Master-Thesis

Architekturentwicklung eines erweiterbaren Expertensystems zur Anwenderunterstützung bei flugphysikalischen Simulationen

Carsten Bochner
Fachhochschule Bonn-Rhein-Sieg

DLR

Architekturentwicklung eines erweiterbaren Expertensystems zur Anwenderunterstützung bei flugphysikalischen Simulationen

Master-Thesis

zur Erlangung des Grades Master of Science in Computer Science

Hochschule Bonn-Rhein-Sieg
University of Applied Sciences
Fachbereich Informatik

Vorgelegt von:

Carsten Bochner
Vondelstr. 56
50677 Köln
Tel.: 0221 - 2784747
E-Mail: carsten@bochner.de

Erstgutachter und Betreuer	:	Prof. Dr. P. Becker Lehrgebiet Wissens- und Informationsmanagement
Zweitgutachter	:	Prof. Dr. S. Bürsner Lehrgebiet Software-Technologien
Betreuer vor Ort	:	Dipl.-Math. J. Rühmkorf Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Köln

Köln, im Februar 2009

You build on failure. You use it as a stepping stone; close the door on the past. You don't try to forget the mistakes, but you don't dwell on it. You don't let it have any of your energy, or any of your time, or any of your space. If you analyse it as you're moving forward, you'll never fall in the same trap twice ...

- Johnny Cash

Danksagung

An dieser Stelle möchte ich jenen danken, die durch ihre Unterstützung zu dieser Master Thesis beigetragen haben.

Allen voran möchte ich meinen Eltern, Brigitte und Dieter Bochner, für ihre jahrelange Unterstützung und das entgegengebrachte Vertrauen, danken. Ohne sie wäre mir dieses Studium nicht möglich gewesen. Großer Dank gilt auch meiner wundervollen Freundin Birgit Horn, die mich fortwährend psychologisch unterstützte und das Korrekturlesen übernahm.

Besonderer Dank gilt den Betreuern und Gutachtern dieser Arbeit. Herr Prof. Becker hat mir in regelmäßigen Besprechungsterminen wichtige Hinweise geliefert und dadurch sehr geholfen. Frau Prof. Bürsner erklärte sich bereit, die Zeit für die Rolle des Zweitgutachters aufzubringen. Jens Rühmkorf, mein Betreuer beim DLR, verdanke ich aus vielen Gesprächen wichtige Anregungen.

Mein abschließender Dank gilt meinen Vorgesetzten und Mitarbeitern in der Einrichtung „Simulations- und Softwaretechnik“ des DLR. Die dortigen Bedingungen für die Erstellung dieser Arbeit waren hervorragend. Besonders die Möglichkeit zur Teilnahme an der October Rule Fest Konferenz in Dallas, TX möchte ich hervorheben.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Heutzutage ist die Entwicklung von Luft- und Raumfahrzeugen ein komplexer und standardisierter Prozess, der verschiedene Disziplinen der Wissenschaft und des Ingenieurwesens vereint. Die Kenntnis flugphysikalischer Eigenschaften, insbesondere Aerodynamik und Strömung, ist für den Entwurf von Luft- und Raumfahrzeugen unerlässlich. Um den Aufwand zur Berechnung dieser Eigenschaften zu verringern, wurden Methoden und Werkzeuge zur computergestützten Simulation entworfen. Diese werden in integrierten simulationsbasierten Entwicklungsprozessen zusammengefasst. Dadurch ist es beispielsweise möglich, Zeitersparnisse von bis zu mehreren Jahren, gegenüber physikalischen Tests in Windkanälen, zu erzielen [Bec08].

Bei der simulationsbasierten Entwicklung werden verschiedene Softwarewerkzeuge zu einer Prozesskette verknüpft. Im Deutschen Zentrum für Luft- und Raumfahrt (DLR) werden beispielsweise, zur numerischen Berechnung von Strömungen, die folgenden Schritte angewendet. Zunächst wird ein Modell des Luft- oder Raumfahrzeugs mit einer CAD-Software konstruiert. Die CAD-Ausgabedaten können in verschiedenen Formaten vorliegen und müssen vor der weiteren Verwendung durch ein Werkzeug aufbereitet werden. Anschließend wird auf Basis dieser Daten mit einem Netzgenerator ein numerisches Netz erzeugt. Für die numerische Berechnung von Strömungen wird im DLR das CFD-Programm (CFD = Computational Fluid Dynamics) TAU [SGH06] verwendet. Vor dem Einsatz muss TAU konfiguriert werden, wobei die Parameter abhängig von den Eingabedatenformaten sind, der Rechenarchitektur und den physikalischen Rahmenbedingungen der Simulation. Nach der Konfiguration erfolgt abschließend die Berechnung der Strömung mit TAU.

Dieser vereinfachte Prozess zeigt, dass der Anwender sowohl physikalisches Wissen benötigt, als auch Kenntnisse über den Ablaufprozess (Workflow), die Bedienung und die Konfiguration der Werkzeuge. Ein erfahrener Anwender dieser Prozesskette besitzt daher Fachwissen und besondere Fähigkeiten, die den meisten Personen nicht zur Verfügung stehen. Ein solcher Anwender wird als Experte definiert und sein Wissen wird als Expertenwissen bezeichnet [GR04].

Das Expertenwissen kann in so genannten Expertensystemen strukturiert abgebildet und wiedergeben werden. Dies bietet unter anderem den Vorteil, dass das Experten-

wissen einem unerfahrenen Anwender, etwa zu Schulungszwecken, zur Verfügung gestellt werden kann, ohne den menschlichen Experten zeitlich zu belasten [GR04]. Eine zeitliche Beanspruchung des Experten entsteht nur bei der Definition beziehungsweise Eingabe und Pflege von Wissen. Es ist zu beachten, dass sich die Methoden und Best-Practices der Aerodynamik und Strömungstechnik kontinuierlich weiterentwickeln. In dieser Domäne muss die Wissensbasis eines Expertensystems daher effektiv erweitert und gepflegt werden können, so dass der hiermit verbundene Zeitaufwand für alle Beteiligten gering ist. Die Repräsentation von Wissen, anhand von Regeln, ist oftmals hilfreich, da diese dem Denkmuster von Menschen entsprechen und daher leicht zu definieren sind [Jac99].

Eine Evaluation im DLR zeigte, dass die, auf dem Rete-Algorithmus [For79] basierende Regelmaschine JBoss Drools [JBo] Funktionen zur Verfügung stellt, die eine effiziente Unterstützung bei der Wissenspflege erwarten lassen. JBoos Drools kann über ein Plugin in die Entwicklungsumgebung Eclipse [Eclb] integriert werden. In vielen Bereichen des DLR ist Eclipse bereits bekannt und verbreitet.

1.2 Zielsetzung

Aufgabe dieser Arbeit ist der Entwurf einer erweiterbaren Systemarchitektur für ein Expertensystem auf Basis von JBoss Drools und der Eclipse Rich-Client-Platform. Dieses Expertensystem soll an die Anforderungen des simulationsbasierten Entwicklungsprozesses von Luft- und Raumfahrzeugen des DLR angepasst sein und das Wissen dieser Domäne strukturiert festhalten. Eine der Hauptaufgaben dieses Systems ist die Schulung unerfahrener Anwender von computergestützten Simulationswerkzeugen. Das Expertensystem muss daher in der Lage sein, diesen Personen das festgehalten Expertenwissen vollständig, verständlich und in einer geeigneten Form zur Verfügung zu stellen. Das System soll die einfache Erweiterung und Pflege von Wissen erlauben, so dass Wissen auch von Personen eingepflegt werden kann, die nur grundlegende Programmierkenntnisse besitzen. Um einen langfristigen Einsatz des Expertensystems zu ermöglichen, soll die Architektur in der Form aufgebaut sein, dass diese eine zukünftige Abbildung von Wissen aus anderen Domänen mit hoher Wahrscheinlichkeit ermöglicht.

Das Expertensystem stellt den Kern eines größeren Gesamtsystems dar, das in einem separaten Projekt entwickelt wird. Dieses Gesamtsystem umfasst ein Managementsystem für statische Wissensdaten (beispielsweise Anleitungen oder wissenschaftliche Veröffentlichungen), eine Suchmaschine für diese Daten, sowie eine Rechteverwaltung. Die zu entwickelnde Expertensystemarchitektur soll in dieses Gesamtsystem eingebunden werden. Die vorliegende Arbeit grenzt sich von der konzeptionellen und praktischen Entwicklung der anderen Komponenten des Gesamtsystems ab, da diese Aufgaben im Folgenden nicht betrachtet werden.

Im Rahmen dieser Arbeit sollen die wesentlichen Kernkomponenten der entworfenen Architektur prototypisch implementiert und mit beispielhaften Wissen befüllt werden, so dass eine abschließende Evaluation des entwickelten Konzepts möglich ist.

1.3 Vorgehensweise und Struktur

Die methodische Vorgehensweise setzt sich aus vier Schritten zusammen: Anforderungsermittlung, Konzeption, prototypische Implementierung und Evaluation. Diese Schritte werden iterativ durch die, im Rahmen der Erstellung dieser Arbeit, ermittelten Erkenntnisse und Erfahrungen verfeinert. Die vorliegende Arbeit stellt die gewonnenen Erkenntnisse dar und dokumentiert das methodische Vorgehen. Dabei besitzt diese Ausarbeitung die folgende Struktur.

Das an diesen Abschnitt anschließende Kapitel 2 beschäftigt sich mit den theoretischen Grundlagen regelbasierter Expertensysteme. Dort wird zunächst eine allgemeine Einordnung und Charakterisierung von Expertensystemen geliefert. Im Folgenden werden die Expertensysteme betracht, deren Wissensabbildung auf Regeln basiert. Im Rahmen dessen wird der Rete-Algorithmus, ein weit verbreiteter Ansatz zur Regelnauswertung, vorgestellt. Abschließend erfolgt die Betrachtung der theoretischen Architektur, sowie der Vor- und Nachteile von regelbasierten Expertensystemen.

Die wichtigsten, in dieser Arbeit verwendeten Technologien, sind die Eclipse Rich-Client Platform und JBoss Drools, deren Eigenschaften und Fähigkeiten in Kapitel 3 vorgestellt werden.

In Kapitel 4 erfolgt die Ermittlung der Anforderungen an die zu entwerfende Systemarchitektur. Das Ergebnis ist eine Anforderungs- und Problemdefinition, sowie die Kenntnis über die Struktur des Expertenwissens und der Einsatzdomäne. Es werden Anwendungsfälle und Benutzertypen definiert.

Die theoretische Konzeption der Expertensystemarchitektur erfolgt in Kapitel 5. Es wird zunächst ein Konzept entwickelt, das die strukturierte Abbildung und Wiedergabe des Expertenwissens erlaubt. Dem Anwender werden anhand der Architektur tiefergehende Erklärungen über Zusammenhänge und Entscheidungen angeboten. Anschließend wird ein Konzept für die Abläufe der Wissenswiedergabe erstellt, das die gezielte Ermittlung von Benutzereingaben und die Bereitstellung von Informationen berücksichtigt.

Die wesentlichen Aspekte der technischen Realisierung werden in Kapitel 6 vorgestellt. Es wird ein Überblick über die entwickelte Softwarearchitektur und die Komponenten gegeben, sowie die umgesetzte Lösung von technischen Herausforderungen dargelegt. Darüber hinaus werden die entdeckten Probleme aufgezeigt und Erweiterungsmöglichkeiten beleuchtet.

Abschließend wird die entworfene Architektur und der darauf basierende Prototyp anhand von Beispielen evaluiert und ein Ausblick auf mögliche Weiterentwicklungen gegeben.

Die Arbeit wurde in der Einrichtung „Simulations- und Softwaretechnik“ des Deutschen Zentrums für Luft- und Raumfahrt verfasst.

2 Regelbasierte Expertensysteme

In diesem Kapitel werden die grundlegenden Kenntnisse zu regelbasierten Expertensystemen vermittelt. Es werden zunächst die allgemeinen Eigenschaften von Expertensystemen charakterisiert, ohne Beachtung der verwendeten Wissensrepräsentation. Dem folgt eine Vorstellung der für regelbasierte Expertensysteme relevanten Grundlagen und Algorithmen. Abschließend wird ein Überblick geboten über die wesentlichen Architektureigenschaften regelbasierter Expertensysteme und deren Vor- und Nachteile.

2.1 Einordnung und Charakterisierung von Expertensystemen

2.1.1 Einführung in Expertensysteme

Das Forschungsgebiet der Künstlichen Intelligenz (KI) entstand Mitte des 20. Jahrhunderts und befasst sich mit der Erstellung von Intelligenz auf Basis von Maschinen. Eine der populärsten Definitionen der KI ist „Computer dazu zu bringen, wie Menschen zu denken“ [GR04]. Einen umfassenden Einstieg in das Gesamtgebiet KI bieten insbesondere [RN04], [Lug01] und [GRS03].

Expertensysteme sind ein spezielles Teilgebiet der Künstlichen Intelligenz, welches sich mit der Lösung von komplexen Problemen in einem begrenzten Einsatzbereich beschäftigt [GR04]. Ein solches klassisches Einsatzgebiet ist beispielsweise die medizinische Diagnose für konkrete Krankheitsbilder, bei der durch die Eingabe patientenbezogener Daten in das System ein Befund erstellt wird.

Expertensysteme sind typischerweise stärker praktisch ausgerichtet als andere Forschungsfelder der KI. Im Gegensatz zu frühen Ansätzen der Künstlichen Intelligenz, die versuchten Strategien für allgemeine, unspezifische Probleme anzubieten (so genannte General Problem Solver [NS63]), vereinfachen Expertensysteme diese Aufgabe durch eine fachliche Spezialisierung. Es gibt bei diesen somit eine Beschränkung auf einen konkreten, wohldefinierten Problemraum. Ein Problemraum ist als wohldefinert zu bezeichnen, wenn ein menschlicher Experte in der Lage ist, die Schritte und Schlussfolgerungen zu spezifizieren, welche zur Lösung des Problems führen können [GR04]. Die Lösungsstrategie wird aus den Erfahrungen und dem Wissen eines menschlichen Experten vorgegeben und die Aufgabe des Expertensystems ist es, diese nachzubilden und dadurch das Handeln des Experten zu simulieren.

Expertenwissen kann als eine Kombination aus dem theoretischen Verständnis des Problems und effizienten heuristischen Problemlösungsregeln definiert werden [Lug01]. Um dieses Expertenwissen zu erfassen und für die Systementwicklung bereitzustellen, werden häufig Fachexperten an den Entwicklungsprozess beteiligt. Diese erörtern die allgemeine Problemlösungsmethodik und demonstrieren diese anhand von Beispielproblemen. Systementwickler oder spezialisierte Wissensingenieure erfassen, kodieren und implementieren dieses Expertenwissen in das Expertensystem.

Ein fertiggestelltes Expertensystem verfügt über die Fachkenntnisse eines Spezialisten und ist anhand dieser Kompetenzen in der Lage, spezifische Probleme zu lösen oder Ratschläge zu erteilen. Hierbei können einige Expertensysteme den menschlichen Experten vollständig ersetzten und seine Aufgaben selbstständig ausführen, beispielsweise bestimmte Entscheidungen eigenständig treffen. Andere Expertensysteme wiederum dienen nur der Unterstützung von Experten und geben diesen Ratschläge oder Hinweise. Der endgültige Entscheidungsträger bleibt in diesem Fall jedoch der Experte selbst, seine Produktivität und die Qualität seiner Entscheidungen kann durch das Expertensystem aber gesteigert werden.

Da Expertensysteme Lösungen auf Grundlage ihrer Wissensbasis erarbeiten, wird für diese oft das Synonym wissensbasierte Systeme verwendet. Expertensysteme werden in der Literatur auch als ein Unterbereich der wissensbasierten Systeme betrachtet.

2.1.2 Charakterisierung von Expertensystemen

In diesem Abschnitt wird eine allgemeine Charakterisierung von Expertensystemen geboten. Diese Charakterisierung erfolgt anhand einer Abgrenzung von Expertensystemen gegenüber konventionellen Anwendungsprogrammen und orientiert sich an der entsprechenden Ausführung von [Jac99].

Expertensysteme unterscheiden sich insbesondere dadurch von konventionellen Anwendungen, indem sie das formalisierte Schließen zur Lösung von Problemen eines Fachgebiets simulieren, nicht jedoch das Fachgebiet selbst. Ein Expertensystem versucht somit in erster Linie die Problemlösungsstrategien eines menschlichen Experten nachzubilden, während konventionelle Anwendungen den Problemraum, etwa durch mathematische Modelle, nachbilden.

Des Weiteren grenzen sich Expertensysteme dadurch ab, dass sie typischerweise wissensbasiert sind und menschliches Wissen in Sprachen repräsentieren, welche speziell hierzu entwickelt wurden. Dieses Wissen wird darüber hinaus von den Systemmodulen getrennt festgehalten, welche die logischen Schlüsse auf Basis dieses Wissens ziehen. Die Architektur eines solchen Systems unterscheidet sich somit von konventionellen Anwendungsprogrammen. Ein typisches Expertensystem ist charakterisiert durch die beiden folgenden lose gekoppelten Komponenten [FS90]: Eine Komponenten dient der

Repräsentation des vorhandenen Wissens und wird als Wissensbasis bezeichnet. Die andere charakteristische Komponente wird als Inferenzmaschine bezeichnet und stellt die zentrale Problemlösungskomponente dar Die Inferenzmaschine operiert auf Grundlage der Wissensbasis und leitet durch logische Schlussfolgerungen neue Aussagen ab.

Ein weiterer charakteristischer Unterschied zu konventioneller Software ist, dass Expertensysteme Probleme oftmals durch heuristische oder approximative Methoden lösen, die auf Erfahrungswerten und Faustregeln basieren. Im Gegensatz zu den algorithmischen Lösungen konventioneller Programme, ist die Korrektheit einer Lösung bei solchen Expertensystemen nicht garantiert.

2.2 Grundlagen regelbasierter Expertensysteme

Die verwendeten Modelle zur Repräsentation von Wissen und zum Erlangen von Schlussfolgerungen können in einem Expertensystem unterschiedlichen Ansätzen folgen. Die drei Hauptgruppen sind die folgenden [Wol03]:

- **Fallbasierte Expertensysteme:** Das Fachwissen besteht aus bereits bekannten Fällen und Beispielen. Es wird versucht das Problem zu lösen, indem die Lösung eines möglichst ähnlichen Falles adaptiert wird.
- **Modellbasierte Expertensysteme:** Diese werden häufig bei der Fehlerdiagnose technischer Systeme eingesetzt. Hierbei besteht das Fachwissen aus dem Modell des zu diagnostizierenden Systems und die Problemlösung wird aus diesem Modell abgeleitet.
- **Regelbasierte Expertensysteme:** Das Wissen ist in Form von Regeln abgelegt, welche allgemein formuliert und grundsätzlich gültig sind. Die Lösung wird aus den Regeln und fallspezifischen Fakten abgeleitet.

Im weiteren Verlauf dieser Arbeit werden wir uns ausschließlich mit regelbasierten Expertensystemen beschäftigen. Daher werden in den folgenden Abschnitten die Grundlagen und Konzepte dieser Systeme vorgestellt.

2.2.1 Regeln und deren Eigenschaften

Die auf Regeln basierenden Expertensysteme sind die heute am weitesten verbreiteten [GR04]. Bei ihnen wird das Fachgebietswissen in Form von Regelausdrücken repräsentiert. Regeln sind formalisierte Konditionalsätze die sich aus einem Prämissen-Teil (*Wenn*-Abschnitt) und der Konklusion (*Dann*-Abschnitt) zusammensetzen und die folgende einfache Grundform aufweisen:
Wenn *A*
Dann *B*

Dieser Fall stellt eine einfache Implikation dar, wie man sie aus der Mathematik kennt. Ein einfaches Beispiel ist:
Wenn *es regnet*
Dann *ist die Erde nass*
Eine Regel kann angewendet werden, wenn ihr Prämissen-Teil erfüllt ist. Dies wird im Allgemeinen als das Feuern einer Regel bezeichnet.
Im Kontext von Expertensystemen werden häufig Regeln eingesetzt, deren Konklusion mit einer expliziten Aktion verknüpft ist. Diese Regeln werden als Produktionsregeln bezeichnet.

