powerJava* eine Sprache, die nahe an das allgemeine Rollenkonzept kommt. Rollen sind an ihre Spieler gebunden, besitzen Zustand und Verhalten und müssen immer in einem Kontext gebunden sein. Die Sprache wurde zur Erforschung von Multi-Agenten-Systemen eingesetzt und ist eine Erweiterung der Programmiersprache Java. *Rava* [76] ist ebenfalls eine Erweiterung dieser Programmiersprache, die Rollen unterstützt. Die Umsetzung der Rollen basiert auf den Konzepten des Role Object Pattern, vermischt dies aber zusätzlich mit dem *Mediator* Pattern [63].

DOOR ist ein dynamisches, objektorientiertes Datenbank-Management-System mit Rollenunterstützung [195, 194], kann aber auch als eine objektorientierte Programmiersprache, mit Schwerpunkt auf Datenbanken gesehen werden. Das DOOR-Datenmodell beinhaltet Klassen und Rollen sowie deren Instanzen. Klassen und Rollen bilden eine Art Vererbungshierarchie, wobei die Klassen- und Rollen-Hierarchien orthogonal zueinander sind, d. h. eine Rolle kann keine Klasse spezialisieren und umgekehrt.

EpsilonJ [112] ist eine weitere Erweiterung der Programmiersprache Java. Sie basiert auf einem sprachunabhängigen Modell – Epsilon [168]. Das Modell der Rollen kommt dem Idealmodell nahe, d. h. es bildet Kontext, Rolle und Spieler ab. In Epsilon beinhalten Kontexte Rollen, die an Objekte außerhalb des Kontextes, Spieler genannt, gebunden werden können. Rollen definieren zusätzlich eine Schnittstelle, die von Objekten zur Verfügung gestellt werden muss, um als Spieler in Erscheinung zu treten. Die Bindung von Rollen und Spieler wird dabei explizit angegeben.

ObjectTeams [80] ist die am weitesten fortgeschrittene Implementierung und basiert ebenfalls auf Java. Hauptkonzepte von ObjectTeams sind *Teams*, Rollen, die playedBy-Beziehung sowie *callins* und *callouts*. Rollen werden jeweils in einem Kontext definiert, d. h. sie werden hier nicht dynamisch gebunden. Kontexte können, wie Klassen und Rollen, Attribute und Methoden besitzen, und sind deshalb statusbehaftet. In ObjectTeams werden die Kontexte *Teams* genannt. Rollen sind rigide, müssen demnach an einen Spieler gebunden werden. Dies geschieht mithilfe der *playedBy*-Beziehung. Der Spieler wird als *base* bezeichnet (ähnlich dem Kern). Rolleninstanzen sind stark an ihre Spieler gebunden, d. h. sie können nicht von einem Spieler zu einem anderen migrieren. Der Grund

[6] Object Oriented Role Analysis and Modeling.
[7] Dies ist auch eine der Hauptmotivationen für die Aspektorientierung.
[8] Dies gilt, insofern die lokalen Methodenvariablen außer acht gelassen werden.

dafür ist, dass Rollen auch Attribute besitzen, und deshalb einen Status besitzen. Dieser kann nicht auf beliebige Spieler übertragen werden. Rollen können aber jederzeit abgelegt oder neu an den Spieler gebunden werden. In ObjectTeams verändern Rollen stark das Verhalten der Spieler. Dafür gibt es zwei Möglichkeiten: *callins* und *callouts*. Ein *callin* wird in einer Rolle festgelegt und gibt an, welche Methoden, die am Spieler aufgerufen werden, an die Rolle umgeleitet werden. *Callouts* sind das Gegenteil, sie leiten den Kontrollfluss von der Rolle auf den Spieler um. Beide Konzepte sind bereits aus der Aspektorientierung bekannt und wurden auch mit ähnlichen Mitteln umgesetzt.

7.1.3 Zusammenfassung

Mithilfe der Objektorientierung ist eine adaptive Workflow-Beschreibungssprache nur schwer umsetzbar. Eine adäquate Lösung ist mittels Rollen realisierbar. Formal haben Rollen zwei wichtige Eigenschaften:

1. Sie sind *non-rigid*. Rigidity bedeutet, dass eine Instanz eines Typs, diese nicht verlieren kann, ohne selbst aufzuhören zu existieren.

2. Rollen sind *founded*, das heißt, dass sie auf Zusammenarbeit mit anderen Rollen angewiesen sind. Rollen sind demnach immer an einen Kontext gebunden.

Der entscheidende Vorteil von Rollen ist ihre dynamische Natur. Rollenorientierte Programmierung überwindet die Beschränkung der Objektorientierung, die Objekte an einen bestimmten Typ für ihre gesamte Lebensdauer bindet. Während ihrer Lebenszeit können Objekte sowohl unterschiedliche Rollen spielen, als auch mehrere Rollen gleichzeitig und mehrere Rollen von der gleichen Art zur gleichen Zeit. Diese Dynamik ermöglicht die Entwicklung flexibler, adaptiver und erweiterbare Systeme.

7.2 Kontextorientiertes erweiterbares Workflowmodell basierend auf Rollen

Wichtig für das Design einer rollenorientierten Workflow-Beschreibungssprache sind die funktionalen Anforderungen, die an sie gestellt werden. Diese entsprechen den Perspektiven von Workflows, die bereits in Abschnitt 2.2.2 vorgestellt worden:

- *Verhaltensperspektive*: Beschreibung des Kontrollflusses
- *Kontextbezogene Perspektiven:*
  - *Funktionale Perspektive*: Beschreibung der einzelnen Aktivitäten
  - *Informationelle Perspektive*: Beschreibung der Daten und des Datenflusses
  - *Operationale Perspektive*: Beschreibung der internen und externen Dienste
  - *Organisationale Perspektive*: Beschreibung von Ressourcen und Ressourcenzuweisungen zu den Aktivitäten

Eine ideale Workflow-Beschreibungssprache sollte mithin alle Workflowperspektiven (siehe 2.2.2) unterstützen. Ein Blick auf die aktuellen WfMS-Implementierungen zeigt, dass viele nur einen Teil der Anforderungen umsetzen. Dies liegt auch daran, dass verschiedene Ansätze sich spezialisieren, so dass eine vollständige Abdeckung nicht notwendig erscheint. Die durch die Aspekte implizierten Anforderungen sind zudem umfangreich, und dadurch schwer in Gänze umzusetzen.

7.2.1 Modulares rollenbasiertes Workflowmodell

Die Lösung für dieses Problem liegt in einer *modularen*, auf Rollen basierenden, Workflow-Beschreibungssprache. Die Verwendung von Rollen ermöglicht zweierlei: (1) Laufzeitadaptivität und (2) Modularität. Bei der Laufzeitadaptivität werden die dynamischen Rollenbindungen ausgenutzt. Die Modularität ermöglicht es, zu einem späteren Zeitpunkt Elemente hinzuzufügen, um nicht alle Anforderungen sofort erfüllen zu müssen.

Der Vorteil des Einsatzes von Rollen besteht darin, dass die Kerne ausschließlich das Grundgerüst bilden, wobei die einzelnen Objekte wenig eigene Attribute besitzen. Die Funktionalität wird in den Rollen fixiert, die gleichzeitig die dafür notwendigen Daten speichern können. Der Kern übernimmt in diesem Fall nur die Funktion des Ankerpunkts und Vermittlers für die, dynamisch durch Rollen, hinzugefügten Erweiterungen.

Workflows basieren immer auf einer definierten Reihenfolge von Aktionen. Es gibt dabei zwei unterschiedliche Lesarten. Zum einen existiert die Beschreibung des Kontrollflusses, in der festgelegt wird, welche Aktionen aufeinander folgen. Diese Sichtweise ist bei Workflows die am häufigsten vorkommende. Zum anderen kann der Workflow auch mithilfe des Datenflusses beschrieben werden, wie es beispielsweise in Datenflussdiagrammen [13] gemacht wird. Diese beschreiben die Veränderung der Daten mithilfe von Aktionen. Der Kontrollfluss ist hierbei nicht vorhanden, es gibt demzufolge auch keine komplexen Schleifenkonstrukte. Mit diesem Ansatz können programminterne Abläufe beschrieben werden, allerdings sind komplexe Arbeitsabläufe in der realen Welt nicht umsetzbar, weshalb der Ansatz demzufolge auch wenig eingesetzt wird. Daher kann diese Sichtweise bei der Konzeption der Workflow-Beschreibungssprache nicht in Betracht gezogen werden. In kontrollflussorientierten Sprachen, ergibt sich der Datenfluss nur implizit, durch die Auswertung der Nutzung von Variablen in bestimmten Aktionen. Es gibt jedoch auch Arbeiten [71], die den Datenfluss zusätzlich zum Kontrollfluss beschreiben. Diese werden nur zur Optimierung der Ausführung genutzt. Die *informationsbezogene Perspektive* sieht hingegen eine Beschreibung des Datenflusses vor.

Der Kern einer Workflow-Beschreibungssprache sollte demnach der *Verhaltensperspektive* sein, der den Kontrollfluss repräsentiert. Petri-Netze, bzw. die vereinfachten *Workflow-Nets*, können als Grundlage jeder Workflow-Beschreibungssprache genutzt werden, wie in [187] gezeigt wurde. Die Autoren untersuchten die Abbildbarkeit aktueller Workflow-Beschreibungssprachen (mit dem Fokus auf Webservices) auf Workflow-Nets, mit dem Ergebnis, dass dies „einfach“ möglich ist[9]. Sie präsentierten einen Zuordnungskatalog für die grundlegenden Sprachelemente von BPEL auf Workflow-Nets. In BPEL sind die gebräuchlichsten Kontrollflusskonstrukte, wie Sequenz, Parallelität, Entscheidung, Schleifen und Ausnahmebehandlung vertreten, so dass Workflow-Nets als Basis für diese Sprachen gelten können. In [183] wird eine Studie vorgestellt, wie Workflow-Beschreibungssprachen die gängigen Workflowmuster [184] unterstützen und wie sie auf

[9] Zitat: „This shows that it is not hard to deploy an erroneous BPEL process model [...]“ [187].

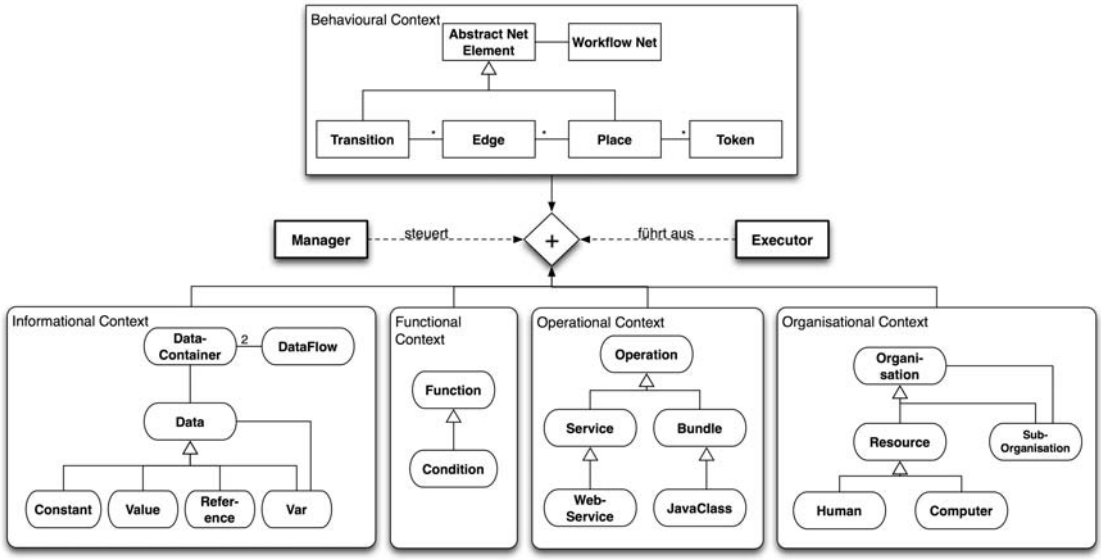


Abbildung 7.4: Rollenorientiertes Workflowmodell

Workflow-Nets abgebildet werden können. Die Autoren fanden heraus, dass Workflow-Nets die Workflowmuster abbilden können, so dass es möglich ist, fast jede Workflow Sprache mit Workflow-Nets darzustellen.

Workflow-Nets [185, 178], sind eine spezielle Klasse von Petri-Netzen, die einigen Einschränkungen unterliegt:

1. Es existiert nur ein Startknoten i.
2. Es existiert nur ein Endknoten o.
3. Für jedes Element – Stelle oder Transition – existiert ein Pfad von i nach o, den dieses nutzt.

Abbildung 7.4 zeigt ein vereinfachtes[10] Modell eines Workflow-Nets, als Basis für die modulare und adaptive Workflow-Beschreibungssprache. Um diesen *Behavioral Context* gruppieren sich Kontexte, die die verbleibenden Workflowaspekte repräsentieren. Diese bilden abgeschlossene, modulartige Einheiten und beinhalten die Rollen, gekennzeichnet durch die abgerundeten Ecken. Die Elemente der einzelnen Kontexte können dynamisch durch die Workflow-Net Klassen gespielt werden.

Als Kern für die Rollen dienen die Elemente des *Behavioural Context*: *Transition, Place Edge, Token,* und das *Workflow-Net* selbst. Diese fünf Typen sind rigide, d. h. sie können in ihrer Struktur zur Laufzeit nicht durch Rollen verändert werden[11]. Dies bedeutet auch, dass Adaptionen sich auf die Rollen beschränken, und den Kontrollfluss des Workflows nicht betreffen. Die Transition in Workflow-Nets repräsentiert eine auszuführende Aktion. Transitionen sind atomare Vorgänge, die, wenn sie schalten, Token aus den Eingangsstellen verbrauchen und Token an die Ausgangstellen verteilen. Diese speichern die Token, bis die folgende Transition wiederum schaltet.

[10] Auf die Beschreibung der Workflow-Net Contraints mittels OCL wurde an dieser Stelle verzichtet.
[11] Für die Veränderung des Kontrollflusses ist die Adaption durch Verweben vorgesehen.

Transitionen stellen die Verbindung zwischen der operationalen Perspektive – der Beschreibung der einzelnen Dienste/Funktionen – und dem funktionalen Aspekt – der Konfiguration der verschiedenen Operationen, her. Der *Functional Context* beschreibt Daten für die Ausführung der verschiedenen Funktionen in der Workflow-Engine. Es gibt nur zwei Basis-Interfaces für Rollen: *function* und *condition*, die je nach aufzurufender Operation durch eine entsprechende Rolle verfeinert werden müssen. Eine Operation könnte der Aufruf beispielsweise eines Webservices sein. Die verfügbaren Dienste und Operationen sind in dem *Operational Context* beschrieben. Er enthält ein zentrales Interface: *operation*. Die abgeleiteten *service* und *bundle*[12] sind Beispielrollen für BPEL und jBPM.

Der *Organizational Context* beschreibt die Struktur von Organisation oder Ressourcen. Dies können sowohl menschliche, als auch technische Ressourcen, wie Drucker etc. sein. Dadurch kann der *Functional Context* mit Mitarbeitern kollaborieren, um damit User Tasks, wie sie beispielsweise in Oracle BPEL[13] und WebSphere[14] vorhanden sind, zu unterstützen. Auch kann damit klassische Zugriffskontrolle modelliert werden.

Token können durch Rollen des *Informational Context* erweitert werden. Dieser ist verantwortlich für die Daten und den Datenfluss. Im Gegensatz zu den anderen Kontexten sind hier viele Rollen vorgegeben, da dieser Kontext sich in den verschiedenen Workflowsprachen wenig unterscheidet. Das Basisinterface ist der *DataContainer*, der sich auf mehrere *data*-Rollen bezieht. Komplexe Datenkonstrukte können über die Rekursion an der Rolle *Var*[15] beschrieben werden, wie sie für BPEL oder anderen Webservice-Kompositionssprachen benötigt werden. Auch können durch Verfeinerung der Rolle *Var* eigene Datentypen angebunden werden. Einfache Variablen werden über die Rolle *Value*, Konstanten über *Constant* dargestellt. Für Datenreferenzen[16] ist die Rolle *Data Reference* vorgesehen, wie sie beispielsweise für *data-grey-box Webservices* [71] benötigt wird.

Der Datenfluss – *DataFlow* – kann hier explizit angegeben werden, im Gegensatz zu klassischen Workflow-Beschreibungssprachen, die diesen zumeist nur implizit unterstützen. Einige Workflow-Engines unterstützen die explizite Beschreibung des Datenflusses, weshalb die explizite Variante integriert ist[17]. Für die Modellierung eines Datenflusses werden zwei Ankerpunkte vom Typ Data-Container benötigt. Einer bildet den Beginn, der andere das Ende des Datenflusses.

Der Effekt dieses Ansatzes auf das Modell eines Workflows ist in Abbildung 7.5 dargestellt. Es zeigt einen Workflow zur Überprüfung einer Kreditkartennummer. Der Workflow besteht aus einer einzigen Transition und einem Token. Der Übergang transportiert den Token zum Ende der Instanz, d. h. er hat Zugriff auf den Token und die mit ihm verbundenen Daten. Die Transition selbst spielt die Funktion `CheckCreditCard`, der ein Java-Klassenaufruf zugeordnet ist. Beide haben Zugang zu den `DataContainter` (DC) CreditCard durch `CheckCreditCard`.

Ein entscheidender Vorteil eines rollenbasierten Workflowmodells ist die Erweiterbarkeit. Wie in Abschnitt 2.2.2 beschrieben, existieren neben den fünf Hauptperspektiven noch

[12] Stellt den Aufruf einer Java-Klasse dar.
[13] http://www.oracle.com/technetwork/middleware/bpel/overview/index.html (zuletzt aufgerufen am 25.02.2011).
[14] http://www-01.ibm.com/software/de/websphere/ (aufgerufen am 27.06.2011).
[15] Composite Design Pattern [63].
[16] Konstrukt ähnlich zu Pointern.
[17] Jede implizite Nutzung kann auf eine explizite abgebildet werden.

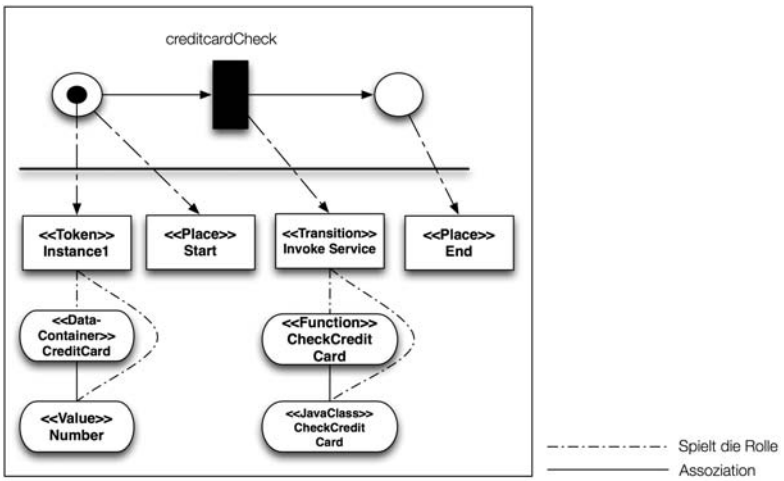


Abbildung 7.5: Workflow mit Rollen

weitere. Diese können ebenfalls durch Einführung neuer Kontexte realisiert werden, die modular an das bestehende Modell angegliedert werden können.

7.2.2 Manager und Executor

Neben diesen Basiskontexten existieren zwei weitere Komponenten, die in Abbildung 7.4 dargestellt sind. Die erste ist der *Manager*. Diese Komponente überwacht und kontrolliert zur Laufzeit alle Kollaborationen und Constraints. Da zur Laufzeit auch neue Rollen hinzugefügt werden können, ist der Manager selbst als erweiterbare Komponente konzipiert. Andere Manager können sich bei einem zentralen Manager registrieren und werden während der Validierung iterativ aufgerufen. Dadurch ist es möglich, neue Rollen-Contraints zur Laufzeit hinzuzufügen. Die grundlegenden Rollen-Constraints[18] des vorgestellten Rollenmodells sind:

1. *Workflow-Nets* können nur Rollen des *Organizational Contexts* spielen, bis auf *human*.
2. *Token* können nur Rollen des *Informational Context* spielen.
3. *Edges* und *Transitions* können keine Datenrollen spielen.
4. *Places* können nur *Organizational* und *Functional Context*-Rollen spielen.
5. Rollen der Kontexte *Operational, Organizational* und *Informational* können keine Kollaboration eingehen.
6. Ein korrekter Datenfluss benötigt zwei *Data-Container*-Rollen.

Die zweite Komponente ist der *Executor*, der das Workflow-Net interpretiert und als Laufzeitumgebung für die Rollen dient. Dazu bringt es eine entscheidende Eigenschaft mit: der Executor kann *Gegenrollen* spielen. Diese sind wichtig, denn eine Rolle kann nur ihre Funktionen wahrnehmen, wenn sie eine Rolle zur Kollaboration besitzt. Das

[18] Die Rollen-Constraints sind nicht in Abbildung 7.4 dargestellt, da diese zu einem überladenen Diagramm führen würde.

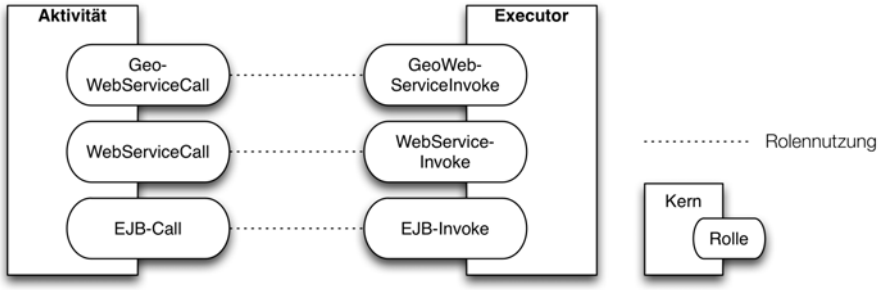


Abbildung 7.6: Rollen und Gegenrollen

oben beschriebene Rollenmodell bildet nur eine Seite der Kollaboration ab. Der Executor muss für jede dieser Rolle eine Gegenrolle besitzen, um diese auszuwerten. Zu diesem Zweck besitzt der Executor für jede Basisrolle, die passende Gegenrolle, wie dies in Abbildung 7.6 dargestellt ist. Die Rolle `WebServiceCall` beinhaltet alle Konfigurationsdaten und ggf. die Verbindung zu anderen Rollen, die für den Aufruf notwendig sind. Dazu zählen die Variablen des Token, die für die Parameter der Webservice-Methode wichtig sind. Die Zuordnung der Rollen geschieht über eine interne Abbildung des Managers, dass zur Laufzeit veränderbar ist. Dieser Ansatz entspricht einer strikten Trennung von Konfigurationsdaten und der Ausführung, was vor allem bei der Umsetzung von Vorteil ist.

7.2.3 Rekonfiguration und Erweiterung

Der Hauptvorteil bei der Verwendung von Rollen liegt in der dynamischen Erweiterbarkeit der Workflow-Engine sowie der Laufzeitadaptivität der einzelnen Workflowinstanzen. Im Gegensatz zu klassischen Lösungen für Erweiterbarkeit – wie beispielsweise die Design Pattern *Decorator* und *Composite* [63] – können Rollenbindungen zur Laufzeit geändert werden, was zudem die Funktionalität der Basiskomponente verändert.

Die Basis-Rollen und Schnittstellen werden durch das Modell zur Verfügung gestellt. Jede potentielle Erweiterung kann diese Schnittstellen nutzen oder sie durch Rollenvererbung verfeinern, solang keine Rollen-Constraints verletzt werden. Neue Rollen können zur Laufzeit, durch die Registrierung neuer *Role-Bundles*[19], hinzugefügt werden. Ein Role-Bundle benötigt zwei wesentliche Komponenten: (1) eine Erweiterung für den Manager, die die Registrierung der neuen Rollen und der neuen Contraints vornimmt, (2) die neuen Rollen, wobei jede Rolle ihre Gegenrolle benötigt, die der Executor spielen kann.

Role-Bundles können durch die dynamischen Eigenschaften von Rollen zur Laufzeit hinzugefügt oder gelöst werden. Bei der Integration neuer Bundles müssen die Erweiterungen registriert werden, so dass die Rollen offiziell eingebunden werden. Dadurch können der Manager und der Executor die Rollen adressieren. Danach können sie an die Kernelemente gebunden werden. Bei der Herauslösung eines Role-Bundles kann dies zu Inkonsistenzen führen, denn die darin verfügbaren Rollen sind dementsprechend für die laufenden Instanzen nicht mehr verfügbar. Dies kann zu einem ungültigen Zustand der Instanz führen. Die Role-Bundles können deshalb nur herausgelöst werden, wenn alle

[19] Bundles entsprechen klassischen Plug-Ins, wie sie beispielsweise in der Entwicklungsumgebung Eclipse verwendet werden.

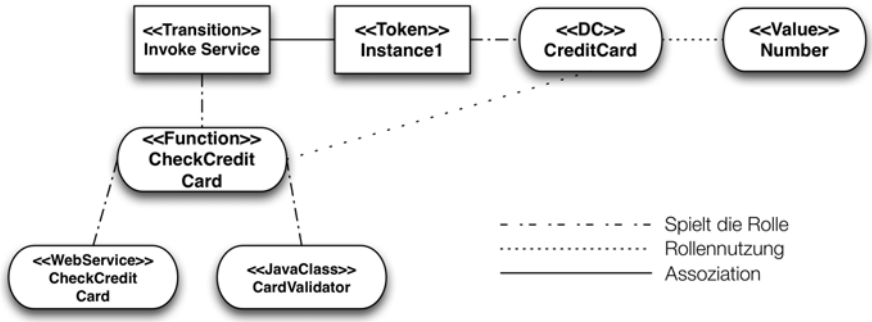


Abbildung 7.7: Beispiel Rollenraum für eine Transition und einen Token

abhängigen Workflowinstanzen beendet wurden[20]. Es ist aber zusätzlich möglich, das Bundle zu deaktivieren, so dass keine Rollen neu instanziiert werden können. Die bestehenden Rolleninstanzen sind davon nicht betroffen, so dass alle Instanzen regulär enden. Der Vorteil dieses Ansatzes besteht darin, dass keine neuen Instanzen mit diesen Rollen gestartet werden können. Bei dem Versuch, eine deaktivierte Rolle zu instanziieren, wenn eine neue Workflowinstanz gestartet werden soll, interveniert der Manager mit einer *invalid*-Antwort. Dadurch kann der Executor diese Workflowinstanz nicht ausführen.

Abbildung 7.7 zeigt eine modifizierte Beispiel-Konfiguration aus Abbildung 7.5. Die Transition *Invoke Service* spielt die Funktion `CheckCreditCard`, die mit zwei operativen Rollen verbunden ist: `Webservice` und `JavaClass`. Dies bedeutet, dass die Transition die `Function CheckCreditCard` von beiden operativen Rollen ausgeführt werden kann. Sie ist demnach überkonfiguriert. Beide haben über die Funktion CheckCreditCard Zugang zum DataContainter[21] `CreditCard`. Executor und Manager können zur Laufzeit entscheiden, welche Rolle verwendet werden soll. Im Fall eines fehlgeschlagenen Aufrufs bei einer der beiden Rollen, kann der Executor die andere Rolle, als eine Möglichkeit einer Exception Handling-Strategie, ausführen. Wenn beide Rollen bei der Workflow-Engine registriert sind, kann diese wählen, welche Rolle mit einer Gegenrolle gebunden wird. Die Entscheidung kann auf einer Zufallsfunktion basieren oder auf der Entscheidung eines der Direktoren. Auch können verschiedene Qualitäten berücksichtigt werden. Wird beispielsweise erkannt, dass ein Webservice hohe Antwortzeiten aufweist, kann auf einen anderen Dienst umgeschaltet werden. Dadurch können Aktivitäten im Workflow flexibler gestaltet werden. Anstatt *compensation handling* (Ausweichhandlungen) zu modellieren, z. B. wenn die Antwortzeit nicht ausreichend ist, werden Modellvarianten zur Designzeit angegeben, um aufwändige Ausweichhandlungen zu vermeiden.