Im Zusammenhang mit regelbasierten Expertensystemen wird der Prämissen-Teil der Regel oftmals als LHS (left hand side) bezeichnet und die Konklusion als RHS (right hand side).

Der Prämissen-Teil kann sich aus einzelnen Prämissenelementen (Bedingungen) zusammensetzen, die über Junktoren (logische Verknüpfungsoperatoren) verbunden sind. Diese Prämissenelemente basieren auf den Attributen der Regel. Ob die gesamte Regel erfüllbar ist, wird letztlich durch die logische Verknüpfung aller Prämissenelemente ermittelt. Im folgenden Beispiel besteht der Prämissen-Teil der Produktionsregel aus zwei Prämissenelementen die über den Junktor der UND-Verknüpfung verbunden sind:
Wenn *Flugzeug im Landeanflug UND das Fahrwerk eingefahren ist*
Dann *schalte Warnsignal ein*

Regeln bilden zusammen mit den Attributen ihrer Aussagen den abstrakten Teil der Wissensbasis eines Expertensystems. Regeln definieren hierbei die Zusammenhänge zwischen diesen Attributen, im letzen Beispiel also die Zusammenhänge zwischen den Attributen Landeanflug, Fahrwerk und Warnsignal. Die von den Regeln getroffenen Definitionen sind allgemein gültige Strategien und Vorgaben, erlauben dem System aber noch nicht neue Schlüsse zu ziehen. Erst bei der Anwendung des Expertensystems auf einen speziellen Fall wird konkretes fallspezifisches Wissen in die Wissensbasis des Expertensystems aufgenommen und die Attribute werden mit konkreten Werten befüllt. Diese konkreten Werte werden als Fakten bezeichnet. Im vorherigen Beispiel etwa die Attribute und Fakten „*Landeanflung is true*“ und „*Fahrwerk eingefahren is true*“. Anhand dieser Fakten, die sich für jeden einzelnen Anwendungsfall unterscheiden können, ist das System in der Lage die Regeln abzuarbeiten und auf neues Wissen zu schließen. Dieses Vorgehen wird als datengetriebene oder vorwärtsgerichtete Inferenz (Vorwärtsverkettung) bezeichnet. Aus erfüllten Prämissen-Teilen wird auf die Wahrheit der Konklusion geschlossen und hierdurch neues Wissen erzeugt. Dieses neue Wissen wird selbst zum Fakt und geht in den Inferenzprozess ein. Dieser Prozess ist beendet, wenn alle Fakten abgearbeitet sind.

Dem entgegen steht die zielorientierte Inferenz (Rückwärtsverkettung), bei der nicht

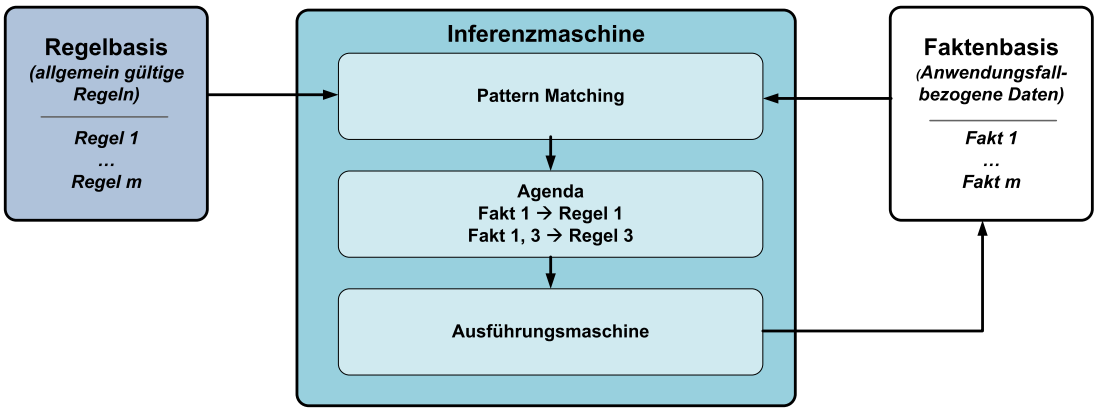

Abbildung 2.1: Allgemeine Architektur eines Regelsystems

die Fakten als Ausgangsbasis verwendet werden, sondern ein Zielelement der Konklusion, dessen Zustand ermittelt werden soll. Ausgehend von diesem Zielelement werden die Regeln ausgewählt und miteinander verkettet, die es erlauben den gesuchten Zustand zu ermitteln. Der Inferenzprozess ist erfolgreich beendet, wenn der Zustand des Zielelements, unter Berücksichtigung der fallbezogenen Fakten, hergeleitet werden konnte.

Es gibt regelbasierte Expertensysteme, die nur die vorwärtsgerichtete Inferenz verwenden und Systeme, die nur die rückwertsgerichtete Inferenz verwenden. Die meisten aktuellen regelbasierten Expertensysteme verwenden die Vorwärtsverkettung. Einige unterstützen beide Methoden und überlassen somit dem Systemingenieur die Auswahl. Die klassische Logik bildet das Fundament der Regelsysteme. Weitergehend sei hierzu auf [Lig06] und [Sch05] verwiesen.

2.2.2 Regelsystemarchitektur und Pattern Matching

Systeme, die auf Basis von Regeln und Fakten operieren und hieraus Schlüsse ableiten werden als Regelsysteme bezeichnet. Diese bestehen aus drei Hauptkomponenten:

- **Regelbasis:** Die Regeln, also das abstrakte Wissen des Systems, werden in der Regelbasis (englisch: Production Memory) abgelegt.
- **Faktenbasis:** Das fallspezifische Wissen, also die aktuellen Fakten, befinden sich in der Faktenbasis (englisch: Working Memory).
- **Inferenzmaschine:** Diese verknüpft die Faktenbasis mit der Regelbasis und ermittelt die zu aktivierenden Regeln. Diese werden auf eine so genannte Agenda-Liste gesetzt und an die Ausführungskomponente weitergereicht.

Abbildung 2.1 veranschaulicht den Zusammenhang zwischen diesen Komponenten.

Die Regelbasis und die Faktenbasis werden zusammengefasst als die Wissensbasis des Systems bezeichnet. Um neue Schlüsse zu ziehen und die zu aktivierenden Regeln zu

ermitteln, arbeitet die Inferenzmaschine mit beiden Wissensbereichen und verknüpft die Elemente der Faktenbasis mit den Regelprämissen. Dieser Prozess wird in der Literatur als Pattern Matching definiert.

Der naive Ansatz des Pattern Matching funktioniert wie folgt: Es werden alle Regeln der Regelbasis einzeln betrachtet. Dabei wird jede Bedingung der Regel einzeln daraufhin geprüft, ob Fakten in der Faktenbasis existieren, die diese Bedingung erfüllen. Wenn alle Bedingungen einer Regel erfüllt sind, wird diese aktiviert und auf die Agenda-Liste gesetzt.
Dieser Ansatz ist sehr ineffektiv, da für jede existierende Bedingung gegebenenfalls die komplette Faktenbasis durchsucht werden muss, um die Erfüllbarkeit zu überprüfen. Darüber hinaus muss bei jeder Änderung an der Faktenbasis, das komplette Pattern Matching für alle Bedingungen und Regeln erneut gestartet werden, da die Ergebnisse des vorherigen Durchlaufs nicht festgehalten wurden und daher nicht nachvollziehbar ist, wie sich diese Änderung auswirkt.

Die angenäherte Komplexität dieses Ansatzes für einen Zyklus liegt laut [FH03] im schlechtesten Fall bei $\mathcal{O}(RF^P)$, wobei R die Anzahl der Regeln, F die Menge aller Fakten und P die durchschnittliche Anzahl von Prämissen pro Regel ist.

2.2.3 Konflikt-Lösung

Für den Fall, dass eine Menge von Regeln existiert, die aktiviert sind und ausgeführt werden sollen, so muss die Reihenfolge bestimmt werden, in der diese Ausführung erfolgt. Da die zu aktivierenden Regeln, in Hinblick auf den Ausführungszeitpunkt, untereinander im Konflikt stehen, wird die Menge dieser Regeln als Konfliktset bezeichnet. Regelsysteme wählen anhand einer Strategie zur Konflikt-Lösung jeweils eine einzelne Regel aus und entfernen diese dann aus dem Konfliktset. Die verwendeten Strategien variieren zwischen den Regelsystemen und sind darüber hinaus abhängig vom Anwendungsfall. Häufig werden in der Praxis verschiedene Ansätze miteinander kombiniert. Die drei meist verbreiteten Strategien sind laut [Jac99] die folgenden:

- **Neuheit:** Diese Strategie bewertet Regeln anhand der Aktualität ihrer Faktenbasis und bevorzugt Regeln, die auf den zuletzt hinzugefügten Fakten beruhen.
- **Spezifität:** Hierbei erhalten Regeln, die spezifischer sind, eine höhere Ausführungspriorität. Regeln die viele Bedingungen besitzen und daher schwerer zu erfüllen sind, werden bei dieser Strategie zu Beginn ausgeführt.
- **Feuerbeständigkeit:** Diese Strategie verhindert, dass eine Regel auf Grund derselben, unveränderten Fakten mehrfach feuert. Dies wird erreicht, indem eine gefeuerte Regel aus dem Konfliktset entfernt wird,

Darüber hinaus wird in vielen Systemen dem Anwender die Möglichkeit geboten, die Priorität der Regeln selbst zu definieren. Diese Funktion wird als Salience bezeichnet.

Das Festlegen dieser Priorität erfolgt gleichzeitig mit der Definition der Regel.

2.3 Rete-Algorithmus

2.3.1 Konzepte und Aufbau

Ende der 1970er Jahre entwickelte Charles L. Forgy an der Carnegie-Mellon University den Rete-Algorithmus [For79] [For82], welcher effizientes vorwärtsgerichtetes Pattern-Matching für regelbasierte Expertensysteme bietet und eine weite Verbreitung in Expertensystemen erlangt hat. Im Laufe der Zeit haben sich verschiedene Varianten des Rete-Algorithmus herausgebildet. Diese verfolgen unterschiedliche Ziele, beispielsweise die Integration in moderne objektorientierte Programmiersprachen. Im weiteren Verlauf dieses Abschnitts wird nur der klassische Rete-Algorithmus vorgestellt und an diesem die grundlegenden Konzepte verdeutlicht. Der ursprüngliche Rete-Algorithmus definiert nur den Algorithmus, um die zu aktivierenden Regeln zu ermitteln, er macht jedoch keine Angaben über Strategien zur anschließenden Konflikt-Lösung und überlässt dies den konkreten Implementierungen.

Der Grundansatz zur Leistungssteigerung beim Pattern Matching basiert auf zwei empirisch gewonnenen Erkenntnissen Forgys [FH03]. Die erste Erkenntnis ist die Tatsache, dass die Regelbasis typischer Expertensysteme über eine längere Folge von Auswertungszyklen stabil bleibt und im Allgemeinen auch an den Fakten des Working Memory nur wenige Änderungen vorgenommen werden. Forgy nennt dies die Eigenschaft der temporären Redundanz [For79]. Der naive Pattern-Matching Algorithmus ignoriert diese Eigenschaft der Expertensysteme und berechnet in jedem Zyklus die Muster aller Fakten und Regeln neu. Er betrachtet somit auch Kombinationen, die bereits vorher untersucht wurden. Im Gegensatz hierzu speichert der Rete-Algorithmus die bereits erzeugten Zwischenergebnisse für Faktenobjekte ab und verwendet diese, solange es keine Änderungen gibt, in späteren Zyklen erneut. Es müssen in jedem Zyklus nur noch die geänderten Faktenobjekte überprüft werden, wodurch eine erhebliche Verringerung der Laufzeit möglich wird.

Die zweite Erkenntnis in Forgys Arbeit ist, dass die Prämissen der Regeln oftmals eine hohe Ähnlichkeit aufweisen. Dies bezeichnet er als Eigenschaft der strukturellen Ähnlichkeit [For79]. Diese Tatsache nutzt der Rete-Algorithmus aus und steigert die Effizienz, indem er die Regelprämissen auf Ähnlichkeiten untersucht. Gefundene Redundanzen werden durch das Zusammenfügen der Regelprämissen beseitigt. Hierdurch kann der Aufwand für die Mustererkennung noch weiter gesenkt werden, da die redundanten Teile der Regeln nur einmal betrachtet werden müssen und nicht, wie im naiven Algorithmus, für jede Regel erneut.

Um diese Ansätze konkret umzusetzen, betreibt der Rete-Algorithmus zunächst ein

Preprocessing, welches die Eigenschaft der strukturellen Ähnlichkeit ausnutzt. Hierbei erzeugt der Rete-Algorithmus aus den Prämissenelementen für jede Regel einen gerichteten Baumgraphen. Dieser stellt ein Entscheidungsnetzwerk dar, dessen Knoten sich in zwei Hauptkategorien aufteilen:

- **Alpha-Knoten:** Diese Knoten repräsentieren eine Selektion auf Basis eines einzelnen Attributes der Faktenbasis, also eines einzelnen Elements einer Regelprämisse ohne Verknüpfung. Ein Alpha-Knoten hat genau einen Vaterknoten.
- **Beta-Knoten:** Hierbei handelt es sich um Verbindungsknoten zwischen Attributen. Beta-Knoten dienen der Repräsentation von Selektionsprüfung, die auf mehreren Attributen der Faktenbasis operieren. Hierzu stellt ein Beta-Knoten eine Relation (Junktor oder Variabelbindung) zwischen den Elementen der Regelprämisse dar. Ein klassischer Beta-Knoten hat zwei Vaterknoten und repräsentiert eine logische UND-Verknüpfung.

Aufgrund ihrer Eigenschaft, nur einen Vaterknoten zu besitzen, werden Alpha-Knoten in der Literatur auch als One-Input Nodes bezeichnet, Beta-Knoten entsprechend als Two-Input Nodes.

Die Knoten werden entsprechend der Aussagelogik der Regeln so lange miteinander verknüpft, bis diese in einem einzigen gemeinsamen Ergebnis enden. Dieses Ergebnis ist der Eingangswert für den untersten Knoten des Rete-Netzes:

- **Aktionsknoten:** Dieser Knoten repräsentiert den Dann-Teil der Regel. Erhält er eine Eingabe (Signal) von seinem Vorgängerknoten, so aktiviert er, ohne weitere Prüfung, die ihm zugeordnete Regel. Dies bedeutet, dass er diese Regel für die Ausführung auf die Agenda-Liste setzt.

Abbildung 2.2 stellt exemplarische Rete-Netze mit ein, zwei und drei Attributen dar.

Die drei genannten Knotentypen sind nur die wesentlichen Basistypen. Bereits der ursprüngliche Rete-Algorithmus kennt in seiner einfachsten Form sechs Knotentypen [For79] und neuere Varianten des Algorithmus führen weitere Knotentypen ein. Die hier vorgestellten Typen sind für das grundlegende Verständnis ausreichend.

Der Rete-Algorithmus speichert zu jedem Knoten die Zwischenergebnisse der Selektionen ab, so dass diese in späteren Zyklen nicht mehr erneut erzeugt werden müssen. Der Rete-Algorithmus verwendet zwei unterschiedliche Speicherbereiche, die traditionell als Alpha-Speicher und Beta-Speicher bezeichnet werden [FH03]. Der Alpha-Speicher speichert zu einem Alpha-Knoten die Fakten ab, die die Selektionsbedingung erfüllt haben, also weiter gereicht wurden. Der Beta-Speicher enthält für jeden Beta-Knoten eine Liste der bisherigen Faktenkombinationen, die die Verbundsbedingungen erfüllen. Nur die in einem Zyklus neu eintreffenden, also veränderten Fakten werden an den einzelnen Knoten untersucht und mit bereits vorhandenen Ergebnissen verknüpft. Bereiche des Baums, die in einem Zyklus von keiner Veränderung betroffen sind, werden

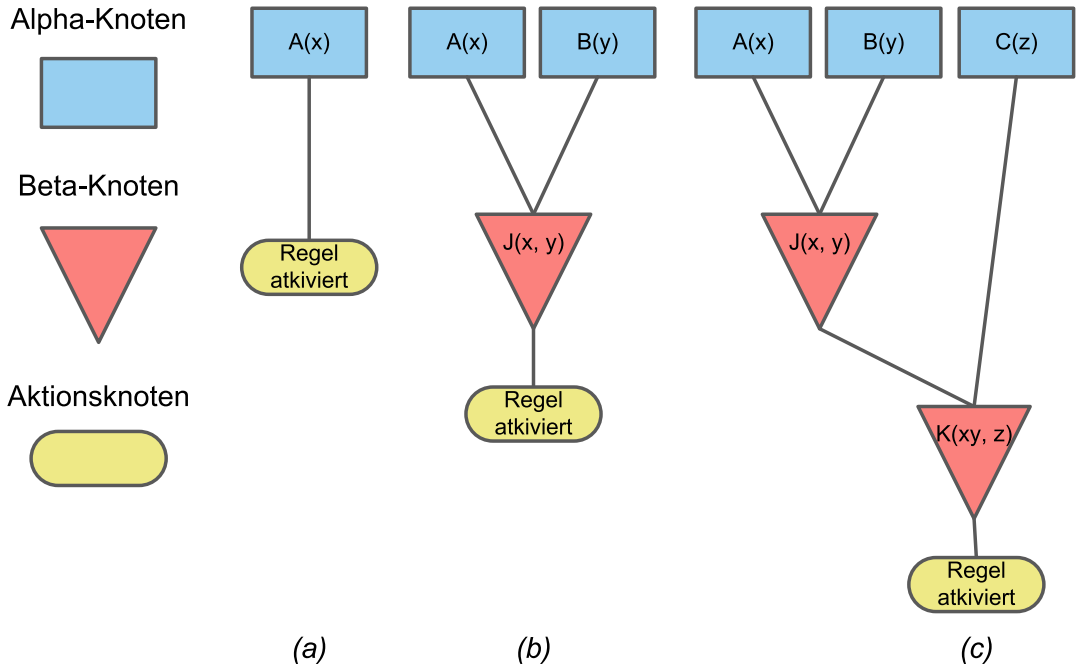

Abbildung 2.2: Beispielhafte Darstellung von Rete-Netzen mit ein, zwei und drei Attributen.

nicht erneut betrachtet. Es werden stattdessen die abgespeicherten Ergebnisse dieser Bereiche verwendet.

2.3.2 Beispielhafte Erzeugung eines Rete-Netzes

Der objektorientierten Nomenklatur folgend, werden in diesem Beispiel Objekte verwendet, die Attribute und Eigenschaften haben und durch diese die fallbasierten Fakten abbilden. Gegeben sei eine Objektklasse für den Faktentyp *Kunde* (mit den Fakten *„suchtFlugreise“* und *„lieblingsziel“*), eine Objektklasse für den Faktentyp *Hotel* (mit den Fakten *„verfügbar“*, *„ort“* und *„nächsterFlughafen“*) und eine Objektklasse für den Faktentyp *Flug* (mit den Fakten *„verfügbar“* und *„zielflughafen“*). Eine Beispielregel um einem Kunden eine Flugreise zusammenzustellen könnte demnach lauten:
Wenn
Kunde.suchtFlugreise == true,
Hotel.verfügbar == true, Hotel.ort == Kunde.lieblingsort,
Flug.verfügbar == true, Flug.zielflughafen == Hotel.nächsterFlughafen
Dann
Empfehle dem Kunden das Hotel und den Flug

Aus diesem Beispiel ergeben sich drei Alpha-Knoten:

1. SELECT: *Kunde.suchtFlugreise == true*

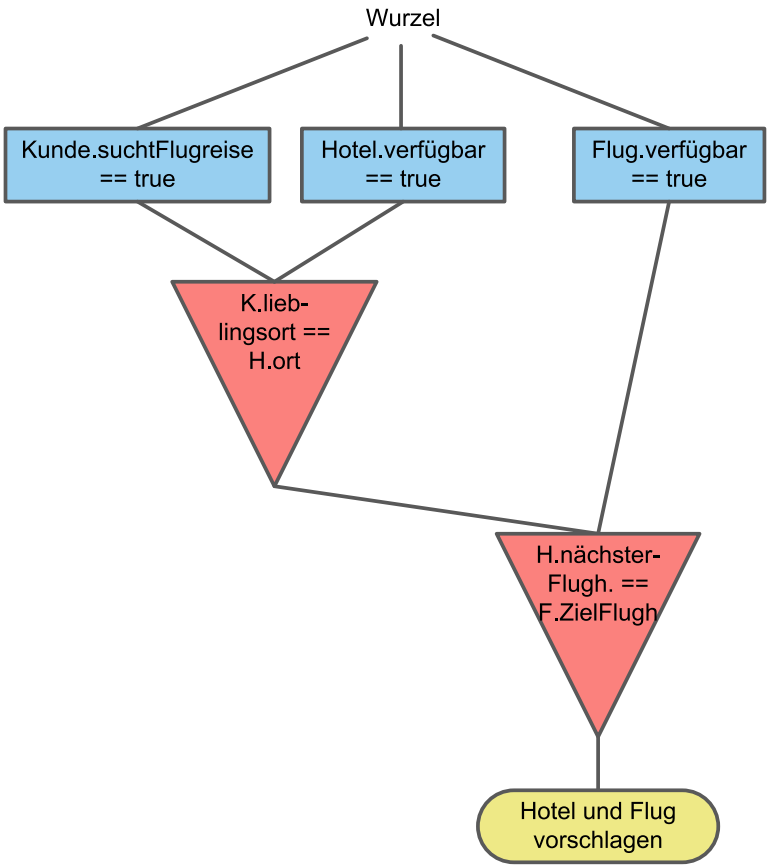

Abbildung 2.3: Rete-Netz zum Flugreise-Beispiel aus Abschnitt 2.3.2

2. SELECT: *Hotel.verfügbar == true*
3. SELECT: *Flug.verfügbar == true*

Es ergeben sich zwei Beta-Knoten:

1. JOIN: *Kunde.Lieblingsziel == Hotel.Ort*
2. JOIN: *Hotel.NächsterFlughafen == Flug.Zielflughafen*

Das hieraus entstehende Rete-Netz ist in Abbildung 2.3 dargestellt. Die Fakten wandern durch dieses Netzwerk von oben nach unten. Von der Wurzel ausgehend werden die neu zu überprüfende Fakten des Working Memory den ihren Typen entsprechenden Alpha-Knoten zugewiesen. Im Beispiel wählt der Faktentyp *Kunde* den linken Weg, der Faktentyp *Flug* den rechten. Somit beschreitet jeder Faktentyp einen anderen Weg durch das Netzwerk. Die neuen Fakten werden an den Alpha-Knoten einer Selektion unterzogen und nur die Fakten, die die Selektionsbedingungen des Knotens erfüllen, werden nach unten an den nächsten Knoten weitergereicht. Das Ergebnis dieser Selektionsprüfung wird im Alpha-Speicher abgelegt. Eine erneute Überprüfung dieser Fakten ist in späteren Zyklen also nicht mehr nötig [Doo95].

Wird ein Fakt von einem Alpha-Knoten an einen Beta-Knoten, der zwei Eingänge besitzt, weitergegeben, so soll dieses Fakt mit anderen Fakten verknüpft werden. Hierzu müssen die Verbundbedingungen überprüft werden, beispielsweise ob zwei Fakten denselben Wert aufweisen. Diese geschieht indem das neu ankommende Fakt mit den gegebenenfalls bereits am zweiten Eingang vorhandenen gültigen Fakten verknüpft wird und diese Kombinationen auf die Erfüllung der Verbundbedingung überprüft wird. Im Beispiel wird etwa der *Lieblingsort* des *Kunden* mit dem *Ort* des *Hotels* abgeglichen. Die Faktenkombinationen, die diese Prüfung überstehen, werden wiederum nach unten an den Folgeknoten weitergegeben. Das Ergebnis dieser Prüfung wird im Beta-Speicher abgelegt und kann im nächsten Zyklus wieder verwendet werden. Dieses Vorgehen wird, falls die entsprechenden erfüllenden Fakten vorhanden sind, bis zum Aktionsknoten fortgesetzt, was ein Feuern der Regeln auslöst. Auf diese Weise werden alle neuen Fakten in das Rete-Netz eingefügt und die Alpha- und Beta-Speicher aufgefüllt. Das Ergebnis dieses Prozesses ist die Ermittlung aller zu aktivierender Regeln.

2.3.3 Änderungen am Rete-Netz

Werden Änderungen am Working Memory vorgenommen, so müssen diese in das bereits erstellte Rete-Netz eingepflegt werden. Hierzu werden die Änderungen durch das Netzwerk propagiert. Ein geändertes Fakt wird also noch einmal durch das Netzwerk geschickt. Es werden die betroffenen Bereiche des Rete-Netzes erneut untersucht und die entsprechenden Speicher aktualisiert. Durch die Änderung von Fakten können Alpha- und Beta-Knoten neu aktiviert werden, was zu einem veränderten Feuern der Regeln führen kann. Auch können bereits aktivierte Regeln wieder zurückgezogen werden.

Wird ein eines zusätzliches Faktenobjekt neu in das Working Memory aufgenommen, dann wird dieser Fakt von der Wurzel aus, entsprechend seinem Faktentyp, nach unten durch das Netzwerk gereicht wird. Wird im Beispiel aus Abschnitt 2.3.2 etwa ein neuer *Flug* in die Faktenbasis aufgenommen, wird dieser im rechten Alpha-Knoten eingefügt. Dort wird zunächst geprüft, ob dieser *Flug* verfügbar ist *(Flug.verfügbar == true)*. Ist dies der Fall, wird dieses Ergebnis in den Alpha-Speicher des Knotens aufgenommen. Anschließend wird das Fakt an den folgenden Beta-Knoten weiter gereicht. Dessen Beta-Speicher verfügt bereits über alle gültigen Kombinationen von *Kunden* und *Hotels*, so dass nun nur diese mit dem neuen *Flug* verknüpft werden müssen. Dies führt gegebenenfalls zu einer Aktivierung der Regel *„Hotel und Flug vorschlagen“*. Es wird deutlich, dass der Aufwand für das Einfügen eines neuen Fakts wesentlich geringer ist, als beim naiven Pattern Matching. Bei diesem hätten alle Fakten erneut untersucht werden müssen.

Das Entfernen von Fakten ähnelt dem Ändern von Fakten. Das zu löschende Fakt wird durch das Netz gereicht und die Speicher in denen es enthalten ist werden bereinigt. Sollte es bis zum Aktionsknoten gelangen, so müssen die Regeln, zu deren Aktivierung

dieser Fakt beigetragen hat wieder von der Agenda-Liste entfernt werden.

2.3.4 Ausnutzung der temporären Redundanz

Das in vorherigen Abschnitten vorgestellte Vorgehen erlaubt die effiziente Ausnutzung der temporären Redundanz-Eigenschaft der Faktenbasis, da einmal erzielte Erkenntnisse für spätere Zyklen festgehalten werden. Es ist jedoch zu beachten, dass bisher für jede Regel ein eigenes Rete-Netz erzeugt wurde. Dies hat zur Folge, dass eine Änderung an einem Fakt in jedes Rete-Netz eingepflegt werden muss, welches dieses Fakt enthält. Auf Grund der von Forgy beschriebenen Eigenschaft der strukturellen Ähnlichkeit von Regelnprämissen, ist in typischen Anwendungsszenarien zu erwarten, dass ein Fakt häufig an mehreren Rete-Netzen beteiligt ist. Daher erkannte Forgy hier einen weiteren Optimierungsansatz und entwickelte die Idee, die Einzelnetze miteinander zu verknüpfen [For79]. Der Ansatz ist, die gleichen Anfangsstücke der Netze in einem neuen Netz miteinander zu verschmelzen. Dies wird in der Literatur häufig als Node Sharing bezeichnet. Der Aufwand für eine doppelte Überprüfung eines Fakts kann vermieden werden. Dies führt zu einer Reduzierung der Zahl der zu überprüfenden Fakten. Laut Untersuchungen [Doo95] steigt der Faktor dieser Reduzierung, in typisch strukturierten Wissensbasen, linear mit der Anzahl der Regeln. Dies ermöglicht eine deutliche Zeitersparnis zur Ausführungszeit. Darüber hinaus wird die Größe des nötigen physikalischen Speichers reduziert, da das Verknüpfen der Netze zu einer geringeren Netzanzahl führt [Gup86].

Das folgende Beispiel verdeutlicht, ausgehend von dem in Abbildung 2.4(a) dargestellten Netz, das Vorgehen beim Node Sharing. Hierbei kann man zwei Stufen unterscheiden [FH03]. In der ersten werden zunächst nur die Alpha-Knoten der beiden Netze miteinander verknüpft. Es ist in Abbildung 2.4(b) erkennbar, dass die Alpha-Knoten, welche in den beiden Netzen die gleiche Funktion für das gleiche Fakt repräsentierten, zusammengelegt wurden. Hierdurch sind die beiden Netze miteinander verknüpft.

In dem neu entstandenen Netz ist jedoch weiterhin Redundanz enthalten, da es zwei Beta-Knoten gibt, die genau die gleiche Selektionsprüfung vornehmen. In der zweiten Stufe des Node Sharings werden auch diese Beta-Knoten zusammengeführt. Abbildung 2.5 stellt das Ergebnis dieses Schrittes dar. Auf Grund des hohen Aufwands für die Variabelbindungen, sind die Beta-Knoten im Allgemeinen die dominierenden Faktoren der Systemlaufzeit. Das Zusammenlegen der Beta-Knoten ist daher besonders ertragreich und reduziert den Aufwand stärker als das Zusammenlegen von Alpha-Knoten [FH03].

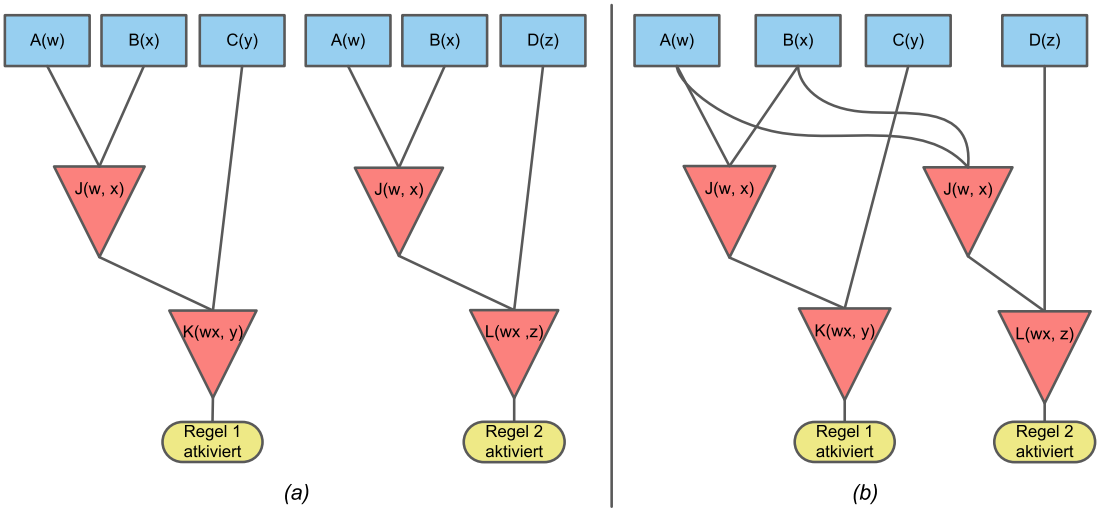

Abbildung 2.4: Node Sharing zwischen Alpha-Knoten

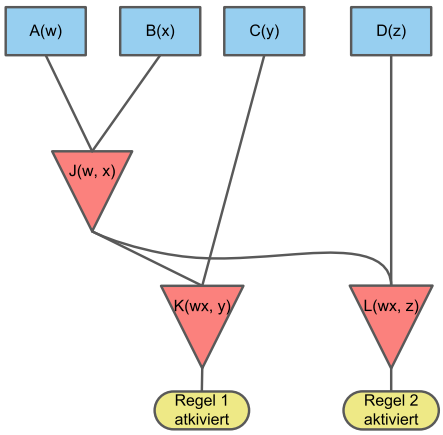

Abbildung 2.5: Node Sharing zwischen Beta-Knoten

2.3.5 Performance und Komplexität

Die Betrachtung der Performance des Rete-Algorithmus ist schwieriger als beim naiven Algorithmus, der in jedem Zyklus eine vorhersehbare Anzahl von Operationen ausführt. Die Performance des Rete-Algorithmus hingegen ist abhängig von Änderungen an der Wissesnsbasis zur Laufzeit. Dies bedeutet, dass sich der konkreten Komplexität nur grob genähert werden kann.

Im ersten Zyklus, in dem noch kein Netz erzeugt wurde und die Speicher noch leer sind, muss der Rete-Algorithmus alle Fakten gegen alle Prämissenelemente testen. Hierbei ist der Aufwand im Allgemeinen genauso hoch wie bei dem naiven Algorithmus [FH03]. Durch das aufwendige Preprocessing, das für das Erzeugen der Rete-Netze notwendig ist, kann im ersten Zyklus die Performance offensichtlich noch unter der des naiven Algorithmus liegen. Der Rete-Algorithmus zeigt seine Stärke erst in den folgenden Zyklen, wenn nur noch die Fakten betrachtet werden müssen, die sich geändert haben und für diese durch das Node Sharing eine effektive Strategie zur Verfügung steht. Dies bedeutet jedoch auch, dass im ungünstigsten Fall, also wenn sich alle Fakten ändern und kein Node Sharing möglich ist, die Performance weiterhin auf dem Niveau des naiven Algorithmus bleibt. Für typische Szenarien, in denen sich die Wissensbasis nur langsam ändert und ein moderates Node Sharing möglich ist, sowie effektive Indexmethoden bereit stehen, nennt [FH03] als Annäherung die folgende Komplexität. Demnach ist die Laufzeit proportional zu $\mathcal{O}(R'F'^{P'})$, wobei R' eine Zahl kleiner als die Anzahl der Regeln, F' die Menge der Fakten die sich in jeder Iteration ändern und P' eine Zahl größer als 1 aber kleiner als die durchschnittliche Anzahl von Prämissen pro Regel.

Darüber hinaus sei anzumerken, dass die durch den Rete-Algorithmus ermöglichte Leistungssteigerung durch einen wesentlich höheren Speicherbedarf erzielt wird. Der von Rete benötigte Bedarf wird von [Mad03] definiert als $\mathcal{O}(RFP)$, wobei R die Anzahl der Regeln, F die Anzahl der verwendeten Fakten und P die durchschnittliche Anzahl der Prämissen pro Regel. Im Vergleich hierzu benötigt der naive Algorithmus lediglich $\mathcal{O}$(F) Speicherplatz. Daher ist in verschiedenen Szenarien, etwa bei eingebetteten Systemen, abzuwägen, ob der Einsatz des Rete-Algorithmus sinnvoll ist.

Weiterführende Untersuchungen zu Komplexität, Performance und Speicherbedarf des Rete-Algorithmus wurden angestellt von [AF88] und [Doo95].

2.3.6 Strategien zur Optimierung

Einen entscheidenden Einfluss auf die Leistung des Rete-Algorithmus in einem Expertensystem hat die Beschaffenheit seiner Regelbasis. Insbesondere die Anordnung der Prämissen und die Spezifität der Regeln besitzen einen starken Einfluss auf die Größe der Zwischenergebnisse an den einzelnen Knoten. Da der Rete-Algorithmus die

teilerfüllten Zwischenergebnisse zwischen den Zyklen speichert und im nächsten Durchlauf neu untersucht, wirkt sich deren Anzahl direkt auf die erzielte Geschwindigkeit und den benötigten Speicherverbrauch aus.

Um eine möglichst effiziente Regelbasis zu erhalten, wurden daher die folgenden Strategien entwickelt [GR04]. Diese schließen sich jedoch gegenseitig nicht aus und die Anwendung kann zu Widersprüchen führen, in diesem Fall muss zwischen den erzielbaren Optimierungsergebnissen abgewogen werden.

- **Regeln so spezifisch wie möglich:** Regeln sollten generell so spezifisch wie möglich formuliert werden. Dies führt direkt zur Verkleinerung der Zwischenergebnisse an den Knoten, da im Allgemeinen weniger Fakten nach unten weitergereicht werden. Es ist jedoch zu beachten, dass allgemeiner formulierte Regeln meist bessere Möglichkeiten zum Node Sharing bieten.
- **Spezifischste Prämisse zuerst:** Die spezifischste Prämisse einer Regel sollte am weitesten links in einer Regel erscheinen, da zu erwarten ist, dass diese Prämisse die größte Anzahl von Fakten ausschließt und gleichzeitig viele Variablen mit den anderen Prämissen teilt. Die Anzahl der zu überprüfenden Fakten in den folgenden Beta-Knoten wird somit reduziert.
- **Prämissen mit wenigen Fakten zuerst:** Prämissen, die die mengenmäßig wenigsten Fakten betreffen, sollten möglichst am Anfang der Regel angesiedelt werden. Dies führt zu einer Reduzierung der teilerfüllten Ergebnisse und ist erstrebenswert, da der Aufwand für die weitere Untersuchung teilerfüllter Ergebnisse besonders hoch ist.
- **Prämissen mit volatilen Fakten zuletzt:** Die Prämissen die Fakten einbeziehen, welche regelmäßigen Änderungen unterliegen, sollten am Ende der Regeln platziert werden. Hierdurch kann die geringste mögliche Anzahl von Änderungen an den Zwischenergebnissen erreicht werden. Da häufig die spezifischsten Regeln auf den volatilsten Fakten basieren, kann hier ein Widerspruch entstehen.

Weiterführende Strategien und Überlegungen zur Optimierung der Regelbasis liefert insbesondere [GR04].