Das obige Beispiel kann ebenfalls statisch mittels verschiedener Design-Pattern, wie z. B. *Template Method* [63], die auf Vererbung basieren, abgebildet werden. Fallen die zur Verfügung stehenden Dienste aus, ist es nicht möglich, diese durch neue Dienste auszutauschen. Es besteht lediglich die Möglichkeit, die Workflowinstanz abzubrechen oder zu pausieren und zu warten, bis einer der Dienst wieder verfügbar ist. Mit einer rollenbasierten Workflow-Engine ist es möglich, eine neue operative Rolle zur Transition hinzufügen. Dies kann auf Basis einer Direktor-Entscheidung oder einer Adaption durch den Administrator geschehen. Dadurch wird beispielsweise die Ausnahmebehandlung flexibler. Klassische Exception-Handling Strategien [137, 165, 39, 148, 4] und viele weitere

[20] Ein Abbrechen der Workflowinstanzen wird hier nicht als adäquate Lösung angesehen.
[21] In Abbildung 7.7 mit DC gekennzeichnet.

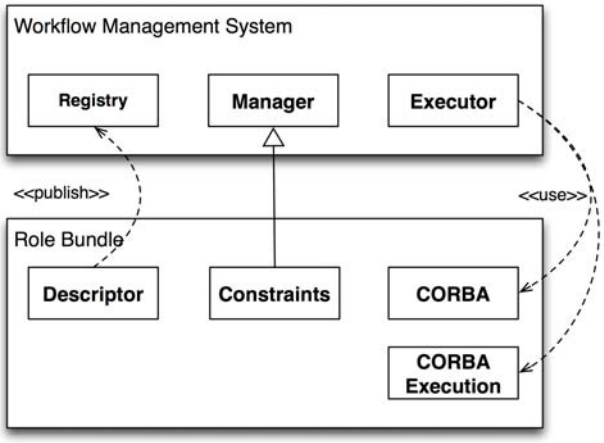


Abbildung 7.8: Aufbau eines Role-Bundles

ähnliche Ansätze können nicht derart auf unerwartete Ereignisse reagieren. Die Ausnahmebehandlungsstrategien müssen im Voraus feststehen. Diese Art von Flexibilität durch (Über-)Konfiguration ist eine Form der Adaption. In [182] wird ein Ansatz für die Adaption von Workflows durch Vererbung vorgestellt. Dieser Ansatz ist mit einer rollenbasierten Workflow-Engine vollständig umsetzbar, mit dem Zusatz, dass auch Adaptionen zur Laufzeit möglich sind. Ein Ansatz auf Vererbungsbasis ist zu statisch. In dem einfachen Beispiel ist es zwar möglich, eine Aktivität zu implementieren, der beide Funktionen – Webservice und JavaClass – unterstützt. Um aber eine dritte Funktionalität zu unterstützen, muss die Anwendung neu gestaltet werden.

7.3 Rollenbasierte Rekonfiguration der Workflow-Engine

Die Rollenorientierung ermöglicht zudem noch eine erweiterte Flexibilität, die über die der Instanzrekonfiguration hinausgeht: Die dynamische Erweiterung der Workflow-Engine selbst. Durch die Abschaffung des statischen Typsystems und der Einführung flexibler Rollentypen, können dynamische Softwarearchitekturen entstehen, darunter auch flexible Workflow-Engines.

In einer auf Rollentypen basierenden Workflow-Engine ist die Funktionalität nicht in den Klassen, sondern in den Rollen deklariert. Diese Rollen werden in abgeschlossenen *Role-Bundles* verwaltet (siehe Abschnitt 7.2.3), die sowohl die Rolle und Gegenrolle sowie alle notwendigen Rollen-Constraints, in Form von Managererweiterungen, beinhalten. Wird bei einer Workflow-Engine eine neue Funktion benötigt, die im Auslieferungszustand nicht vorhanden ist, kann diese durch Nachladen eines Role-Bundles nachträglich integriert werden. Werden die Role-Bundles auf einem zentralen Server gelagert, kann jede Engine nach Bedarf Role-Bundles nachladen.

Abbildung 7.8 zeigt ein Beispiel für die Einbindung eines Role-Bundles. Hierbei wird die Rolle zur Nutzung von CORBA-Diensten eingebunden. Der Role-Bundle Descriptor enthält dabei alle notwendigen Informationen, welche neuen Rollen enthalten sind und welche neuen Constraints gelten sollen. Dieser wird von der Registry ausgelesen, die die neuen Informationen dem Manager und dem Executor übermittelt, die ihrerseits die neuen Komponenten nutzen.

7.3.1 Nachladen von Rollenmodellen

Im Bereich der Webservices wird die *Universal Description, Discovery and Integration* [22] (UDDI) benutzt, um Dienste zu beschreiben und zu klassifizieren. Dies ermöglicht es, Webservices nach bestimmten Kriterien aufzufinden. Die Technik kann z. B. dazu verwendet werden, zu einem bestimmten Service ähnliche Services aufzufinden, um diese dann in einem Late-Binding Ansatz auszutauschen [137]. In UDDI werden Services auf verschiedenen Ebenen (*Pages* genannt) beschrieben:

- In den *White Pages* werden die Stammdaten verwaltet, wie beispielsweise Name, textuelle Beschreibung, Kontakt, Person etc.
- Die *Yellow Pages* geben Auskunft über die Servicekategorie, Art der Industrie, Art der Dienstleistungen, geographische Position
- Wurden über diese Angaben bereits Dienste gefunden, vervollständigen die *Green Pages* das Bild. Hier finden sich Informationen über die Adressierung und den Zugriff auf den Dienst sowie die Dokumentation einzelner Methoden.

Mithilfe dieser Registratur werden Services klassifiziert und können gefunden werden, um sie beispielsweise in Workflows einzubinden. Grundsätzlich stellt ein Role-Bundle auch einen Service dar. Es kapselt Funktionalität und bietet sie über verschiedene Methoden an. Es kann von einem Server nachgeladen und in die Engine integriert werden. Wenn eine Engine einen Workflow ausführen soll, prüft der Manager, ob alle benötigten Rollen sowie Gegenrollen vorhanden sind. Ist dies nicht der Fall, meldet er dies und der Workflow kann nicht ausgeführt werden. Mithilfe eines UDDI Servers für Role-Bundles, können die benötigten Rollen aufgefunden, nachgeladen und integriert werden. Danach kann der Workflow planmäßig ausgeführt werden.

Abseits der Adaption von Workflowinstanzen ergeben sich weitere Einsatzfelder. Eine Workflow-Engine auf Basis des hier vorgestellten rollenbasierten Workflowmodells ist grundsätzlich in der Lage, die verschiedensten Workflow-Beschreibungssprachen zu importieren und auszuführen, solang die Sprache grundsätzlich mit den Kontrollflusskonstrukten von Workflow-Nets kompatibel ist. Wie in [187] gezeigt, ist dies bei den gebräuchlichsten Sprachen ohne weiteres möglich. Die größten Unterschiede zwischen den einzelnen Workflow-Beschreibungssprachen existieren bei den unterstützten Funktionalitäten. Bei der Migration eines Workflowmodells zu einer anderen Engine, die grundsätzlich die gleiche Sprache unterstützt, kann es trotzdem zu Problemen kommen. Ein Beispiel dafür ist die Sprache BPEL, die zwar standardisiert wurde, allerdings wurden bei der Umsetzung der einzelnen Hersteller proprietäre Erweiterungen oder Einschränkungen eingebracht. So kann beispielsweise der User Task in Oracle BPEL nicht mit der Workflow-Engine ActiveBPEL ausgeführt werden. Problematisch sind demnach nicht die Kontrollflussstrukturen, sondern die nicht vorhandenen Funktionalitäten. Eine rollenbasierte Workflow-Engine besitzt diesen Nachteil nicht, denn sie kann mit neuen Funktionalitäten nachgerüstet werden. Zum Importieren von Workflows verschiedener Sprachen in diese Workflow-Engine ist ein zweistufiges Verfahren notwendig. Der erste Schritt besteht aus der Abbildung der Kontrollfluss-Strukturen auf die des Workflow-Nets. Ein weiterer Schritt bildet dann die Ressourcen und Aktivitätsbeschreibungen auf die Rollen ab, wobei nicht vorhandene Rollen über UDDI nachgeladen werden können. Sind diese dort nicht auffindbar, kann der Importvorgang nicht vollständig abgeschlossen werden. Die Rolle kann aber zu jedem beliebig späteren Zeitpunkt nachinstalliert werden.

7.3.2 Domänenübergreifende Workflow-Beschreibungssprachen

Neben dem Import von verschiedenen Workflowmodellen aus verschiedensten Domänen, gibt es einen weiteren Vorteil einer rollenbasierten Workflow-Engine. Die verschiedenen Rollen, die in einer Engine eingesetzt werden, vereinen verschiedene Domänenkonzepte, wie sie in den bisherigen Implementierungen nicht vorkommen. So können Rollen zum Aufruf von Open Geospatial Webservices [157] Seite an Seite mit SOAP- oder EJB-Rollen verwendet werden. Wird ein BPEL-Workflow importiert, stehen dem Designer anschließend nicht nur die Basis-BPEL Funktionen bereit, sondern auch BPEL-fremde. Er ist nicht mehr an das BPEL-Sprachkorsett gebunden und ist offen für verschiedene Domänen, die durch die Rollen der Engine bereitgestellt werden. Zum Beispiel können EJB-RMI Aufrufe ein Webservice-Invoke ersetzen oder ergänzen. Mit diesem Mashup aus verschiedenen Domänenkonzepten entstehen einzigartige Workflow-Beschreibungen.

Diese Möglichkeiten gehen über den aktuellen Stand der Technik hinaus. Es gibt verschiedene WfMS, die auch frei zu definierende Aktivitäten zulassen, diese aber sind nur bedingt vergleichbar. Beispielsweise werden in jBPM die Funktionen durch Java-Klassen beschrieben. In diesen Java-Klassen können beliebige Aufrufe oder Berechnungen ausgeführt werden. Es können hierbei auch Konzepte verschiedener Domänen verschmolzen werden, so dass eine rollenbasierte Engine ebenfalls in der Lage ist, EJB oder Open Geospatial Webservices aufzurufen. Diese Lösung ist aber auf eine einzelne Instanz der jBPM-Engine beschränkt und die verschiedenen Domänenkonzepte sind nicht erstrangige Workflow-Konstrukte, so dass ihre Wiederverwendbarkeit nicht gegeben ist.

Ein rollenbasiertes WfMS kann nicht nur verschiedene Domänenkonzepte miteinander verbinden, sondern es lassen sich auch verschiedene Varianten für spezielle Anwendungsfälle oder Plattformen maßschneidern. Eingebettete Systeme (Embedded Systems), wie sie in der Automobil- oder Unterhaltungselektronikbranche genutzt werden, besitzen zumeist eingeschränkte Ressourcen und Kommunikationsmöglichkeiten. Ein voll ausgestattetes WfMS benötigt zu viele Ressourcen, die nicht zur Verfügung stehen. Auch können komplizierte Berechnungen nicht lokal ausgeführt werden, da die Rechenleistung nicht vorhanden ist. Ein rollenorientieres WfMS ist in der Lage, auf diese Bedürfnisse zugeschnitten zu werden, indem nur ein Basissatz an Rollen, maßgeschneidert für den Anwendungsfall, ausgeliefert wird. Die Rollen dienen als flexibler Unterbau einer Produktlinie für WfMS. Diese können für sowohl für eingebettete Systemen als auch für High-End-Server konfiguriert werden und zudem zur Laufzeit verlängert werden (siehe Abschnitt 7.2.3).

Ein auf spezielle Anwendungsfälle zugeschnittenes WfMS benötigt auch eine Beschreibungssprache für die einzelnen Workflows. Da die Funktionalitäten aus verschiedenen Workflowdomänen stammen, existiert hierfür keine einheitliche Sprache. Einzige Konstante sind die Kontrollflussstrukturen, die auch die Basis für die Rollen bilden. Es ist demnach notwendig, für diese Anpassungen die passende domänenspezifische Workflow-Beschreibungssprache (DSL[22]) zu entwickeln. Diese beinhaltet anschließend nur die Konzepte für den Kontrollfluss und die jeweils verfügbaren Rollen, womit das speziell angepasste WfMS beschreibbar wird. Entwicklungswerkzeuge zur (automatischen) Erstellung auf Basis eines Modells existieren bereits. Vertreter sind beispielsweise EMFText [77] oder XText[23], die in diesem Falle eine solche Sprache erzeugen können.

[22] Domain specific language.
[23] http://www.eclipse.org/Xtext/ (zuletzt aufgerufen am 25.02.2011).

Damit geht der hier vorgestellt Ansatz über die einfache Erweiterbarkeit von Workflowmodellen, wie sie beispielsweise in [115, 160, 153] vorgestellt wurden, hinaus.

7.4 Autonomer Rekonfigurator

Auf Grundlage des rollenbasierten Workflowmodells und der rollenbasierten Workflow-Engine sind eine Vielzahl von Laufzeitadaptionen möglich. Diese befinden sich auf einer anderen Ebene als die bisherigen Kontrollflussadaptionen aus Kapitel 3. Das neue Workflowmodell besitzt als Kern ein Workflow-Net, das für die Kontrollflussbeschreibung zuständig ist. Dieses bildet den Kern, der durch Rollen verfeinert wird, der Kontrollfluss bleibt jedoch stabil. Die Stärke dieses Ansatzes besteht in der Einfachheit, Funktionen bzw. Konfigurationen von Workflows zu verändern, bzw. neu zu setzen. Konfiguration bezeichnet dabei eine oder mehrere Rollen eines Kontextes, die einen gemeinsamen Zweck erfüllen, wie z. B. der Aufruf eines Webservices.

Es gibt drei grundsätzliche Rekonfigurationen im Zusammenhang mit Konfigurationen:

1. *Wechsel zwischen Konfigurationen:* Das bereits angesprochene Beispiel aus Abschnitt 7.2.3 zeigt, dass zwischen verschiedenen Konfigurationen einer bestimmten Aktivität[24] gewechselt werden kann, indem die playedby-Verbindung zu der Rolle `WebService` aufgelöst wird und eine neue Verbindung zu *JavaClass* gesetzt wird. Dadurch ist es möglich, zwischen verschiedenen Konfigurationen zu wechseln.

2. *Hinzufügen von Rollen:* Bei dem Hinzufügen einer weiteren Rolle entsteht eine weitere Konfiguration. Bei diesem Vorgang müssen alle Rollen-Constraints beachtet werden. Es kann keine Rolle an eine Klasse gebunden werden, wenn dadurch beispielsweise ein *role-prohibition*-Constraint (Abschnitt 7.1.1) verletzt wird. Dabei ist zusätzlich zu beachten, dass nicht nur die Beziehungen zwischen Rolle und Klasse betrachtet werden, sondern auch die Beziehungen zwischen den Rollen selbst. Dabei spielen die Constraints der Role-Bundles eine entscheidende Rolle, denn diese werden vom Manager in die Überprüfung mit einbezogen. Sind eine oder mehrere Rollen erfolgreich hinzugefügt worden, können sie beim Wechsel zwischen Konfigurationen berücksichtigt werden.

3. *Entfernen von Rollen:* Für das Entfernen einer Rolle gelten ebenfalls Constraints. Dabei muss darauf geachtet werden, dass die Rolle nur entfernt werden kann, wenn dadurch keine Abhängigkeitsbedingungen von anderen Rollen verletzt werden, wie z. B. *role-implication*.

Ein Wechsel der Konfiguration beschränkt sich jedoch nicht nur auf Operationen, sondern kann auch die Bindung eines Nutzers zu einem User Task betreffen. Ist der bisher vorgesehene Bearbeiter der Aufgabe nicht verfügbar, kann die ihn repräsentierende Rolle, aus dem *Organizational Context*, durch eine andere ersetzt werden. Dies geschieht durch den Wechsel der Konfiguration, wobei die neue Personenrolle vorher hinzugefügt worden sein kann.

Das Hinzufügen von Rollen zur Laufzeit ermöglicht auch Adaptionen, die unter dem Begriff des *Late Bindings*[25] bzw. Unterspezifikation (Abschnitt 2.3.3) bekannt sind. Bei

[24] Tatsächlich Transition, denn es handelt sich um ein Workflow-Net.
[25] Auch *Late Modeling*.

diesem Ansatz werden bestimmte Aspekte[26] zur Designzeit nicht definiert. Dies kann z. B. eine nicht zugeordnete Operation oder User Task ohne feste Personenzuweisung sein. Der Prozess ähnelt einem Template, das zur Laufzeit ergänzt werden muss. Das Template gibt dabei nur durch das Interface der Platzhalter die grundlegenden Anforderungen vor. Dieser Ansatz ermöglicht eine einfach zu handhabende Laufzeitflexibilität, da sie sich auf die Platzhalter im Template beschränkt. Die Auswahl der einzufügenden Elemente kann manuell, durch einen Administrator oder automatisch geschehen, wie in [137] gezeigt wurde.

Dieser Ansatz ist mit den oben genannten Rekonfigurationen abzubilden. Dabei kann auf zwei Arten vorgegangen werden. Ist es gewollt, beispielsweise den allgemeinen Typ der Operation zur Designzeit festzulegen, kann eine allgemeine Rolle *AbstractWebService* eingebunden werden, von der alle Webservice-Aufrufe erben. Damit ist sichergestellt, dass nur Webservices an dieser Stelle eingebunden werden können. Zur Laufzeit kann dann durch Hinzufügen einer Rolle diese Basisrolle verfeinert werden, so dass ein Aufruf des Webservices gelingt. Ist eine derartige Einschränkung nicht notwendig, kann auf die allgemeine Rolle verzichtet werden, bzw. die allgemeinste des betreffenden Kontextes gewählt werden.

Diese verschiedenen Rekonfigurationen können durch einen Administrator jederzeit manuell durchgeführt werden. Es ist zudem möglich, die in Kapitel 4 - 6 genannten Direktoren zur Steuerung der Adaption zu nutzen. Durch die neuen Rekonfigurationen wird aus dem *autonomen Weber* ein *autonomer Rekonfigurator*.

Dabei muss beachtet werden, dass die Situationsbeschreibungen der Cases für diese Adaptionsform und die Intention für die rollenbasierte Adaptionen genutzt werden können. Die Situationsbeschreibung umfasst sowohl alle instanzrelevanten Variablen, als auch die Ausnahmen und die Laufzeiten aller ausgeführten Aktivitäten. Diese Indikatoren sind ausreichend für diese Adaptionsart. Es können über die Ausnahme Ausfälle eines Webservices registriert werden, wodurch ein Konfigurationswechsel initiiert werden kann. Auch kann anhand der Laufzeiten einzelner Aktionen festgestellt werden, welche eine zu lange Antwortzeit aufweist. Wird ein gewisser Schwellwert überschritten, kann ebenfalls ein Konfigurationswechsel gestartet werden. Gleiches gilt für das Late Binding.

Weiterhin weiterverwendet werden kann das Konzept der Intentions. Diese beschrieben bei der Adaption durch Verweben das Einweben eines Subprozesses mit einem expliziten Rücksprungpunkt. Durch die Einführung eines neuen Typs der Intention ist jedoch eine Unterscheidung der beiden Rekonfigurationsansätze notwendig. Ein Typ für die bekannte Verwebung, ein weiterer für die Rollenadaption. Diese Intention beschreibt die Aktivierung oder Deaktivierung einer Rolle (Konfigurationsänderung), bzw. ob neue Rollen hinzugefügt bzw. gelöscht wurden.

Die Berechnung der Adaptionen übernimmt weiterhin der Direktor. Der Executor hingegen ist der Kooperationspartner für den Koordinator. Dieser sendet die Adaptionsanweisungen nicht an die Workflow-Engine sondern an den Executor, der für die Verwaltungen der Rollenbindungen zuständig ist. Der Manager überwacht parallel die Einhaltung der Rollen-Constraints.

Durch die Vereinbarkeit der Ansätze können beide Rekonfigurationsansätze gemeinsam in einem WfMS genutzt werden. Zum einen können Kontrollflussänderungen durch

[26] Aspekte deshalb, da sich hierbei auf die Workflowaspekte bezogen wird.

Einweben, zum anderen Konfigurationsänderungen durch Rollenadaption gelöst werden. Diese Kombination ermöglicht komplexe Adaptionen und Exception Handling-Strategien. Abbildung 7.1 zeigt ein Beispiel für eine solche Strategie. Ein User Task kann nicht ausgeführt werden, da der zugewiesene Bearbeiter nicht zur Verfügung steht. Damit der Workflow nicht abgebrochen werden muss, ist eine Adaption notwendig, die aus verschiedenen Teilen besteht: Der Zuweisung eines neuen Bearbeiters und die wiederholte Ausführung der Aufgabe. Beide Adaptionsansätze einzeln betrachtet stoßen in diesem Szenario an Grenzen. Die Rollenadaption kann eine Aktivität nicht nochmals ausführen, aber den Bearbeiter wechseln. Die Kontrollflussadaption ermöglicht die Wiederausführung der Aktivität, allerdings ist es nur schwer möglich, einen neuen Bearbeiter zuzuweisen. Zusammen können beide Adaptionsarten dieses Problem lösen. Eine Intention besteht in diesem Fall aus zwei Teilzielen. Im Ersten erfolgt eine Rollenadaption, die den Bearbeiter austauscht, im Zweiten wird ein Leer-Workflow eingewoben, wobei der Rücksprungpunkt vor dem User Task liegt. Dadurch wird dieser erneut ausgeführt. Beide Adaptionsansätze ergänzen sich demnach und ermöglichen ein hoch-rekonfigurierbares WfMS.

7.5 Zusammenfassung

Verwebte Workflows basieren auf einer generischen Adaption durch das Einweben von Teilworkflows in eine laufende Workflowinstanz. Gesteuert wird dies durch einen Direktor, der auf Grundlage einer Wissensbasis Adaptionen vorschlägt und je nach Ausgestaltung neue Adaptionsmöglichkeiten lernt[27]. Dieser Adaptionsansatz beeinflusst den Kontrollfluss einer Instanz und eignet sich gut für komplexe Eingriffe in den Workflow. Für kleine Änderungen ist dieser Ansatz allerdings umständlich einzusetzen, denn er bedingt immer einer Kontrollflussadaption, auch wenn nur der aufzurufende Webservice auszutauschen ist. Um auf diese Probleme reagieren zu können, wurde ein rollenbasiertes Workflowmodell entwickelt.

Die Rollenorientierung ist ein erweitertes Konzept der Objektorientierung, das für dynamische Anwendungen bestens geeignet ist. Kern dieses Ansatzes ist das Konzept der Rolle, die der einer Theaterrolle eines Schauspielers ähnelt. Die Rolle wird von einer Objektinstanz gespielt, wobei diese Relation zur Laufzeit aktiviert und deaktiviert werden kann. Wird eine Rolle gespielt, kann sie das Verhalten der Basisklasse beeinflussen und verändern. Dadurch kann zur Laufzeit aus einem *Invoke* für einen Webservices ein *RMI*-Aufruf für ein EJB werden. Basierend auf dem Konzept der Rolle wurde ein neues Workflowmodell erstellt. In diesem bildet das Modell eines Workflow-Nets gleichzeitig die Basis für die Rollen und die *Verhaltensperspektive* der Workflows. Zudem wurden für die anderen Basisaspekte grundlegende Rollen eingeführt, die von Implementierungen verfeinert oder erweitert werden können.

Dieses Rollen-Workflowmodell ist die zweite Basis für vollständig rekonfigurierbare Workflows. Es bietet Adaptionsmöglichkeiten, um alle Perspektiven eines Workflows[28] zu rekonfigurieren. Damit können Adaptionen durchgeführt werden, wie sie aus den Bereichen des Late-Bindings und *flexibility by selection* bekannt sind.

[27] Ausgenommen ist hier der Statechart-Direktor.
[28] Außer der Verhaltensperspektive.

Die Beschreibung der Rekonfigurationen in rollenbasierten Workflowmodell ist in die vorgestellten Direktoren-Ansätze integrierbar. Dadurch wird eine Kombination mit der Adaption durch Verweben möglich, wodurch komplexe, zusammengesetzte Adaptionen erreicht werden können. Durch die neuen Adaptionsmöglichkeiten entsteht aus dem *autonomen Weber* (siehe Abschnitt 3.2.2) ein *autonomer Rekonfigurator*.

Das rollenbasierte Workflowmodell ermöglicht nicht nur vollständig rekonfigurierbar Workflows, sondern auch ein rekonfigurierbares WfMS. So können Funktionalitäten über Role-Bundles zur Engine hinzugefügt werden, was auch die Unterstützung verschiedenster Workflow-Beschreibungssprachen begünstigt. Ebenfalls möglich sind auf den Kunden zugeschnittene Workflow-Beschreibungssprachen bzw. je nach Anwendungsfall maßgeschneiderte WfMS. Dies geht weit über die ursprüngliche Idee – der Laufzeitadaption von Instanzen – hinaus.

Kapitel 8

Realisierung und Evaluation

In den vorangegangenen Kapiteln 3 - 7 wurden sowohl Adaptionsverfahren als auch Methoden zur Adaptionssteuerung vorgestellt. Wie Abbildung 8.1 zeigt, existieren die beiden Adaptionsmöglichkeiten: Adaption durch Verweben und das rollenbasierte Workflowmodell. Durch die Kombination beider Verfahren ist eine vollständige Abdeckung der fünf zentralen Workflowperspektiven für die Änderungsoperationen gegeben. Für die Adaptionssteuerung wurden sowohl der allgemeine Direktor-Ansatz eingeführt als auch drei verschiedene Lösungskonzepte – (semantischer) BDI-Direktor und Statechart-Direktor – vorgestellt, die je nach Anwendungsgebiet Vor- und Nachteile besitzen. Ein wichtiger Aspekt, der beim semantischen BDI-Direktor eingeführt wurde, ist zudem die Kommunikation zwischen den einzelnen Direktoren.

Diese verschiedenen Ansätze wurden in der *Open Service Process Platform* (OSPP) [136] umgesetzt. Die Entwicklung dieser Plattform erfolgte unter der Maßgabe, ein adaptives WfMS zu entwickeln. Dabei sollten nicht nur die Adaptionssteuerungen umgesetzt werden. Es war zudem wichtig neuartige Adaptionssteuerungen zu integrieren sowie die Funktionalitäten des WfMS erweiterbar oder austauschbar zu gestalten. Beispielsweise kann das WfMS je nach Einsatzzweck sowohl mit als auch ohne graphische Modellierung der Workflows eingesetzt werden. Dieses Kapitel gibt einen Überblick über die Konzeption und Realisierung der *Open Service Process Platform*. Die Schwerpunkte lagen dabei auf der Realisierung der einzelnen Direktoren, der Integration unterschiedlicher Adaptionstechniken in einem einzelnen WfMS und der Umsetzung der rollenbasierten Workflow-Beschreibungssprache, inklusive der Adaption durch Verweben.