2.4 Regelbasierte Expertensysteme

2.4.1 Architekturmodell und Komponenten

Regelbasierte Expertensysteme können als eine spezielle Form von Regelsystemen aufgefasst werden. Daher ähneln sich Architektur und Komponenten dieser Systeme. Ein regelbasiertes Expertensystem baut auf der klassischen Architektur eines Regelsystems auf, erweitert diese aber um Komponenten für die Interaktion mit dem Wissensinge-

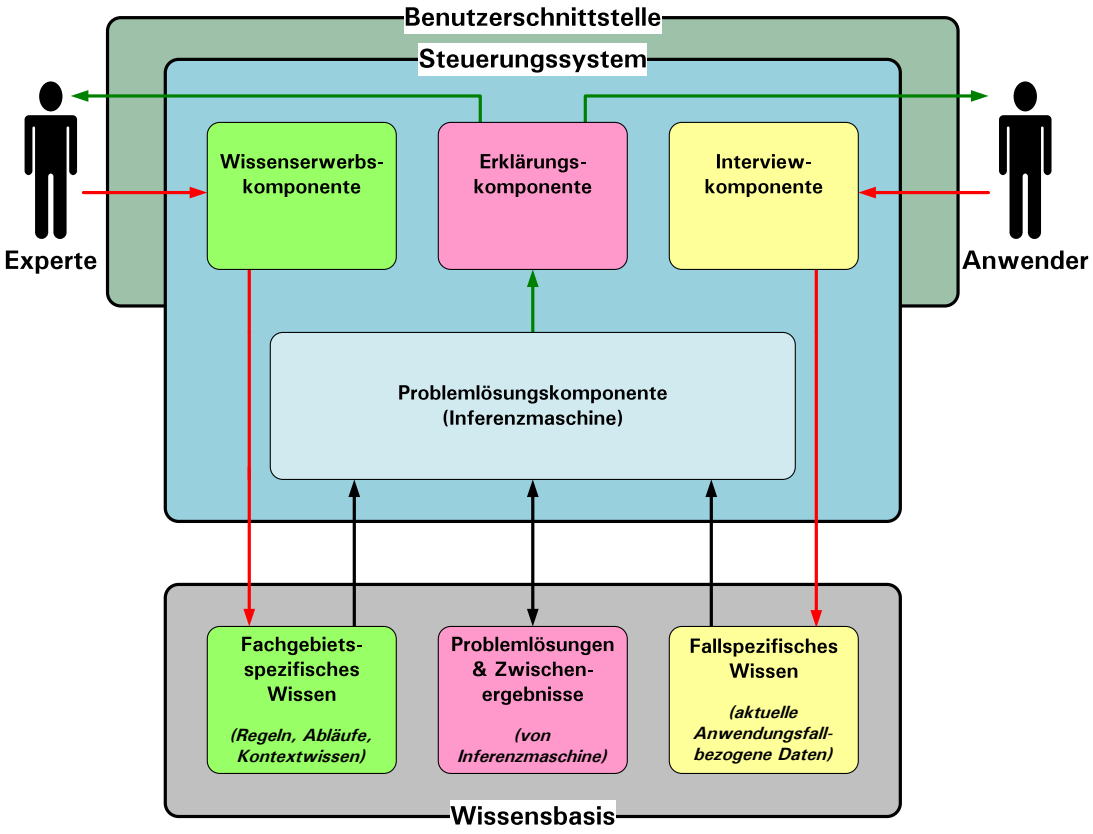

Abbildung 2.6: Theoretische Architektur eines regelbasierten Expertensystems

nieur und dem Endanwender. Gemeinsam ist beiden Systemen die Verwendung einer Inferenzmaschine und einer Wissensbasis, wobei auch im Expertensystem die Inferenzmaschine ein Pattern Matching zwischen fallspezifischen Fakten und Regeln durchführen muss. Zusätzlich enthält eine klassische Architektur für regelbasierte Expertensysteme die folgenden theoretischen Komponenten, deren Zusammenhänge in der Abbildung 2.6 dargestellt sind:

- **Benutzerschnittstelle:** Diese Komponente bildet das Grundgerüst für die Kommunikation des Systems mit dem Endanwender und dem Wissensingenieur. Bei der Ausführung des Systems werden dem Endanwender die ermittelten Informationen angezeigt. Die anderen Komponenten verwenden die Benutzerschnittstelle um mit dem Benutzer Informationen auszutauschen.
- **Erklärungskomponente:** Diese liefert dem Endanwender Informationen über die Entstehung der angebotenen Lösungen und bietet Erklärungen, die über das reine Anzeigen der Resultate und Schlussfolgerungen hinausgehen. Es sollen Fragen beantwortet werden wie beispielsweise: *Wie wurde die Lösung des Problems gefunden? Warum wird eine bestimmte Information verlangt?*
- **Wissenserwerbskomponente:** Diese Komponente unterstützt den Wissensin-

genieur bei dem Aufbau, der Pflege und der Erweiterung von fachbereichsspezifischen Wissen.

- **Interviewkomponente:** Die Interviewkomponente dient der Kommunikation mit dem Endanwender und ermöglicht die Eingabe von fallspezifischen Faktenwissen.

Darüber hinaus kann der Begriff der Regelbasis weitergefasst werden als bei klassischen Regelsystemen, da zusätzlich auch Wissen über Abläufe und Fachinformationen enthalten sein können, welche nicht in Form von Regeln hinterlegt sind. Somit kann diese Komponente auch als Wissensbasis für fachgebietsspezifisches Wissen betracht werden und die Regelbasis als ein Teilelement hiervon.

2.4.2 Vor- und Nachteile regelbasierter Expertensysteme

Damit ein effizienter Einsatz von Expertensystemen möglich ist, müssen sich Entwickler und Anwender solcher Systeme und derer zahlreichen Vor- und Nachteile bewusst sein. Im Nachfolgenden werden die wesentlichen Aspekte der Leistungsfähigkeit von Expertensystemen dargestellt.

Positiv ist bei allen Expertensystemen, dass diese den menschlichen Experten weitestgehend ersetzen können. Dies bringt mehrere Vorteile mit sich. Zum einen wird eine Kostenersparnis möglich, da Expertensysteme beliebig häufig verbreitet und eingesetzt werden können. Die Aufwendungen hierfür sind meist geringer als die nötigen Aufwendungen für die Beschäftigung von menschlichen Experten [GR04]. Zum anderen können Expertensysteme jederzeit und praktisch überall eingesetzt werden, was eine erhöhte Verfügbarkeit von Expertenwissen erlaubt, beispielsweise auch nachts oder an gefährlichen Orten. Außerdem bleibt das in den Expertensystemen festgehaltene Wissen prinzipiell unbegrenzt erhalten, während dies bei menschlichen Experten nicht gewährleistet ist.

Die Geschwindigkeit mit der Expertenwissen bereitgestellt wird, ist meist wesentlich höher als bei menschlichen Experten [GR04]. Dies ermöglicht beispielsweise die Überwachung von Systemen zur Echtzeit oder die besonders schnelle und emotionslose Begutachtung in Notfallsituationen.

Ein weiterer Vorteil ist die meist höhere zeitliche Verfügbarkeit von Expertensystemen für den einzelnen Anwender. Diese erlaubt es dem Anwender, sich die erstellten Problemlösungen und die dahinter liegenden Strategien umfangreich erklären zu lassen und alternative Szenarien durchzuspielen. Hierdurch kann ein höherer Lerneffekt erzielt werden, als bei einem menschlichen Experten, der oftmals keine Zeit für diese Erklärungen aufbringen kann.

Eine Besonderheit von regelbasierten Expertensystemen ist es, dass das Wissen des

Experten in sehr direkter und bekannter Repräsentation abgelegt werden kann. Regeln entsprechen den bekannten Denkmustern der meisten Menschen und sind daher leichter zu erfassen und zu formulieren als andere Wissensrepräsentationen [GR04]. Dies bietet den Vorteil, dass der Experte sein Wissen leicht bereitstellen kann. Auf Grund der Möglichkeit diese Regeln in einer Form, ähnlich der natürlichen Sprache, in das Expertensystem zu integrieren, kann eine erhöhte Wartbarkeit und Qualitätssicherung erzielt werden. Der Experte kann direkt in diesen Prozess integriert werden, wodurch die Gefahr einer fehlerhaften Transformation von natürlichsprachlichen Regeln in systemverständliche Regeln verringert wird.

Darüber hinaus erlauben Regeln eine klare Trennung von Fachlogik, Daten und Anwendungscode, da alle Regeln in einem eigenen Repository gesammelt werden können. Dies ermöglicht eine einfachere Pflege aller Elemente.

Dem Gegenüber stehen einige Nachteile, die vor dem Einsatz abgewogen werden müssen. Zunächst ist der Aufwand für die Erstellung eines Expertensystems verhältnismäßig hoch, da in der Regel sowohl Experten, als auch Softwareentwickler und Endanwender für einen längeren Zeitraum in den Erstellungsprozess involviert sind, ohne das direkt ein Nutzen entsteht. Darüber hinaus kann auch die anschließende Wartung und Pflege des Systems und der Wissensbasis mit hohen Aufwendungen verbunden sein.

Problematisch ist auch, dass einige Experten nicht in der Lage sind ihr Wissen und ihre Lösungsstrategien explizit und hinreichend darzustellen, so dass dieses Wissen nicht vollständig in das System integriert werden kann.

Des Weiteren ist es mit Problemen verbunden, dass regelbasierte Expertensysteme Schwierigkeiten haben, ein tiefer gehendes Verständnis für den Problembereich zu entwickeln [Lug01]. Stattdessen werden nur Strategien auf Basis allgemeingültiger Regeln angewendet, wobei die zugrunde liegenden Zusammenhänge oft nicht vollständig abgebildet werden können. Dies führt dazu, dass regelbasierte Expertensysteme nicht selbstständig tiefer gehende Erklärungen liefern können, sondern nur die einzelnen Schritte und Regeln, die zur Lösungsfindung beitrugen.

Außerdem besteht häufig ein Mangel an Flexibilität und Robustheit gegenüber neuen und abgewandelten Problemstellungen. Expertensysteme sind nicht in der Lage neue Strategien auf Basis von Grundprinzipien zu generieren [Lug01] und somit kreativ neue Regeln aufzustellen.

Expertensysteme besitzen ein Wissen, dass auf einen sehr speziellen Fachbereich beschränkt ist. In diesem Gebiet verfügen diese im Allgemeinen über eine hohe Kompetenz. Dies macht diese Systeme zum einen besonders leistungsfähig und eignet sie für den zuverlässigen praktischen Einsatz. Zum anderen fehlt jedoch Allgemeinwissen, so dass die Kompetenz des Expertensystems beim Verlassen des Fachbereichs steil abfällt.

Ein Anwender von Expertensystemen muss sich dieses Gefälles bewusst sein und kritisch darauf achten, ob er bei der Anwendung den Kompetenzbereich des Systems verlässt.

Eine weitere Herausforderung stellt die Verifizierung der Korrektheit des Expertensystems dar, welche noch schwieriger zu überprüfen ist, als bei konventionellen Computersystemen [Lug01]. Dies ist unter Anderem mit der hohen Schwierigkeit der Validierung von Regeln zu begründen.

Ein besonderer Nachteil von regelbasierten Expertensystemen ist, dass diese häufig heuristische Regeln verwenden, welche die Eigenschaft haben wenig robust zu sein und nur schwer mit fehlenden oder unerwarteten Fakten umgehen können [Lug01].

Ausführliche Informationen zu den Vor- und Nachteilen von regelbasierten Expertensystemen bietet [GR04] und [Lug01], welche auch als Grundlage für die hier gegebenen Informationen dienten.

3 Verwendete Technologien

Im Rahmen dieses Kapitels werden die Grundlagen, Eigenschaften und Funktionen der beiden Technologien vermittelt, die in dieser Arbeit von besonderer Bedeutung sind. Zum einen ist dies die Eclipse Rich-Client Platform, die das Framework für die zu entwickelnde Architektur bildet. Zum anderen JBoss Drools, welche, in dem zu entwerfenden Expertensystem, als Regelmaschine zum Einsatz kommt.

3.1 Eclipse Rich-Client Platform

3.1.1 Grundlagen von Eclipse RCP

Die Eclipse Rich-Client Platform (RCP) [Eclc] ist ein Framework für die Entwicklung von Anwendungen auf Basis des Eclipse Projekts. Hierbei handelt es sich um ein Open Source Projekt der Eclipse Foundation, welches seine Wurzeln in dem Ziel hat, eine freie und erweiterbare Java-Entwicklungsumgebung (IDE) zu entwickeln. Die Eclipse Rich-Client Platform definiert die wesentlichen Komponenten der Eclipse IDE, die nötig sind, um eigene Applikationen zu entwickeln. Zu Beginn dieser Arbeit stellte die Eclipse Version 3.3 die aktuellste verfügbare Version dar, und ist daher die verwendete Grundlage für diese Arbeit.

Die interne Architektur von Eclipse besteht aus einzelnen Komponenten und basiert auf dem Konzept der Erweiterbarkeit durch Plugins. Diese Plugins lassen sich dynamisch in das Framework einbinden und vergrößern die Funktionalität der Gesamtapplikation. Hierzu verwendet Eclipse definierte Schnittstellen. Diese Plugin-basierte Architektur ist Grundkonzept des gesamten Eclipse Projekts und wird auch für die Basiskomponenten der Eclipse IDE selbst verwendet. [GB04] bezeichnet Eclipse daher „als eine Sammlung von Plätzen an den Funktionalität eingefügt wird (Erweiterungspunkten) und eingefügter Funktionalität (Erweiterungen)“.

Als Grundlage für die Kollaboration zwischen den einzelnen Plugins verwendet Eclipse den OSGi-Standard der OSGi Alliance [OSG]. Dieser definiert ein dynamisches Modulsystem für Java und ermöglicht es zur Laufzeit Softwarekomponenten (Bundles) und Dienste (Services) zu installieren, starten, stoppen und deinstallieren [WHKL08]. Die Kommunikation zwischen einzelnen Komponenten erfolgt durch ein serviceorientiertes Konzept. Eclipse verwendet die OSGi Open Source Implementierung Equinox.

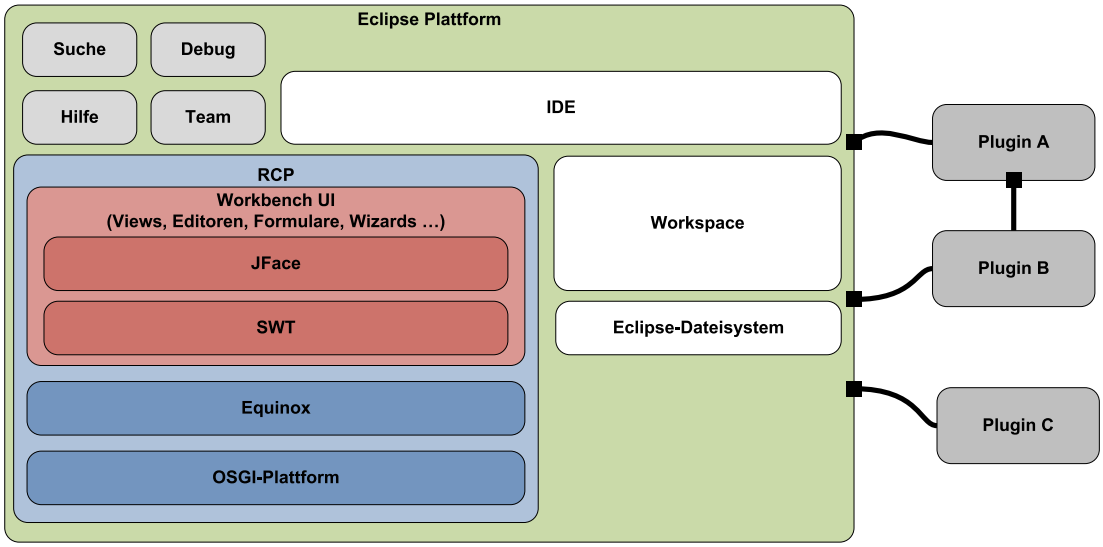

Abbildung 3.1: Darstellung der Kernkomponenten der Eclipse-Plattform in Version 3.3

Abbildung 3.1 zeigt die wesentlichen Komponenten der Eclipse-Plattform in Version 3.3 nach [Dau08] und [Ecla].

3.1.2 Eclipse durch Plugins erweitern

Um Anwendungen zu entwickeln, die auf der Eclipse Rich-Client Platform aufsetzen, müssen diese selbst in Form eines oder mehrerer Plugins implementiert und in die Eclipse-Plattform eingebunden werden.

Ein Eclipse Plugin entspricht vollständig einem OSGi Bundle [ML05] und muss daher die entsprechenden Spezifikationen des OSGi Standards erfüllen. Ein typisches Plugin besteht aus mindestens einem Java-Archiv (JAR), welches die Klassen des Plugins enthält. Darüber hinaus besitzt ein Plugin grundsätzlich ein OSGi-Manifest, das das Plugin beschreibt (Name, Version, Bereitsteller), seine Abhängigkeiten zu anderen Plugins angibt, die Sichtbarkeit der eigenen Pakete für andere Plugins definiert und nötige Klassen für den eigenen Klassenpfad mitteilt. Darüber hinaus kann im Manifest noch eine Aktivator-Klasse angegeben werden, die den Lebenszyklus des Plugins verwaltet. Anhand dieser Informationen wird der Klassenpfad des Plugins dynamisch erzeugt und das Plugin kann von Eclipse intern über einen speziellen Klassenlader gestartet und gestoppt werden. Es ist anzumerken, dass jedes Plugin einen eigenen Klassenpfad und Namensraum besitzt. Es wird nicht, wie bei einer Standard-Java-Anwendung, ein globaler Klassenpfad verwendet. [Dau08].

Für die Kommunikation zwischen einzelnen Plugins bietet OSGi eine serviceorientierte Architektur an, die es einem Plugin erlaubt seine Dienste anderen Plugins zur Verfügung zu stellen. Hierzu muss ein Plugin seinen Dienst zunächst mittels eines Java-

Interfaces formal spezifizieren. Anschließend wird der Dienst beim OSGi-Framework registriert, wobei seine Eigenschaften und Parameter angegeben werden. Andere Plugins können den angebotenen Dienst nun beim OSGi-Framework auffinden und anschließend die angebotenen Methoden verwenden.

Die Verwendung dieser serviceorientierten Kommunikation eignet sich insbesondere für den Einsatz zwischen Plugins, die nicht zwingend gemeinsam in einer Anwendung ausgeliefert oder weitergepflegt werden, die also eine besonders starke Entkopplung erfordern. Darüber hinaus können Plugins ihre eigenen Pakete offen legen und gegenseitig auf diese zugreifen. Diese Methode gleicht dem Zugriff zwischen einzelnen Paketen in Java. Der direkte Paketzugriff ist in vielen Anwendungsfällen hinreichend und gleichzeitig einfacher umzusetzen.

Ein Eclipse Plugin verfügt historisch bedingt neben dem OSGi-Manifest über einen weiteren Mechanismus, der die Kommunikation zwischen Plugins unterstützt. Eclipse erlaubt es dem Plugin sogenannte Erweiterungen (die konkrete Dienstimplementierungen) und Erweiterungspunkte (die Dienstspezifikation) zu definieren, die es ermöglichen Dienste und Funktionen zwischen Plugins anzubieten. Diese werden in der Plugin-Manifest Datei plugin.xml definiert. Im Gegensatz zum OSGi-Manifest genügt die Spezifizierung über ein Java-Interface hierbei nicht aus. Stattdessen muss ein Erweiterungspunkt explizit beschrieben werden, was eine stärkere Ausdruckskraft besitzt und daher eine genauere Spezifikation erlaubt [Dau08].

Dem Entwickler einer Eclipse Rich-Client Anwendung werden auf Grund dieses Plugin-Konzepts zahlreiche Funktionen anderer Plugins des Eclipse Projekts zur Verfügung gestellt, die direkt in die zu entwickelnde Anwendung eingebunden werden können. Das Eclipse Projekt wird auch als Werkzeugkasten bezeichnet [Dau08], dessen Funktionen über standardisierte Konzepte in der eigenen Applikation verwendet werden können.

3.1.3 Grafische Elemente von Eclipse RCP

Die Benutzeroberfläche (GUI) einer Eclipse Rich-Client Anwendung baut auf die Darstellungslogik der Eclipse IDE auf und besitzt Eigenschaften und Bezeichnungen, die an diese Entwicklungsumgebung erinnern. Eine Eclipse RCP Applikation verfügt über eine so genannte Workbench, die den konzeptionellen Rahmen der Benutzeroberfläche darstellt und praktisch betrachtet, alle anderen Elemente enthält. Innerhalb der Workbench befinden sich im Wesentlichen beliebig viele Anschichten (Views) und typischerweise ein Editor. Anders als der Name vermuten lässt, ist ein Editor nicht zwingend für die Bearbeitung von Text oder Grafik einzusetzen, sondern stellt stattdessen eine für die Anwendung besonders wichtige View dar [ML05]. Der Editor ist die zentrale GUI-Komponente der Applikation und kann nur einmalig existieren. Eine View hingegen ist meist eine ergänzende oder unterstützende Ansicht auf Informationen oder ein

Formular für eine besondere Aufgabe.

Die Kombination von verschiedenen Views und einem Editor wird als Perspektive bezeichnet und kann als eine Ansichtsseite innerhalb der Workbench aufgefasst werden [CR06]. Perspektiven bündeln Elemente die eine logische Zusammengehörigkeit besitzen und einer bestimmten Aufgabenstellung unterliegen.

Eclipse bietet verschiedene Grafikbibliotheken an, um Views und Editoren mit konkreten Grafikelementen zu füllen. Das Standard Widget Toolkit (SWT) ist von IBM als Alternative zu Suns AWT/Swing entwickelt worden und enthält einen Standardsatz von nativen Grafikkomponenten (Widgets) für das jeweilige Windowing-System. Nur wenn die gewünschten Komponenten nicht nativ verfügbar sind, werden diese in Java simuliert [GB04]. Dies führt zu einem komfortableren Verhalten für den Anwender und einer gesteigerten Performance der Anwendung.

Die JFace Bibliothek baut auf SWT auf und bietet eine fertige Sammlung höherwertiger GUI Komponenten, wie etwa Dialoge oder Wizards [Dau08]. Des Weiteren verfügt JFace seit Eclipse Version 3.3 über eine Databinding-Funktion, die es erlaubt das Datenmodell direkt mit einer grafischen Sicht zu verknüpfen und Änderungen in beide Richtungen zu propagieren.

Speziell für die Gestaltung von Formularen wurde die Forms API Bibliothek entwickelt. Diese baut ebenfalls auf die Elemente von SWT auf und erlaubt die vereinfachte und vereinheitlichte Gestaltung von Formularelementen. Das Design der Elemente ist gezielt an dem von Webformularen angelehnt [ML05]. Durch das FormText Widget wird die aufbereitete Ausgabe von HTML-ähnlichem Text, inklusive der automatischen Umwandlung von URLs in Hyperlinks, möglich. Das Forms API wird von Eclipse RCP selbst in zahlreichen Komponenten verwendet. Ein Beispiel stellen die Editoren für die OSGi- und Plugin-Manifeste dar.

3.2 Regelmaschine JBoss Drools

3.2.1 Grundlagen von JBoss Drools

JBoss Drools [JBo] ist eine frei verfügbare Java-basierte Open Source Regelmaschine unter der Apache Software Lizenz 2.0 und wird seit 2001 von einer offenen Gemeinschaft entwickelt. Zu Beginn dieser Arbeit ist Version 4.0.7 des Produkts verfügbar.

Den Kern von Drools bildet eine Java Bibliothek, die eine vorwärtsgerichtete Regelmaschine enthält. Diese verwendet eine objektorientierte Variante des Rete-Algorithmus. Daneben verfügt Drools über weitere Bibliotheken, die optional zusätzlich zu der Kernkomponente verwendet werden können. Eine der Bibliotheken enthält ein Plugin für die Eclipse Entwicklungsumgebung und unterstützt den Anwender bei der Erstellung von Regeln und Drools-basierten Anwendungen.

Zusätzliche bietet Drools ein komplettes Business Rule Management System (BRMS) auf Basis der Java Enterprise Edition, das die Verwaltung von Regeln über einen Webbrowser ermöglicht. Dieses System wird, auf Grund der Aufgabenstellung der vorliegenden Arbeit nicht verwendet.

Drools unterstützt Java vollständig und bietet für die Steuerung der Regelmaschine eine Java Programmierschnittstelle an. Auch erlaubt Drools die Repräsentation von Fakten in Form von Java Objekten und ermöglicht es Regeln in einer Java-ähnlichen Syntax zu formulieren. Darüber hinaus verwendet Drools intern typische Java-Objektklassen für die Abbildung von Regeln [Wun06]. Neben der Java-Version existiert eine Portierung für die Microsoft .NET Umgebung, welche auf älteren Versionen von Drools aufbaut.

Die Vorgaben dieser Arbeit fordern den Einsatz der Regelmaschine JBoss Drools für das zu entwickelnde Expertensystem. Diese Forderung entstand im Vorfeld anhand einer durchgeführten Evaluation. Diese ergab, dass JBoss Drools insbesondere auf Grund seiner guten Java Unterstützung, der Eclipse Anbindung, der aktiven Entwickler- und Nutzergemeinde, dem objektorientieren Regelansatz und der kostenlosen Open Source Software gegenüber Konkurrenzprodukten zu bevorzugen ist.

3.2.2 Umsetzung der theoretischen Architektur in Drools

Das Kernsystem von Drools stellt eine Implementierung der klassischen Architektur für regelbasierte System dar (vgl. Abbildung 2.6). Somit verwendet Drools eine Wissensbasis, die aufgeteilt ist in eine Faktenbasis (genannt Working Memory) und eine Regelbasis (genannt Rulebase), die Regelpakete enthält. Diese Regelpakete, so genannte Packages, enthalten eine Sammlung von definierten Regeln, welche typischerweise in einem fachlichen Zusammenhang stehen. Fakten werden in Drools im Allgemeinen durch Java Beans repräsentiert.

Die Faktenbasis und die Regelbasis werden durch die Drools Inferenzmaschine miteinander verknüpft, wobei der objektorientierte ReteOO-Algorithmus, eine Abwandlung des klassischen Rete-Algorithmus, für das Pattern Matching verwendet wird.

Die Kernarchitektur von Drools wird von den Entwicklern selbst [PNF+] in die folgenden zwei Hauptkomponenten eingeteilt. Diese sind in Abbildung 3.2 veranschaulicht. Anhand der Beschreibung dieser Komponenten lässt sich auch der generelle zeitliche Ablauf bei der Erstellung der Wissensbasis und dem Aktivieren von Regeln erkennen:

- **Zusammenstellungs-Komponente:** Diese Komponente steuert den Zusammenstellungsprozess von Regelpaketen, die anschließend in die Regelbasis aufgenommen werden können.

  Es werden zunächst vom Anwender Regeldateien erzeugt, wobei Drools verschiedene Formate unterstützt. Anschließend werden diese Dateien an den Regelparser

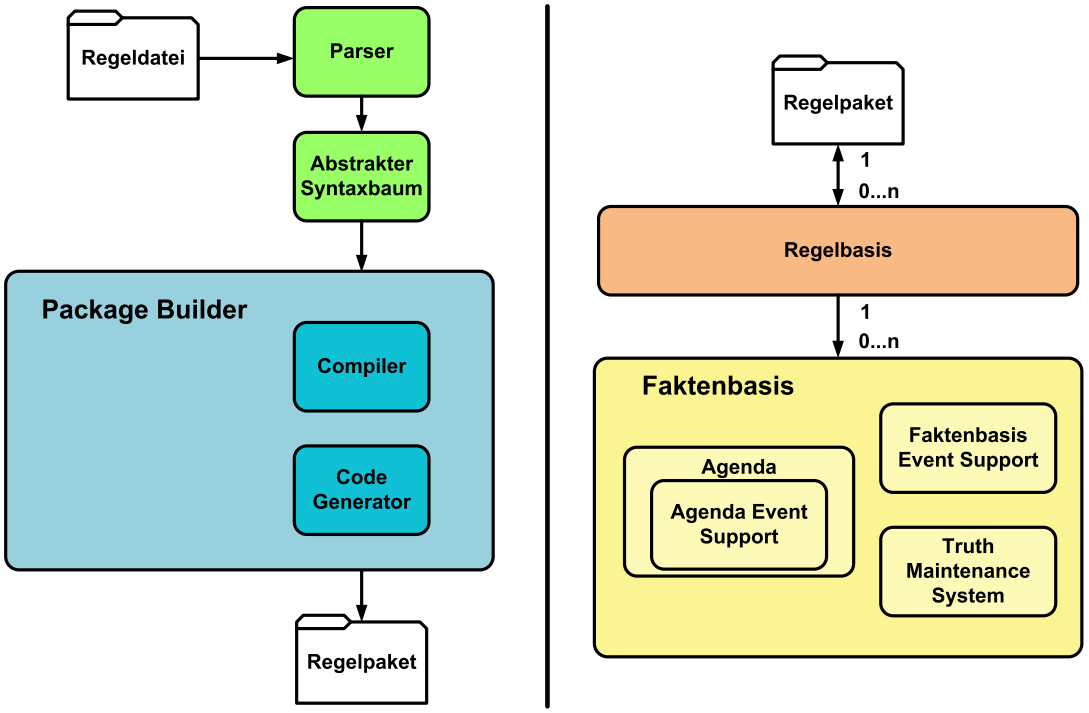

Abbildung 3.2: Die wesentlichen Kernkomponenten von Drools 4

übergeben, der über eine Antlr v3 Grammatik [ant] definiert ist. Nachdem die grammatikalische Korrektheit der Regeln vom Parser überprüft wurde, werden diese in einen Syntaxbaum überführt, der die Regeln beschreibt. Dieser Syntaxbaum wird anschließend an die Package Builder Komponente von Drools übergeben, welche aus dem Baum ein kompiliertes Regelpaket in Binärcode erzeugt. Dieses Regelpaket kann ein oder mehr Regeln enthalten und ist in sich abgeschlossen, sie besitzen keine Abhängigkeiten zu anderen Elementen.

- **Laufzeit-Komponente:** Diese Komponente stellt die von der Inferenzmaschine verwendete Wissensbasis von Drools dar.

  Die in der Zusammenstellungs-Komponente erzeugten Regelpakete können in die Regelbasis eingefügt werden und stehen damit der Regelmaschine zur Verfügung. In der Regelbasis können ein oder mehrere Regelpakete enthalten sein. Drools erlaubt es, jederzeit Regelpakete in die Regelbasis aufzunehmen und wieder zu entfernen. Jede Regelbasis kann mit beliebig vielen Faktenbasen verknüpft sein und initiiert selber deren Erzeugung. Sobald eine Faktenbasis erzeugt ist, können Faktenobjekte in diese eingefügt werden. Die Inferenzmaschine betrachtet alle miteinander verbundenen Regel- und Faktenbasen und führt für diese das Pattern Matching durch, das gegebenenfalls zu einer Aktivierung der Regeln führt.

  Drools bietet die Möglichkeit, dass die Inferenzmaschine automatisch auf alle Änderungen an der Faktenbasis reagiert und direkt ein neues Pattern Matching

durchführt. Das Feuern der aktivierten Regeln hingegen erfolgt grundsätzlich erst nachdem dies explizit angestoßen wurde, etwa durch die Methode `fireAllRules()`.

Die Drools-Architektur siedelt darüber hinaus verschiedene Subkomponenten innerhalb der Faktenbasis an. Dies sind das Truth Maintenance System, das Konflikte zwischen Fakten handhabt, sowie das Ereignisunterstützungssystem der Faktenbasis, die Agenda-Liste und deren Ereignisunterstützungssystem.

3.2.3 Objektorientierte Regeln und Fakten in Drools

Um eine einfache Anbindung an die Programmiersprache Java zu ermöglichen, unterstützt Drools die objektorientierte Definition von Regeln und Fakten. Darüber hinaus können viele, einem Java-Entwickler bekannte, Sprachkonstrukte und Mechanismen verwendet werden, was vielen Personen das Erlernen von Drools vereinfacht.

In Drools werden Fakten durch Java Beans Objekte repräsentiert. Dies bedeutet, dass die Objekte Getter- und Setter-Methoden besitzen, die es Drools erlauben die Attribute eines Fakts abzurufen und zu verändern. Auf Grund dieser Konvention ist es in den Regeln möglich, direkt auf die Attributwerte zuzugreifen, ohne deren get-Methoden verwenden zu müssen. Dies vereinfacht die Regelsyntax. Die Faktenobjekte werden beim Einfügen in die Faktenbasis intern nicht kopiert, sondern über Zeiger referenziert [PNF+].

Um der Inferenzmaschine automatisch Veränderungen an einem Fakt mitzuteilen, verwendet Drools das Konzept der Java PropertyChangeListener. Bei diesem Konzept werden Ereignisnachrichten an einen registrierten Empfänger versendet, hier an die Inferenzmaschine, sobald eine Veränderung an einem Faktenattribut erfolgt ist. Dies veranlasst die Inferenzmaschine zu einem erneuten Pattern Matching.

Regeln können in verschiedenen Formen abgelegt werden, abhängig vom Einsatzzweck. Die grundlegende Form ist die Verwendung der Drools Rules Language (DRL), die einen mächtigen Sprachumfang besitzt und objektorientierte Definitionen ermöglicht. Die Verwendung setzt geschulte Wissensingenieure voraus. Es werden zwei Dialekte angeboten, der eine ist MVEL [Coda], aufbauend auf einer speziellen Ausdruckssprache für Java. Der zweite Dialekt ermöglicht die direkte Verwendung von Java-ähnlicher Syntax. Die DRL erlaubt das Importieren anderer Javaobjekte und die Verwendung derer Funktionen, darüber hinaus können in der DRL-Datei selbst Java-Funktionen definiert werden.
Das folgende einfache Beispiel veranschaulicht die Regelstruktur in Drools:

Listing 3.1: Regel in Java Dialket

```
rule "Check for Country"
        agenda-group "SelectGenre"
        when
                person : Person(musicalPreference matches "Country")
        then
                person.setRecommendation("Johnny Cash")
                update (person);
end

rule "Check for Jazz"
        agenda-group "SelectGenre"
        when
                person : Person(musicalPreference matches "Jazz")
        then
                person.setRecommendation("Miles Davis")
                update (person);
end
```

Weitere Möglichkeiten zur Definition von Regeln, die Drools unterstützt sind:

- **Domain Specific Language (DSL):** Auf Basis einer DSL können Regeln in natürlicher Sprache formuliert werden, so dass auch unerfahrene Anwender einen Zugang zu den Regeln haben und diese erzeugen oder ändern können. Drools übernimmt selbstständig die Übersetzung der Regeln aus der natürlichen Sprache und verbindet diese mit den Faktenobjekten. Damit dies möglich wird, muss die DSL zuvor von einem erfahrenen Anwender definiert werden und hierbei verschiedene Zuordnungen zwischen natürlicher Sprache und Faktenmodell getroffen werden.
- **Decision Tables:** Diese Entscheidungstabellen richten sich insbesondere an unerfahrene Anwender, die mit Tabellenkalkulationsanwendungen vertraut sind. Es wird ermöglicht Regeln auf Basis von Entscheidungstabellen zu definieren, die in den Formaten Excel, Impress oder CSV vorliegen [PNF+]. Wie bei der DSL werden die Elemente der Tabelle automatisch in Regeln übersetzt und mit dem Faktenmodell verknüpft. Für die Definition dieser Verknüpfung ist in vielen Fällen ein erfahrener Anwender nötig.
- **XML - Regelsprache:** Diese native Regelsprache ist nur als Alternative zur DRL gedacht. Da die Lesbarkeit wesentlich geringer ist, sollte XML nur in begründeten Einzelfällen verwendet werden, etwa in Umgebungen, in denen XML das Standardformat ist. [PNF+]

3.2.4 ReteOO und Konflikt-Lösung in Drools

Der von Drools verwendete Pattern Matching Algorithmus ReteOO ist eine Weiterentwicklung des klassischen Rete-Algorithmus und wurde von Bob McWhirter entworfen [JBo]. Das Ziel von ReteOO ist der optimierte Einsatz in objektorientierten Regelmaschinen. Die verfügbare Literatur zu diesem Algorithmus ist äußerst knapp und nur online veröffentlicht. Der folgende Abschnitt versucht, anhand dieser beschränkten Informationen, eine Erklärung für die wesentlichen Unterschiede zwischen dem klassischen Rete-Algorithmus und ReteOO zuliefern.

Der Hauptansatzpunkt für die Änderungen ist das Einfügen von Faktenattributen in die Regelmaschine. [Codb] beschreibt diesen Ansatz wie folgt: Während in klassischen Sprachsystemen wie Lisp die Attribute direkt in die Regelmaschine eingebunden werden können, müssen in objektorientierten Sprachen, auch die Faktenobjekte und deren Klassentypen, beachtet werden. ReteOO vermeidet dies und verwendet hierzu lediglich einen einzelnen Typ von Wurzelobjekten für das Netz. Dieses Wurzelobjekt besitzt ein Attribut für Objekttypen und ein Namensattribut, anhand dessen die konkreten Objekte unterschieden werden können. Diese Verwendung eines Namens entspricht der Variabelzuweisung von Objekten in Java und existiert nicht im klassischen Rete-Algorithmus, wo alle eingefügten Faktentupel namenlos sind.

Eine weitere Besonderheit ist die Verwendung von so genannten ObjectTypeNodes [PNF+]. Diese befinden sich direkt unterhalb des Wurzelknotens und prüfen den Typ von neu eingefügten Objekten anhand der Java Methode `instanceof()`. Es werden nur die Objekte weitergereicht, deren Typ für die nachfolgenden Knoten relevant ist. Dies erlaubt eine effektive Vorauswahl und vermeidet unnötige Untersuchungen an den Folgeknoten. Die Prüfung erfolgt intern anhand einer HashMap und wird beim Neueinfügen von Faktenobjekten angestoßen.

ReteOO nutzt die Fähigkeit objektorientierte Sprachen aus, Attribute und Beziehungen durch Sprachkonstrukte abzubilden. Hierzu wurden zusätzliche Knotentypen eingeführt, welche nur die Aufgabe haben, Attribute zu ermitteln und diese an die vorbeiziehenden Faktentupel anzuhängen. Auf Grund dieser Knoten besitzt ein ReteOO-Baum eine Struktur in der Alpha-Knoten und Beta-Knoten verschachtelt sind und sich abwechseln. Der klassische Baum besitzt hingegen Alpha-Knoten nur direkt unter der Wurzel.

Die HashMap wird auch verwendet um das Propagieren von Faktenobjekten durch den Baum zu beschleunigen [PNF+]. Hierzu werden die Alpha-Knoten in die HashMap aufgenommen. Dies ermöglicht, dass ein neu eingefügtes Faktenobjekt nicht mehr an jeden Alpha-Knoten weitergereicht werden muss, sondern direkt anhand der HashMap zu den relevanten Knoten springen kann.

Auch beim Aufbau des Baums unterscheiden sich die beiden Rete-Varianten in einem

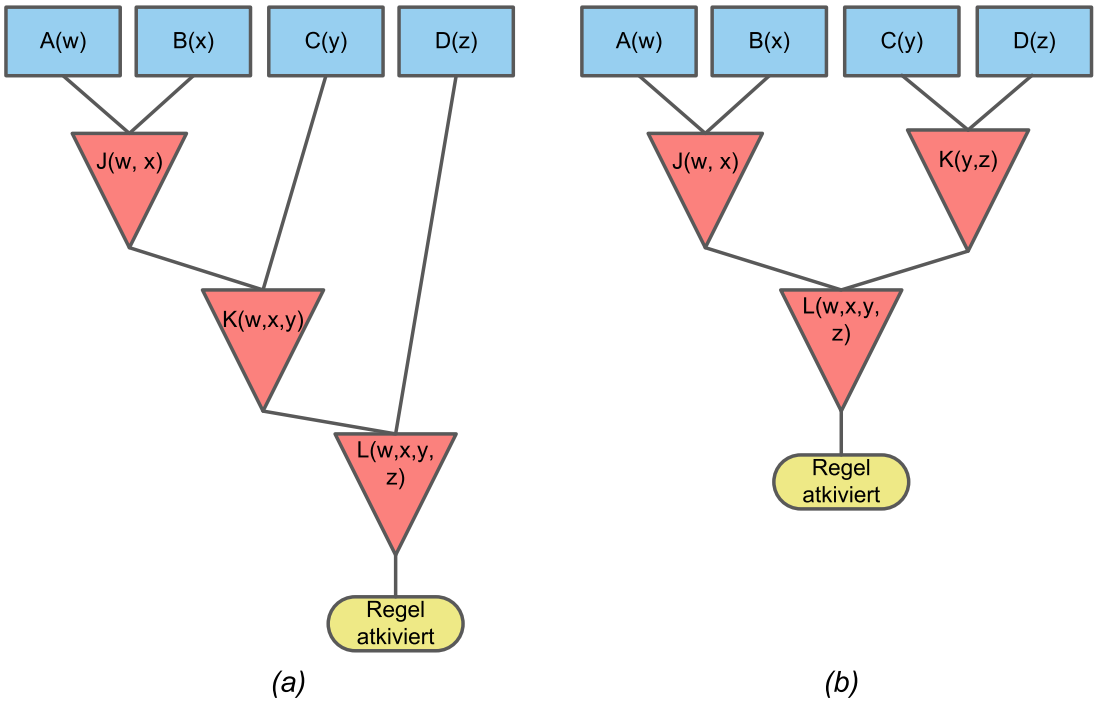

Abbildung 3.3: Darstellung der unterschiedlichen Netzgenerierung bei Rete und ReteOO

wesentlichen Punkt. Während im klassischen Baum nur ein einzelnes Objekt auf der rechten Seite des Baums eingefügt werden kann und dieser mit der kompletten linken Seite des Baums über einen Beta-Knoten verbunden wird, können beim ReteOO-Baum direkt zwei Objekte rechts in den Baum eingefügt werden [Codb]. Erst das Ergebnis des Beta-Knotens, das diese Objekte verbindet, wird mit der linken Seite des Baums verknüpft. Abbildung 3.3 stellt dies anschaulich dar.