Mit der Open Service Process Platform wurde zwei Mal am IEEE Service Computing Contest teilgenommen. Schwerpunkt dieses Wettbewerbs ist es, die verschiedenen Gebiete des Service Computings – Webservices, Cloud Computing, etc. – an praxisnahen Projekten umzusetzen. Die Einreichung umfasst zwei Teile: (1) ein Papier über die theoretischen Grundlagen und die Konzeption der Lösung sowie (2) eine lauffähige Online-Demo der Anwendung. Im Jahre 2008 konnte die Konzeption eines rekonfigurierbaren WfMS auf Basis der BDI-Direktoren und des rollenbasierten Workflowmodells den *Sieg* beim *Service Computing Contest*[1][2] davon tragen [136]. Zwei Jahre später konnte OSPP in das Finale[3] des auf das Cloud-Computing ausgerichteten *Service Computing Contest 2010* [139] einziehen.

OSPP ist auch die Basis für die Evaluation, die den zweiten Teil des Kapitels darstellt. Hierbei wird die Verwendbarkeit der verschiedenen Lösungen an einer bisher nicht vorgestellten Problemstellung gezeigt: Die verteilte Ausführung einzelner Workflowfragmente.

[1] http://iscc.servicescomputing.org/2008/index.htm (zuletzt aufgerufen am 25.02.2011).
[2] http://iscc.servicescomputing.org/2008/Results.htm (zuletzt aufgerufen am 25.02.2011).
[3] http://iscc.servicescomputing.org/2010/Finalists.htm (zuletzt aufgerufen am 25.02.2011).

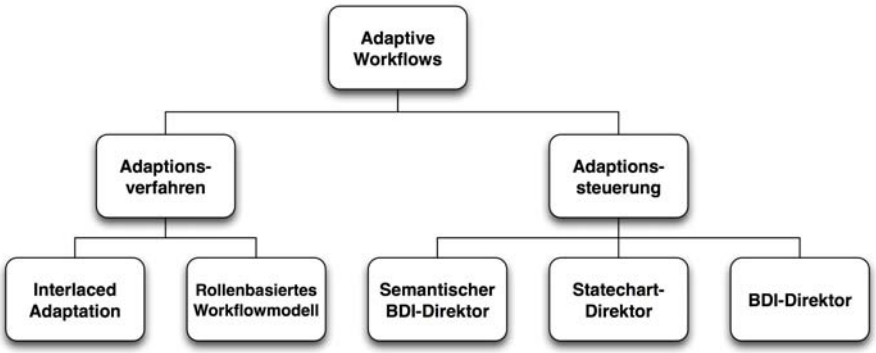


Abbildung 8.1: Vollständige Rekonfiguration

Herkömmliche WfMS werden auf einem Rechnerknoten ausgeführt. Es existieren vereinzelte verteilte WfMS, bei denen ein Workflow auf verschiedenen Rechnerknoten ausgeführt werden kann. In diesem Kapitel wird gezeigt, wie während der Ausführung einer Instanz die Bearbeitung einzelner Workflowfragmente auf verschiedene Rechnerknoten auslagern werden kann. Dies ist beispielsweise bei Genexpressionsprozessen notwendig, denn aufgrund der großen Ressourcenanforderungen dieser Workflows ist eine Lastverteilung zwischen verschiedenen Rechnerknoten notwendig. In der Evaluation konnte gezeigt werden, dass mit Hilfe der Adaption durch Verweben und des rollenbasierten Workflowmodells, Genexpressionsprozesse zur Lastverteilung auf verschiedene Workflow-Engines verteilt werden können. Zugleich konnte der Direktor-Koordinator-Ansatz genutzt werden, um die nötige Steuerung der Adaptionen vorzunehmen.

8.1 Open Service Process Platform

Bei der Umsetzung der Open Service Process Platform wurde neben der Implementierung der Workflow-Engine und der Rekonfigurationsansätze auf *Erweiterbarkeit*, *Vernetzung* und *einfachen Zugriff* geachtet.

Erweiterbarkeit Die Workflow-Engine und die Direktoren wurden in der Programmiersprache Java umgesetzt. Das Rahmenwerk für all diese vorgestellten Komponenten bietet die *OSGi Plattform* [120], hier im speziellen die Equinox-Implementierung durch die Eclipse Foundation. Die OSGi Platform ist ein offener, auf Java basierender, Industriestandard. Entwickelt wurde er als serviceorientierte und komponentenbasierte Laufzeitumgebung für das Management von unterschiedlichen Geräteklassen, wie beispielsweise Fernseher oder Industrieanlagen. Hauptziel ist dabei die Interoperabilität zwischen Anwendungen auf verschiedenen Geräten zu gewährleisten.

Das Hauptkonzept von OSGi ist die Erweiterbarkeit der Anwendung durch *Bundles*. Ein Bundle ist eine Zusammenstellung verschiedener Java-Klassen, die eine abgeschlossene Funktionalität repräsentieren. Im Gegensatz zu klassischen Java-Anwendungen bietet die OSGi-Plattform *Extension Points* (Erweiterungspunkte), die die Bundles adressieren können. Durch diese kann die Basisfunktionalität erweitert werden. Ein Beispiel hierfür ist Eclipse. Die Java-IDE[4] basiert ebenfalls auf OSGi und stellt verschiedenste Extension

[4]Integrated Development Environment (integrierte Entwicklungsumgebung).

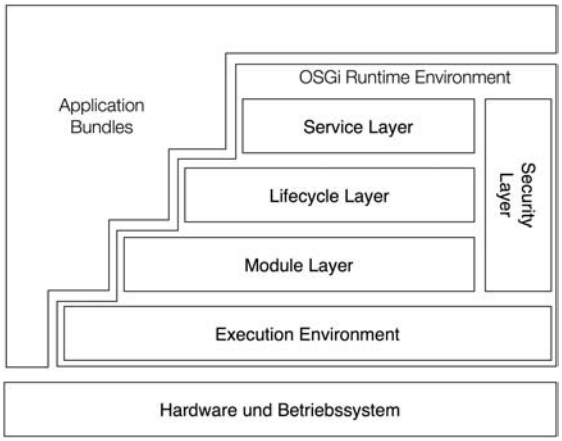


Abbildung 8.2: OSGi Laufzeitumgebung

Points zur Verfügung. Das folgende Beispiel zeigt die Nutzung des Extension Points `org.eclipse.ui.editors`.

Listing 8.1: Nutzung eines Extension Points

```
<extension name="workflowEditor" point="org.eclipse.ui.editors">
  <editor class="org.tud.ospp.editors.WorkflowEditor"
          default="false" icon="icons/workflow.gif"
          id="org.tud.ospp.editors.WorkflowEditor"
          name="Workflow"/>
</extension>
```

Zur Nutzung eines Extension Points muss jedes Bundle zwei Dateien bereitstellen. Das Manifest (`MANIFEST.MF`) beschreibt, welche anderen Bundles benötigt werden, damit die gewünschte Funktionalität zur Verfügung gestellt werden kann. Die `plugin.xml` beschreibt anschließend, welche Extension-Points das Bundle nutzt. Im oberen Beispiel wird die Anwendung um einen neuen Editor zur Bearbeitung von Workflows erweitert. Dies bedeutet, dass Eclipse diesen Editor in die Liste der verfügbaren Editoren aufnimmt und auf Wunsch des Nutzers darstellt. Bundles können dabei selbst Extension Points definieren und diese zur Erweiterung freigeben.

Zur Handhabung der Bundles stellt die OSGi Laufzeitumgebung verschiedene Funktionalitäten zur Verfügung, wie in Abbildung 8.2 dargestellt. Aufbauend auf Java (*Execution Environment*) definiert der *Module Layer* nicht nur das zentrale Kompomentenmodell, sondern ist auch für das Laden der Klassen und der Bundles sowie das Auflösen der Bundle-Abhängigkeiten (auf Basis des Manifests) zuständig. Über den *Lifecycle Layer* werden die Installation, das Starten und Stoppen sowie die Deinstallation eines Bundle gesteuert. Diese Funktionen stehen auch zur Laufzeit zur Verfügung. Über den *Service Layer* können zusätzlich Dienste bereitgestellt oder genutzt werden. Der querschneidende *Security Layer* nutzt und erweitert die bestehende Java Security Architektur. Hier werden die Berechtigungen verwaltet und kontrolliert, so dass beispielsweise festgestellt werden kann, welche Bundles bestimmte Services verwenden dürfen.

Somit stellt OSGi eine flexible Plattform zur Verfügung, die ideal für die Umsetzung von OSPP ist. Abbildung 8.3 zeigt die entstandene Architektur der Open Service Process Platform.

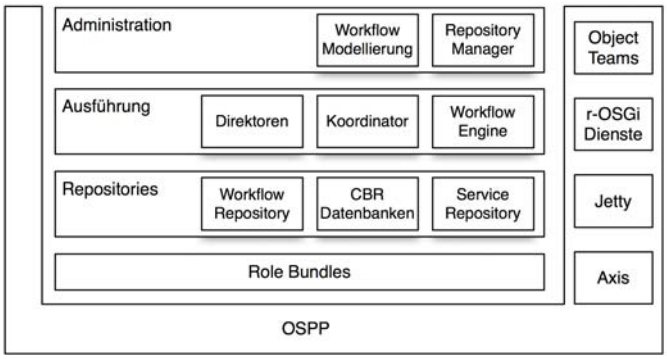


Abbildung 8.3: Überblick über die Open Service Process Platform

Mit OSGi und Java lassen sich die in Kapitel 7 vorgestellten Rollenkonzepte nicht umsetzen. Deshalb wurde zusätzlich *ObjectTeams* (OT) [81] in die Architektur eingeführt. Bei ObjectTeams handelt es sich um eine Spracherweiterung von Java, die die Konzepte der rollenbasierten Programmierung implementiert. Diese ermöglicht, dass normale Java-Klassen weiterhin genutzt werden können. Mit dieser Erweiterung kann ein Objekt durch Rollen verändert werden, d. h. eine Rolle spielen.

Darauf aufbauend wurden *Role-Bundles* umgesetzt, die bereits in Abschnitt 7.2.3 beschrieben wurden. Damit werden nicht nur die Grundrollen bereitgestellt, vielmehr können dadurch dynamisch neue Rollen der Workflow-Engine zur Verfügung gestellt werden. Dazu enthält jedes Role-Bundle die neuen Rollen (Rolle und Gegenrolle) und die notwendigen Rollen-Constraints. Durch die Verwendung der Extension Points des Executors und des Manager[5] in der `plugin.xml` werden diese der Workflow-Engine bekannt gemacht. Role-Bundles können auch zur Laufzeit hinzugefügt werden. OSGi lädt automatisch[6] alle Bundles eines bestimmten Verzeichnisses. Einmal gestartet, bindet es allerdings keine neu hinzugekommenen Bundles mehr ein. Allerdings kann das Laden und Starten eines neuen Bundles jede auf OSGi basierende Anwendung auslösen, sofern sie Kenntnis von neuen Bundles hat.

Darüber befindet sich die Schicht mit den verschiedenen Repositories. Sowohl für das *Workflow Repository* als auch das *Service Repository* wurden eigene Bundles erstellt. Das Service Repository ist zum einen eine Datenbank, die verschiedene Links auf Webservices bereitstellt. Zum anderen werden hier auch die WSDL-Spezifikationen zwischengespeichert und für jeden Service ein Java-Stub mit Axis[7] generiert, so dass dieser Webservice und seine Datentypen in die Workflows eingebunden werden können. Für die drei Case-Datenbanken und die Ontologie wurden jeweils eigenständige Bundles geschrieben. Aufbauend auf diesen Repositories arbeiten die Direktoren, der Koordinator und die Workflow-Engine.

Vernetzung Ein wichtiger Aspekt der Direktoren ist, wie in Kapitel 6 beschrieben, die Kooperation. Diese muss dabei nicht auf die Direktoren innerhalb einer OSPP-Instanz beschränkt sein. Vielmehr ist es sinnvoll, auch andere Installationen von OSPP mit

[5] siehe Abschnitt 7.2.1.

[6] Durch Verwendung bestimmter Parameter im Manifest kann das automatische Starten eines Bundles verhindert werden.

[7] Java-Implementierung für das Simple Object Access Protocol, um sowohl WSDL-Dateien zu erzeugen, als auch Java-Stubs zu generieren. Verwendet wurden die Versionen 1.x und 2.x.

einzubeziehen. Um dies umzusetzen wurde das r-OSGi Framework benutzt. r-OSGi ist ein Framework zur Kommunikation und Verteilung von OSGi-Anwendungen. Für die Kommunikation kann eine OSGi-Anwendung `Services` (Dienste) definieren, die von anderen OSGi-Anwendungen genutzt werden können. Für OSPP wurden verschiedene Services definiert:

- Workflow Repository
  - Auflistung aller vorhandenen Workflows
  - Importieren eines Workflows
  - Löschen eines Workflows
- Workflow-Engine
  - Starten eines Workflows oder eines Fragments (selektive Adaption)
  - Beenden eines Workflows
  - Pausieren eines Workflows
  - Senden der aktuellen Situation und des Status
- MessageQueue (Blackboard)
  - Publizieren einer Nachricht
  - Löschen einer Nachricht
- Direktor
  - Senden einer Nachricht
  - Empfangen einer Nachricht
  - Empfangen der aktuellen Situation

Jede OSPP Instanz stellt die Services zur Verfügung und enthält zusätzlich auch die Stubs, um selbst Services anderer Instanzen nutzen zu können. Damit die einzelnen OSPP-Installationen wissen, welche weiteren existieren, muss es eine Art Register geben, bei dem sie nachfragen können. Für die Umsetzung gibt es viele Möglichkeiten. In der prototypischen Implementierung wurde dies durch eine Master-Slave Konstruktion gelöst. Dabei spielt eine OSPP-Instanz den zentralen Knoten (Master). Alle anderen Installationen (Slaves) kennen den Master und melden sich beim Start bei ihm an bzw. beim Herunterfahren wieder ab. Somit entsteht ein zentrales Register, bei dem jede Instanz nachfragen kann.

Einfacher Zugriff Um einen einheitlichen und einfachen Zugriff auf alle Funktionen von OSPP zu ermöglichen, wurde eine Webanwendung entwickelt, die in der gleichen OSGi-Anwendung wie die Workflow-Engine verwendet werden kann. Diese bildet die oberste Schicht in Abbildung 8.3.

Zur Entwicklung der Webanwendung wurde das Eclipse-Frameworks RAP, die Rich Ajax Platform[8], genutzt. Die Funktionsweise ist vergleichbar mit denen klassischer Web-Frameworks, wie beispielsweise Struts[9] in Verbindung mit JavaServer Pages (JSP) und

[8] http://www.eclipse.org/rap/ (zuletzt aufgerufen am 25.02.2011).
[9] http://struts.apache.org/ (zuletzt aufgerufen am 25.02.2011).

Ajax[10]. Allerdings ist die Ausgangsbasis nicht HTML. Genutzt wird hingegen Java-Code, der auch in der Programmierung von Plugins für Eclipse zum Einsatz kommen könnte[11].

Um diese RAP-Anwendung in OSPP zu integrieren, wurde der *Jetty* Webserver verwendet, der über spezielle OSGi-Bundles eingebunden wurde. Dieser führt die Webanwendung aus.

Abbildung 8.4 zeigt die Web-Oberfläche von OSPP. Zentral ist das Modellierungswerkzeug zu sehen, in dem der Kontrollfluss des Workflows modelliert werden kann. Verfeinert werden die einzelnen Aktivitäten, indem aus dem *Object Rolespace* Rollen zugewiesen und konfiguriert werden. An der rechten unteren Seite sind verschiedene Werkzeuge für den Administrator angeordnet. Hier können die einzelnen Instanzen überwacht oder das Log betrachtet werden. Zudem wurden verschiedene *Views* (Ansichten) für die Repositories umgesetzt. In der Mitte unten ist das Workflow Repository dargestellt. Weitere Views für die Webservices oder die Case-Datenbanken sind ebenfalls vorhanden.

8.1.1 Interne Struktur von OSPP

Neben diesen übergreifenden Designzielen wurde bei der Entwicklung der *Open Service Process Platform* (OSPP) auf die Integration aller vorgestellten Adaptionslösungen Wert gelegt. OSPP wurde entworfen um jederzeit neue Direktoren oder Rekonfigurationsansätze integriert zu können. Abbildung 8.5 zeigt eine vereinfachte Darstellung der Komponenten, die direkt mit der Ausführung und Adaption in Zusammenhang stehen. Diese können in drei Hauptgruppen untergliedert werden. Die erste beinhaltet die verschiedenen Umsetzungen der Direktoren und alle damit im Zusammenhang stehenden Interfaces. Die zweite Hauptgruppe ist für die Interaktion zwischen den Direktoren und der Workflow-Engine zuständig. Die Workflow-Engine inklusive des rollenbasierten Workflowmodells bildet die dritte Hauptgruppe. Zusätzlich zu diesen Hauptgruppen existieren für die Analyse von Log-Files der Case-Extractor (siehe Abschnitt 5.2.3) und das Workflow Repository. Dieses beinhaltet nicht nur die Workflows, die durch Rollen beschrieben wurden, sondern es unterstützt zusätzlich jBPM und BPEL Workflows. In den folgenden Abschnitten werden diese Gruppen detaillierter vorgestellt.

Eingliederung unterschiedlicher Direktoren Das Workflow Repository (siehe Abbildung 8.5) beinhaltet alle Workflowmodelle von OSPP, die durch die Workflow-Engine instanziiert werden können. Damit die Workflow-Engine und vor allem der Direktor nicht verschiedene Schnittstellen für die unterschiedlichen Workflow-Beschreibungssprachen bereitstellen müssen, wurde eine einheitliche Schnittstelle zum Laden von Workflows entwickelt. Sie besteht aus mehreren Adaptern [63], welche die verschiedenen Workflowmodelle auf die internen Modelle abbilden. Es existieren Adapter für BPEL (1.x)[12] und jBPM[13]. Eine beispielhafte Abbildung des in Kapitel 7 vorgestellten Workflows ist in Abbildung 8.6 dargestellt. Damit ist sowohl der Direktor, als auch die Workflow-Engine

[10] Asynchronous JavaScript and XML.
[11] http://www.eclipse.org/swt (zuletzt aufgerufen am 25.02.2011).
[12] Die Beschränkung auf Version 1.x ist nicht fachlich bedingt, sondern ist auf den Implementierungszeitpunkt zurückzuführen.
[13] jBPM besitzt keine über die BPEL Spezifikation hinausgehenden Sprachkonstrukte, so dass der Adapter eine vereinfachte Implementierung des BPEL Adapters ist.

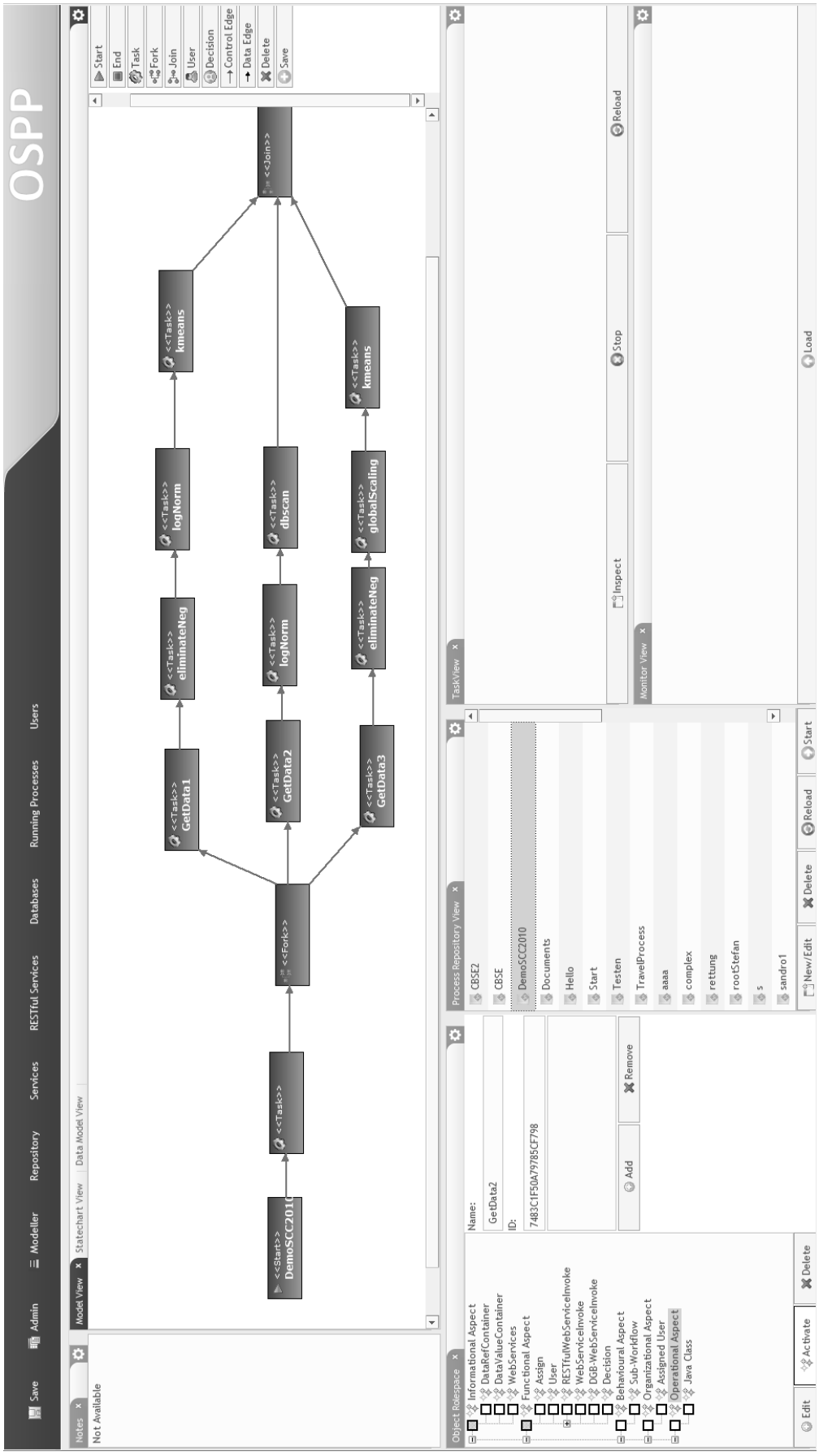


Abbildung 8.4: Webinterface der Open Service Process Platform

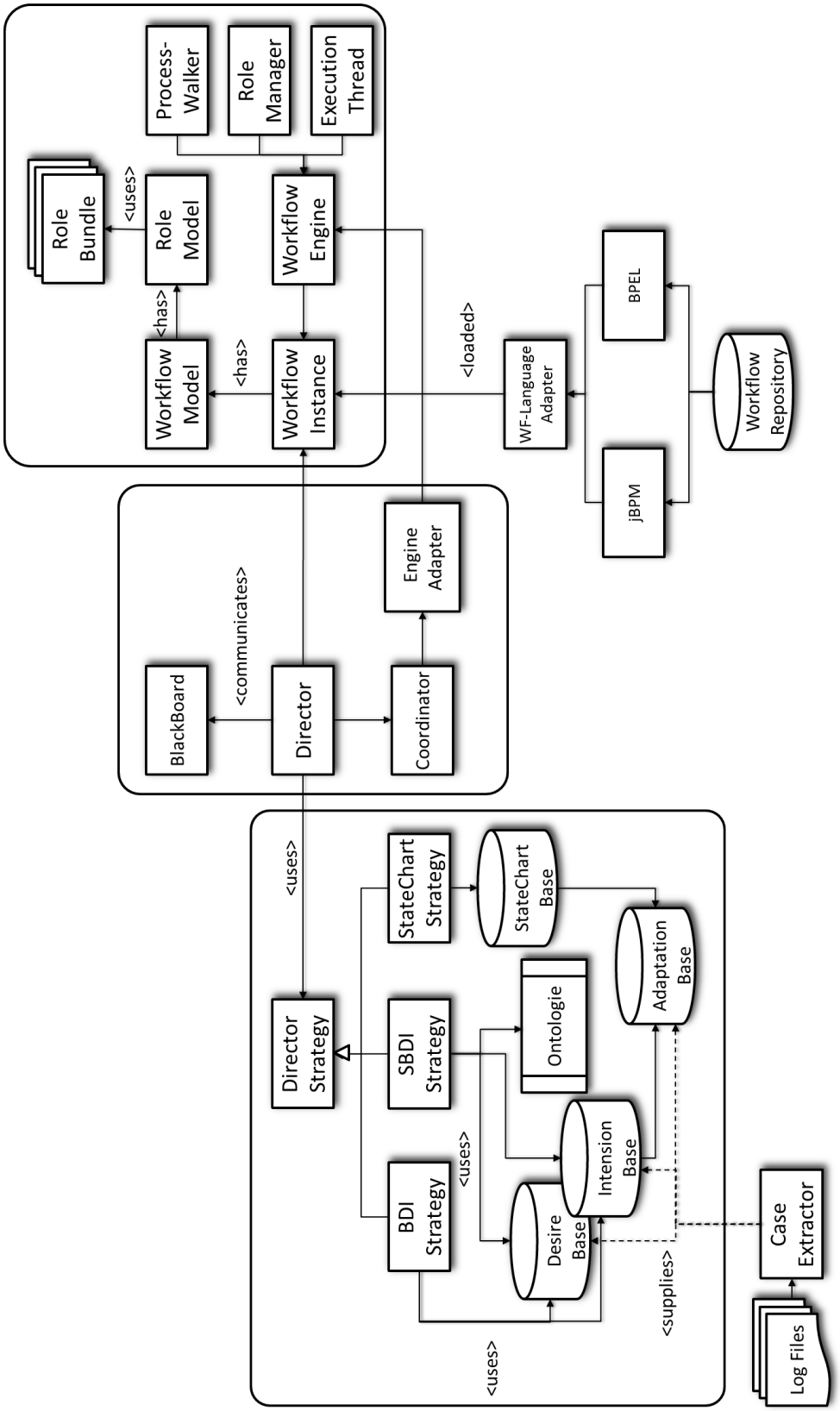


Abbildung 8.5: Architektur der Open Service Process Platform

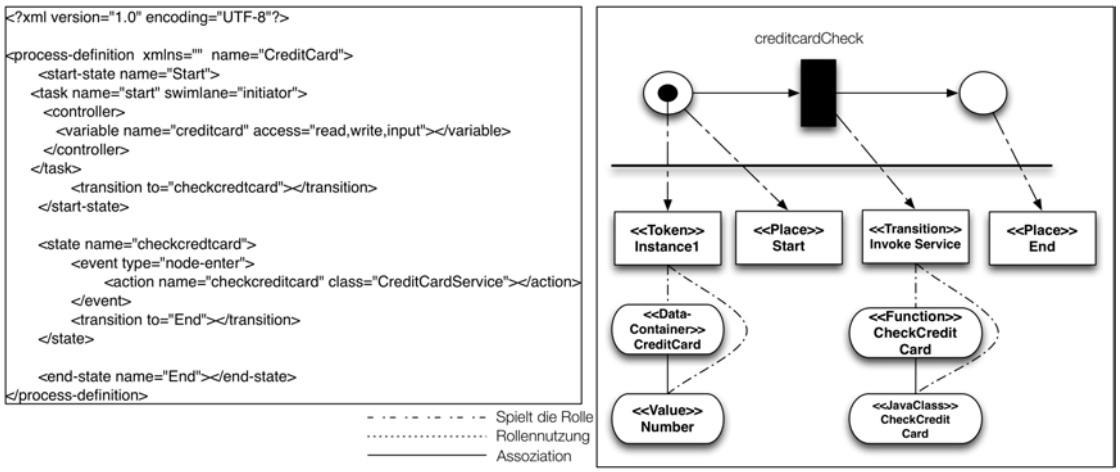


Abbildung 8.6: Abbildung von jBPM auf das rollenorientierte Workflowmodell

in der Lage, unterschiedliche Workflow-Beschreibungssprachen zu unterstützen. Die Adapter basieren dabei auf den Vorarbeiten von [187], die zeigen, wie verschiedene BPEL Konstrukte auf Workflow-Net-Fragmente abzubilden sind.