Drools bietet zahlreiche Strategien zur Konflikt-Lösung für den Fall, dass mehrere Regeln aktiviert wurden. Hierbei unterstützt Drools Strategien, die auf den zuvor vorgestellten Standardstrategien basieren (vgl. 2.2.3) und erlaubt darüber hinaus eigene Strategien zu integrieren. Die von Drools verwendete Voreinstellung benutzt die Salience Strategie und LIFO (last in, first out), wobei Regeln die durch die gleiche Aktion aktiviert wurden, denselben LIFO-Wert besitzen und beliebig ausgeführt werden [PNF+].

Neben der Verwendung von Prioritäten über die Salience Strategie, ermöglicht es Drools, Regeln in so genannte Agenda-Gruppen einzuteilen und dadurch die Ausführungsreihenfolge zu beeinflussen. Durch diese Funktion ist es möglich beim Feuern von Regeln, jeweils nur eine Gruppe zu beachten und andere aktivierte Regeln zu ignorieren. Die Auswahl der Gruppen kann über die API erfolgen oder durch Regeln angestoßen werden.

3.2.5 Ruleflow-Funktion von Drools

Eine Besonderheit von Drools sind so genannten Ruleflows, die es erlauben die Abläufe der Regelbearbeitung, ähnlich wie bei einem Workflowsystem zu steuern. Die Möglichkeit der Einflussnahme auf den Ablauf der Regelausführung ist bei Ruleflows expliziter und anschaulicher als bei der Verwendung von Salience und Agenda-Gruppen. Modelliert werden die Abläufe über Flussdiagramme, wobei im Diagramm verschiedene Komponenten (Knoten) verwendet werden können, die jeweils spezifische Funktionen repräsentieren. Drools bietet in der aktuellen Version 4 die folgenden acht Knotentypen zur Modellierung an [PNF+]:

1. **Start:** Dies ist der Ausgangspunkt des Ruleflow. Es kann nur ein Startpunkt existieren.
2. **Ende:** Ein Ruleflow muss über mindestens einen Endpunkt verfügen. Sobald einer dieser Endpunkte erreicht ist, wird die Ausführung des Ruleflows beendet.
3. **Ruleflow-Gruppe:** Ähnlich dem Konzept der Agenda-Gruppe lassen sich Regeln zu Gruppen zusammenfassen, die an einem bestimmten Punkt des Ruleflows betrachtet werden. Innerhalb des Diagramms erlaubt eine „Ruleflow-Gruppe“ festzulegen, dass eine gruppierte Menge von Regeln dem Pattern Matching unterzogen wird. Anschließend werden die aktivierten Regeln gefeuert. Die Regelmaschine betrachtet, aktiviert und feuert die Regeln der Ruleflow-Gruppe solange in einer Schleife, bis keine neuen Regeln mehr aktiviert und gefeuert wurden. Dies erlaubt es, dass Regeln erst durch andere Regeln derselben Gruppe angestoßen werden können. Die Schleifenausführung kann durch das Regel-Statement `no-loop` explizit unterbunden werden.
4. **Split:** Diese Komponente stellt eine Gabelung im Flussdiagramm dar. Es können Bedingungen definiert werden, die festlegen in welchen Zweigen die Ausführung fortgesetzt wird. Hierzu werden drei Splittypen angeboten: AND (alle Zweige simultan), XOR (nur genau ein Zweig), sowie OR (alle Zweige deren Bedingungen erfüllt sind).
5. **Join:** Verschiedene Zweige werden durch einen Join-Noten wieder zusammengefügt. Es kann definiert werden, ob mit dem Fortfahren gewartet wird, bis alle Zweige den Verbindungsknoten erreicht haben oder ob bereits die Ankunft eines Zweiges ausreicht.
6. **Milestone:** Dieser Knoten ermöglicht es den Ruleflow anzuhalten bis eine bestimmte Bedingungen erfüllt ist. Zum Beispiel bis ein Attribut einen bestimmten Wert annimmt. Sobald diese Bedingung wahr ist, wird die Ausführung fortgesetzt.
7. **Subflow:** Durch die Verwendung eines Subflow können Ruleflows ineinander verschachtelt werden. Ein Subflow stellt einen eigenen Ruleflow dar, der innerhalb

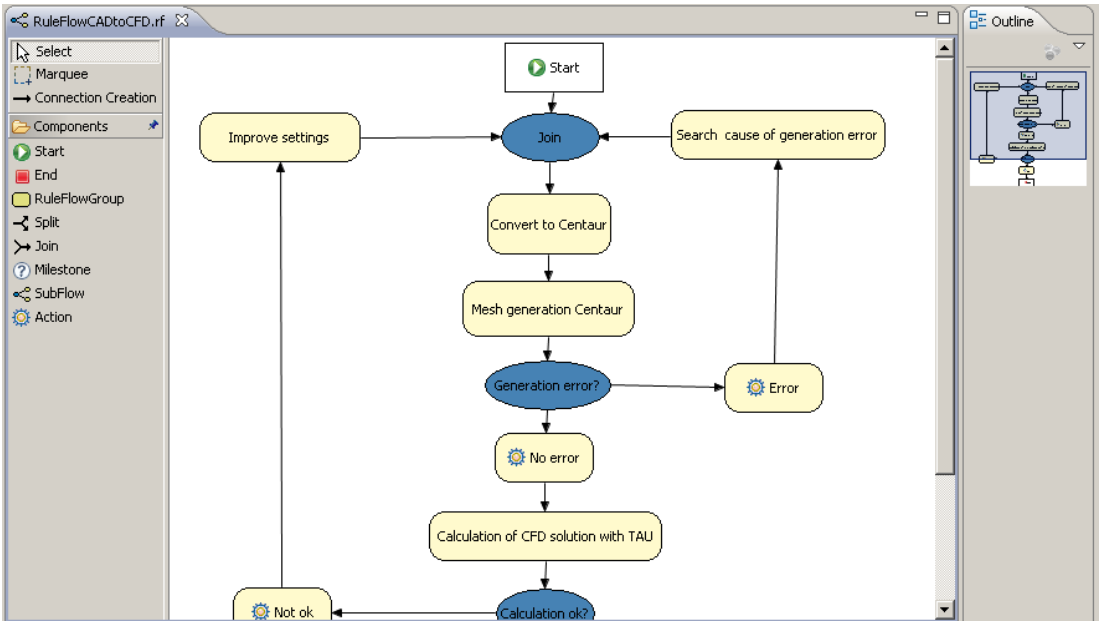

Abbildung 3.4: Beispielhafter Ruleflow in der Eclipse IDE

eines anderen Ruleflows ausgeführt wird. Sobald der Subflow abgearbeitet wurde, fährt der aufrufende Ruleflow mit seiner Arbeit fort.

8. **Action:** Sobald dieser Knoten erreicht ist, ruft dieser Ausführung einer Aktion oder Funktion auf. Er besitzt keine Bedingungen und stellt praktisch gesehen eine Regel ohne Prämisse dar. Aktionsknoten können nicht auf die Objekte des Working Memory zugreifen, sondern nur auf Objekte die im Drools-Kontext explizit als global verfügbar definiert sind.

Für die Modellierung von Ruleflows bietet Drools einen Grafikeditor an. Dieser steht lediglich für das Eclipse Plugin zur Verfügung (siehe Abbildung 3.4). Das JEE-basierte Business Rules Managment System bietet hingegen keinen visuellen Editor an. Intern werden Ruleflows durch zwei XML-Dateien repräsentiert, wobei eine Datei nur das reine Flussdiagramm und dessen Verknüpfungen repräsentiert, die andere die logischen Inhalte der einzelnen Komponenten.

3.2.6 Besondere Eigenschaften von Drools

Drools verfügt über einige weitere erwähnenswerte Eigenschaften, die im Folgenden kurz zusammengefasst werden.

Dem Anwender wird die Möglichkeit geboten Truth Maintenance zu verwenden. Drools ermöglicht dadurch, auch dann die Korrektheit und Konsistenz der gelieferten Lösung zu garantieren, wenn sich Fakten während der Inferenz ändern und beispielsweise bereits aktivierte Regeln wieder ungültig werden. Um dies umzusetzen verwendet

Drools so genannte Shadow Facts, die eine interne Kopie der Faktenobjekte darstellen und außerhalb der Regelmaschine nicht erreichbar sind [PNF+]. Alle Änderungen an einem Faktenobjekt, die während des Inferenzprozesses stattfinden, werden nicht in die Shadow Facts übertragen, sondern existieren nur im originalen Faktenobjekt. Da die Regelmaschine während des Inferenzprozesses nur auf Basis der Shadow Facts arbeitet, ist die am Ende gelieferte Lösung konsistent. Sollen Fakten während des Inferenzprozesses geändert werden, so muss diese explizit der Regelmaschine mitgeteilt werden und diese führt diese Änderungen zu einem geeigneten sicheren Zeitpunkt aus.

Eine weitere Eigenschaft von Drools ist, dem Benutzer des Eclipse Plugin einige Optionen anzubieten, die es erlauben die Regelmaschinen-internen Prozesse anzuzeigen. So ist es möglich, während der Ausführung alle wichtigen Ereignisse (Änderungen an der Faktenbasis, Aktivierungen, Deaktivierungen und Feuern von Regeln) in Textform auszugeben und zu speichern oder sich einen Überblick über die Agenda-Liste zu verschaffen. Der Eclipse Debugmodus wird von Drools dadurch unterstütz, dass eigene Debug-Views für die Inhalte und Struktur der Faktenbasis angeboten werden.

Erwähnenswert ist letztlich noch, dass Drools optional die Java Rule Engine API [Suna] anbietet, die einen Sprachstandard für Java Regelmaschinen spezifiziert. Da dieser Standard den kleinsten gemeinsamen Nenner zwischen verschiedenen Implementierungen definiert, werden nicht alle Fähigkeiten von Drools unterstützt. Ein einfacher Austausch zwischen verschiedenen Regelmaschinen ist trotzdem nicht möglich, da der Standard keine Regelsprache definiert. Der Einsatz wird von den Drools-Entwicklern daher nicht empfohlen [PNF+].

4 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die zu entwerfende Systemarchitektur ermittelt. Die Grundlage hierfür bildeten Interviews mit Experten, potenziellen Anwendern und weiteren Stakeholdern. Das Ergebnis dieser Ermittlung ist eine Anforderungs- und Problemdefinition, sowie die Kenntnis der Einsatzdomäne.

Dieses Kapitel bietet kein vollständiges und strenges Lasten- beziehungsweise Pflichtenheft in der für Industrieprojekte empfohlenen Form, da es ein explizites Ziel der vorliegenden wissenschaftlichen Ausarbeitung ist, zunächst neue Erkenntnisse und Erfahrungen über die generelle Machbarkeit zu sammeln und zu bewerten.

4.1 Einsatzszenario und Ziele

4.1.1 Einsatzdomäne und Expertenwissen

Das zu entwickelnde System soll in erster Linie wissenschaftliche Mitarbeiter, welche Luft- und Raumfahrzeuge entwerfen, bei der Durchführung von computergestützten Simulationen von flugphysikalischen Eigenschaften unterstützen. Um ein grundlegendes Verständnis dieser Domäne zu ermöglichen, wird im Folgenden eine kurze Einführung in diese Domäne gegeben.

Der simulationsbasierte Entwicklungsprozess hat sich in den letzten Jahren zu einem weit verbreiteten Verfahren am DLR Institut für Aerodynamik und Strömungstechnik entwickelt. Insbesondere die Simulation von Strömungsvorgängen anhand von numerischen Berechnung kommt zum Einsatz, wobei das vom DLR entwickelte CFD-Programm (CFD = Computational Fluid Dynamics) TAU verwendet wird [SGH06]. Das Ziel einer numerischen Strömungssimulation ist die Berechnung der Strömungsgrößen Geschwindigkeit, Druck und Temperatur.

Nach Aussage des Instituts [Bec08] liegt die Attraktivität dieser numerischen Simulation von Strömungsvorgängen darin, dass eine deutliche Kostenreduktion im Vergleich zu anderen Verfahren, wie beispielsweise Windkanäle, erzielt wird.

Die Simulation unterteilt sich in die folgenden vier Arbeitsschritte [Ins07]:

1. **Geometriedefinition:** Grundlage der Simulation ist ein Geometriemodell des Flugkörpers, das in der Regel von einem Ingenieur mit Hilfe eines CAD-Programms

(CAD = Computer Aided Design) definiert wurde. Das CAD-Programm und seine Ausgabeformate können variieren.

2. **Netzgenerierung:** In diesem Schritt wird ein numerisches Netz generiert, welches die Oberfläche der CAD-Geometrie umschließt. Dies wird als Diskretisierung bezeichnet, da einzig an den Gitterpunkten des Netzes die Strömungsgrößen berechnet werden. Der Abstand der Netzpunkte ist daher eine wichtige Größe für die Qualität der erzeugten Simulationsergebnisse, da zu grobmaschige Netze die Struktur des Modells gegebenenfalls nicht korrekt abbilden. Der Rechenaufwand steigt mit der Dichte des Netzes an.
3. **Simulation:** Die Strömungssimulation erfolgt beim DLR mit dem CFD-Programm TAU. Die Software besteht aus drei Hauptkomponenten die nacheinander zum Einsatz kommen:
   - Preprocessing: In diesem Schritt wird das Netz für die Simulationsrechnung aufbereitet, was verbesserte Simulationsergebnisse ermöglicht.
   - Strömungslöser: Dieser berechnet die konkrete Strömung auf Basis der Punkte des numerischen Netzes. Es stehen verschiedene Methoden für die Lösung der Strömungsgleichung zur Verfügung.
   - Adaption: Nach der Berechnung der Lösung wird das Netz gegebenenfalls an kritischen Stellen durch Einfügen zusätzlicher Knoten verfeinert.
4. **Visualisierung:** Die erzeugten Simulationsergebnisse liegen abschließend in Form von ASCII-Tabellen vor. Durch Verwendung des Werkzeuges tau2plt können diese konvertiert werden, so dass die Ergebnisse mit Visualisierungstools anschaulich betrachtet werden können.

Es wird deutlich, dass im Rahmen eines simulationsbasierten Entwicklungsprozesses verschiedene Softwarewerkkomponenten konfiguriert und miteinander verknüpft werden. Grundlage der Konfiguration sind insbesondere physikalische Faktoren, unterschiedliche Geometriemodelle, verschiedene Simulationsanforderungen sowie das Zusammenspiel variierender Softwarekomponenten.

Im Rahmen dieser Arbeit wurde eine Expertenbefragung am DLR Institut für Aerodynamik und Strömungstechnik durchgeführt, wobei beispielhaftes Expertenwissen ermittelt und grafisch aufbereitet wurde. Die Abbildungen des Anhangs A stellen den Prozess der Berechnung einer CFD-Lösung mit TAU dar. Die Ausgangsbasis für diese Grafiken bilden Präsentationsfolien. Diese sind auf der CD, welche dieser Arbeit beiliegt, enthalten. Anhand der grafischen Darstellung lässt sich die wesentliche Struktur des Expertenwissens erkennen, wobei deutlich wird, dass das Wissen einerseits einen prozeduralen Charakter aufweist, andererseits anhand verschiedener Regeln Entscheidungen für das weitere Vorgehen getroffen werden. Nach Aussage der Domänenexperten stellen diese Grafiken einen typisch strukturierten Fall dar und können daher als ers-

tes Musterbeispiel für die Wissensmodellierung und die Konzeption verwendet werden. Es ist laut Information der Stakeholder zu erwarten, dass zukünftig zu verwaltendes Expertenwissen, einen stärker regelbasierten Charakter aufweist, da beispielsweise physikalische Randbedingungen geprüft werden sollen.

4.1.2 Grundlegende Ziele

Die im Rahmen dieser Arbeit zu entwickelnde Architektur wird Teil eines Systems, das als zentrale Wissensbasis für den Einsatz numerischer Anwendungen (CFD-Software) am DLR Institut für Aerodynamik und Strömungstechnik dient und sich derzeit in der prototypischen Konzeption und Entwicklung befindet. Das Gesamtsystem soll insbesondere die vorhandenen Kenntnisse im Umgang mit numerischen Anwendungen verwalten und wiedergeben können. Die Wissensbasis enthält hierbei sowohl Erfahrungen und vorgehensspezifisches Wissen von Experten für numerische Anwendung, als auch eine Sammlung von relevanten Dokumenten. Die Software unterstützt den Anwender daher begleitend beim Erstellen, Durchführen und Verifizieren von Simulationsszenarien, und dient gleichzeitig als Nachschlagewerk. Das zu entwickelnde System besitzt somit die Eigenschaften eines Expertensystems zur Anwenderunterstützung, als auch die Eigenschaften einer zentralen Wissenssammlung. Das System soll nicht mit externen Systemen, wie etwa der Simulationssoftware oder dem CAD-Programm, kommunizieren, sondern wird vom Anwender begleitend und unabhängig verwendet.

Laut Aussage des DLR sind standardisierte Vorgehensweisen eine wichtige Anforderung bei der industriellen Anwendung numerischer Simulationsverfahren, da hierdurch die Vergleichbarkeit von Ergebnissen unterschiedlicher Simulationen sichergestellt werden kann. Das zu entwickelnde System soll hierzu einen Beitrag leisten.

Im Rahmen einer Anforderungsanalyse wurden die folgenden grundlegenden Ziele, welche das System erfüllen soll, ermittelt:

- Gewährleistung von fehlerfreien, konsistenten und hochwertigen Simulationsergebnissen
- Bei unerfahrenen Anwendern soll primär eine Reduktion von Unsicherheiten und Vermeidung von Fehlern erzielt werden. Darüber hinaus soll sekundär ein Lerneffekt eintreten.
- Bei erfahrenen Anwendern sollen bereits bekannte Fehler vermieden werden und eine Unterstützung bei wiederkehrenden Herausforderungen geboten werden.
- Höhere Effizienz der eingesetzten Ressourcen und hierdurch geringere Kosten
- Zeitlich unbeschränkte Sicherung und Nutzbarmachung des vorhandenen Expertenwissens.

Um diese Ziele zu erreichen, wurden von den Stakeholdern die folgenden Kernfunktionen des Systems definiert:

- Die Software stellt die zentrale Anlaufstelle für die Unterstützung von CFD-Anwender dar und dient diesen als Hilfe und Dokumentation. Das im DLR vorhandene Wissen zu einem Problem wird durch die Software strukturiert bereit gestellt und die Anwender werden bei der Lösung von Problemen unterstützt.
- Neben dem Zugriff auf vorhandenes Wissen bietet das System auch die Möglichkeit, neues Wissen strukturiert in das System einzupflegen, und somit Wissen einzelner Personen für das DLR zu sichern.
- Da das einzupflegende Wissen teilweise sensible Daten beinhaltet, muss ein Sicherheits- und Rechtekonzept existieren, das es erlaubt, den Zugriff auf die im System befindlichen Daten zu kontrollieren und diese zu verwalten. Hierzu wird eine Benutzerverwaltung benötigt.

Im Rahmen des Gesamtprojekts wird die Wissensbasis des zu entwickelnden Systems zunächst auf den Bereich des DLR Strömungslösers TAU eingeschränkt. Das System soll jedoch bewusst für eine Erweiterung auf andere Wissensgebiete offen gehalten werden. Aktuelles Ziel des Gesamtprojekts ist zunächst die Konzeption, Implementierung und Evaluation eines Prototyps, welcher die grundlegenden Funktionalitäten darstellt und als Diskussionsgrundlage für eine endgültige Implementierung dienen soll.