Nachdem der Workflow geladen und instanziiert wurde, muss er durch den Direktor überwacht werden, damit dieser notwendige Adaptionen vorschlagen kann. Dazu ist es notwendig, dass der Direktor über alle Änderungen bzgl. einer Workflowinstanz informiert wird. Zu diesem Zweck registriert sich jeder Direktor bei der Workflow-Engine für eine Workflowinstanz. Umgesetzt wurde dies über das Entwurfsmuster *Observer* [63]. Übertragen werden dabei zwei verschiedene Datensätze. Der erste beinhaltet das aufgetretene Ereignis, repräsentiert durch die Klasse `Event` und die betroffene Workflowkomponente. Es wurden die grundlegenden Events implementiert, wie sie in [176] beschrieben und in Abbildung 4.5 dargestellt sind. Dieser Datensatz ist vor allem wichtig für den Statechart-Direktor. Der zweite Datensatz wird durch die Klasse `Situation` repräsentiert. Sie beinhaltet das vollständige Bild der Workflowinstanz: Alle Variablen und aufgetretenen Events (zeitlich geordnet), Ausnahmen und das Laufzeitverhalten der Aktivitäten, d. h. wie lang beispielsweise ein Funktionsaufruf gedauert hat. Diese Informationen sind wichtig für die BDI-Direktoren. Die Situation fungiert gleichzeitig auch als standardisierte Schnittstelle zur Kommunikation mit dem Direktor. Dies ermöglicht eine vereinfachte Integration in andere WfMS.

Ziel bei der Entwicklung von OSPP war, eine Plattform für die existierenden und zukünftigen Direktoren zu erschaffen. Die verschiedenen Umsetzungen der Direktoren wurden unabhängig voneinander angefertigt. Dies machte die Entwicklung einer einheitlichen Schnittstelle und eines standardisiertes Protokolls für die Methoden notwendig. Das Interface `IDirector` besitzt neben verschiedenen Helfermethoden die zentralen Methoden `receiveSituation` und `direct`, die auch in dieser Reihenfolge aufgerufen werden müssen. Der Methode `receiveSituation` werden die beiden oben genannten Datenstrukturen übergeben, während `direct` die Berechnung einer Adaption anstößt. Beide Methoden müssen von den Implementierungen realisiert werden. Dies entspricht dem Entwurfsmuster *Strategy* [63] und wurde in der Klasse `DirectorStrategy` umgesetzt.

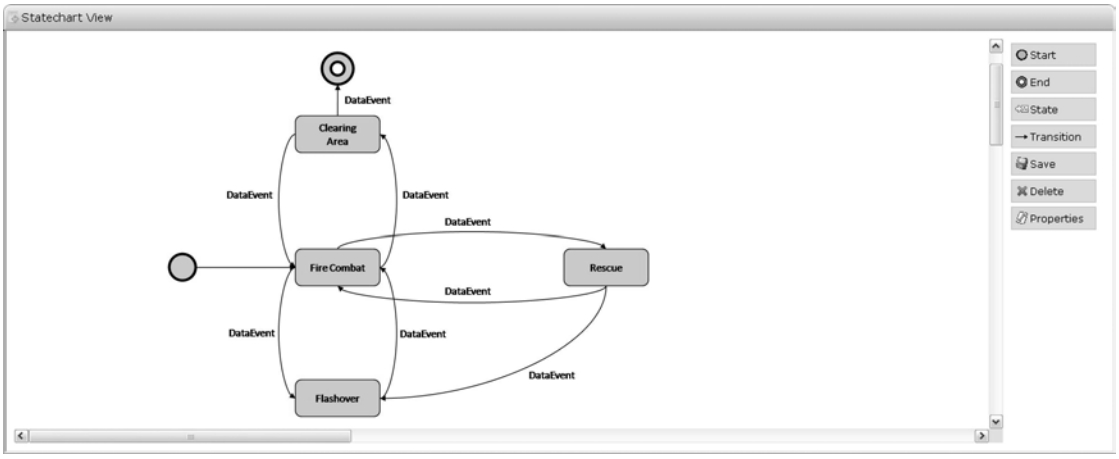


Abbildung 8.7: Editor für Statecharts

Statechart-Direktor Beim Statechart-Direktor werden die möglichen Adaptionen und deren Reihenfolgen über ein Zustandsdiagramm beschrieben. Damit die Adaptionsbeschreibung nicht mit der Workflow-Beschreibung vermischt wird, wird das Statechart separat gespeichert. Dazu wurde das Statechart-Repository eingeführt, dass sowohl die Statechart-Beschreibung enthält als auch die Verknüpfung von Workflow und Statechart.

Neben der Integration in OSPP musste ein Metamodell für die Modellierung von Zustandsdiagrammen erstellt werden. Ein `Statechart` besteht aus einer Menge an `States`, wobei genau ein `Start`-State sowie mindestens ein `End`-State vorhanden sein müssen. Des Weiteren enthält das Diagramm mindestens eine *StateTransition*, welche die *States* miteinander verbindet. Verbindungen zu ein und demselben State sind ebenso möglich. Die Eigenschaften eines States begrenzen sich auf einen Namen sowie eine ID zur eindeutigen Identifizierung. Zusätzliche Eigenschaften werden für die Auswertung an dieser Stelle nicht benötigt. Die Transition enthält neben der ID eine Reihe zusätzlicher Attribute, die für die Steuerung der Adaptionen von Bedeutung sind: Die `EventDefinition` beschreibt den Typ des Events, bei dem die Transition aktiv wird. Bedingungen werden über eine Liste von `ConditionDefinition` definiert. Der Nutzer besitzt die Möglichkeit, verschiedene Bedingungen logisch miteinander zu verknüpfen, anhand einer zuvor festgelegten `conditionDisjunction`, die über die Auswertung entscheidet. Die `AdaptationDefinition` enthält die aus der *Adaptionsdatenbank* ausgewählte Adaption. Beschrieben werden die Adaptionen in der Adaptionsdatenbank über das Interface `IAdaptation`. Mit diesem ist es möglich sowohl Änderungen am rollenorientierten Workflowmodell als auch serielle und parallele *Verwebungen* zu beschreiben. Die notwendigen Informationen wurden in den vorangegangenen Kapiteln vorgestellt. Um die Adaptions-Zustandsdiagramme zu modellieren, wurde ein grafischer Editor implementiert (siehe Abbildung 8.7).

BDI-Direktor Im Gegensatz zum Statechart-Direktor ist eine explizite Modellierung, unter welchen Umständen eine Adaption durchgeführt werden soll, bei dem BDI-Direktor nicht notwendig. Adaptionsentscheidungen werden auf Basis bekannter, bereits geschehener, Adaptionen getroffen. Dabei wird die aktuelle Situation mit den bereits gelernten in der Case-Datenbank verglichen. Der BDI-Direktor arbeitet auf Grundlage zweier Datenbanken. Die `Desire-Base` wird zur Ermittlung der Änderungen an Desires benötigt. Zur Umsetzung dieser Abbildung wurde das Interface `ICase` entwickelt,

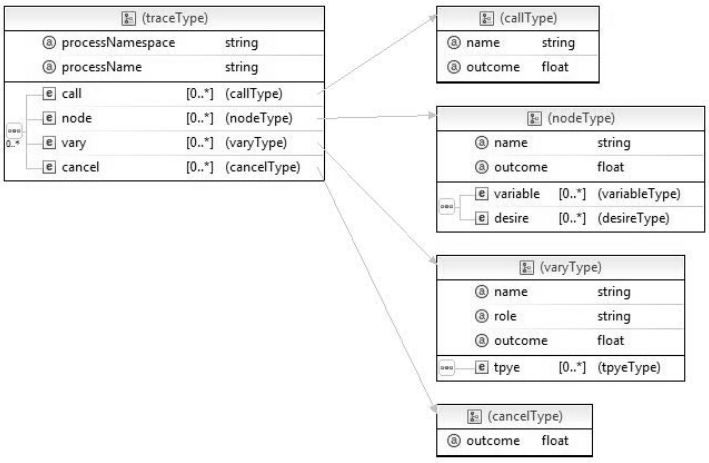


Abbildung 8.8: Schema eines Traces

das in seiner Funktionsweise einer erweiterten HashMap aus Java entspricht. Ein Case wird dabei beschrieben als `ICase<S extends Situation, C extends IAdaptation>`. Für die Desire-Base existiert die Implementierung `DesireAdaptation`. Für die zweite Datenbank, die `Intention-Base`, wird die Klasse `IntentionAdaptation` genutzt.

In der Intention-Base befindet sich die Abbildung, aus der, ausgehend von der aktuellen Situation und den neu berechneten Desires, eine Adaption ableitet werden kann. Dazu wird in der Intention-Base eine ähnliche Situation gesucht, die ein positives Feedback (Outcome) besitzt[14]. Die dazugehörige Adaption wird dementsprechend in der aktuellen Situation angewandt. Die Adaption ist, wie beim Statechart-Direktor, in der `Adaptation-Base` gespeichert.

Um die Ähnlichkeit von zwei Situationen zu bestimmen, werden für alle Variablen der Situationen die Einzelähnlichkeiten bestimmt. Demnach werden paarweise Variablen der Situationen verglichen und das Ergebnis einer jeden Berechnung in die gewichtete Summe der Ähnlichkeiten einbezogen.

Wichtig für die BDI-Direktoren ist es, dass die Case-Datenbanken ausreichend gefüllt sind, da sonst keine (sinnvollen) Entscheidungen getroffen werden können. Wie in Abschnitt 5.2.3 bereits beschrieben, kann dieses Wissen von Experten geliefert werden. Eine Möglichkeit dazu ist die Nutzung von *Traces*, wie sie in Abschnitt 5.2.2 eingeführt wurden. Ziel ist es, aus angereicherten Log-Dateien, Fälle für die Case-Datenbanken zu extrahieren. Dadurch wird eine Einlernphase des BDI-Direktors vermieden.

Traces sind Log-Dateien von beendeten Instanzen, die ohne einen Direktor ausgeführt wurden. Ein Experte muss im Anschluss daran den Inhalt des Logs erweitern, denn aus dem Trace der Ausführung kann nicht auf die Desires geschlossen werden. Diese Zusatzinformation muss der Experte liefern. Ebenso wird der Outcome einer jeden Adaption angegeben. Die Traces werden in OSPP als einzelne XML-Datei pro Instanz angelegt.

[14] Die Ähnlichkeitssuche wurde, wie in Abschnitt 5.2.1 beschrieben, umgesetzt.

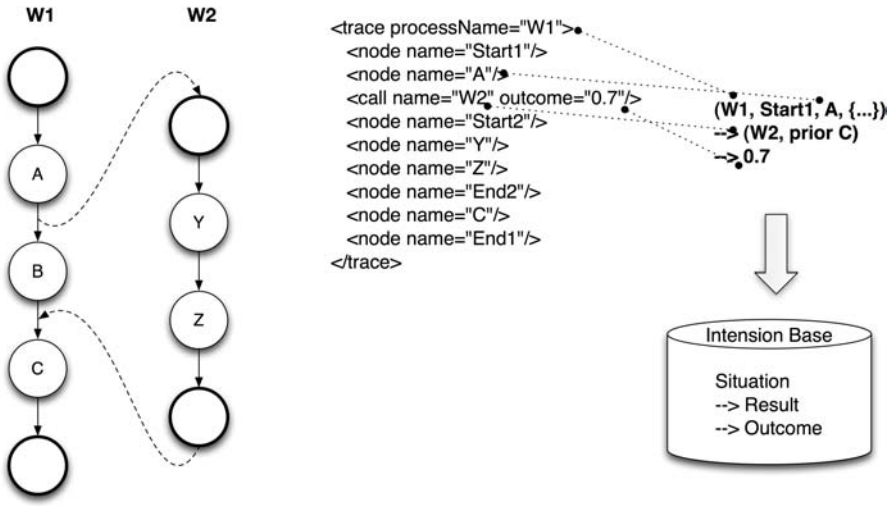


Abbildung 8.9: Abbildung von Trace auf die Intention Base

Es wird durch eine XSD-Datei beschrieben, die schematisch in Abbildung 8.8 dargestellt ist.

Grundsätzlich besteht ein Traces aus einer Abfolge von `node`-Einträgen, die gelegentlich von `call`, `vary` oder `cancel`-Elementen unterbrochen werden. Für jede ausgeführte Aktivität einer Instanz wird dabei ein Eintrag in den Trace geschrieben. Jede Änderung einer Variable oder eines Desires wird ebenfalls festgehalten, wobei diese Änderung einer bestimmten Aktion zugeordnet wird. `variable` und `outcome` besitzen zusätzlich ein `type`-Attribut. Für Variablen bezeichnet es den Datentyp des Werts. Wenn dieser einem XML-Schema Standardtype – `double`, `string`, etc. – aus dem Namespace `http://www.w3.org/2001/XMLSchema` entspricht, werden die entsprechenden Java-Datentypen (Integer, Double, String, ...) zur Interpretation genutzt. Das `type`-Attribut kann aber auch auf Datentypen aus anderen Namensräumen verweisen, die beispielsweise aus einer externen WSDL[15] stammen. Für diese werden extra Klassen erzeugt, die für die Verarbeitung in OSPP integriert werden. Für das Desire kann das `type`-Attribut die Werte `create` und `drop` annehmen. Dies bedeutet, dass Desires erstellt oder gelöscht werden sollen.

Kontrollflussadaptionen, die während der Ausführung angewandt wurden, werden mit `call` und `cancel` markiert. Ersteres bedeutet, dass eine neue Instanz mit dem angegebenen Namen in die aktuelle Instanz eingewoben wurde. Alle folgenden `node`-Elemente beziehen sich implizit auf das Modell der eingewobenen Instanz. Beide Typen enthalten den Parameter `outcome`, der den Nutzen der Adaption angibt. Für Adaptionen an der Rollenzuordnung ist `vary` vorgesehen. Hierbei wird eine Reihenfolge von Rollen definiert. Wie bei den Nodes besitzt jede Rollenangabe einen eindeutigen Namen und ein zusätzliches Attribute `type`, das die Werte `create`, `drop`, `active` und `deactivate` annehmen kann. Mit den ersten beiden Typen können Änderungen an den playedby Beziehungen beschrieben werden, während `active` angibt, ob diese Rolle aktiv genutzt wurde. Dies ist wichtig, wenn eine Aktivität überkonfiguriert wurde (wie in Abschnitt 7.4 beschrieben).

[15] Web Services Description Language.

Diese Informationen werden vom `Case-Extractor` in die jeweiligen Datenbanken übertragen. Dazu werden für jeden Zeitpunkt im Trace die zu diesem Zeitpunkt gültigen Beliefs ermittelt. Bei Änderungen an den Desires wird dann ein Case in die Desire-Base eingetragen. Ähnlich verhält es sich bei den Adaptionen, wobei die Adaptionsvorschriften extra in der `Adaptation-Base` gespeichert werden. Die Fälle in der Intention-Base verweisen dementsprechend auf die Adaptionen in der Adaptation-Base. Diese Vorgehensweise ist in Abbildung 8.9 schematisch dargestellt. Die beispielhafte Workflowausführung auf der linken Seite wird zuerst in einen Trace umgewandelt. Anschließend werden die notwendigen Informationen für die Intention Base aufgebaut und gespeichert.

8.1.2 Kommunikation und Kooperation für Direktoren

Als eine sinnvolle Erweiterung wurde in Kapitel 6 der BDI-Direktor um semantisches Wissen erweitert. Diese ermöglicht nicht nur die Kommunikation zwischen Direktoren, sondern auch die korrekte Vergleichbarkeit von Variablenwerten, vor allem bei Zeichenketten.

Die Ontologie (siehe Abbildung 6.8) wurde mithilfe von Protégé [162] umgesetzt. Für die Verarbeitung in der prototypischen Implementierung wurde das Jena Semantic Web Framework [94] verwendet. Jena ermöglicht die Einbindung von RDF- und OWL-Dokumenten und stellt alle nötigen Funktionen bereit, um mit den Daten zu arbeiten.

Ein zentraler Aspekt des semantischen BDI-Direktors ist Kommunikation und die darauf aufbauende Kooperation zwischen den Direktoren. Um die Kommunikation zwischen Direktoren umzusetzen, wurde *OpenJMS* genutzt. Die Open Source Implementierung der Java Message Services [172] ist für die nachrichtenbasierte verteile Kommunikation entwickelt worden. Durch den Einsatz von OpenJMS können die Direktoren unterschiedlicher OSPP-Instanzen miteinander kommunizieren. Dazu wurde der OpenJMS Server in OSPP integriert.

Mit OpenJMS wurde eine `MessageQueue` entwickelt, die dem Blackboard aus Kapitel 6 entspricht. Zur Kommunikation stellt die MessageQueue die beiden Methoden `publishQuestion(Message m)` und `deleteQuestion(Message m)` zur Verfügung. Die Nachricht vom Typ `Message` ist die Umsetzung der ACL-Nachricht, wie sie in Listing 6.2 definiert ist. Bei seiner Initialisierung registriert sich jeder Direktor als Observer bei der MessageQueue und wird bei Aufruf der Methode `publishQuestion` über das Vorhandensein einer neuen Nachricht informiert. Mit `deleteQuestion` kann eine veröffentlichte Nachricht wieder gelöscht werden.

Der Kommunikationsablauf wurde bereits in Abschnitt 6.2.2 beschrieben und dementsprechend umgesetzt. Die Kommunikation wird durch die `Update`-Methode in einem jedem Direktor realisiert. Bei seiner Initialisierung meldet sich jeder Direktor als Observer bei der MessageQueue an. Dazu implementiert die Klasse Director das Interface Observer. Die gesamte Behandlung der Kommunikation findet innerhalb der Methode `update` statt. Beim Eintrag einer Nachricht in die MessageQueue wird diese den registrierten Direktoren mitgeteilt. Die Nachricht ist ein Objekt der Klasse `Message`. Die Nachrichteninhalte werden im `Content`-Bereich übertragen. Dieser besteht aus einer zur Übertragung serialisierten `Situation`, die alle notwendigen Informationen enthält. Gültige Vorschläge werden als eine Liste von `IAdaptation` übertragen. In der aktuellen Umsetzung findet bei Reaktion auf eine Nachricht durch einen Kommunikationspartner eine Peer-to-Peer Kommunikation zwischen zwei Partnern statt. Hierbei wird entschieden, mit welchem

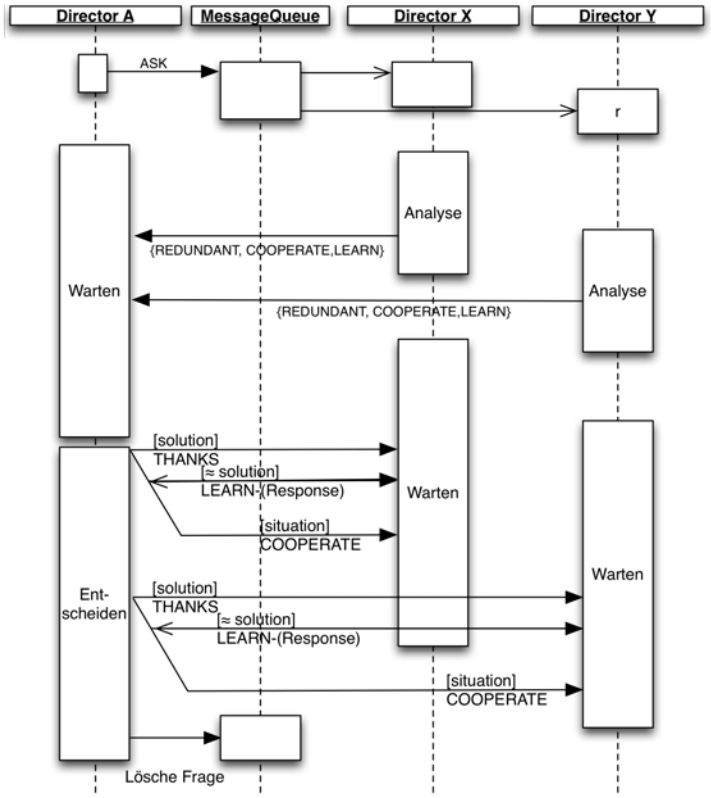


Abbildung 8.10: Kommunikation zwischen Direktoren

antwortenden Direktor kommuniziert wird. Erhält ein antwortender Direktor nach einem, in der Konfiguration festgelegtem, Zeitintervall keine Antwort, setzt er seine Arbeit fort. Abbildung 8.10 stellt in einem Sequenzdiagramm die möglichen Abläufe visuell dar.

Den Einstiegspunkt in eine mögliche Kommunikation bildet `publishQuestion` im Direktor. Diese Methode veranlasst den Direktor, eine Nachricht an die MessageQueue zu senden und zu warten. Dies geschieht, wenn der Direktor keine passenden, d. h. ähnlichen Situationen in der `Intention-Base` findet. Die Wartezeit wird in Millisekunden in der Konfiguration festgelegt und kann vorzeitig beendet werden, wenn eine definierte Mindestanzahl an Antworten eingetroffen ist. Auch die minimale Nachrichtenanzahl wird in der Konfiguration bestimmt. Nach Ablauf der Wartezeit oder dem vorzeitigen Beenden des Wartens werden die Nachrichten ausgewertet und die publizierte Frage wird gelöscht. Jede publizierte Nachricht führt zu einem Ausführen der Update-Methode in den anderen Direktoren.

Erhält der Direktor einen Adaptionsvorschlag von einem anderen Direktor, ist sicher zu stellen, dass dieser mit den lokalen Vorschlägen verglichen wird, ob er eine höhere Ähnlichkeit aufweist. Ist dies der Fall, wird er in die lokalen Falldatenbanken eingepflegt. Dies verhindert zusätzlich, dass ein lokaler, als schlecht eingestufter Vorschlag erneut verwendet wird. Neben der Entscheidung gemeinsam zu lernen, kann auch die Entscheidung für einen `Merge` getroffen werden. Wie die Entscheidung getroffen wird und welche Konsequenzen ein Merge nach sich zieht, wurde in Kapitel 6 beschrieben.

8.1.3 Realisierung von Rekonfigurationen

Die Workflow-Engine muss Adaptionen auf verschiedenen Ebenen unterstützen. Dies ist zum einen die Kontrollflussadaption, die Adaption durch Verweben, und zum anderen die Adaption über das rollenbasierte Workflowmodell. Beide Rekonfigurationsansätze wurden in OSPP umgesetzt, wie der folgende Abschnitt zeigen wird.

Adaption durch Verweben Das rollenbasierte Workflowmodell besitzt als Kern ein Workflow Net, ein spezielles Petri-Netz. Bereits in Abschnitt 3.1.3 wurde gezeigt, dass die Adaption durch Verweben auch in Petri-Netzen anwendbar ist.

Die Umsetzung des Workflow-Net wurde auf Basis der Arbeit von Pellegrini und Giacomini [121] durchgeführt. Sie zeigen in ihrer Arbeit eine auf C++ basierende Implementierung einer Petri-Netz Workflow-Engine. Für die Ausführung einer Instanz ist der `ProcessWalker` zuständig, der einen `Token` durch die Instanz leitet. Dieser zeigt sich verantwortlich für die Überprüfung der Schaltbedingungen der Transitionen (entspricht den Aktivitäten). Die Ausführung einer Aktivität wird in einen `ExecutionThread` ausgelagert. Dies ist notwendig, um die parallele Ausführung von Aktivitäten zu ermöglichen.

Wie in Abschnitt 3.1.2 beschrieben, muss eine Workflow-Engine folgende Operationen unterstützen:

1. Pausieren einer Instanz, um auf die Entscheidung des Direktors zu warten.
2. Instanziierung beliebiger Workflowfragmente inkl. der Übertragung des Tokens in selbige
3. Einfügen eines virtuellen And-Joins
4. Die Übertragung der Variablenbelegungen in die eingewobene Instanz.

All diese Operationen wurden im `ProcessWalker` umgesetzt. Die Entscheidungsfindung durch den Direktor wird im `ProcessWalker` vor und nach jeder Aktivitätsausführung ausgelöst. Während dieser Zeit wird die Instanz pausiert. Da aber die Berechnung möglicher Adaptionen zum Teil rechenintensiv ist, kann der Nutzer festlegen, an welchen Stellen auf eine Adaption geprüft werden soll. Dies vermindert den Aufwand erheblich.

Wird eine Adaption vorgeschlagen, initialisiert der ProcessWalker einen neuen ProcessWalker für die einzuwebende Instanz. Dabei übergibt er auch alle aktuellen Variablenbelegungen an den neuen ProcessWalker. Für den möglichen Rücksprungpunkt wird zudem die Schaltbedingung für die Transition geändert, so dass ein `And-Join` virtuell eingefügt wird. Bei einer seriellen Adaption wird der `Token` an den neuen ProcessWalker übergeben. Ist ein Rücksprungpunkt gesetzt, wird auf die Beendigung des neuen ProcessWalker gewartet, wenn nicht, wird der ursprüngliche ProcessWalker beendet. Hingegen wird das Token bei einer parallelen Adaption aufgeteilt.

Der ProcessWalker ist dabei nur für die Interpretation des Petri-Netzes zuständig. Die Funktion, beispielsweise der Aufruf eines Webservices, ist hingegen für den `Execution-Thread` von Bedeutung, der die Funktion ausführt.

Rollenbasiertes Workflowmodell Der ProcessWalker ist demnach nicht für die Adaptionen des rollenbasierten Workflowmodells zuständig, sondern der ExecutionThread. Er ist nicht nur für die Ausführung der Funktionen zuständig, sondern überprüft beispielsweise, ob die erforderlichen Rechte vorhanden sind (organisatorische Perspektive) und alle Datenabhängigkeiten eingehalten werden (informationsbezogene Perspektive).

Die Rollen, bzw. das gesamte rollenbasierte Workflowmodell, wurden mit *ObjectTeams* umgesetzt. Zentrales Konzept von ObjectTeams ist das `Team`. Dieses Konstrukt ist klassenähnlich strukturiert, d. h. es kann ebenfalls Attribute und Methoden enthalten. Teams bilden zusätzlich den Kontext für Rollen. Ein Team kann mehrere Rollen beinhalten und gruppiert diese dadurch semantisch. Ein Team entspricht demzufolge den in Kapitel 7 eingeführten und in Abbildung 7.4 dargestellten Kontexten.