4.2 Einordnung und Abgrenzung

Die vorliegende Arbeit befasst sich mit dem zentralen Teilbereich des Gesamtsystems und entwickelt die Systemarchitektur des Expertensystems. Das Expertensystem muss sich in die Gesamtarchitektur einfügen und übernimmt dabei eine vermittelnde Funktion zwischen verschiedenen Komponenten. Abbildung 4.1 zeigt die gemeinsam im Team entworfene Architektur des Gesamtsystems auf. Die im Rahmen dieser Arbeit zu entwickelnde Komponente wird als „Core Expert Layer“ bezeichnet. Die weiteren dargestellten Systemkomponenten sind nicht direkter Bestandteil dieser Arbeit. Dieses sind:

- **Tool Layer**: Diese Komponente bietet eine Schnittstelle zum zentralen Active Directory- und LDAP-Dienst an, auf welchem die Rechte- und Benutzerverwaltung des Gesamtsystems aufbaut. Diese sieht vor, dass abgelegtes Wissen nur für bestimmte Benutzergruppen freigegeben werden kann. Darüber hinaus übernimmt das Tool Layer bei Bedarf das Management von einzelnen Sessions. Zum Zeitpunkt der Erstellung dieser Arbeit ist das Tool Layer noch nicht fertig gestellt. Entsprechende Komponenten und Anforderungen werden in dieser Arbeit nur am Rande betrachtet, die Rechte- und Benutzerverwaltung werden in der Implementierung vernachlässigt.

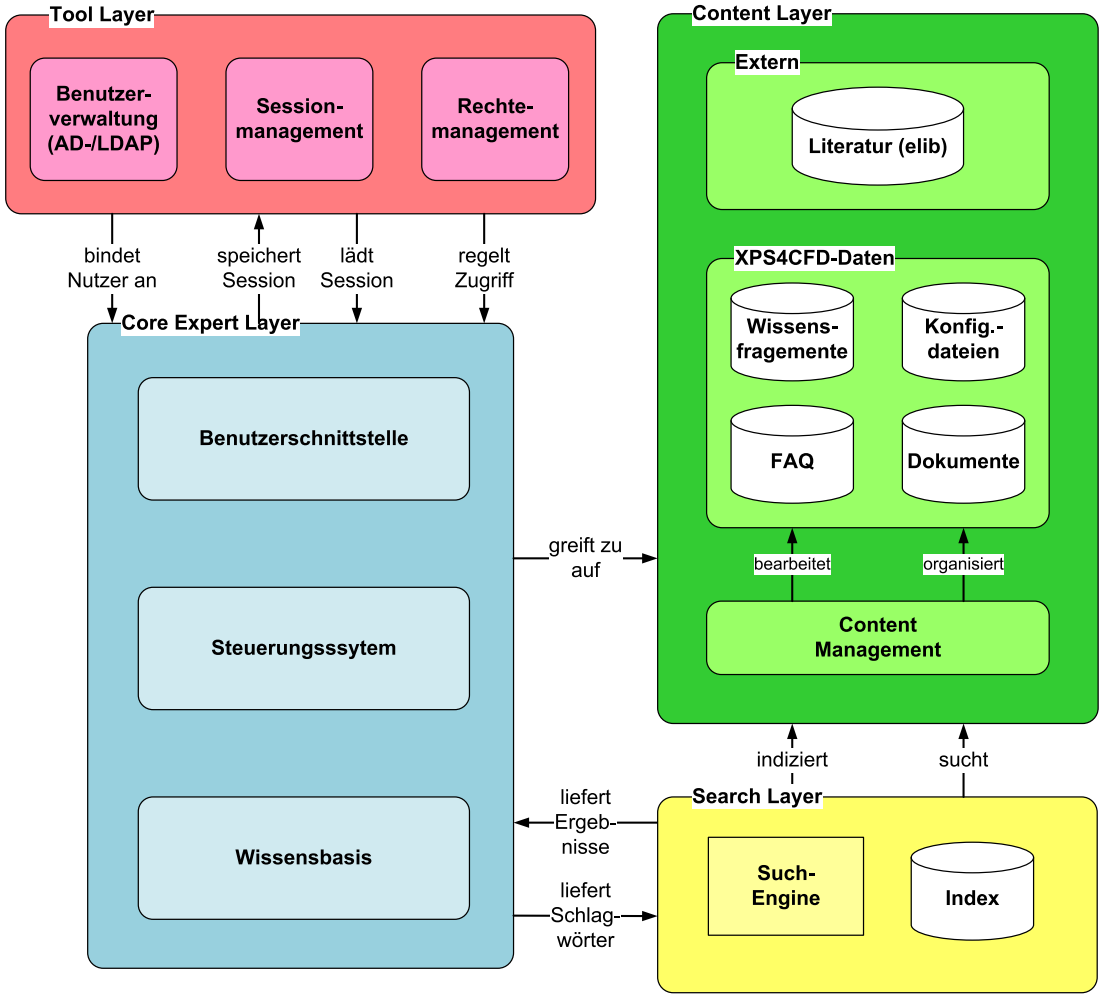

Abbildung 4.1: Überblick über das Gesamtsystem. Im Rahmen dieser Arbeit wird das Core Expert Layer umgesetzt.

- **Content Layer**: Diese ist die zentrale Wissenssammlung des Systems und dient dem Speichern verschiedener Inhalte. Hierzu gehört Wissen in natürlichsprachlicher Form, Dokumentationen, Konfigurationsdateien, sowie Expertensystembezogenes Wissens wie Anwendungsszenarien und Regeln. Das Content Layer bietet darüber hinaus ein System für eine dateibasierte Wissensverwaltung an. Grundlage dieser Verwaltung wird die vom DLR entwickelte Software Datafinder [Deu] bilden. Des Weiteren wird eine relationale Datenbank in das Content Layer integriert, in welcher die für das Expertensystem relevanten Daten abgelegt werden. Da sich der Content Layer derzeit in Entwicklung befindet, werden für den Prototypen alle Daten im lokalen Dateisystem abgelegt.
- **Search Layer**: Das Search Layer umfasst einen Suchdienst für die Inhalte der Wissensbasis und der Literaturdatenbank des DLR. Die Suche kann anhand von Begriffen oder komplexen Anfragen spezifiziert werden. Das Search Layer verwendet die freie Suchmaschine Apache Lucene [Apa]. Die Ergebnisse der Suche sollen später vom Search Layer in aufbereiteter Form weitergeleitet werden können, so dass diese Ergebnisse direkt in den Expertensystemkern eingebunden werden. In dieser Arbeit stellt das Search Layer einen entsprechenden Dienst zur Verfügung und bietet für das Betrachten der Ergebnisse eine eigene Benutzerschnittstelle in Form eines Eclipse Plugins.

Primäres Ziel dieser Arbeit ist die Entwicklung eines Architekturkonzepts für die zentrale Expertensystemkomponente des Gesamtsystems. Anschließend wird ein Prototyp implementiert, der die wichtigsten Funktionen und Konzepte umfasst und hierdurch die grundlegende Machbarkeit aufzeigt.

4.3 Funktionale und nichtfunktionale Anforderungen

Aus den geplanten Anwendungsbereichen und der Befragung der zukünftigen Anwender werden die Anforderungen für das im Rahmen dieser Arbeit zu entwickelnde System abgeleitet. Hierzu wurde jeweils ein Workshop für Experten und zukünftige Endanwender am DLR Institut für Aerodynamik und Strömungstechnik in Braunschweig abgehalten und Interviews mit weiteren Stakholdern geführt.

4.3.1 Funktionale Anforderungen

1. Das System muss dem Benutzer das im DLR vorhandene Wissen für die Anwendung flugphysikalischer Simulationen strukturiert zur Verfügung stellen.
2. Wissen muss in Form von Szenarien gesammelt und wiedergegeben werden können. Ein Szenario umfasst einen konkreten Anwendungsfall der CFD-Software oder bildet die Lösung eines domänenspezifischen Problems ab.

3. Szenarien müssen mit der Suchmaschine des Search Layer kommunizieren können und die gefundenen Elemente darstellen können.
4. Szenarien müssen Daten des Content Layer verwenden und einbinden können. Dies sind beispielsweise Texte oder Grafiken.
5. Szenarien müssen generell die Möglichkeit bieten externe Funktionen und Programme aufzurufen.
6. Der Endanwender kann die Szenarien gezielt anhand von Infotexten auswählen.
7. Für den Endanwender muss die angebotene Lösung nachvollziehbar sein. Ihm soll deutlich werden, welche Entscheidungen vom System getroffen wurden, wann diese geschah und warum.
8. Der Endanwender kann ein Szenario an einer gewünschten Stelle speichern, und später an dieser Stelle fortfahren.
9. Bestehende Szenarien und deren Elemente müssen editierbar und neue Szenarien erzeugbar sein, so dass das System um strukturiertes neues Wissen erweitert werden kann.
10. Änderungen und Erweiterungen des Wissens müssen einen Release-Prozess durchlaufen. Hierzu ist das Release Management des Tool Layer zu verwenden.
11. Bei allen Endanwenderaktionen sind das Rechtemanagement und die Benutzerverwaltung des Tool Layer zu verwenden.
12. Es muss die Möglichkeit bestehen, Informationen in verschiedenen Sprachen anzubieten. Die primäre Sprache ist hierbei Englisch, ergänzend soll Text in Deutsch angeboten werden können.

4.3.2 Nichtfunktionale Anforderungen

1. Die Erweiterung der Wissensbasis muss von einer geschulten Person, die keine tiefgehenden Programmierkenntnisse besitzt, in angemessener Zeit durchgeführt werden können.
2. Das Gesamtsystem soll als Arbeitsmittel in der wissenschaftlichen Forschung sowie bei industriellen Partnern eingesetzt werden. Dabei ist von weitestgehend heterogenen Rechner- und Softwarelandschaften auszugehen ist. Dies führt dazu, dass der Einsatz der zu entwickelnden Software somit auf den gängigen UNIX-Plattformen sowie unter Microsoft Windows möglich sein muss.
3. Es existiert eine klare Definition des Prozesses zur Wissenspflege, so dass eine Qualitätssicherung ermöglicht wird.

4.3.3 Randbedingungen

1. Das Expertensystem wird auf Basis der Regelmaschine JBoss Drools 4.0.7 konzipiert und aufgebaut.
2. Als Programmiersprache ist Java in Version 6 zu verwenden.
3. Grundlage der Entwicklung bildet die Eclipse Rich-Client-Platform 3.3. Das Expertensystem soll auf Basis dieses Frameworks konzipiert werden.

Die oben beschriebenen Anforderungen sind in dieser Form den Stakeholdern vorgelegt und als geeignet akzeptiert worden. Im Rahmen dieser Arbeit sollen nur die wesentlichen Kernanforderungen umgesetzt werden. Es wurde von allen Beteiligten übereinstimmend erkannt, dass auf Grund der zeitlichen Beschränkung eine vollständige Konzeption und prototypische Umsetzung aller Anforderungen nicht zu erreichen ist. Ziel dieser Arbeit ist daher zunächst der Nachweis der Machbarkeit und das Sammeln von Erfahrungen anhand eines Grundkonzepts. Aufbauend auf dieser Arbeit sollen später weitere Konzepte integriert werden. Insbesondere die Aspekte, die den Veröffentlichungsprozess und die Rechteverwaltung betreffen werden in der vorliegenden Arbeit weitestgehend ignoriert. Diese Entscheidung ist damit zu begründen, dass es sich hierbei um keine Kernfunktionen des Expertensystems handelt, sondern um Komponenten für das Management von Freigaben, wie es auch häufig in Content Management Systemen oder Wikis vorzufinden ist. Des Weiteren müssen diese Komponenten eng mit den anderen Layern des Gesamtsystems, insbesondere dem Tool Layer [vgl. 4.2], zusammenarbeiten, was auf Grund des aktuellen Entwicklungsstands dieser Komponenten problematisch ist.

4.4 Theoretische Anforderungen an Expertensysteme

Neben den spezifischen Anforderungen für den vorliegenden konkreten Einsatz, werden an dieser Stelle abschließend die allgemeinen theoretischen Eigenschaften aufgezählt, die ein optimales Expertensystem aufweisen sollte. [FS90] nennt hierbei die folgenden Eigenschaften, welche auch als ein Gradmesser für die Güte des Systems verwendet werden können. Demnach soll ein Expertensystem:

1. Wissen eines oder mehrerer Experten zur Lösung von Problemen in begrenzten Domänen anwenden.
2. das Expertenwissen explizit und möglichst deklarativ darstellen.
3. das Einpflegen des Expertenwissens in das System möglichst gut unterstützen.
4. die Wartbarkeit und Pflege des Expertenwissens im System möglichst gut unterstützen.
5. das Wissen in einfacher und verständlicher Form präsentieren.

6. die Verwendung unsicheren Wissens unterstützen.
7. mit dem Benutzer über eine möglichst intuitive und anschauliche Schnittstelle interagieren.
8. die gelieferten Ergebnisse und Lösungen dem Benutzer begründen und erklären.
9. das fallspezifische Faktenwissen und die Heuristiken zur Problemlösung trennen.
10. die Widerverwendbarkeit von eingepflegtem Wissen in verwandten Domänen ermöglichen.

[FS90] merkte an, dass ihm zum Zeitpunkt der Veröffentlichung kein Expertensystem bekannt war, dass alle genannten Eigenschaften aufweist. Auch wenn gegebenenfalls in der Zwischenzeit ein solches System entwickelt worden sein sollte, so zeigt diese Aussage, welche Herausforderung die vollständige Umsetzung dieser Eigenschaften darstellt. Das hier prototypisch entwickelte System sollte daher auch nur als ein Versuch der Annäherung an diese theoretischen Anforderungen verstanden werden.

4.5 Akteure des Systems

Vor der weiteren Konzeption des Systems werden im Folgenden die mit dem System interagierenden Akteure definiert. Die Basis bildeten die zuvor ermittelten Anforderungen, auf Grund derer eine Aufteilung der Akteure in verschiedene Rollen vorgenommen wird. Eine natürliche Person kann, jeweils abhängig von der momentanen Tätigkeit, mehrere Rolle annehmen.

Endanwender

Die Software wendet sich primär an wissenschaftliche Mitarbeiter, die im CFD-Umfeld tätig sind. Diese Endanwender lassen sich grob in zwei Gruppen einteilen. Zum Einem gibt es unerfahrene Anwender von CFD-Software, die durch das zu entwickelnde System geschult werden um neues Wissen zu erlangen. Die andere Zielgruppe sind erfahrene Anwender, die mit der CFD-Software prinzipiell vertraut sind, aber bei besonders spezifischen oder selten auftretenden Problemstellungen unterstützt werden sollen.

Diese beiden Gruppen werden aus Sicht des Systems zunächst nicht weiter unterschieden, jedoch sollten die angebotenen Informationen so gekennzeichnet sein, dass jeder Anwendertyp das für ihn relevante Wissen zügig entdecken kann.

Domänen-Experten

Dies sind Personen, die über Expertenwissen in der CFD-Domäne verfügen und deren Kenntnisse in dem Expertensystem hinterlegt werden soll. Die Rolle des Domänen-

Experten sieht vor, dass diese Personen über keine umfangreichen technischen Kenntnisse zur Pflege und Erweiterung des Expertensystems verfügen. Domänen-Experten agieren bei der Wissenspflege nicht direkt mit dem System, sondern mittels eines Wissensingenieurs. Daher ist diese Rolle innerhalb des Expertensystems nicht vorhanden.

Wissensingenieure

Diese Rolle übernehmen Personen, die das Expertenwissen vom Domänen-Experten akquiriert und in das Expertensystem einpflegen. Es wird angenommen, dass Personen, die diese Rolle ausführen, umfangreiche Kenntnisse im Umgang mit der Expertensystemkomponente zum Wissenserwerb besitzen. Darüber hinaus verfügen Wissensingenieure über ein fundiertes Verständnis für das zu pflegende Domänenwissen.

Es ist davon auszugehen, dass Wissensingenieure Programmierkenntnisse in Java besitzen und in der Lage sind, selbstständig kleinere Softwarekomponenten zu entwickeln und diese in eine vordefinierte Expertensystemarchitektur integrieren können. Darüber hinaus sind diese mit den Grundkenntnisse von JBoss Drools vertraut und in der Lage Regeln und Ruleflows zu definieren. Die Fähigkeit zur Erstellung von XML-Dateien auf Basis vordefinierter XML-Schema kann vorausgesetzt werden. Entsprechend geschulte Dömanen-Experten können selbst die Rolle des Wissensingenieurs übernehmen und ihr Wissen selbstständig einpflegen. Wissensingenieure werden im Folgenden auch als Knowledge-Engineers bezeichnet.

Release-Manager

Personen, die das im System hinterlegte Wissen verwalten übernehmen, die Rolle des Release-Managers. Seine Aufgabe ist es, Änderungen und Erweiterungen, des in Expertensystem abgelegten Expertenwissens, zentral freizugeben oder abzulehnen. Darüber hinaus sieht diese Rolle die Rechteverwaltung des Wissens für bestimmte Benutzergruppen vor. Diese Rolle ist stark mit dem Tool Layer des Gesamtsystems verbunden, das zum Zeitpunkt der Erstellung dieser Arbeit noch nicht verfügbar ist. Daher wird, im Rahmen der vorliegenden Arbeit, diese Rolle bei der praktischen Umsetzung nicht beachtet.

4.6 Anwendungsfälle des Systems

Auf Basis der Anforderungen an das System und den vorher dargelegten Rahmenbedingungen, werden Anwendungsfälle definiert. Diese spezifizieren die Funktionalität des Systems und die Aktionen der einzelnen Rollen. Auf Grund des prototypischen Charakters des zu entwickelnden Systems werden zunächst nur die wesentlichen Funktionen beschrieben.

Die Anwendungsfälle der Dömanen-Experten liegen außerhalb des Expertensystems und werden daher an dieser Stelle nicht betrachtet. Dömanen-Experten treten lediglich als Informanten für die Wissensingenieure auf. Es ist Aufgabe der Wissensingenieure die Anwendungsfälle und hierzu gehörige Methoden zu ermitteln, die es ihnen erlauben das Wissen der Domänen-Experten korrekt, vollständig und strukturiert zu erfassen.

4.6.1 Anwendungsfälle für Endanwender

Szenario auswählen und ausführen

Zusammenfassung: Der Endanwender wählt ein vordefiniertes Szenario aus, welches sein Problem abbildet. Bei der Ausführung des Szenarios wird er interaktiv geführt.
Akteure: Endanwender
Vorbedingungen: Der Benutzer ist als Endanwender am System angemeldet
Beschreibung:

1. Der Benutzer wählt Button „Neues Szenario starten“.
2. Der Benutzer erhält eine kategorisierte Liste von vorgegebenen Szenarien inklusive einer kurzen Beschreibung jedes Szenarios.
3. Der Benutzer wählt das vorgegebene Szenario, welches seinem Problem entspricht.
4. Das System initialisiert und startet das Szenario.
5. Der Benutzer wird interaktiv und schrittweise durch das Szenario geführt. Ihm werden bei jedem Schritt Hinweise und Erklärungen angezeigt.
6. Gegebenenfalls ist zur korrekten Bestimmung des weiteren Ablaufs, die Eingabe von Informationen (fallbezogene Fakten) durch den Nutzer nötig ist. In diesem Fall werden ein oder mehrere entsprechende Interview-Formulare angezeigt und vom Benutzer ausgefüllt. Die Interview-Formularfelder sind der Fragestellung angepasst und können Multiple-Choice-, Zahlen- oder Freitextfelder umfassen.
7. Anhand der Eingaben wird der weitere Ablauf definiert und das Szenario bis zur nächsten nötigen Benutzereingabe oder bis zum Ende vorgesetzt.
8. Während des gesamten Prozesses soll dem Endanwender nachvollziehbar erklärt werden, welche Entscheidungen, wann und warum vom System getroffen wurden.

Ausnahmen:

- Ungültige Eingaben im Interviewformular: Diese sollten bereits durch die Definition der Eingabefelder vermieden werden, beispielsweise sollten eher ein Multiple-Choice-Element verwendet werden als ein Freitext-Eingabefeld. Wenn trotzdem fehlerhafte Eingaben erfolgen, müssen diese vom System erkannt und der Benutzer mit einem entsprechenden Hinweis zur erneuten Eingabe aufgefordert werden.