Rollen werden als innere Klassen eines Teams deklariert. Jede Klasse, die innerhalb eines Teams deklariert wird, ist demzufolge eine Rolle. Beispielsweise wurde der *functional context* (siehe 7.2.1) in ObjectTeams folgendermaßen[16] umgesetzt:

Listing 8.2: Functional Context

```
1  package org.tud.ospp.engine.model.roles;
2
3  ...
4
5  public team class FunctionalTeam
6  {
7      public void registerWebServiceInvokeConfigurator(Transition as
           WebServiceInvokeConfigurator task, WebServiceInvoke wsi) {
8          task.setWebServiceInvoke(wsi);
9      }
10
11     public void unregisterWebServiceInvokeConfigurator(Transition as
           WebServiceInvokeConfigurator task) {
12         unregisterRole(task, WebServiceInvokeConfigurator.class);
13     }
14
15 ...
16
17     public class WebServiceInvokeConfigurator playedBy Transition
18         base when (FunctionalTeam.this.hasRole(base,
               WebServiceInvokeConfigurator.class)) {
19       ...
20     }
21
22     public class DataGreyBoxWebServiceInvokeConfigurator playedBy
           Transition
23         base when (FunctionalTeam.this.hasRole(base,
               DataGreyBoxWebServiceInvokeConfigurator.class)) {
24       ...
25     }
26
27     public class OWSInvokeConfigurator playedBy Transition
28       base when (FunctionalTeam.this.hasRole(base, OWSInvokeConfigurator.
             class)) {
29       ...
30     }
31
32     public class RESTfulWebServiceInvokeConfigurator playedBy Transition
```

[16] Vereinfachte Darstellung.

```
33        base when (FunctionalTeam.this.hasRole(base,
              RESTfulWebServiceInvokeConfigurator.class)) {
34          ...
35      }
36
37      public class AssignConfigurator playedBy Transition
38          base when (FunctionalTeam.this.hasRole(base, AssignConfigurator.
                class)) {
39          ...
40      }
41  }
```

Die Rollen im `FunctionalTeam` werden von der `Transition` gespielt. Die Aktivierung einer Rollenbindung geschieht für die Rolle `WebServiceInvokeConfigurator` in der Methode `registerWebServiceInvokeConfigurator`. Die Rolle kann mit `unregisterWebServiceInvokeConfigurator` abgelegt werden[17].

Wie in Kapitel 7 beschrieben, ist für eine Rollenkollaboration immer eine Gegenrolle notwendig. Im Falle von ObjectTeams ist eine solche Konstruktion nicht notwendig. Die Methoden einer Rolle können jederzeit aufgerufen werden. Trotzdem wurde in dieser Umsetzung das Kollaborationsverhalten realisiert, da es eine Trennung von Konfigurationsdaten und Ausführung ermöglicht. Das `FunctionalTeam` enthält `Configurators`, welche die Konfigurationsdaten für eine bestimmte Funktion speichern. Soll eine bestimmte Aktivität ausgeführt werden, analysiert der `ExecutionThread` die gespielten Rollen und bindet die passende Gegenrolle, einen `Handler`. Im Falle des WebService wäre dies der `WebServiceInvokeHandler`. Diese beiden Rollen kollaborieren anschließend, wobei der Configurator die Konfigurationsdaten an den Handler übergibt, der die entsprechende Funktionalität bereitstellt. Durch dieses Verhalten übernimmt der `ExecutionThread` die Aufgabe des *Executors* aus Kapitel 7. Er arbeitet eng mit dem `Manager` zusammen, der die Einhaltung aller Rollen-Constraints überwacht.

Das rollenbasierte Workflowmodell kann, wie in Abschnitt 7.4 beschrieben, auf drei verschiedene Arten adaptiert werden:

1. Wechsel zwischen Konfigurationen
2. Hinzufügen von Rollen
3. Entfernen von Rollen

Schlägt der Direktor eine Adaption am Rollenmodell vor, kann der Koordinator sich nicht an den ProcessWalker wenden, denn dieser kann nur Änderungen an der Ablaufstruktur vornehmen. Für diese Art von Adaption ist der `RoleManager` zuständig. Dieser besitzt Zugriff auf alle instanziierten Workflowmodelle und damit auch auf die aktivierten Rollenbindungen, die er auf Befehl des Koordinators jederzeit ändern kann. Damit ist es ihm auch möglich, zwischen verschiedenen Konfigurationen zu wechseln. Dabei ist zu beachten, dass der ExecutionThread keine aktive Rollenkollaboration mit zu entfernenden Rollen besitzt.

[17] Diese Methoden existieren auch für die anderen Rollen. Zu Gunsten der Übersichtlichkeit wurde auf eine Darstellung verzichtet.

8.2 Evaluation

Die Flexibilität von OSPP ermöglicht es, verschiedene Anwendungsszenarien umzusetzen. Ein immer wichtiger werdender Aspekt einer jeden IT-Infrastruktur ist das *Cloud Computing*. Im Zusammenhang mit der Teilnahme am *IEEE Service Computing Contests* wurde OSPP auf dieses Anwendungsfeld angepasst.

Der Vorteil des Cloud Computings ist die einfache Skalierbarkeit großer Anwendungen. Dies hat den Vorteil, dass Teile der IT-Infrastruktur eines Unternehmens in die Cloud migriert werden können. Ein wichtiges Element dieser Infrastruktur ist das eingesetzte WfMS. Auch dieses kann in die Cloud integriert werden. Dieses Vorgehen bietet Vorteile bei der Milderung von Lastspitzen. Dabei werden zusätzliche Ressourcen für das WfMS zur Verfügung gestellt.

Eine dezentrale und komplexere Umsetzungsmöglichkeit stellt ein WfMS-Cluster dar, wobei parallel verschiedene Instanzen des WfMS auf unterschiedlichen Rechnerknoten ausgeführt werden. Die Instanzen können zudem unterschiedlich ausgestattet werden, d. h. dass sie z. B. unterschiedliche Funktionen bereitstellen. Dies kann beispielsweise bedeuten, dass nur wenige Instanzen in der Lage sind, datenintensive Dienste aufzurufen, da der Rechnerknoten genügend Ressourcen zum Speichern dieser Daten bereithält.

Eine solche Differenzierung ermöglicht es, bestimmte Fragmente eines Workflows auf anderen WfMS-Knoten auszuführen. Dies kann aus unterschiedlichen Gründen geschehen: Verteilung von Lasten oder Einsparung von Energie. Verallgemeinernd kann in diesem Zusammenhang von Kosten gesprochen werden. Sind für verschiedene Faktoren die Kosten bekannt, kann für jedes Fragment bestimmt werden, auf welchem Knoten dessen Ausführung am kostengünstigsten ist.

Im Rahmen des *CoolSoftware*[18]-Projekts wurde deshalb untersucht, ob mit den in OSPP realisierten Ansätzen eine fragmentierte Verteilung von Workflows umsetzbar ist. Das Ziel war, Workflows möglichst ressourcenschonend auszuführen. Im gewählten Szenario – Genexpressionsprozesse [137, 70] – bedeutet dies, dass die Workflows optimal verteilt werden, so dass einzelne Rechnerknoten nicht überlastet wird. Genexpressionsprozesse sind der Klasse der *Productive Workflows* zuzuordnen und sind besonders datenintensive Workflows, d. h. es müssen viele Daten gespeichert und verarbeitet werden. Dies beansprucht vor allem die CPU, den Arbeitsspeicher und Festplattenplatz. Werden viele Genexpressionsprozesse gleichzeitig ausgeführt, kann ein Rechnerknoten diese Last nicht allein verarbeiten. Ein zweites Merkmal von Genexpressionsprozessen sind die hohen Flexibilitätsanforderungen. Da die Durchführung einzelner Aktivitäten im Workflow von der Interpretation der Daten abhängig ist, muss es möglich sein, einen Genexpressionsprozess zur Laufzeit anzupassen.

Dabei wird die Evaluation anhand der Genexpressionsprozesse, als Vertreter der *Productive Workflows*, zeigen, dass mittels der neuen Adaptionsmechanismen *Strukturelle Rekonfiguration*, *kontextbezogene Rekonfiguration* und die Verteilung realisierbar sind. Die Ergebnisse sind auch auf die *Administrative Workflows* übertragbar, denn auch sie basieren auf einer vorgegebenen Ablaufstruktur. Wie Tabelle 8.1 zeigt, werden die *Ad-hoc-Workflows* davon nicht abgedeckt, denn alle Rekonfigurationsansätze setzen modellierte Workflows voraus.

[18] Projekt im Rahmen des Spitzenclusters *Cool Silicon*, gefördert vom Bundesministerium für Bildung und Forschung (BMBF) (Förderkennzeichen: 13N10782).

	Strukturelle Rekonfiguration	Kontextbezogene Rekonfiguration	Verteilung
Production Workflows	x	x	x
Administrative Workflows	x	x	x
Ad-hoc-Workflows			

Tabelle 8.1: Evaluationskriterien

Bevor die Ergebnisse vorgestellt werden, soll zunächst der Problembereich der WfMS-Cloud und der Problembereich der Genexpressionsprozess ausführlicher erläutert werden. Des Weiteren werden die Erweiterungen an OSPP – die Kostenfunktionen sowie die Fragmentierung – näher betrachtet. Anschließend wird die flexible Workflowausführung zur Lastverteilung vorgestellt.

8.2.1 Kooperative WfMS

WfMS sind ein wesentlicher Bestandteil der unternehmensweiten IT-Infrastruktur. Die Bandbreite der von Workflows abgebildeten Prozesse reicht von klassischen Produktionsprozessen über Prozesse zur Datenanalyse und -auswertung bis hin zu klassischen Geschäftsprozessen.

Derartige Systeme verbrauchen jedoch viele Ressourcen, sei es die Auslastung der CPU bei komplexen Berechnungen oder Speicherplatz für die gesammelten Daten. Für die Ausführung von datenintensiven Workflows, wie beispielsweise Genexpressionsprozesse, müssen die verfügbaren Ressourcen überwacht und verwaltet werden. Bei der gleichzeitigen Ausführung kann es zu einer verlangsamten Ausführung kommen, wenn nur ein Rechnerknoten genutzt wird. Eine mögliche Lösung für dieses Problem ist, eine Workflowinstanz während der Ausführung in Fragmente aufzuteilen und diese auf verschiedene Workflow-Engines zu verteilen. Dadurch entsteht ein Geflecht von kooperativen Workflow-Engines.

Der Workflow wird weiterhin auf der zentralen Instanz gestartet und von einem Direktor überwacht. Dieser entscheidet allerdings nicht über das Einweben eines Workflowfragmentes, sondern entscheidet, wie die Workflowfragmente auf verschiedenen Workflow-Engines verteilt werden. D. h. dass die Kontrollflusssteuerung von der Ursprungsinstanz auf andere übertragen wird. Eine Instanz kann dabei nach verschiedenen Kriterien aufgeteilt werden, wie beispielsweise Lastverteilung. Die Ergebnisse eines jeden Fragments werden wieder an die Ursprungsinstanz gemeldet, die in diesem Fall die Rollen eines Verteilungskoordinators einnimmt. Ein mögliches Beispiel für dieses Vorgehen ist in Abbildung 8.11 dargestellt. Die Instanz wird hierbei in mehrere Fragmente[19] aufgespalten. Verschiedene Workflow-Engines führen die Workflowfragmente aus und senden die Ergebnisse zurück an die Ursprungsengine. Die Zuordnung einzelner Fragmente zu den Engines ist dabei für jede Instanz unterschiedlich. Das Fragment mit `Task 4` und `Task 5` kann auch von Engine 3 ausgeführt werden.

Die Aufteilung des Gesamtworkflows in Fragmente ist demnach nicht nur von Kontroll- und Datenfluss abhängig, sondern auch von verschiedenen Parametern, die durch eine

[19] Dargestellt durch die Ellipsen.

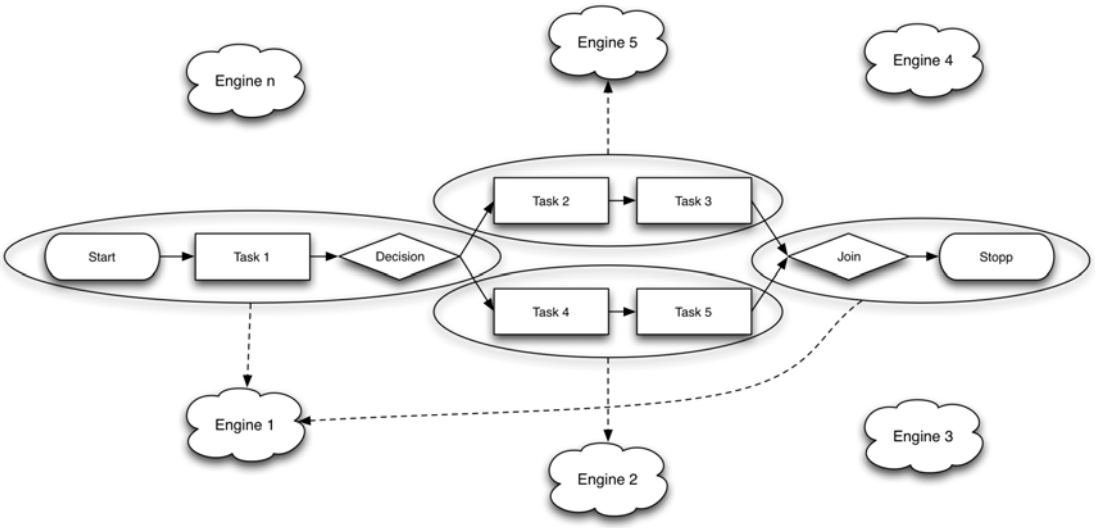


Abbildung 8.11: Verteilung von Workflowfragmenten

Kostenfunktion beschrieben werden können. Dies entspricht einem Optimierungsproblem, bei dem die kostengünstigste Abarbeitung einzelner Teilschritte gefunden werden muss. Das Beispiel in Abbildung 8.11 macht dies deutlich. Es gibt allein für diesen kleinen Workflow eine Vielzahl mögliche Fragmentierungen[20]. Für alle Fragmente müssen die Kosten berechnet werden.

Einen ähnlichen Ansatz beschreiben Baresi et al. in [15]. Sie zeigen eine Möglichkeit Workflows für mobile Umgebungen zu fragmentieren. In solchen ist es nötig, die begrenzten Kommunikationsmedien und den Energieverbrauch der mobilen Geräte zu schonen. Zudem entsteht durch das ständige Hinzufügen und Entfernen der Geräte ein sich ständig veränderndes Ad-Hoc-Netzwerk. Die Fragmente des Workflows müssen möglichst unabhängig voneinander ausgeführt werden können. Um die Belastung der Netzwerkinfrastruktur zu minimieren, sind möglichst wenige, große Fragmente zu erzeugen. Die mobilen Geräte können dann längere Zeit ohne Kommunikation mit den anderen Teilnehmern die in den Fragmenten enthaltenen Aufgaben durchführen. Ein nicht auf Energie bezogener Ansatz wird in [130] beschrieben. Auch hier wird der Workflow in einzelne Fragmente unterteilt, die anschließend verteilt ausgeführt werden können. Basis für diese Arbeit war ADEPT. Weitere Ansätze finden sich in [169, 124, 170].

In den folgenden Abschnitten wird gezeigt, wie mit den Rekonfigurationsansätzen im Zusammenspiel mit dem Direktor diese verteilte Workflowausführung umgesetzt werden konnte.

8.2.2 Genexpressionsprozesse

Die Bestimmung der Funktion eines oder mehrerer Gene der DNA ist vor allem bei der Erforschung von Krankheiten ein zentrales Problem. Ziel ist es, wenn ein Gen als Auslöser einer Krankheit identifiziert wird, dieses bei den Patienten gezielt auszuschalten, um die

[20] Abbildung 8.11 zeigt nur eine konkrete Ausprägung.

Krankheit aufzuhalten. Die Funktion von Genen wird mit Hilfe von *Genexpressionsprozessen* ermittelt. Dabei wird durch verschiedene Clusterverfahren versucht, ein Zusammenhang zwischen Genen und Krankheiten zu finden. Der Begriff der Genexpression beschreibt einen mehrstufigen Prozess. Der Ausgangspunkt ist die Transkription, bei der die genetischen Informationen der DNA in die kurzlebige *messenger RNA* (mRNA) übertragen werden. Um die Funktionalität der Gene und deren gegenseitige Beeinflussung zu ermitteln wird die mRNA-Menge des zu untersuchenden Genes gemessen.

Eine Möglichkeit für die Messung sind *Microarrays* [6, 7, 109]. Diese ermöglichen die Messung der relativen Häufigkeit von Genen, in Bezug auf den Zelltyp, Gewebe und Auswirkungen auf die Umwelt für tausende Gene gleichzeitig. Die Genexpression Experimente führen zu einer großen Menge an Daten, die nicht nur direkt durch die Microarray-Experiment erzeugt werden. Die Daten können in *Image Data* (Bild-Daten), *Expression Data* und *Annotation Data* [50] eingestuft werden. All diese Daten werden für den Analyse-Prozess benötigt

In Abschnitt 2.1.2 und Abbildung 2.3 wurde der Genexpressionsprozess bereits vorgestellt. Kern sind die folgenden Aktivitäten.

1. *Datenintegration:* Die Genexpressionsdaten müssen in ein gemeinsames Schema integriert werden. In [50] wurden mögliche Integrationsansätze und Probleme untersucht. Das Standardformat für den Expressionsdaten ist die Genexpression-Matrix [29], wobei Zeilen die Gene und Spalten die Proben repräsentieren.

2. *Datennormalisierung:* Eine Normalisierung der integrierten Daten ist notwendig, um Schwankungen zu kompensieren. Derzeit existieren viele Strategien für die Normalisierung dieser Daten. Da es keine Standard-Normalisierungs-Strategie gibt, ist es notwendig, dass der Benutzer die Möglichkeit hat, die Normalisierungart selbst zu bestimmen.

3. *Datenanalyse:* Die Analyse erfolgt auf den normalisierten Daten und sucht nach Beziehungen zwischen den Genen und Proben. Hierfür existiert ebenfalls eine Vielzahl an Methoden, wie bzw. Clustering, Assoziationsregeln oder Klassifizierung. Aufgrund des fehlenden Standard-Analyseverfahrens sollten die Nutzer auch hier die Möglichkeit besitzen, die Analysemethode zu wählen und zu parametrisieren.

4. *Interpretation und Verifikation:* Für die Interpretation der Analyseergebnisse müssen die Ergebnisse in einer verständlichen Form zusammengefasst werden. Insbesondere wird hier eine Visualisierung benötigt, welche die Analyseergebnisse darstellt. Nach Interpretation und Überprüfung der Analyseergebnisse können Normalisierung oder Datenanalyse mit veränderten Parametern oder Methoden wiederholt werden.

Anforderungen des Genexpressionsprozesses Bei der Realisierung von Genexpressionsprozessen mit einem WfMS treten zwei Hauptprobleme auf:

- Datenvolumen: Die Menge der bei diesen Prozessen verarbeiteten Daten ist groß. Für ein kleines Experiment können 2GB[21] an Gen-Sequenzdaten entstehen. Hinzu kommt, dass viele Experimente gleichzeitig stattfinden und Daten anderer Forschergruppen integriert werden müssen. Diese Daten müssen nicht nur gespeichert, sondern auch verarbeitet werden. Ein WfMS auf einem einzelnen Rechnerknoten

[21] Zusätzlich zu 0,5 TB Bild-Daten und 100GB Metadaten.

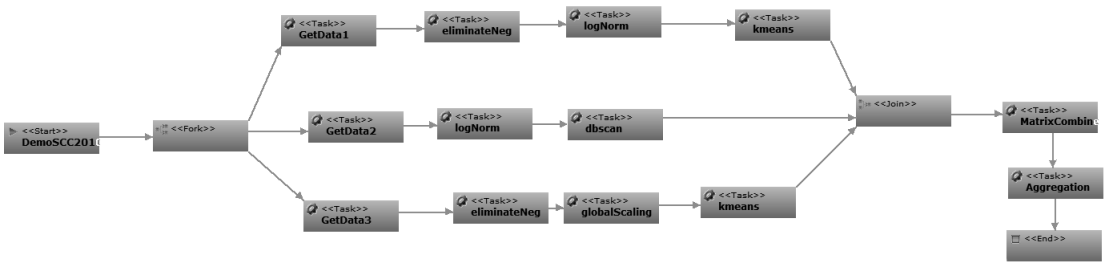


Abbildung 8.12: Beispiel eines Genexpressionsprozess in OSPP

wäre hier nicht ausreichend. Eine Verteilung der Aufgaben auf verschiedene WfMS-Instanzen ist notwendig. Dies kann mit Hilfe von verteilten OSPP-Instanzen und der Adaption durch Verweben umgesetzt werden.

- Kontextbezogene Rekonfiguration: Der Benutzer muss während der Ausführung des Analyseprozesses die Verarbeitungsmethoden auswählen oder verändern können. Ein statischer Workflow ist in diesem Falle nicht verwendbar. Das rollenbasierte Workflowmodell bietet die notwendige Flexibilität.
- Strukturelle Rekonfiguration: Durch die Möglichkeit, dass nach der Interpretation, Normalisierung oder Datenanalyse mit veränderten Parametern oder Methoden wiederholt werden können, muss auch die Ablaufstruktur rekonfigurierbar sein, wofür die die Adaption durch Verweben benötigt wird.

Adaptiver Genexpressionsprozess Das Ziel dieses Szenarios ist es, die Last, die durch die einzelnen Aktivitäten in Genexpressionsprozessen entstehen, auf verschiedene OSPP-Instanzen zu verteilen. Dazu muss gleichzeitig die funktionale Flexibilität sichergestellt werden.

Im Rahmen des *Service Computing Contest 2007 und 2010* [137, 139] wurde ein vollständiger Genexpressionsprozess angefertigt, der in Abbildung 8.12 dargestellt ist. In einem ersten Schritt werden aus drei verschiedenen Datenquellen die Rohdaten gelesen. Für jede Datenquelle werden anschließend verschiedene Verarbeitungsverfahren durchgeführt, die ein Webservice zur Verfügung stellt:

- *Normalisierung*
  - *eliminateNeg*: Elimination aller negativen Werte
  - *logNorm*: Logarithmische Normalisierung
  - *globalScaling*: Global Scaling Normalisierung
- *Clustering*
  - *dbscan*: Density-Based Spatial Clustering of Applications with Noise [152]
  - *kmeans*: k-Means-Algorithmus, ein Verfahren zur Clusteranalyse

Der Beispielworkflow in Abbildung 8.12 ist statisch. Die Anzahl der zu verarbeiteten Datenquellen ist auf drei beschränkt und die einzelnen Verarbeitungsschritte (Normalisierung und Clustering) sind auf die jeweilige Datenquelle zugeschnitten. Die notwendige

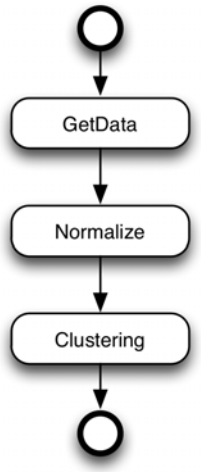


Abbildung 8.13: Vereinfachter Genexpressionsprozess

Flexibilität kann nicht gewährleistet werden. Die gewünschte Flexibilität unterteilt sich in zwei Hauptkriterien:

- *Rekonfiguration:* Der Workflow sollte manuell zur Laufzeit rekonfigurierbar sein. Dies beinhaltet, dass sowohl die Funktionalitäten wähl- und austauschbar sind als auch, dass die Anzahl der Datenquellen oder der Normalisierungen zur Laufzeit vom Benutzer festgelegt werden können.
- *Verteilung:* Die Workflowaktivitäten sollen automatisch auf verschiedene Rechnerknoten verteilt werden, so dass zum einen eine gleichmäßige Auslastung, zum anderen eine zügige Bearbeitung gewährleistet werden kann.

Beide Kriterien sind mit den vorgestellten Adaptionsmethoden umsetzbar. Diese werden auf einen stark vereinfachten Genexpressionsprozess angewandt, der in Abbildung 8.13 dargestellt ist. Dieser besteht nur aus drei Schritten:

1. *GetData*: Laden der zu verarbeitenden Daten
2. *Normalize*: Die Normalisierung der Daten
3. *Clustering*: Berechnung von Clustern

Jede der drei Berechnungen können durch unterschiedliche Rollen umgesetzt werden. Dadurch ist es möglich, die Aktivitäten durch mehrere Rollen konfiguriert werden, d. h. sie sind überkonfiguriert (siehe Abschnitt 7.4). Es wurden für die Normalisierung und das Clustering Rollen entwickelt, die die gewünschte Funktionalität bieten. Zudem wurden für verschiedene Datenquellen eigene Rollen beschrieben, welche die Daten auslesen können (zumeist über Datenbankzugriffe). OSPP wurde zudem dahingehend angepasst, dass der Nutzer vor jeder Aktivität die Rollen selbst auswählen kann (oder bei der Standardrolle belässt).

Der Nutzer kann jedoch nicht nur über die Funktionalitäten bestimmten, sondern auch selbst den Kontrollfluss des Workflows ändern. Dafür steht ihm in OSPP ein spezieller Dialog zur Verfügung, indem er die Art der Adaption (zur Verfügung stehen die unterstützen Adaptionsmuster), das einzuwebende Fragment und den Rücksprungpunkt

bestimmen kann. Dies ermöglicht ihm zum einen die Veränderung der Anzahl der Arbeitsschritte. Er kann statt nur einer Normalisierung eine zweite durchführen lassen, wie es bspw. in Abbildung 8.12 der Fall ist. Dort werden `eliminateNeg` und `logNorm` nacheinander ausgeführt. Dafür ist lediglich eine serielle Adaption notwendig[22]. Des Weiteren kann er ein Experiment wiederholen, wenn bspw. die Ergebnisse des Clusterings unzureichend sind. Dazu kann er einen einmaligen `Loop` einfügen. Zum anderen kann er die Anzahl der parallel zu untersuchenden Datenquellen auswählen[23]. Durch diese Adaptionsmöglichkeiten entstehen für jedes Experiment angepasste Workflows. Eine Automatisierung der Adaption ist hier nicht möglich, da die Ausführung von der menschlichen Interpretation der Daten abhängt.

Jeder Benutzer besitzt dabei nur das Wissen über die von ihm aktuell ausgeführten Workflows. Über parallel ausgeführte Workflows, wie auch über die Auslastung der zur Verfügung stehenden Server, weiß er nichts. Dies macht eine manuelle Optimierung bzgl. der Auslastung der Server unmöglich. Der Direktor kann dies gewährleisten. Dieser wertet die Auslastungsdaten der verschiedenen OSPP-Instanzen aus und verteilt die einzelnen Aktivitäten auf nicht hinreichend ausgelastete Instanzen, so dass die Workflows schnellstmöglich ausgeführt werden können. Dazu muss der Direktor über einen Bewertungsmaßstab verfügen, der die Verteilungsadaptionen ermöglicht. Dazu wurde ein spezielles Kostenmodell erstellt, das im Folgenden vorgestellt wird.

8.2.3 Kostenbewußter BDI-Direktor

Die verteilte Workflowausführung basiert auf der Fragmentierung des Workflows, wobei jedes Fragment auf einer anderen Engine ausgeführt werden kann. Die Entscheidung, welches Fragment wo ausgeführt wird, ergibt sich aus den verursachten Kosten. Im optimalen Fall wird jedes Fragment auf der Engine ausgeführt, auf der es die wenigsten Kosten verursacht. Dabei ergeben sich zwei grundlegende Probleme: (1) das Aufstellen der Kostenfunktionen und (2) die Ermittlung der Fragmente. Bei den Kosten ist es notwendig zu wissen, welche Kostenarten existieren und wie diese berechnet werden.

Kosten in Workflows Kosten sind sowohl in der technischen Welt als auch im wirtschaftlichen Kontext bekannt. Beispielsweise kann die Geschwindigkeit durch Zeitkosten ausgedrückt werden oder ein QoS[24]-Parameter, wie die Übertragungsgeschwindigkeit, wird durch Kommunikationskosten repräsentiert. Andererseits können Kosten, die bei der Nutzung von nicht technischen Unternehmensressourcen entstehen, berücksichtigt werden. Beispiele sind Personalkosten, sowohl in ihrer zeitlichen als auch monetären Ausprägung. Auch kann man Kosten leicht mit Zuverlässigkeit in Beziehung setzen. Die Wahrscheinlichkeit der Verfügbarkeit einer Ressource bestimmt, wie sich die Kosten durchschnittlich durch das Aussetzen der Ressource erhöhen.

Im Allgemeinen kann zwischen zwei Kostenarten unterschieden werden:

- *Unternehmensausgaben:* Kosten die mit der Ausführung des Workflows in Zusammenhang stehen.

[22] Entspricht dem Adaptionsmuster *Insert*.
[23] Entspricht dem Adaptionsmuster *Parallelize*.
[24] Quality of Service.

- *Technische Kosten:* Kosten der verwendeten IT-Infrastruktur. Dazu zählen laufende Kosten, wie Auslastung, Energieverbrauch oder auch statische Kosten, wie beispielsweise Anschaffungen.

Beide Größen treten in Workflows auf und müssen beschrieben werden. Bei technischen Kosten legt der Entwickler Eigenschaften des Systems zugrunde, beispielsweise die Auslastung der Netzwerk- und Rechnerressourcen oder die Zeit, die ein System mit der Bearbeitung einer Aufgabe beschäftigt ist:

- *System:* Verbrauch (CPU / RAM / Netzwerk), Verbrauch von Festplattenspeicherplatz, etc.
- *Qualitative Eigenschaften:* Effizienz, Wartbarkeit, Verfügbarkeit
- *Quality of Service:* Service-Level Agreements[25]

Kostenfunktionen Wesentlich für die Auswahl der optimalen Zuteilung der Workflowfragmente zu den Engines ist die Prognose der Kosten für jede Ressource in einem Fragment. Da die Zuteilung anhand von Kosten erfolgt, muss für jede Ressource eine Kostenfunktion berechnet werden, die den Kosten zu einem bestimmten Zeitpunkt entspricht. Dieser Zeitpunkt ist geprägt durch die folgenden Größen:

- Zustandsgrößen des Workflows sowie
- Zustandsgrößen der Workflow-Engines.

Der Zustand der Instanz ist durch die Belegung seiner Variablen beschrieben. Insofern diese als Parameter eines späteren Funktionsaufrufes dienen oder sie von einer Ressource verarbeitet werden, können sie sich direkt auf die entstehenden Kosten auswirken. Enthält der Workflow beispielsweise eine Stückzahl-Variable, beeinflusst diese die Zeitdauer der Bearbeitung durch eine Funktion. Technische Kosten und Unternehmensausgaben bilden zusammen die Zustandsgrößen $x_1 \dots x_n$ die Eingabedimensionen der Kostenfunktion[26] f_K:

$$K_{R_1} = f_K(x_1, \dots, x_n)$$

Das Ergebnis ist ein Kostenwert. Da in Workflowfragmenten mehrere Ressourcen verwendet werden, entstehen automatisch mehrere Einzelkosten für ein Fragment[27]. Diese müssen in einer Gesamtkostenfunktion gebündelt werden. Die einzelnen Kostenfunktionen müssen durch den Modellierer des Workflows angegeben werden.

[25] Dienstgütevereinbarung.
[26] Kosten für Ressource R_1 für n Zustandsgrößen.
[27] Für jede Ressource eine Kostenfunktion.

Gesamtkostenfunktion Bei der Vereinheitlichung verschiedener Kosten in einer Gesamtkostenfunktion müssen die unterschiedlichen Maßeinheiten beachtet werden. Diese können übernommen werden, wenn sie in ein entsprechendes Verhältnis gestellt werden. Eine Normalisierung muss vorgenommen werden, wenn in der Gesamtkostenfunktion verschiedene Kostenarten miteinander verglichen werden. Weiterhin kann eine Priorisierung von einzelnen Kosten notwendig sein, wenn es beispielsweise wichtig ist, einen Workflow möglichst schnell auszuführen und dabei höhere (monetäre) Kosten zweitrangig sind.

Dazu wird jeder Kostenfunktion für eine Ressource eine Liste von Faktoren bzw. Prioritäten zugeordnet. Diese Prioritäten $p_1 \dots p_n$ und Normalisierungsfaktoren $c_1 \dots c_n$ sind Koeffizienten einer Gesamtkostenfunktion K_G und deren Variablen $x_1 \dots x_n$ die prognostizierten Kosten[28]:

$$K_G(x_1, \dots, x_n) = \sum_{a=1}^{b} p_a \times c_a \times f_K(x_1, \dots, x_n)$$

Durch die Normalisierungsfaktoren für eine Größe findet eine Umrechnung aller Kostengrößen in eine gemeinsame Maßeinheit statt. Die einfachste Bezugsgröße sind, da es sich bei WfMS meist um betriebswirtschaftliche Anwendungen handelt, die Stückkosten. Der Verbrauch von Ressourcen, wie Betriebsmitteln, Verbrauch von CPU und Speicherkapazität, Personalzeit und Energie, kann in eine Währung abgebildet werden. Durch Modelle findet sich oft auch eine Annäherung, welchen finanziellen Wert eine Rechendauer hat.

Fragmentierung Die Gesamtkosten müssen für jedes Fragment inkl. seiner Ressourcen berechnet werden. Die Fragmentierung des Workflows entspricht einer Zerlegung in alle möglichen Kombinationen. Für n Aktivitäten in einem Workflow existieren dabei maximal $n!$ mögliche Fragmente. Für den Workflow, der in Abbildung 8.11 dargestellt ist, bedeutet dies, dass für 5040 unterschiedliche Fragmente die Gesamtkostenfunktion für jede zur Verfügung stehende Engine berechnet werden müsste. Für diesen kleinen Workflow ist dies eine aufwendige Berechnung. Deshalb muss der Workflowdesigner die Fragmente vorgeben, wie dies beispielhaft in Abbildung 8.11 geschehen ist.

Wenn die Kostenfunktionen korrekt sind und die Fragmente sinnvoll gewählt wurden, kann eine Verteilungsentscheidung getroffen werden[29].

Kostenbewußter BDI-Direktor Aufbauend auf den gegebenen Kostenfunktionen und Fragmenten sollte der BDI-Direktor die Entscheidung für die Verteilung der Fragmente auf die verschiedenen Engines vornehmen. Der BDI-Direktor, wie er in Kapitel 5 und 6 beschrieben wurde, ist jedoch nicht für die Verwendung von Kostenfunktionen ausgelegt. Es musste deshalb eine Vorverarbeitung der *aktuellen Situation* stattfinden, wie in Abbildung 8.14 dargestellt ist.

Die aktuelle Situation entspricht dabei der bereits beschriebenen Situation, bestehend aus den aktuellen Instanzvariablen, Ausnahmen und dem Laufzeitverhalten. Dem Direktor fehlen allerdings die Informationen, wie viel die Ausführung eines Fragmentes zu

[28] Gesamtkostenfunktion für b Ressourcen bei n Zustandsgrößen.

[29] Eine erweiterte Betrachtung, wie Kostenfunktionen und Fragmente automatisch bestimmt werden können, wurde in [139] vorgenommen.

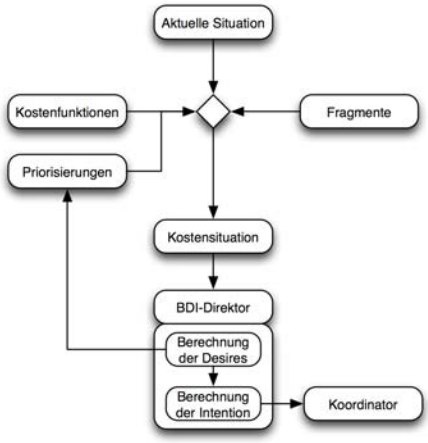


Abbildung 8.14: Kostenbewusste Adaption

einem bestimmten Zeitpunkt kosten würde. Wird der Direktor nach einer Verteilungsadaption gefragt, muss er diese Informationen zuvor berechnen. Die Anfrage für einen Verteilungsvorschlag kann in diesem Szenario von der Workflow-Engine immer nur am Ende eines Fragmentes gestellt werden. Zur Berechnung der *Kostensituation* bestimmt der Direktor alle folgenden – und damit zu berechnenden – Fragmente. Dies kann ein oder mehrere Fragmente betreffen. Im Falle des Workflows aus Abbildung 8.11 folgen auf das erste Fragment zwei weitere Fragmente. In einem zweiten Schritt werden die verfügbaren Engines (siehe Abschnitt 8.1) ermittelt, auf denen die Fragmente ausgeführt werden können. Diese Berechnung prüft, ob bei den zur Verfügung stehenden Engines die notwendigen Role-Bundles vorhanden sind.

Hier zeigen sich die Vorteile des rollenbasierten Workflowmodells. Neben den Adaptionen durch (Über-)konfiguration ist auch ein spezieller Zuschnitt der Workflow-Engine an bestimmte Hardwarevoraussetzungen oder spezielle Anforderungen möglich, wie in Abschnitt 7.3.1 beschrieben. Dies wird ermöglicht durch die dynamisch konfigurierbaren Role-Bundles. Dadurch können in der WfMS-Cloud die unterschiedlichsten Konfigurationen gleichzeitig aktiv sein und je nach Anforderung genutzt werden.

Damit muss jedoch geprüft werden, ob das Fragment auf einer Engine ausgeführt werden kann. Jede Workflow-Engine registriert sich deshalb beim Start, unter Angabe aller installierten Role-Bundles, bei der zentralen Workflow-Engine. Soll ein Fragment auf einer Engine ausgeführt werden, ist es notwendig, dass sie die erforderlichen Funktionalitäten bereitstellt. Ist bekannt, welche Fragmente für welche Engines berechnet werden müssen, wird die *Gesamtkostenfunktion* für jedes Fragment pro Engine berechnet. Diese Ergebnisse werden als Variablen in der Situationsbeschreibung gespeichert und ergeben die *Kostensituation*.

Die Kostensituation wird dem BDI-Direktor übergeben, der die Verteilungsadaption berechnen soll. Die einzelnen Cases in den Datenbanken beziehen sich in diesem Szenario nicht auf die klassischen Workflowvariablen, sondern auf die neuen Kostenvariablen, so dass die Desires und die Intentions auf Basis der Ähnlichkeit der Kostensituationen berechnet werden. In einem ersten Schritt werden die neuen Desires berechnet, die in diesem Szenario eine zusätzliche Funktion besitzen. Wird beispielsweise auf Basis der Kostenfunktion das Desire *Server auslasten* dem Direktor hinzugefügt, werden gleichzeitig die *Priorisierungen* für die Gesamtkostenfunktionen neu justiert. Soll die Auslastung

erhöht werden, werden alle auslastungsbezogenen Kosten, durch die Anhebung des Priorisierungsfaktors, zusätzlich erhöht[30]. Dies wurde über eine statische Abbildung in einer zusätzlichen Komponente realisiert, die den Direktor überwacht.

Bei der Ermittlung der Desires und Intentions erwies sich die Verwendung der Ähnlichkeitsfunktionen bei der Berechnung als Vorteil. Dadurch war es nicht notwendig, alle möglichen Kostensituationen vollständig abzubilden, sondern es reichten wenige Fälle. Nach Berechnung der Desires werden die Intentions, die den Verteilungsadaptionen entsprechen, ermittelt und dem Koordinator übergeben.

8.2.4 Verteilung durch Adaption

Die Verteilungsadaptionen entsprechen dem *Compose*, wie es in Abschnitt 5.2.1 beschrieben wurde. Diese Adaptionsentscheidung wird genutzt, um das Einweben eines Workflows zu beschreiben. Dazu wurde die Webeart (parallel oder seriell) und der *einzuwebende Workflow* (inklusive der Startposition) sowie der *Rücksprungpunkt* angegeben.

Im Falle einer Verteilungsadaption musste dies nicht grundsätzlich geändert werden. Eine Verteilung ist eine serielle Adaption, denn der Kontrollfluss verbleibt für die Zeit der Adaption nicht in der Ausgangsinstanz. Der einzuwebende Workflow entsprach dem der laufenden Instanz, wobei die Startposition der ersten Aktivität im Fragment und die Endposition dem Ende der letzten Aktivität entsprechen. Der Rücksprungpunkt ist mit der Endposition in diesem Fall identisch, da es sich um dieselben Workflowmodelle handelt. Diese Adaptionsanweisung wurde lediglich um die Information, welche Workflow-Engine das Fragment ausführen soll, erweitert[31].

Der Koordinator wurde dahingehend erweitert, dass er auch mit den externen Workflow-Engines zusammenarbeiteten konnte. Dazu war es notwendig, dass die externe Workflow-Engine das Workflowmodell auch auffindet. Da in einem heterogenen System nicht davon ausgegangen werden kann, dass jede Workflow-Engine über die gleichen Workflowmodelle verfügt, war es notwendig, dass der Koordinator diese an die externe Workflow-Engine übergibt. Dazu konnte auf die R-OSGI Dienste zurückgegriffen werden. Der Koordinator fragt nach allen vorhandenen Workflowmodellen. Ist das gesuchte Modell vorhanden, sendet er die Adaptionsanweisung an die Workflow-Engine, welche veranlasst, ein bestimmtes Workflowfragment zu instanziieren. Damit alle notwendigen Daten vorhanden sind, wird die aktuelle Situation (mit allen Variablen) übergeben. Diese Funktionen wurden ursprünglich für die selektive Adaption implementiert (siehe Abschnitt 8.1), sind jedoch für die Verteilung gut geeignet. Die zentrale Instanz wartet anschließend auf die Beendigung der externen Ausführung, bis sie weiter fortfahren kann. Dazu überwacht der Koordinator immer wieder die Ausführung und integriert an deren Ende auch die geänderte Situation in den zentralen Workflow.

Ist das aktuelle Workflowmodell nicht vorhanden, wird ein temporäres Workflowmodell erzeugt und der externen Engine übergeben. Dazu wird das Workflowfragment als eigenständiger Workflow angesehen. Dieses Workflowmodell wird dem externen Workflow Repository zum Import übergeben (siehe Abschnitt 8.1) und erst anschließend wird der Workflow, der dem Fragment entspricht, gestartet.

[30] Die Erhöhung geschieht nur minimal, damit die Kosten nicht unrealistisch werden.
[31] Dementsprechend auch die Intention-Datenbank.

8.2.5 Verhalten der Direktoren

Mit diesen Erweiterungen an Direktor, Koordinator und der Intention-Base konnte das gewählte Szenario erfolgreich umgesetzt werden. Ziel war es, dass ein BDI-Direktor viele unterschiedlich konfigurierte Genexpressionsprozesse auf mehreren OSPP-Instanzen optimal verteilt. Um Messungen zur Auslastung automatisch durchzuführen wurde der Nutzer simuliert, d. h. es wurden per Zufall verschiedene Adaptionen bzgl. der Rollenkonfigurationen und des Kontrollflusses ausgewählt und durch den Koordinator angewandt. Die einzelnen Aktivitäten bzw. Fragmente der völlig unterschiedlichen Workflows sollte der BDI-Direktor auf die verschiedenen Workflow-Engines verteilen.

Für die Auslastung wurden zwei Kriterien betrachtet: Auslastung von CPU und Arbeitsspeicher. Für die Messungen standen vier Server zur Verfügung. Zwei Server verfügten über 8 Kerne und 8GB RAM, die beiden anderen über 16 Kerne und 16GB RAM. Auf jedem der vier Server befindet sich einen OSPP-Instanz, die dem Direktor bekannt ist. Zudem erhält er von jeder Engine auf Anfrage die aktuellen Auslastungsdaten. Jeder Server verfügte zusätzlich über eine Datenbank, die mehrere Testdatensätze mit der Größe 2GB speichern kann.

Dem Direktor wurden zudem verschiedene Situationen in den Case-Datenbanken vorgegeben, die allerdings keine vollständige Abdeckung aller möglichen Situationen darstellen. Demnach musste der Direktor lernen. Fand er keine ähnliche Situation, war er dazu programmiert nach dem Prinzip Try&Error vorzugehen, d. h. er wählte per Zufall eine Engine aus. Eine Bewertung der Adaption, die für das Lernen notwendig ist, erfolgt automatisch über die Ausführungszeit der Workflow-Instanz. Liegt diese deutlich über dem Durchschnittswert für diese Workflow-Konfiguration, wird die Adaption schlecht bewertet. Ist die Ausführungszeit gleich dem Durchschnitt, wird sie neutral bewertet. Positiv bewertet wird sie, wenn die Ausführungszeit unter der Durchschnittszeit ist. Der BDI-Direktor lernt demnach, dass wenn die Auslastung von CPU und RAM sich dem Maximum annähert, die Ausführungszeit schlechter wird. Wurde diese Situation gelernt, wird er die Ausführung einer Aktivität nicht auf diese Engine migrieren.

Über einen festgelegten Testzeitraum wurden per Zufall automatisch Workflows gestartet[32], die anschließend automatisch verteilt wurden. Dabei konnten folgende Beobachtungen gemacht werden:

- *Last*: Die Aktivitäten, sowohl Normalisierung als auch Clustering, erzeugen während ihrer Laufzeit fast konstant eine Maximalauslastung eines Kernes. Ein Server mit 8 Kernen ist somit bei 8 gleichzeitigen Aktivitäten vollständig ausgelastet. Da zudem nebenläufige Prozesse durch das Betriebssystem ebenfalls Einfluss haben, steigt die Ausführungszeit bei der 8. Aktivität deutlich.

- *Priorisierung*: Werden nur wenige Instanzen gleichzeitig ausgeführt, priorisiert der BDI-Direktor immer eine bestimmte OSPP-Engine. Dies ist immer die Engine, für die die meisten positiven Cases in der Datenbank existieren. Diese Priorisierung hat bis zu einer Auslastung von 85%[33] bestand. Danach verteilt sie die weiteren Aktivitäten auf eine andere Engine, wobei wiederrum die mit den meisten positiven Cases genutzt wird. Dieser Vorgang wiederholt sich, bis jede Engine ausgelastet ist.

[32] Die Rate der Starts konnte variiert werden.
[33] Ab 85% CPU-Auslastung stiegen die Ausführungszeiten deutlich.

- *Vollständige Auslastung*: Waren alle Engines ausgelastet, konnte der Direktor keine Cases mit positiven Ergebnissen finden, da die Ausführungszeiten immer über dem Durchschnitt lagen. Er verteilte daraufhin alle Anfragen zufällig (Try&Error).

Da auf Mechanismen wie Warteschlange, wie sie bspw. in [97] beschrieben sind, verzichtet wurde, konnte das System bei permanenter Überlast eine Verlängerung der Ausführungszeit nicht verhindern[34]. Der Priorisierung kann entgegengewirkt werden, indem die Cases beispielsweise ein Verfallsdatum erhalten, so dass der Direktor gezwungen ist, wieder neu zu lernen. Zudem ist es möglich, dass der Direktor regelmäßig zufällige Entscheidungen einstreut, um auch andere Adaptionen zu wählen.

Eine erweiterte Lösung für dieses Szenario[35] konnte erfolgreich beim *Service Computing Contest 2010*[36][37] live gezeigt werden.

8.3 Zusammenfassung

Zur Umsetzung der vorgestellten Konzepte wurde ein neues WfMS erstellt: die *Open Service Process Platform* (OSPP). Bei der Umsetzung der Open Service Process Platform wurde neben der Implementierung der Workflow-Engine und der Rekonfigurationsansätze auf *Erweiterbarkeit*, *Vernetzung* und *einfachen Zugriff* geachtet.

OSPP wurde auf Basis des OSGi-Frameworks[38] entwickelt. Das Hauptkonzept von OSGi ist die *Erweiterbarkeit* der Anwendung durch *Bundles*. Ein Bundle ist eine Zusammenstellung verschiedener Java-Klassen, die eine abgeschlossene Funktionalität repräsentieren. Für die Realisierung der vorgestellten Ansätze wurden verschiedene Bundles entwickelt. Zentral ist das Workflow-Engine- und Direktor-Bundle. Letzteres kann durch verschiedene Direktor-Implementierungen erweitert werden, wobei die drei vorgestellten Direktoren implementiert wurden. Zur Einbindung zukünftiger Direktoren wurde eine einheitliche Schnittstelle entwickelt. Ebenfalls erweiterbar wurde das rollenbasierte Workflowmodell konzipiert. So wurden zentrale Rollen mit ObjectTeams umgesetzt und in einem Role-Bundle konzentriert. Weitere Rollen können zu diesen zur Laufzeit hinzugefügt werden. Dadurch ist es ebenfalls möglich, das WfMS rekonfigurierbar zu gestalten. So können Rollen zur Laufzeit nachgeladen werden, um neue Funktionalitäten in das WfMS zu integrieren. Es ist dadurch auch möglich, ein auf eine bestimmte Domäne zugeschnittenes WfMS zu erstellen. Dieses beinhaltet beispielsweise nur eine minimale Menge an Rollen, die für den Anwendungskontext benötigt werden.

Um einen einheitlichen und *einfachen* Zugriff auf alle Funktionen von OSPP zu ermöglichen, wurde eine Webanwendung entwickelt, die in der gleichen OSGi-Anwendung wie die Workflow-Engine verwendet werden kann. Über diese Web-Anwendung sind das Design von Workflows und die Administration von Workflowinstanzen möglich. Zudem können die Adaptionen der Direktoren überwacht oder auch manuell in die Abarbeitung eingegriffen werden (manuelle Adaption).

[34] Dies war jedoch nicht der Fokus der Evaluation.
[35] Zur Berechnung und Ermittlung der Kostenfunktionen wurde ein neuronales Netz verwendet, das auf Basis von Map-Reduce [48] entwickelt wurde.
[36] http://iscc.servicescomputing.org/2010/index.htm (zuletzt aufgerufen am 25.02.2011).
[37] http://iscc.servicescomputing.org/2010/Finalists.htm (zuletzt aufgerufen am 25.02.2011).
[38] Open Services Gateway Initiative.

Semantische BDI-Direktoren sind für die *vernetzte* Kommunikation mit anderen Direktoren konzipiert. Deshalb wurde OSPP offen gestaltet, so dass verschiedene OSPP-Instanzen sich untereinander austauschen können. Diese Funktion wurde auch für die Evaluation verwendet, bei die verteilte Ausführung einzelner Workflowfragmente mit den vorgestellten Rekonfigurationsansätzen untersucht wurde. Hier konnte gezeigt werden, wie einzelne Workflowfragmente einer Instanz auf verschiedene Workflow-Engines auslagern werden können. Dies ist beispielsweise bei Genexpressionsprozessen notwendig, denn aufgrund der großen Ressourcenanforderungen dieser Workflows ist eine Lastverteilung zwischen verschiedenen Workflow-Engines notwendig. In der Evaluation konnte gezeigt werden, dass mithilfe der Adaption durch Verweben und des rollenbasierten Workflowmodells, Genexpressionsprozesse zur Lastverteilung auf verschiedene Workflow-Engines verteilt werden können. Zugleich konnte der Direktor-Koordinator-Ansatz genutzt werden, um die Steuerung der Adaptionen vorzunehmen.

Kapitel 9

Zusammenfassung und Ausblick

Ziel der vorliegenden Arbeit waren vollständig rekonfigurierbare Workflows. Vollständig bedeutet in der Domäne der Workflows, dass alle fünf Hauptperspektiven veränderlich sind. Im Gegensatz zu evolutionären Lösungen sollen die Änderungen jederzeit zur Laufzeit vorgenommen werden, um auf unvorhergesehene Veränderungen reagieren zu können. Während Änderungen an der Verhaltensperspektive Kontrollflussmodifikationen zur Folge haben, bedeuten Änderungen an den *kontextbezogenen Perspektiven* die Veränderung von Eigenschaften einzelner Aktivitäten. Diese Vielfältigkeit der Veränderungen kann durch die in dieser Arbeit vorgestellten Lösungen abgedeckt werden.

9.1 Ergebnisse

Die Lösungsansätze für die Rekonfiguration von Workflowinstanzen lassen sich in zwei Bereiche aufteilen: (1) Kontrollflussänderungen und (2) Änderungen von Eigenschaften. Der erste Bereich deckt die Verhaltensperspektive ab. Die *kontextbezogenen Perspektiven* – informationelle, funktionale, operationale und organisatorische – sind im zweiten Bereich enthalten.

Die Ergebnisse dieser Arbeit decken zudem die Bereiche der autonomen Steuerung ab. Weiterhin konnte gezeigt werden, dass nicht nur Workflows sondern auch das WfMS rekonfigurierbar gestaltet werden können. Wichtig für das Design der Lösung war, dass sie unabhängig von bestehenden WfMS-Implementierungen zum Einsatz kommen können.

Adaption durch Verweben Für die Kontrollflussänderungen wurde die Adaption durch Verweben vorgestellt. Bisherige Lösungen setzen Änderungen am Workflowablauf zumeist durch eine kleine, lokale Änderung an der Struktur um. Bei der Adaption durch Verweben werden alle Änderungen durch das Verweben von separaten Workflowmodellen realisiert. Die Idee ist, dass bei Bedarf beliebige Workflowfragmente in die zu adaptierende Instanz eingewoben werden. Dies geschieht über eine eingeführte Verzweigung, die in den Kontrollfluss eingebracht wird. Über diese Verzweigung können zwei unterschiedliche Adaptionen realisiert werden:

- *Parallele Adaption:* Hierbei wird der Kontrollfluss aufgeteilt. Ein Token verbleibt dabei in der Ursprungsinstanz und setzt die Arbeit wie vorgesehen fort, der andere wird in die einzuwebende Instanz überführt.
- *Serielle Adaption:* Im Gegensatz zur parallelen Adaption wird das Token nicht geteilt, sondern wird in die neue Instanz überführt.

Um einen vollständig rekonfigurierbaren Kontrollfluss zu erreichen, wurde das Konzept des *variablen Rücksprungpunkt* eingeführt. Dieser ermöglicht es, dass nach Abarbeitung des eingewobenen Workflows das Token an einem beliebigen Punkt des Ursprungsworkflows zurückkehren kann, solang dadurch keine Inkonsistenzen, wie sie beispielsweise durch die Verletzung von Datenabhängigkeiten auftreten, entstehen.

Die Kombination aus Verweben und variablen Rücksprungpunkt ermöglicht es, die Vielzahl von Adaptionsmustern, wie sie in [191] beschrieben sind, zu unterstützen. Der Umstand, dass fertige Workflowfragmente eingewoben werden, bietet zudem den Vorteil, dass sich, im Vergleich zu den vielen kleinen lokalen Adaptionen, die Anzahl der möglichen Adaptionen verringert. Die Workflows bilden einen eingeschränkten Variantenraum für Adaptionen. Für die Modellierung von Workflows bedeutet dies zudem ein hohes Maß an Wiederverwendung. Workflowmodelle können auf Basis der Adaption durch Verweben mit vorgefertigten Workflow(fragmenten) bei Bedarf erweitert werden, ohne das Urspungsmodell dauerhaft zu ändern.

Rekonfigurierbares rollenbasiertes Workflowmodell Die Adaption durch Verweben eignet sich für komplexe Eingriffe in den Kontrollfluss. Für Eigenschaftsänderungen an Aktivitäten ist diese Lösung nicht geeignet. Beispiele hierfür sind der Wechsel des Bearbeiters einer Aktivität oder eine neue Ressourcenzuweisung. Um auf diese Probleme effizient reagieren zu können, wurde das *rollenbasierte Workflowmodell* entwickelt.

Es basiert auf dem Konzept der Rollen, einer Erweiterung der objektorientierten Programmierung. Kern dieses Ansatzes ist das Konzept der Rolle, die von einem Objekt gespielt wird. Diese Bindung zwischen Objekt und Rolle kann zur Laufzeit aktiviert und deaktiviert werden. Wird eine Rolle gespielt, kann sie das Verhalten der Basisklasse verändern oder zusätzliche Daten einbringen. Dadurch kann zur Laufzeit aus einem Aufruf für einen Webservices ein *RMI*-Aufruf für ein EJB werden.

Basierend auf dem Konzept der Rolle wurde ein neues modulares Workflowmodell entwickelt. In diesem bildet das Modell eines Workflow-Nets[1] das Zentrum, das den Kontrollfluss des Workflows beschreibt. Alle weiteren Workflowartefakte, seien es Variablendefinitionen, Funktionen oder Ressourcen werden durch Rollen beschrieben, die von den Workflow-Net-Komponenten gespielt werden. Erst durch die Aktivierung verschiedener Rollen wird aus einer einfachen Transition im Workflow-Net ein Webservice-Aufruf. Dadurch, dass alle Eigenschaften durch Rollen beschrieben sind, können alle Aspekte zur Laufzeit frei rekonfiguriert werden. Durch die flexiblen Laufzeiteigenschaften von Rollen entsteht ein rekonfigurierbares Workflowmodell, wobei jede Perspektive[2] verändert werden kann.

Referenzarchitektur zur autonomen Rekonfiguration von Workflows Für die praktische Verwendbarkeit der Adaption durch Verweben und des rollenbasierten Workflowmodells ist eine funktionierende Steuerung notwendig. Manuell durchgeführte Adaptionen sind für viele Anwendungsfälle bei *Production Workflows* zu langsam und zu komplex. Es war deshalb notwendig, eine Möglichkeit zur autonomen Adaptionen zu entwickeln. Dabei ist es wichtig, dass die Adaptionssteuerung sich an neue Bedingungen anpassen kann, die durch unvorhergesehene Änderungen eintreten. Eine wichtige Eigenschaft dafür ist

[1] Derivat eines Petri-Netzes.
[2] Außer der Verhaltensperspektive.

die Reflexion. Des Weiteren sollte die Steuerung unabhängig von einem konkreten WfMS einsetzbar sein.

Zur Umsetzung einer reflexiven Steuerung wurden zwei neue Komponenten eingeführt. Der *Koordinator*, der die Adaptionen für verschiedene Workflow-Engines umsetzt und der *Direktor*, der diese Adaptionen berechnet. Dies geschieht auf Basis von gelerntem Wissen. Diese Wissensbasis kann zur Laufzeit erweitert werden, so dass sich der Direktor an verändernde Umstände anpassen kann. Mittels des Direktors können deshalb sowohl kurzfristige Änderungen, als auch langfristige, evolutionäre, Änderungen abgebildet werden.

In dieser Arbeit wurden drei verschiedene Realisierungsformen für einen Direktor vorgestellt:

- *BDI-Direktor*: Der BDI-Direktor ist eine Realisierungsform, der in verschiedenen Domänen eingesetzt werden kann. Zur Umsetzung wurde eine Technik aus der Forschung zu künstlicher Intelligenz verwendet. Diese Technik basiert auf der Idee der Nachahmung menschlicher Entscheidungen, wobei drei essentielle Konzepte eingeführt werden: Belief, Desire und Intention (BDI). Die Beliefs bilden das Wissen über den Workflow und seine Umgebung ab. Desires sind Ziele, die eine Workflowinstanz verfolgt und Intentions repräsentieren die Absichten, die mit den Zielen (eingewobenen Workflow) verfolgt werden. Der BDI-Direktor untersucht bei jeder Adaptionsentscheidung, welche Adaptionen er in der Vergangenheit bei einer ähnlichen Situation durchgeführt hat. Dazu vergleicht er die Beliefs, Desires und Intention der aktuellen Situation mit den Daten in seinem Hintergrundwissen, das mittels Case-based Reasoning realisiert wurde. Die Domänenunabhängigkeit besitzt allerdings Nachteile bei der Verarbeitung von Zeichenketten. Der Direktor kann hierbei nicht auf Synonyme zurückgreifen. Dadurch kann er nicht korrekt feststellen, ob die zu vergleichenden Situationen gleich sind. Er ist somit vor allem für *Production Workflows* geeignet.

- *Semantischer BDI-Direktor*: Bei dieser Realisierungsform wurde der BDI-Direktor um eine semantische Komponente erweitert. Dies ermöglichte zum einen die Optimierung beim Vergleich der Situationen, wodurch er für *Administrative Workflows* geeignet ist. Dadurch können mit dem BDI-Ansatz nicht nur *Production Workflows* sondern auch *Administrative Workflows* umgesetzt werden. Zum anderen erweitert sie die Nutzungsmöglichkeiten deutlich. Durch die eingeführte Semantik konnten *kooperative Direktoren* entwickelt werden, die untereinander Lösungen austauschen oder voneinander lernen können. Ermöglicht wurde dies durch die Einführung einer Ontologie, die das gelernte Wissen vergleichbar machte. Eine weitere Anwendungsmöglichkeit stellt die *Redundanzerkennung bzw. -vermeidung* dar. Durch die bessere Vergleichbarkeit der aktuellen Situationen von Workflows kann erkannt werden, welche Aktivitäten oder Teilworkflows zu einem bestimmten Zeitpunkt redundant ausgeführt werden. In Verbindung mit der Adaption durch Verweben können redundante Ausführungen vermieden werden.

- *Statechart-Direktor*: Die Qualität der Adaptionsentscheidungen des (semantischen) BDI-Direktor sind abhängig von dessen Erfahrungen in der Vergangenheit. Dies führt in der Anfangsphase zu häufigen Fehlentscheidungen, da nicht genügend Erfahrungen gesammelt wurden. Bei sicherheitskritischen Systemen sind derartige Fehler jedoch nicht akzeptabel. Es war deshalb notwendig eine Direktor-Variante zu entwickeln, die sich berechenbar verhält. Der Statechart-Direktor setzt deshalb

auf unveränderliche Adaptionsentscheidungen, die durch ein Statechart beschrieben werden. In diesem wird festgelegt, unter welchen Bedingungen eine Adaption erfolgen soll. Jede Adaption führt in einen neuen Zustand, so dass Adaptionen andere bedingen können. Diese Zustandsorientierung ermöglicht zudem einen einfachen Überblick, in welchem Adaptionszustand sich eine Instanz befindet.

Weitere Ergebnisse Über diese Hauptergebnisse hinaus wurde bei der Untersuchung zur Vermeidung von redundanten Workflowfragmenten ein neues Adaptionsmuster gefunden: der *Merge*. Wird ein Workflowfragment in zwei unterschiedlichen Instanzen gleichzeitig ausgeführt, kann das Token von der einen Instanz in die andere überführt werden. Dadurch kann er die bisherigen Ergebnisse (Variableninhalte) übernehmen und so lange in der Instanz verweilen, wie diese mit seiner Ursprungsinstanz gleich ist. Ist sie es nicht mehr, wird die Kooperation beendet und das Token verlässt die Instanz.

Zur Umsetzung der vorgestellten Konzepte wurde ein neues WfMS entwickelt: die *Open Service Process Platform* (OSPP). Als Workflow-Beschreibungssprache benutzt sie das rollenbasierte Workflowmodell, dessen Kontrollfluss sich durch die Adaption durch Verweben verändern lässt. Zudem wurde OSPP in vielerlei Hinsicht als modulare Plattform konzipiert:

- *Eingliederung unterschiedlicher Direktoren:* In OSPP wurden nicht nur die drei vorgestellten Direktoren umgesetzt, sondern es existiert eine einheitliche Schnittstelle mit der weitere Direktoren eingebunden werden können.
- *Rekonfigurierbares WfMS:* Durch die Unterstützung des rollenbasierten Workflowmodells ist es in OSPP möglich, ebenfalls das WfMS rekonfigurierbar zu gestalten. So können Rollen zur Laufzeit nachgeladen werden, um neue Funktionalitäten in das WfMS oder die Workflows selbst zu integrieren. Es ist dadurch auch möglich, ein auf eine bestimmte Domäne zugeschnittenes WfMS zu erstellen. Dieses beinhaltet beispielsweise nur eine minimale Menge an Rollen, die für den Anwendungskontext benötigt werden. Zudem können damit domänenspezifische Workflowsprachen realisiert werden.
- *Kooperative WfMS:* Semantische BDI-Direktoren sind für die Kommunikation mit anderen Direktoren konzipiert. Deshalb wurde OSPP offen gestaltet, so dass verschiedene OSPP-Instanzen sich untereinander austauschen können. Diese Möglichkeit wurde auch bei der Verteilung von Fragmenten der Genexpressionsprozesse verwendet.

9.2 Geltungsbereich des Ansatzes

Der vorgestellte Ansatz ist nicht uneingeschränkt anwendbar. So sind alle Direktoren nicht für die *Ad-Hoc Workflows* geeignet, sondern können ihr volles Potential nur bei den *Production Workflows* und *Administrative Workflows* entfalten.

Eine weitere Einschränkung ist bzgl. der Korrektheit gegeben. Trotz der eingeführten Beschränkungen für den Rücksprungpunkt und der Beachtung von Datenabhängigkeiten bei der Adaption (siehe Abschnitt 3.1.1) kann nicht von korrekten Adaptionen ausgegangen werden. Diese Bedingungen garantieren nur die Korrektheit der Webevorgänge

selbst. Es kann jedoch nicht angenommen werden, dass eine Adaption semantisch korrekt ist, dass keine Deadlocks auftreten oder dass eine Adaption nicht immer wieder ausgeführt wird und dadurch eine Endlosschleife erzeugt wird. Adaptionen sind nur so gut, wie sie gelernt wurden, sei es durch Traces oder durch zur Laufzeit gewonnenes Wissen. Beim Statechart-Direktor ist die Korrektheit direkt abhängig vom Nutzer, der die Reihenfolge und Bedingungen für die Adaptionen vorgibt. Der Direktor kann nur auf dieses Wissen zurückgreifen. Hat er eine falsche Adaption[3] gelernt, wird er diese ausführen. Vorteil der reflexiven Direktoren ist allerdings, dass sie diese Fehler langfristig verlernen können.

Zudem wird bei der Überprüfung der Korrektheit nur eine Überprüfung der Datenabhängigkeiten vorgenommen. Eine vollständige Überprüfung für benötigte Ressourcen, wie sie beispielsweise in [153] vorgestellt wurden, würde hier zusätzliche Sicherheit bieten.

Darüber hinaus können keine Garantien für die Antwortzeiten der BDI-Direktoren[4] gegeben werden. Die Antwortzeit hängt direkt von der Größe der Desire- und Intention-Base. Die Antwortzeit kann beim semantischen BDI-Direktor zusätzlich durch die Kommunikation und Kooperation erhöht werden. Sie hängt somit zusätzlich von den Kommunikationskosten ab, die sich aus den Übertragungszeiten und den Timeouts zusammensetzen.

Eine mögliche Hürde bei der praktischen Anwendung des semantischen BDI-Direktors ist die explizite Beschreibung des semantischen Wissens. Ähnlich zu OWL-S erzwingt dieses Vorgehen einen erhöhten initialen Aufwand in der Erstellung der Ontologie. Es existieren bereits Ansätze zum autonomen Erlernen von Ontologien [73, 83, 146, 119], jedoch hätte eine Integration dieser Technologien den Rahmen der Arbeit gesprengt.

Bei der Kooperation zur Redundanzvermeidung (Merge) ist die Redundanzerkennung unerlässlich. Die bisherige Implementation in OSPP, die auf einer Kombination aus SSA und semantischem Vergleich beruht, ist nur für kleine und nicht komplexe Workflows ausreichend. Für eine bessere Erkennung wird eine erweiterte Erkennungsroutine notwendig, wie sie beispielsweise in [151] vorgestellt wird.

9.3 Weiterführende Arbeiten

Neben diesen Einschränkungen existieren weitere interessante Forschungsfragen, die über die primären Fragestellungen der Arbeit hinausgehen. Während der Bearbeitung der Arbeit ergaben sich im Zusammenhang mit anderen Forschungsprojekten neue Fragen, die im Rahmen dieser Arbeit nicht vertieft werden konnten. Neben vielen Detailfragen lassen sich die weiterführenden Arbeiten in vier große Kategorien einteilen: (1) Einführung neuer Direktoren, (2) Übertragung der Referenzarchitektur, (3) Rollenbasiertes rekonfigurierbares Komponentenmodell und (4) Weiterentwicklung von OSPP. Diese werden im Folgenden detailliert vorgestellt.

Übertragung der Referenzarchitektur In den Projekten CoolSoftware (siehe auch Abschnitt 8.2) und ZESSY[5] wird untersucht, inwieweit sich die Konfiguration und Zu-

[3] D. h. eine Adaption, die mit einem positiven Outcome gespeichert wurde, aber semantisch falsch ist.
[4] Die Antwortzeiten des Statechart-Direktors sind gering, denn die Auswertung der Kantenbedingungen kann vernachlässigt werden.
[5] http://www.qualitune.inf.tu-dresden.de/ (zuletzt aufgerufen am 25.08.2011).

sammenstellung komponentenorientierter Software hinsichtlich verschiedener Qualitätseigenschaften ändern kann. Zu diesen Eigenschaften zählen: (1) Realzeiteigenschaften, (2) Ausfallsicherheit, (3) sichere Bewegung im Raum und (4) der Energieverbrauch. Ändern sich die Anforderungen oder der Kontext eines Systems, soll es automatisch angepasst werden. Dies kann bedeuten, dass Komponenten hinzugefügt oder andere Hardware-Ressourcen genutzt werden. Dazu werden die Komponenten im Cool Componenten Model (CCM) [68] beschrieben. Die funktionalen Eigenschaften und damit verbundenen Energieverbräuche (bzw. Qualitäten) werden derzeit auf Grundlage der Energy Contract Language (ECL) [68] beschrieben, deren Vorbild die Component Quality Modeling Language (CQML) [142] ist. Auf Basis dieser Verträge wird derzeit die beste Konfiguration berechnet. In diesem Zusammenhang wäre eine Integration von Direktor und Koordinator sinnvoll. Beide könnten so angepasst werden, dass sowohl die Rekonfigurationen berechnet werden, als auch die Rekonfigurationen durchgeführt werden. Basis kann dabei sowohl ECL als auch eine andere Wissensbasis sein.

Wird dieser Gedanke weiterverfolgt, so kann die Referenzarchitektur auch bei der Steuerung von in *Cyber-Physical Systems* eingesetzt werden. Diese erweitern den Systembegriff insoweit, indem sie eine Einheit aus virtuellem und physischem System beschreiben. Das physikalische System (zumeist ein eingebettetes System), bestehend aus Aktoren und Sensoren, wird dabei durch das virtuelle System abgebildet. Dieses entscheidet auf Basis der aktuellen Sensorenwerte über auszuführende Aktionen, die an die Aktoren weitergegeben werden. Auch hier können Direktor und Koordinator eingesetzt werden, um die Befehle für die Aktoren zu berechnen.

Einführung neuer Direktoren Die drei vorgestellten Direktoren stellen nur Möglichkeiten für die Umsetzung einer autonomen Steuerung dar. In OSPP können jedoch beliebige Direktoren eingebunden werden (siehe Abschnitt 8.1.1). Denkbar sind verschiedene Möglichkeiten zur Steuerung, wovon zwei hervorstechen:

- *Geschäftsregeln:* Ein wichtiger Faktor beim Erstellen von Geschäftsprozessen ist die Beachtung von Gesetzen und Geschäftsregeln. Bei Geschäftsprozessen werden immer Entscheidungspunkte eingeführt, die auf diesen Regeln basieren. Beispielsweise muss ein Kunde 18 Jahre alt sein, um einen Bestellvorgang durchzuführen. Ist er es nicht, werden andere Aktionen ausgeführt, die explizit spezifiziert werden müssen. Diese Regeln sind ständigen Änderungen unterworfen. Soll ein Kunde nun bereits mit einem Alter von 16 Jahren bestellen dürfen, müssen alle Stellen im Workflow identifiziert werden, in denen die ursprüngliche Regel enthalten ist. In der Praxis setzt sich die Modellierung der Unternehmensaktivitäten aus zahlreichen Geschäftsprozessen zusammen, so dass das Auffinden dieser Stellen aufwändig ist. In [198] untersuchte zur Muehlen Geschäftsprozesse bei einem amerikanischen Versicherungsunternehmen. In einem Workflow zur Verarbeitung von Versicherungsansprüchen wurden 12 Kernaktivitäten und über 5000 Regeln identifiziert. Würden all diese Regeln explizit durch Entscheidungspunkte modelliert, entstände ein Workflow, der nicht mehr überschaut und verwaltet werden kann.

  Mithilfe der Adaption durch Verweben in Verbindung mit einem Direktor, der auf formalisierten Geschäftsregeln basiert, kann das Problem gelöst werden. Ähnlich dem Statechart-Direktor würde eine Regel-Meta-Ebene entstehen, die einfach gepflegt werden kann. Wie in [135] vorgeschlagen, können die Geschäftsregeln aus

einer informalen SBVR-Beschreibung[6] [108] abgeleitet und in OCL[7] [116] oder PRR[8] [167] transformiert werden. Der Direktor könnte mittels dieser Regeln den Geschäftsprozess adaptieren, so dass der Bestellvorgang eines Kunden im Alter von 15 Jahren abgebrochen wird, ohne dass dieses explizit im Geschäftsprozess modelliert wurde.

- Im Rahmen des *Service Computing Contest 2010* wurde eine erste Version eines Direktors auf Basis eines neuralen Netzes vorgestellt [139]. Ähnlich dem semantikfreien BDI-Direktor können damit vor allem Adaptionen in Workflows mit mathematischen Berechnungen eingesetzt werden. Interessant wäre eine Untersuchung über weitere Einsatzmöglichkeiten dieses Direktors. Weiterhin ist für diese Realisierungsform zu untersuchen, wie der Direktor verteilt werden kann und die verschieden Optimierungen bei der Berechnung des neuralen Netzes angewendet werden können[9].

Rollenbasiertes rekonfigurierbares Frameworkmodell Auch die Ideen für das rekonfigurierbare WfMS können weiter entwickelt werden. Neue technische Entwicklungen im Hardwarebereich führten zu immer mobileren und leistungsfähigeren Geräten. Vor allem durch den mobilen Einsatz ergeben sich neue Herausforderungen für die Softwareentwicklung. Mehr als bisher steht der Kontext bei der Nutzung von Software im Mittelpunkt. Anwendungen müssen sich beispielsweise an den Ort oder die Zeit anpassen und sinnvolle Informationen bereitstellen. Die Anpassung an den jeweiligen Kontext erfordert eine auf Veränderungen vorbereitete Anwendungsarchitektur. Große, monolithische, Anwendungen sind schlecht anpassbar und sind aufgrund begrenzter Ressourcen für den mobilen Bereich nicht geeignet. Eine klassische Lösung besteht darin, die Anwendung in einzelne (unabhängige) Komponenten aufzuteilen. Dieser Ansatz kann durch das Konzept der Rollen erweitert werden.

Die in Kapitel 7 und 8 eingeführte dynamische Rekonfiguration eines WfMS kann als Basis für derartige rekonfigurierbare Anwendungen dienen. Der Kern des Konzeptes besteht darin, dass durch das Einbringen neuer Role-Bundles oder Aktivierung bzw. Deaktivierung bestehender Rollen, das WfMS zu adaptieren. Es ist daher möglich, dass eine Anwendung, je nach Kontext, neue Komponenten hinzufügt oder ablegen kann. Neue Komponenten bzw. Role-Bundles können zudem aus einem Repository nachgeladen werden. Es entsteht ein Verbund (Grid) von Kleinstanwendungen (Apps), deren Zusammenstellung sich autonom dem jeweiligen Kontext anpasst. Es entstünde ein *Smart Application Grid*.

Weiterentwicklung von OSPP Auf der Seite der Implementierung stellt OSPP nur eine akademische Realisierung der vorgestellten Konzepte dar. Weder sind alle Funktionen für ein vollständiges WfMS vorhanden, noch sind alle integrierten Komponenten vollständig. Neben der vollständigen Implementierung sind weitere konzeptionelle Arbeiten in der Werkzeugunterstützung denkbar. Dazu zählt die Verbesserung der intuitiven Nutzbarkeit der Webanwendung und des Deployments.

[6] Semantics of Business Vocabulary and Business Rules.
[7] Object Constraint Language.
[8] Production Rule Representation.
[9] In [139] wurde untersucht, inwieweit der Map-Reduce Ansatz [48] helfen kann.

Anhang A

Berechnungsvorschriften

A.1 Berechnung der Nachfolger von Markierung m ($succ(m)$) (aus [21])

gegeben ist $m; succ(m) \leftarrow \oslash$
for all $t \in T$
do if Enabled (t, m)
  then $succ(m) \leftarrow succ(m) \cup m + c_t$;
od;
return $succ(m)$;

A.2 Berechnung der Vorgänger von Markierung m ($pred(m)$) (aus [21])

gegeben ist $m; pred(m) \leftarrow \oslash$
for all $t \in T$
do $m' \leftarrow m - c_i$ /*$m' \in pospred(m)$! */
  if $m' \in S$ /* lookup wheter m' exists at all */
  then if Enabled (t, m') **then** $pred(m) \leftarrow pred(m) \cup m'$;
od;
return $pred(m)$;

Anhang B

Literaturverzeichnis

[1] Aamodt, A. and Plaza, E. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] Abela, C. and Montebello, M. PreDiCtS: A Personal Dicovery and Composition of Services. Website: http://staff.um.edu.mt/cabe2//research/projects/predicts/predicts.html, 2009.

[3] Abela, C. and Montobello, M. CCBR Ontology for Reusable Service Templates. *Demos and Posters of the 3rd European Semantic Web Conference*, 1:3, 2006.

[4] Adams, M., ter Hofstede, A. H. M., Edmond, D., et al. Dynamic and extensible exception handling for workflows: A service-oriented implementation. Technical report, Business Process Management Group Queensland University of Technology, Brisbane, Australia, 2007.

[5] Adams, M. J., ter Hofstede, A. H., Edmond, D., et al. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In R. Meersman and Z. Tari, editors, *the 14th International Conference on Cooperative Information Systems (CoopIS'06)*, pages 291–308. Springer, Montpellier, France, 2006.

[6] Affymetrix Inc. Genechip expression analysis technical manual. Technical report, 2001.

[7] Affymetrix Inc. New statistical algorithms for monitoring gene expression on genechip probe arrays. Technical report, 2001.

[8] Allen, F. E. Control flow analysis. *SIGPLAN Not.*, 5:1–19, 1970.

[9] Alonso, E., Luck, M., Kudenko, D., et al. Learning in Multi-Agent Systems. *Knowledge Engineering Review, 16*, 3:277–284, 2001.

[10] Baldoni, M., Boella, G., and van Der Torre, L. Powerjava: ontologically founded roles in object oriented programming language. In *Proc. of 21st ACM Symposium on Applied Computing, SAC 2006, Special Track on Object-Oriented Programming Languages and Systems (OOPS 2006*, pages 1414–1418. ACM, 2006.

[11] Baldoni, M., Boella, G., and van der Torre, L. The interplay between relationships, roles and objects. In *Proceedings of the 3rd International Conference on Fundamentals of Software Engineering (FSEN'09)*. 2009.

[12] Balzer, S., Gross, T. R., and Eugster, P. A relational model of object collaborations and its use in reasoning about relationships. In E. Ernst, editor, *Proceedings of the 21st European Conference on Object-Oriented Programming, ECOOP*, pages 323–346. 2007.

[13] Balzert, H. *Lehrbuch der Softwaretechnik / Helmut Balzert[2]: Softwaremanagement.* Lehrbücher der Informatik. Spektrum, Akad. Verl., Heidelberg, 2. Aufl. edition, 2008.

[14] Barbara, D., Mehrotra, S., and Rusinkiewicz, M. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1):5–15, 1996.

[15] Baresi, L., Maurino, A., and Modafferi, S. Workflow partitioning in mobile information systems. In E. Lawrence, B. Pernici, and J. Krogstie, editors, *MOBIS*, volume 158 of *IFIP International Federation for Information Processing*, pages 93–106. 2004.

[16] Bäumer, D., Riehle, D., Siberski, W., et al. Role object. In *Pattern Languages of Program Design 4*, pages 15–32. Addison-Wesley, 2000.

[17] Bechhofer, S., van Harmelen, F., Hendler, J., et al. OWL Web Ontology Language Reference. Technical report, W3C, http://www.w3.org/TR/owl-ref/, 2004.

[18] Becker, J. *Prozessmanagement: Ein Leitfaden zur prozessorientierten Organisationsgestaltung.* Springer, Berlin, 5., Überarb. und erw. Aufl. edition, 2005.

[19] Beckett, D. and Berners-Lee, T. Turtle - Terse RDF triple language, W3C team submission, 2008. Http://www.w3.org/TeamSubmission/turtle/.

[20] Beierle, C. and Kern-Isberner, G. *Methoden wissensbasierter Systeme.* Vieweg Verlag, 2005.

[21] Bell, A. and Haverkort, B. R. Sequential and distributed model checking of petri nets. *International Journal on Software Tools for Technology Transfer (STTT)*, 7:43–60, 2005. 10.1007/s10009-003-0129-2.

[22] Bellwood, T., Capell, S., Clement, L., et al. UDDI Version 3.0.2 http://uddi.org/pubs/uddi_v3.htm, 2004.

[23] Ben-Ari, M., Manna, Z., and Pnueli, A. The temporal logic of branching time. In *Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '81, pages 164–176. ACM, New York, NY, USA, 1981.

[24] Berners-Lee, T. and Connolly, D. Notation3 (N3): A readable RDF syntax. Technical report, W3C, 2008.

[25] Bierman, G. M. and Wren, A. First-class relationships in an object-oriented language. In *Proceedings of the 19th European Conference on Object-Oriented Programming, ECOOP*, pages 262–286. 2005.

[26] Bohn, H., Bobek, A., and Golatowski, F. WS-BPEL process compiler for resource-constrained embedded systems. In *AINA Workshops*, pages 1387–1392. IEEE Computer Society, 2008.

[27] Bratman, M. *Intention, plans, and practical reason.* Harvard University Press Cambridge, Mass, 1987.

[28] Brazdil, P., Gams, M., Sian, S., et al. Learning in Distributed Systems and Multi-Agent Environments. *Proceedings of the European working session on learning on Machine learning*, 1:412 – 423, 1991.

[29] Brazma, A. and et al, P. H. Minimum information about a mircoarray experiment (MIAME) - toward standards for microarray data. In *Nature Genetics, suppl. 19*. 2001.

[30] Briggs, P., Cooper, K. D., Harvey, T. J., et al. Practical improvements to the construction and destruction of static single assignment form. *Software – Practice and Experience*, 28(8):859–881, 1998.

[31] Brogi, A., Corfini, S., and Iardella, S. From OWL-S Descriptions to Petri Nets. In E. Nitto and M. Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, pages 427–438. Springer-Verlag, Berlin, Heidelberg, 2009.

[32] Bry, F., Eckert, M., Patranjan, P., et al. Realizing business processes with ECA rules: Benefits, challenges, limits. In *Proc. Int. Workshop on Principles and Practice of Semantic Web*. Springer, 2006.

[33] Bry, F. and Patranjan, P. Reactivity on the web: Paradigms and applications of the language XChange. *J. of Web Engineering*, 5:2006, 2005.

[34] Buford, J., Jakobson, G., and Lewis, L. Extending BDI Multi-Agent Systems with Situation Management. *Proceeding of the 9th International Conference on Information Fusion*, 1:1–7, 2006.

[35] Carlsen, S. Conceptual Modeling and Composition of Flexible Workflow Models. *Norwegian University of Science and Technology*, 1997.

[36] Casati, F. Semantic Interoperability in Interorganizational Workflows. *WACC workshop on cross-organizational workflows, San Francisco, CA, USA, February*, 1999.

[37] Casati, F., Ceri, S., Pernici, B., et al. Deriving active rules for workflow enactment. In *Proceedings of the 7th International Conference on Database and Expert Systems Applications*, DEXA '96, pages 94–115. Springer-Verlag, London, UK, 1996.

[38] Casati, F., Ceri, S., Pernici, B., et al. Workflow Evolution. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Cottbus, Germany, October 7-10, 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 1996.

[39] Casati, F. and Pozzi, G. Modeling exceptional behaviors in commercial workflow management systems. In *Cooperative Information Systems, 1999. CoopIS '99. Proceedings. 1999 IFCIS International Conference on*, pages 127 –138. 1999.

[40] Charfi, A. and Mezini, M. Aspect-oriented web service composition with AO4BPEL. In L.-J. Zhang, editor, *ECOWS*, volume 3250 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2004.

[41] Christophides, V., Hull, R., Karvounarakis, G., et al. Beyond discrete e-services: Composing session-oriented services in telecommunications. *Proc. of Workshop on Technologies for E-Services (TES)*, 2001.

[42] Chung, P., Cheung, L., Stader, J., et al. Knowledge-based process management–an approach to handling adaptive workflow. *Knowledge-Based Systems*, 16(3):149–160, 2003.

[43] Cohen, P. R. and Jensen, D. Overfitting explained, 1997.

[44] Corchado, J., Pavón, J., Corchado, E., et al. Development of CBR-BDI Agents: A Tourist Guide Application. *7th European Conference on Case-based Reasoning*, pages 547–559, 2004.

[45] Courbis, C. and Finkelstein, A. Towards an aspect weaving BPEL engine. *The Third AOSD Workshop on Aspects Components and Patterns for Infrastructure Software ACP4IS Lancaster UK*, (March), 2004.

[46] Cytron, R., Ferrante, J., Rosen, B. K., et al. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13:451–490, 1991.

[47] Dadam, P., Kuhn, K., Reichert, M., et al. ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen. *Proc. 25. GI-Jahrestagung (GISI 95)*, pages 677–686, 1995.

[48] Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, 2008.

[49] Dellen, B., Maurer, F., and Pews, G. Knowledge-based techniques to increase the flexibility of workflow management. *Data Knowledge Engineering*, 23(3):269–315, 1997.

[50] Do, H. H., Kirsten, T., and Rahm, E. Comparative evaluation of microarray-based gene expression databases. In *BTW 2003, Datenbanksysteme für Business, Technologie und Web, Tagungsband der 10. BTW-Konferenz, 26.-28. Februar 2003, Leipzig*, volume 26 of *LNI*, pages 482–501. GI, 2003.

[51] Dobson, S., Denazis, S., Fernández, A., et al. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.

[52] Drysdale, D. *An Introduction to Fire Dynamics, 2nd Edition*. John Wiley & Sons, 1998.

[53] Dumas, M., van der Aalst, W. M., and ter Hofstede, A. H. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

[54] Ellis, C., Keddara, K., and Rozenberg, G. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 10–21. ACM, New York, NY, USA, 1995.

[55] Emerson, E. A. and Lei, C.-L. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.

[56] Fensel, D., Lausen, H., Polleres, A., et al. *Enabling Semantic Web Services*. Springer-Verlag Berlin Heidelberg, 2007.

[57] Fenton-Kerr, T., Clark, S., Cheney, G., et al. Multi-Agent Design In Flexible Learning Environments. *Proceedings of the Ascalite '98*, 1:223–229, 1998.

[58] Ferber, J. *Multiagentensysteme: Eine Einführung in die künstliche Intelligenz*. Addison-Wesley, 2001.

[59] Fipa. *FIPA Abstract Architecture Specification*. FIPA, 2001.

[60] Fipa. *FIPA ACL Message Structure Specification*. FIPA, 2001.

[61] Flocchini, P., Prencipe, G., Santoro, N., et al. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots, 1999.

[62] Fox, G. C. and Gannon, D. Special issue: Workflow in grid systems: Editorials. *Concurr. Comput. : Pract. Exper.*, 18:1009–1019, 2006.

[63] Gamma, E., Helm, R., Johnson, R., et al. *Design Patterns: Elements of Reusable Object-Oriented Languages and Systems*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[64] Georgakopoulos, D., Hornick, M., and Sheth, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, pages 119–153, 1995.

[65] Georgiadis, I., Magee, J., and Kramer, J. Self-organising software architectures for distributed systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM, New York, NY, USA, 2002.

[66] Goh, A., Koh, Y.-K., and Domazet, D. ECA rule-based support for workflows. *Artificial Intelligence in Engineering*, 2001.

[67] Götz, S., Richly, S., and Aßmann, U. Role-based object-relational co-evolution. In *Proceedings of 8th Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE 2011)*. 2011.

[68] Götz, S., Wilke, C., Schmidt, M., et al. Towards energy auto tuning. In *Proceedings of First Annual International Conference on Green Information Technology (GREEN IT)*. 2010.

[69] Guizzardi, G. *Ontological foundations for structural conceptual models*. Ph.D. thesis, University of Twente, Enschede, The Netherlands, Enschede, 2005.

[70] Habich, D., Richly, S., Lehner, W., et al. Data-aware SOA for gene expression analysis processes. In *IEEE SCW*, pages 138–145. IEEE Computer Society, 2007.

[71] Habich, D., Richly, S., Preissler, S., et al. Data-grey-box web services in data centric environments. In *Proceedings of the 2007 International Conference on Web Services (ICWS 2007)*, pages 976–983. 2007.

[72] Halpin, T. A. *A Logical Analysis of Information Systems: static aspects of the data-oriented perspective*. Ph.D. thesis, University of Queensland, 1989.

[73] Hammack, C. and Scott, S. Lasso: A learning architecture for semantic web ontologies. In *Machine Learning and Applications, 2004. Proceedings. 2004 International Conference on*, pages 10 – 17. 2004.

[74] Harel, D. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[75] Hawiyszkienycz, I., and Debenham, J. A Workflow System Based on Agents. *Database and Expert Systems Applications: 9th International Conference, DEXA'98, Vienna, Austria, August 24-28, 1998: Proceedings*, 1998.

[76] He, C., Nie, Z., Li, B., et al. Rava: Designing a Java extension with Dynamic Object Roles. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pages 453 – 459. IEEE Computer Society, 2006.

[77] Heidenreich, F., Johannes, J., Karol, S., et al. Derivation and Refinement of Textual Syntax for Models. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2009)*, volume 5562 of *LNCS*, pages 114–129. Springer, 2009.

[78] Herbst, J. A machine learning approach to workflow management. *Proceedings 11th European Conference on Machine Learning*, 1810:183–194, 2000.

[79] Hernández, A. G., Fallah-Seghrouchni, A. E., and Soldano, H. Distributed Learning in Intentional BDI Multi-Agent Systems. *Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)*, 1:225–232, 2004.

[80] Herrmann, S. Object Teams: Improving modularity for crosscutting collaborations. In *NetObjectDays*, pages 248–264. 2002.

[81] Herrmann, S., Hundt, C., and Mosconi, M. ObjectTeams / Java Language Definition - Version 1.0. Technical report, Technical University Berlin, 2007.

[82] Hitzler, P., Kroetzsch, M., Rudoplh, S., et al. *Semantic Web*. Springer, 2008.

[83] Hlj, H. and you Liu, D. Learing OWL ontologies from free texts. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics*. 2004.

[84] Hollingsworth, D. Workflow management coalition specification: the workflow reference model. Technical report, WfMC specification, 1994.

[85] Hopcroft, J. E., Motwani, R., and Ullmann, J. D. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Pearson Studium, München, 2., überarbeitete aufl. edition, 2002.

[86] Howden, N., Ronnquist, R., Hodgson, A., et al. JACK Intelligent Agents-Summary of an Agent Infrastructure. *5th International Conference on Autonomous Agents*, 2001.

[87] Hsu, M. and Kleissner, C. Objectflow: Towards a process management infrastructure. *Distributed and Parallel Databases*, 4(2):169–194, 1996.

[88] Huang, J. and Pearce, A. R. Distributed Interactive Learning in Multi-Agent Systems. *Proceedings of the 21th National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, 1:665–671, 2006.

[89] Huber, M. JAM: a BDI-theoretic mobile agent architecture. *Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, 1999.

[90] J. Bae and H. Bae and S. Kang and Y. Kim. Automatic control of workflow processes using ECA rules. *Knowledge and Data Engineering*, 2004.

[91] Jablonski, S. Workflow-Management-Systeme: Motivation, Modellierung, Architektur. *Informatik Spektrum*, pages 13–24, 1995.

[92] Jablonski, S., Böhm, M., and Schulze, W. *Workflow-Management*. Dpunkt Verlag, 1997.

[93] Jablonski, S. and Bussler, C. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Itp New Media, 1996.

[94] Jena Team. Jena - A Semantic Web Framework for Java. Website: http://jena.sourceforge.net, 02/2009.

[95] Jennings, N., Sycara, K., and Woolrigde, M. A Roadmap of Agent Research and Development. *Kluwer Academic Publishers, Boston*, 1:1–32, 1998.

[96] Jimenez-Ochoa, I., Begovich, O., Ramirez-Trevino, A., et al. Implementing BDI agents using petri nets. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 1, pages 286 – 291 vol.1. 2003.

[97] Jin, L.-j., Casati, F., Sayal, M., et al. Load balancing in distributed workflow management system. In *Proceedings of the 2001 ACM symposium on Applied computing*, SAC '01, pages 522–530. ACM, New York, NY, USA, 2001.

[98] Joeris, G. and Herzog, O. Managing evolving workflow specifications with schema versioning and migration rules. Technical Report 15-1999, Center for Computing Technologies (TZI), University of Bremen, 1999.

[99] Joncheere, N. and van der Straeten, R. Uniform modularization of workflow concerns using unify. *4th International Conference on Software Language Engineering; SLE 2011*, 2011.

[100] Kammer, P., Bolcer, G., Taylor, R., et al. Techniques for Supporting Dynamic and Adaptive Workflow. *Computer Supported Cooperative Work (CSCW)*, 9(3):269–292, 2000.

[101] Kammer, P. J. *A Distributed Architectural Approach to Supporting Work Practice*. Ph.D. thesis, University Of California, Irvine, 2004.

[102] Kappel, G., Lang, P., Rausch-Schott, S., et al. Workflow management based on objects, rules, and roles. *IEEE Bulletin of the Technical Committee on Data Engineering*, 18:11–18, 1995.

[103] Kolodner, J. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., 1993.

[104] Koulopoulos, T. M. *The Workflow Imperative: Building Real World Business Solutions*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[105] Kramer, J. and Magee, J. A rigorous architectural approach to adaptive software engineering. *J. Comput. Sci. Technol.*, 24:183–188, 2009.

[106] Lawrence, P., Bouzeghoub, M., Fabret, F., et al. Workflow Handbook. In *Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW 99*. 1997.

[107] Leen-Kiat Soh. Multiagent Ditributed Ontology Learing, 2002.

[108] Linehan, M. H. SBVR Use Cases. In *Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*, RuleML '08, pages 182–196. Springer-Verlag, Berlin, Heidelberg, 2008.

[109] Lipschutz, R. J. High-density synthetic ologonucleotide arrays. In *Nature Genetics, suppl. 21*, pages 20–24. 1999.

[110] Martin, D. L., Paolucci, M., McIlraith, S. A., et al. Bringing semantics to web services: The OWL-S approach. In J. Cardoso and A. P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer, 2004.

[111] Müller, R. and U. Greiner, E. R. AGENTWORK: A Workflow-System Supporting Rule-Based Workflow Adaptation. *Data and Knowledge Engineering*, 2004.

[112] Monpratarnchai, S. and Tetsuo, T. The design and implementation of a role model based language, EpsilonJ. In *Proceedings of the 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2008)*, pages 37–40. 2008.

[113] Myers, K. and Berry, P. Workflow management systems: An AI perspective. *Artificial Intelligence Center, SRI International, Menlo Park, CA USA*, 1999.

[114] Nutt, G. The evolution toward flexible workflow systems. *Distributed Systems Engineering*, 3(4):276–294, 1996.

[115] Oberweis, A., Wendel, T., and Stucky, W. Teamwork coordination in a distributed software development environment. In *in Innovationen bei Rechen- und*, pages 423–429. Springer-Verlag, 1994.

[116] Object Management Group. The object constraint language specification, version 2.0. http://www.omg.org/spec/OCL/2.0/PDF/, 2006.

[117] Object Management Group. The unified modelling language specification, version 2.2. http://www.omg.org/spec/UML/2.2/, 2009.

[118] Olivia, C., Chang, C., Enguix, C., et al. Case-Based BDI Agents: An Effective Approach for Intelligent Search on the World Wide Web. *Intelligent Agents in Cyberspace. AAAI Spring Symposium*, 1999.

[119] Omelayenko, B. Learning of ontologies for the web: the analysis of existent approaches. In *In Proceedings of the International Workshop on Web Dynamics*. 2001.

[120] OSGi Alliance. OSGi Service Platform Release 4. Webseite: http://www.osgi.org/Main/HomePage., 2007.

[121] Pellegrini, S. and Giacomini, F. Design of a petri net-based workflow engine. In *GPC-WORKSHOPS '08: Proceedings of the 2008 The 3rd International Conference on Grid and Pervasive Computing - Workshops*, pages 81–86. IEEE Computer Society, Washington, DC, USA, 2008.

[122] Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. Declare: Full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007.

[123] Pokahr, A., Braubach, L., and Lamersdorf, W. Jadex: A BDI reasoning engine. *Multi-Agent Programming: Languages, Platforms and Applications. Springer, Berlin*, 2005.

[124] Purvis, M., Purvis, M., and Lemalu, S. A framework for distributed workflow systems. In *In Procs. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34). IEEE Computer Society*. 2001.

[125] Rao, A. AgentSpeak: BDI Agents Speak Out in a Logical Computable Language. *Agents Breaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, Netherlands, January 22-25, 1996: Proceedings*, 1996.

[126] Rao, A. and Georgeff, M. BDI Agents: From Theory to Practice. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[127] Reenskaug, T., Wold, P., and Lehne, O. *Working with objects - The OOram Software Engineering Method*. TASKON, 1995.

[128] Regev, G., Soffer, P., and Schmidt, R. Taxonomy of flexibility in business processes. In *BPMDS* [129].

[129] Regev, G., Soffer, P., and Schmidt, R., editors. *Proceedings of the CAISE*06 Workshop on Business Process Modelling, Development, and Support BPMDS '06, Luxemburg, June 5-9, 2006*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[130] Reichert, M., Bauer, T., and Dadam, P. Flexibility for distributed workflows. In *Handbook of Research on Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments*. IGI Global, Hershey, PA, 2009.

[131] Reichert, M. and Dadam, P. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10:93–129, 1998.

[132] Reisig, W. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Leitfäden der Informatik. Vieweg+Teubner, 2010. 248 pages; ISBN 978-3-8348-1290-2.

[133] Richly, S., Buecke, W., and Assmann, U. A BDI-Based Reflective Infrastructure for Dynamic Workflows. In *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops*, pages 112–119. IEEE Computer Society, Washington, DC, USA, 2008.

[134] Richly, S., Götz, S., Schmidt, S., et al. Role-based multi-purpose workflow engine architecture. In *International Joint Workshop on Technologies for Context-Aware Business Process Management, Advanced Enterprise Architecture and Repositories and Recent Trends in SOA Based Information*, volume 1, pages 45–54. 2010.

[135] Richly, S., Habich, D., Gietl, F., et al. Joining business rules and business processes. In *Proceedings of the 16th Int. Conference on Information and Software technologies (IT 2010), ed. by A. Targamadze et al, Kaunas, Lithuania, April 21-23, 2010*. 2010.

[136] Richly, S., Habich, D., Ruempel, A., et al. Open service process platform 2.0. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008, 8-11 Jul, Hawaii, USA)*. 2008.

[137] Richly, S., Habich, D., Thiele, M., et al. Supporting gene expression analysis processes by a service-oriented platform. In *IEEE SCC*, pages 739–746. IEEE Computer Society, 2007.

[138] Richly, S., Hartwig, C., Voigt, H., et al. Modellierung komplexer Workflows in jExam. In *Informatiktage*, pages 33–36. 2006.

[139] Richly, S., Pueschel, G., Habich, D., et al. Mapreduce for scalable neural nets training. In *SERVICES*, pages 99–106. IEEE Computer Society, 2010.

[140] Richly, S., Schmidt, S., and Assmann, U. A semantic-BDI-based approach to realize cooperative, reflexive workflows. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 1680 –1685. 2010.

[141] Riehle, D. and Gross, T. Role model based framework design and integration. In *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 117–133. ACM, New York, NY, USA, 1998.

[142] Roettger, S. and Zschaler, S. CQML+: Enhancements to CQML. In *In Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering*, pages 43–56. Cépaduès-Éditions, 2003.

[143] Ross, R. G. The business rule approach. *IEEE Computer*, 36(5):85–87, 2003.

[144] Rumbaugh, J. Relations as semantic constructs in an object-oriented language. In *Proceedings of the 2nd ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 466–481. 1987.

[145] Russel, S. and Norvig, P. *Artificial intelligence*. Prentice-Hall, 2nd edition edition, 2003.

[146] Sabou, M., Wroe, C., Goble, C., et al. Learning domain ontologies for semantic web service descriptions. *Journal of Web Semantics*, 3:40, 2005.

[147] Sadiq, S. Handling dynamic schema change in process models. *Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian*, pages 120–126, 2000.

[148] Sadiq, S. W. and Orlowska, M. E. On capturing exceptions in workflow process models, 2000.

[149] Sadiq, S. W., Orlowska, M. E., and Sadiq, W. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30:349–378, 2005.

[150] Sadiq, S. W., Sadiq, W., and Orlowska, M. E. Pockets of flexibility in workflow specification. In *ER 01*, pages 513–526. 2001.

[151] Saha, D., Samanta, A., and Sarangi, S. R. Theoretical framework for eliminating redundancy in workflows. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, SCC '09, pages 41–48. IEEE Computer Society, Washington, DC, USA, 2009.

[152] Sander, J., Ester, M., Kriegel, H.-P., et al. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[153] Schmidt, R. *Aspektorientierte Komponentensysteme zur Unterstützung weitreichender Geschäftsprozesse*. Ph.D. thesis, Universität Karlsruhe, 1999.

[154] Schmidt, R. Flexibility in service processes. In Regev et al. [129].

[155] Schmidt, R. and Assmann, U. CompFlow: A Component-based and Aspect-separated Architecture for Workflow Support. In *Proceedings EDBT ´98 Workshop on Workflow Management Systems*. 1998.

[156] Schonenberg, M. H., Mans, R. S., Russell, N. C., et al. Towards a taxonomy of process flexibility, 2008.

[157] Schut, P. OpenGIS Web Processing Service. Technical report, 2007.

[158] Seow, K. T., Sim, K. M., Ong, Y. S., et al. A BDI Assignment Protocol With New Cooperative-Concession Strategies. *IEEE Transactions on Systems, Men, and Cybernetics - Part A: Systems and Humans*, 38:688–697, 2008.

[159] Shen, S., Ohare, G. M. P., and Collier, R. Decision-Making of BDI Agents, a Fuzzy Approach. *Proceedings of the Fourth International Conference on Computer and Information Technology*, 1:1022–1027, 2004.

[160] Sheth, A. P., Kochut, K., Miller, J. A., et al. Supporting state-wide immunisation tracking using multi-paradigm workflow technology. In *VLDB'96*, pages 263–273. 1996.

[161] Smolander, K. OPRR: A model for modelling systems development methods. *Next Generation CASE Tools, K. Lyytinen and V.-P. Tahvanainen (Ed.)*, 1991.

[162] Stanford Center for Biomedical Informatics Research. The Protege Ontology Editor and Knowledge Acquisition System (http://protege.stanford.edu/), 02/2009.

[163] Steimann, F. On the representation of roles in object-oriented and conceptual modelling. *IEEE Transactions on Data and Knowledge Engineering*, 35(1):83–106, 2000.

[164] Stone, P. *Layered Learning in Multi-Agent Systems*. Master's thesis, School of Computer Science, Carnegie Mellon University, 1998.

[165] Strong, D. M. and Miller, S. M. Exceptions and exception handling in computerized information processes. *ACM Trans. Inf. Syst.*, 13:206–233, 1995.

[166] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

[167] Tabet, S., Wagner, G., Spreeuwenberg, S., et al. OMG production rule representation - context and current status. In *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*. W3C, 2005.

[168] Tamai, T., Ubayashi, N., and Ichiyama, R. Objects as actors assuming roles in the environment. In *LNCS Software Engineering for Multi-Agent Systems V: Research Issues and Practical Applications*, pages 185–203. 2007.

[169] Tan, W. and Fan, Y. Model fragmentation for distributed workflow execution: A petri net approach. In F. F. R. Corchado, V. Larios-Rosillo, and H. Unger, editors, *ISSADS*, volume 3563 of *Lecture Notes in Computer Science*, pages 207–214. Springer, 2005.

[170] Tan, W. and Fan, Y. Dynamic workflow model fragmentation for distributed execution. *Comput. Ind.*, 58:381–391, 2007.

[171] The Eclipse Foundation. BPEL to Java (b2j), webseite: http://www.eclipse.org/stp/b2j/, 2011.

[172] The OpenJMS Group. OpenJMS. Website: http://openjms.sourceforge.net/, 02/2009.

[173] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services, 09/2008.

[174] Trapp, D. M. *Optimierung objektorientierter Programme*. Springer-Verlag Berlin Heidelberg New York, 2001.

[175] Uchitel, S., Kramer, J., and Magee, J. Behaviour model elaboration using partial labelled transition systems. In *European software engineering conference; (ESEC) 11th SIGSOFT symposium on the foundations of software engineering (FSE-11)*, volume 28. 2003.

[176] Vaculín, R. and Sycara, K. Monitoring execution of OWL-S web services. In *European Semantic Web Conference, OWL-S: Experiences and Directions Workshop*. 2007.

[177] van der Aalst, W., Pesic, M., and Schonenberg, H. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113, 2009. 10.1007/s00450-009-0057-9.

[178] van der Aalst, W. M. P. Verification of workflow nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer-Verlag, London, UK, 1997.

[179] van der Aalst, W. M. P. and Basten, T. Inheritance of workflows – an approach to tackling problems related to change. *Theoretical Computer Science*, 270:2002, 2002.

[180] van der Aalst, W. M. P., Basten, T., Verbeek, H. M. W., et al. Adaptive workflow-on the interplay between flexibility and support. In *International Conference on Enterprise Information Systems*, pages 353–360. 1999.

[181] van der Aalst, W. M. P. and Hofstede, T. YAWL: Yet another workflow language. *Information Systems*, 30:245–275, 2002.

[182] van der Aalst, W. M. P. and Jablonski, S. Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276, 2000.

[183] van der Aalst, W. M. P. and ter Hofstede, A. H. M. Workflow patterns: On the expressive power of (petri-net-based) workflow languages. In *Proc. of the Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, August 28-30, 2002 / Kurt Jensen (Ed.)*, pages 1–20. Technical Report DAIMI PB-560, 2002.

[184] van Der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., et al. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[185] van der Aalst, W. M. P., van Hee, K. M., ter Hofstede, A. H. M., et al. Soundness of workflow nets: Classiffication, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011. Accepted for publication.

[186] van der Aalst, W. M. P., Weske, M., and Grünbauer, D. Case handling: a new paradigm for business process support. *Data Knowledge Engineering*, 53(2):129–162, 2005.

[187] Verbeek, H. M. W. and van der Aalst, W. M. P. Analyzing bpel processes using petri nets. In *PNCWB '05: Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.

[188] Vieira, T. A. and Casanova, M. A. Flexible Workflow Execution through an Ontology-based Approach. *Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada*, pages 24–28, 2004.

[189] Vossen, G. and Becker, J., editors. *Geschäftsprozessmodellierung und Workflow-Management*. Thomson Publishing, 1996.

[190] W3C. SOAP Version 1.2 Part 0: Primer (Second Edition). online, 2007. W3C Recommendation.

[191] Weber, B., Reichert, M., and Rinderle-Ma, S. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66:438–466, 2008.

[192] Weikum, G. and Vossen, G. *Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 2001.

[193] Wermelinger, M. *Specification of Software Architecture Reconfiguration*. Ph.D. thesis, Universidade Nova de Lisboa, 1999.

[194] Wong, R. K. Heterogeneous and Multifaceted Multimedia Objects in DOOR/MM: A Role-Based Approach with Views. *Journal of Parallel and Distributed Computing*, 56:251–271, 1999.

[195] Wong, R. K., Chau, H. L., and Lochovsky, F. H. Door: A dynamic object-oriented data model with roles. Technical Report 12, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, 1996.

[196] Woolridge, M. *Reasoning about Rational Agents*. MIT Press, 2000.

[197] Workflow Management Coalition Specification. *Workflow Management Coalition Terminology & Glossary (Document No. WFMC-TC-1011)*. Workflow Management Coalition Specification, 1999.

[198] zur Muehlen, M. Integrating business processes and business rule. In *Stevens Institue of Technology Management*. Veröffentlichter Vortrag; siehe http://www.slideshare.net/mzurmuehlen/integrating-business-rules-and-business-processes, 2007.

Eidesstattliche Erklärung

Ich erkläre, dass

- ich an keinem zurückliegenden erfolglosen Promotionsverfahren teilgenommen habe;
- ein an die Fakultät Informatik der Technischen Universität Dresden zu übersendendes Führungszeugnis gemäß §30 Abs. 5 Bundeszentralregistergesetz bei der zuständigen Meldebehörde beantragt wurde;
- ich die vorliegende Arbeit ohne unzulässige Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind in der Arbeit als solche kenntlich gemacht;
- die vorliegende Arbeit weder im Inland noch im Ausland in gleicher oder in ähnlicher Form einer anderen Prüfungsbehörde zum Zwecke einer Promotion oder eines anderen Prüfungsverfahrens vorgelegt und auch noch nicht veröffentlicht wurde;
- das Manuskript wurde eigenständig erstellt; bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich die Unterstützungsleistungen von Prof. Dr. rer. nat. habil. Uwe Aßmann erhalten;
- es waren keine weiteren Personen bei der geistigen Herstellung der vorliegenden Arbeit beteiligt, insbesondere auch nicht die Hilfe eines Promotionsberaters in Anspruch genommen wurde. Es erfolgten keine weder unmittelbar noch mittelbar geldwerten Leistungen im Zusammenhang mit dem Inhalt der vorliegenden Dissertation.

Dresden, 30. September 2011 _____

Unterschrift