Nachbedingungen: Der Benutzer hat das Szenario durchgespielt. Er kann das Szenario inklusive seiner Eingaben abspeichern.

Szenario speichern und laden

Zusammenfassung: Der Benutzer speichert sein aktuelles Szenario inklusive seiner Eingaben ab. Später lädt er dies und setzt es fort.
Akteure: Endanwender
Vorbedingungen: Der Benutzer ist als Endanwender am System angemeldet. Er befindet sich in einem Szenario.
Beschreibung:

1. Der Benutzer wählt den Button „Szenario speichern“.
2. Es erscheint ein Dialog-Formular, das den Benutzer zur Eingabe des Speicherpfads und des Namens der Datei auffordert.
3. Das System speichert den derzeitigen Zustand des Szenarios, inklusive der Benutzereingaben in einer Datei.
4. Der Benutzer beendet das Szenario und verlässt das Expertensystem.
5. Der Benutzer kehrt zu einem späteren Zeitpunkt zum Expertensystem zurück und meldet sich wieder als Endanwender an. Er möchte ein Szenario laden und wählt den Button „Szenario laden“. Diese Option steht dem Benutzer jederzeit zur Verfügung.
6. Dem Benutzer wird ein Dialog-Formular angezeigt, das ihn zur Eingabe des Datenpfades und des Dateinamens auffordert.
7. Der Benutzer füllt die entsprechenden Felder aus und definiert hiermit das zu ladende Szenario. Es kann sich um ein von ihm selbst erstelltes Szenario handeln oder um ein Szenario, das ihm von einem anderen Benutzer zur Verfügung gestellt wurde.
8. Das Expertensystem öffnet die Datei und stellt das abgespeicherte Szenario wieder her.

Ausnahmen:

- Der Zeitpunkt des Speicherns ist ungültig: Der Benutzer möchte an ungültiger Stelle abspeichern, beispielsweise vor der Initialisierung eines Szenarios. Der Button „Szenario speichern“ ist an diesen Stellen nicht auswählbar oder es wird eine entsprechende Meldung angezeigt.
- Der Dateipfad ist ungültig: Das System weist den Benutzer daraufhin, dass der Pfad ungültig ist und die Datei nicht gespeichert beziehungsweise geladen werden kann.

Nachbedingungen: Es ist dem Benutzer möglich mit dem Szenario, an der Stelle an der gespeichert wurde, fortzufahren.

4.6.2 Anwendungsfälle für Wissensingenieure

Neues Szenario erstellen

Zusammenfassung: Ein vollständig neues Szenario wird vom Wissensingenieur erstellt.
Akteure: Wissensingenieur
Vorbedingungen: Das szenariospezifische Wissen des Domänen-Experten wurde erfasst und liegt dem Wissensingenieur vor. Die notwendigen externen Wissenskomponenten sind in das Content-Layer eingepflegt. Der Benutzer ist als Wissensingenieur im Expertensystem angemeldet.
Beschreibung:

1. Der Wissensingenieur wählt den Button „Szenario anlegen oder editieren“.
2. Der Wissensingenieur verwendet, die ihm bereitgestellten Ansichten und Editoren innerhalb seiner Eclipse-Perspektive, um eine neues Szenario zu erstellen. Er erzeugt Regeln, Ruleflows und Faktenobjekte sowie die anzuzeigende Informations- und Interviewseiten.
3. Die erzeugten Szenarioelemente werden vom Wissensingenieur gemäß der Spezifikation in die Expertensystemarchitektur integriert.
4. Das fertige Szenario wird von dem Wissensingenieur in ein zentrales Repository übertragen, so dass der Release-Prozess für neues Wissen angestoßen wird.

Nachbedingungen: Das neue Szenario ist angelegt und wird an den Release-Manager weitergereicht. Dieser kann das neue Szenario für die Nutzung durch Endanwender freigeben.

Bestehendes Szenario editieren

Zusammenfassung: Ein bereits bestehendes Szenario wird durch den Wissensingenieur verändert.
Akteure: Wissensingenieur
Vorbedingungen: Es existiert ein bestehendes Szenario, das verändert werden soll. Die zu diesem Szenario gehörenden Dateien befinden sich bereits lokal im System des Wissensingenieurs. Der Wissensingenieur besitzt das szenariospezifische Wissen um die Änderungen sinnvoll durchzuführen. Der Benutzer ist als Wissensingenieur angemeldet.
Beschreibung:

1. Der Wissensingenieur wählt den Button „Szenario anlegen oder editieren“.

2. Die lokal gespeicherten Dateien des zu verändernden Szenarios, können vom Wissensingenieur in einer speziellen Ansicht durchsucht und ausgewählt werden.
3. Der Wissensingenieur verwendet, die ihm bereitgestellten Ansichten und Editoren innerhalb der Eclipse-Perspektive des Expertensystems, um die bestehenden Dateien des Szenarios zu editieren oder neue Elemente zu erstellen.
4. Das editierte Szenario wird vom Wissensingenieur in ein zentrales Repository übertragen, so dass der Release-Prozess für verändertes Wissen angestoßen wird.

Nachbedingungen: Das Szenario wurde ediert und wird an den Release-Manager weitergereicht, der die Änderungen freischalten kann.

5 Konzeption und Architekturentwurf

Im folgenden Kapitel wird die Konzeption des entwickelten Systems dargelegt. Ziel ist der grundlegende Entwurf eines Systems, das in der Lage ist, die gestellten Anforderungen zu erfüllen. Insbesondere soll dem Wissensingenieur eine Architektur zur Verfügung gestellt werden, die es ihm erlaubt, auf einfache Art verschiedenstes Expertenwissen vollständig abzubilden und dem Endanwender zugänglich zu machen.

Detaillierte technische Aspekte werden in dieser Darlegung der Konzeption, zugunsten der Klarheit, zunächst vernachlässigt und erst in den folgenden Kapiteln berücksichtigt.

5.1 Entwurf einer Wissensrepräsentation

Zentraler Kern des Expertensystems ist die Repräsentation von Expertenwissen. Der erste Schritt der Konzeption ist daher der Entwurf einer geeigneten Wissensrepräsentation. Anschließend werden die weiteren Komponenten und Funktionen auf Basis dieses Konzepts aufgebaut.

Anhand der Anforderungsanalyse wird deutlich, dass es das entscheidende Ziel, des zu entwickelnden Systems, ist, Expertenwissen im Bereich von flugphysikalischen Simulationen zur Verfügung zu stellen. Das System muss in der Lage sein, dieses Wissen strukturiert abzubilden und dem Anwender in geeigneter Form zur Verfügung zu stellen. Es ist zu beachten, dass es möglich sein muss, auf zugängliche Weise neues Wissen einzupflegen und vorhandenes zu editieren. Darüber hinaus soll das Expertensystem gegebenenfalls später in anderen Einsatzdomänen verwendet werden. Da es im Vorfeld nicht ersichtlich ist, welche Struktur das Wissen in diesen zukünftigen Einsatzdomänen besitzt, muss die Wissensrepräsentation so konzipiert werden, dass eine Erweiterung auf andere Domänen mit hoher Wahrscheinlichkeit möglich ist.

Die Herausforderung besteht darin, eine Wissensrepräsentation zu konzipieren, die folgende Punkte erfüllt:

1. Vollständige und strukturierte Wissensabbildung
2. Einfache Wissenspflege
3. Potenzielle Erweiterbarkeit auf andere Domänen

5.1.1 Abbildung der Lösungsstrategie

Als Ausgangspunkt der Konzeption dient das im Rahmen der Anforderungsermittlung gesammelte beispielhafte Expertenwissen. Dieses Wissen wurde von den Experten als Folienpräsentationen zur Verfügung gestellt, die für die Schulung neuer Mitarbeiter verwendet werden. Das Wissen liegt daher im Wesentlichen in Form von Stichpunkten vor, die Anweisungen zum Vorgehen bei Simulationen darstellen. Diese Folien stellen jedoch keine explizit formulierte Regelsammlung bereit und bieten keine visuelle Darstellung des Prozesses. Die Sammlung, der zur Verfügung gestellten Folien, befindet sich auf der CD, die dieser Arbeit beiliegt.

Um das Wissen korrekt abbilden zu können, muss ein Verständnis über dessen Charakteristik geschaffen werden. Bei der Analyse des Expertenwissens wird deutlich, dass dieses einen schrittweisen Ablauf beschreibt. Dieser Ablauf führt, ausgehend von einer CAD-Konstruktion, zu einem mit TAU erzeugten CFD-Simulationsergebnis. Im Rahmen dieser Arbeit zeigte sich, dass dieser Ablauf durch Flussdiagramme, in Form der Business Process Modeling Notation (BPMN) [Obj], in sehr geeigneter Form repräsentiert werden kann. Die erstellten Diagramme sind im Anhang A zu finden und stellen den allgemeinen Prozess dar, enthalten jedoch keine detaillierten Anweisungen oder zusätzliche Erklärungen zu den Prozessen. Das hierdurch abgebildete Expertenwissen kann in der Form verstanden werden, dass es lediglich die reine, vom Experten definierte, Strategie der Problemlösung abbildet. Das in den Diagrammen abgebildete Wissen enthält keine expliziten Zusatzinformationen, Anweisungen oder Erklärungen.

Das Ergebnis der Wissensumformung kann als äußerst zufrieden stellend bezeichnet werden, da die zuvor eher unstrukturierten Informationen zu einem Fluss zusammengefasst werden konnten, der den Simulationsprozess sehr anschaulich darlegt. Die Experten bestätigten die Korrektheit der erstellten Diagramme und alle Stakeholder nannten die gewählte Darstellungsform sehr ansprechend und gut geeignet, um die Abläufe abzubilden. Es ist jedoch zu beachten, dass neben den reinen Prozessinformationen noch ergänzende und erklärende Informationen und Anweisungen festgehalten werden müssen.

Ausgehend von dem Erfolg der erstellten Flussdiagramme, wird als nächstes eine Möglichkeit entwickelt, um das Wissen in dem Expertensystem abzulegen. Dabei sind, wie gefordert, die Funktionen von JBoss Drools zu verwenden. Bei Betrachtung der Fähigkeiten von Drools wird deutlich, dass die Ruleflow-Komponente besonders geeignet für die Wissensabbildung ist, da diese eine ähnliche Strukturierung der Abläufe wie die BPMN erlaubt. Testweise Abbildungen der Flussdiagramme in Ruleflows bestätigen dies.

Bei weiterer Analyse der Beispielszenarien wird deutlich, dass die in den Diagrammen enthaltenen Gabelpunkte, also die Entscheidungspunkte für den weiteren Verlauf,

die eigentlichen Regeln enthalten. Diese Regeln entsprechen vollständig der Definition für Regelsysteme und operieren auf fallbezogenen Fakten. So lässt sich etwa die erste Gabelung aus Abbildung 8.1.(unten) durch die folgenden Regeln abbilden:

Wenn *das CAD-Format Catia ist*
Dann *biete die Catia Optionen an*

Wenn *das CAD-Format Iges ist*
Dann *biete die Iges Optionen an*

Diese logisch zusammenhängenden Regeln können zu einer Ruleflow-Gruppe zusammengefasst werden, so dass diese an entsprechender Stelle gemeinsam ausgewertet werden. Dabei operieren die Regeln auf einem fallbasierten Fakt, das das CAD-Format repräsentiert.

Bei der Betrachtung des beispielhaften Expertenwissens wird deutlich, dass die Abbildung nur verhältnismäßig wenige Regeln erfordert. Diese werden für die Definition des weiteren Vorgehens an den Vergabelungen benötigt. Alternativ können diese im Ruleflow, anstatt durch Regelgruppen-Knoten, durch spezielle Split-Knoten ersetzt werden, die das weitere Fortschreiten anhand von vordefinierten Bedingungen entscheiden. Dennoch fokussiert sich das im Folgenden entwickelte Konzept auf die Verwendung von Regeln und Regelgruppen, wobei vom Wissensingenieur frei entschieden werden kann, welche Modellierungsform er wählt. Die Fokussierung auf Regelgruppen, die eher für den Einsatz bei umfangreichen Regelsammlungen angebracht ist, ist dennoch gerechtfertigt. Laut Aussage der Stakeholder wird zukünftig einzupflegendes Expertenwissen eine wesentlich größere Regelsammlung umfassen und auch Regeln enthalten, die sich häufig ändern. Unter Beachtung der bereits diskutierten Eigenschaften von regelbasierter Wissensrepräsentation, wird erwartet, dass diese Abbildungsform für das zukünftige Expertenwissen zu bevorzugen ist. Obwohl dieses Expertenwissen zum Zeitpunkt der Erstellung dieser Arbeit nicht zur Verfügung steht, wird deren zu erwartende Beschaffenheit in der folgenden Konzeption mitberücksichtigt, um eine gute Erweiterbarkeit des Systems für zukünftige Wissensdomänen zu gewährleisten.

Abbildung 5.1 zeigt beispielhaft die Modellierung des ersten Prozesses des Beispielwissens anhand der Drools Ruleflow-Komponente. Die folgenden Prozesse sind strukturell sehr ähnlich aufgebaut, so dass einmal gewonnene Erkenntnisse übertragen werden können. Bei der gewählten Modellierung werden alle Aktionen (bspw. Anzeigen von Informationen) und Entscheidungen von den Regelgruppen angestoßen, nicht von Aktions- und Splitknoten. Die Ruleflow-Komponente dient nur der Anordnung der Regelgruppen. Je nach Bedarf kann der Wissensingenieur auch andere Funktionen der Ruleflow-Komponente einbauen, so dass etwa eine Entscheidung über das weitere Vor-

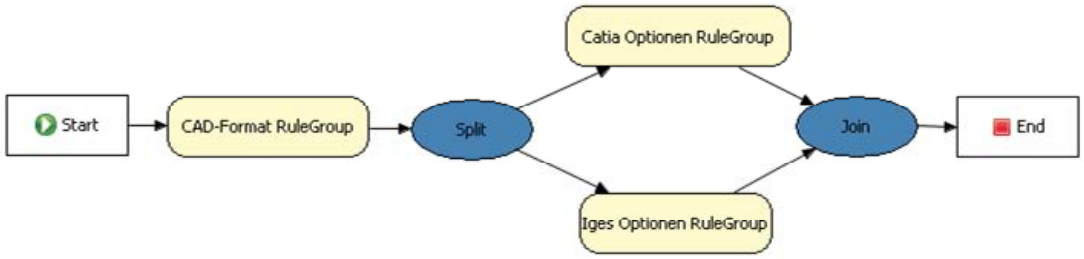

Abbildung 5.1: Drools Ruleflow-Modellierung des ersten Teils des Beispielwissens (vgl. Anhang A Abbildung 8.1.(unten)), der die Konvertierung von einem CAD-Format zu einem Centaur-Format beschreibt. Alle Aktionen und Entscheidungen werden in der gewählten Umsetzung durch Regeln ausgelöst, nicht durch Aktions- oder Splitknoten.

gehen anhand der Bedingungen eines Split-Knoten getroffen wird, nicht durch eine Regelgruppe. Die gewählte Modellierung des Ruleflows ist nur eine generelle Möglichkeit der Abbildung des Expertenwissens, ohne Beachtung der im Folgenden entwickelten Konzepte.

Ein in sich abgeschlossener Prozess ist als Szenario definiert. Ein Szenario repräsentiert einen Ablauf, der zur Lösung einer bestimmten Aufgabe führt. Ein Anwender kann ein solches Szenario auswählen und starten. Anschließend wird er durch den Prozess geführt, der dem von ihm gegebenen Faktenwissen entspricht. Das beispielhafte Expertenwissen kann in mehrere Szenarien aufgeteilt werden. Denkbar sind etwa ein Szenario, das den Prozess von der CAD-Datei zu einem mit Centaur generierten Netz beschreibt und eines, das den Prozess der Berechnung mit TAU abdeckt. Der Anwender erhält durch diese Aufteilung die Möglichkeit, sehr fein gegliedert auf das für ihn relevante Wissen zuzugreifen.

An dieser Stelle kann festgehalten werden, dass die aufgezeigte Kombination aus Fluss- und Regeldarstellung geeignet ist, die im Expertenwissen enthaltenen Prozesse und Entscheidungen vollständig und strukturiert zu repräsentieren. Die Abbildung des Wissens allein durch Regeln, wie in klassischen regelbasierten Expertensystemen üblich, erscheint hingegen ungeeignet, da sich Abläufe und Prozesse hierdurch wesentlich schwerer definieren und veranschaulichen lassen. Daher erscheint es durchaus sinnvoll, neben dem regelbasierten Ansatz, die von Drools angebotene Ruleflow-Funktion in das zu entwickelnde Expertensystem zu integrieren. Der jeweilige Umfang des Einsatzes von Regeln und Abläufen bleibt offen. Theoretisch ist es bei diesem Ansatz möglich nur die Regel- oder nur die Flussdarstellung zu verwenden. Hierdurch kann der Wissensingenieur, je nach Anwendungsfall, selbstständig entscheiden, wie das Wissen optimal in das System eingebunden wird. Diese Modellierungsfreiheit garantiert eine gute zukünftige Erweiterbarkeit des Expertensystems, so dass auch sehr unterschiedlich strukturiertes Wissen eingepflegt werden kann. Ziel der weiteren Konzeption ist es, eine Architektur

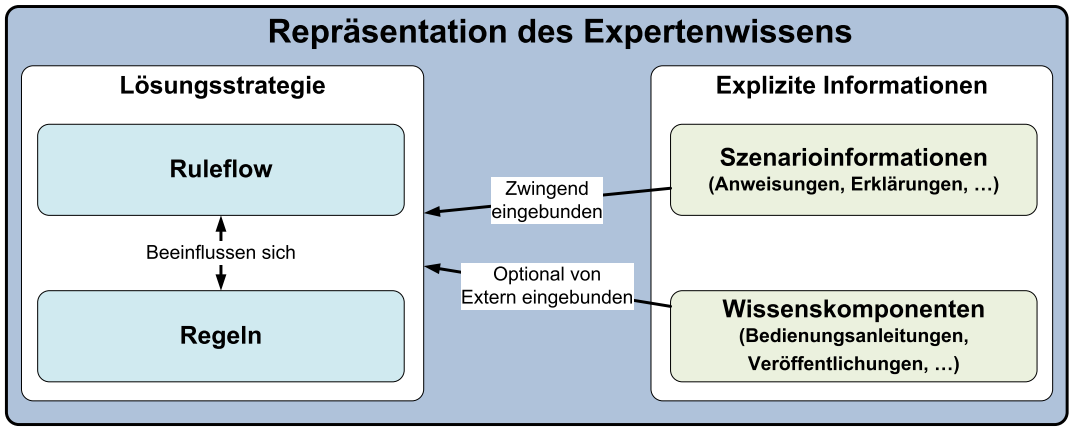

Abbildung 5.2: Darstellung der unterschiedlichen Komponenten zur Wissensrepräsentation

für das Expertensystem zu entwickeln, die diesen Ansatz der Wissensrepräsentation integriert.

5.1.2 Einbindung von expliziten Informationen

Um das Expertenwissen dem Endanwender verständlich darzulegen, ist es nötig, ihm neben dem reinen Wissen über die Lösungsstrategie auch explizite Informationen zur Verfügung zu stellen, etwa als Text oder Bild. Diese Informationen können als eine besondere Form von Expertenwissen aufgefasst werden und lassen sich in zwei Gruppen einteilen. Die erste Gruppe umfasst die Informationen, die sehr eng mit den Prozessen verknüpft sind. Dies sind insbesondere die konkreten Anweisungen oder die wesentlichen Informationen, die dem Endanwender zu jedem Schritt eines Prozesses gegeben werden. Beim Erreichen der oberen Verzweigung von Abbildung 5.1 ist beispielsweise die Information notwendig, dass der direkte Export aus dem Catia-Format in das Iges-Format fehlerbehaftet ist, da dieses Format nur eine begrenzte Genauigkeit bietet. Dieses Wissen ist so stark mit den einzelnen Schritten verknüpft, dass es direkt bei der Erstellung eines Szenarios mit diesen verbunden werden muss und dem Benutzer an der entsprechenden Stelle unaufgefordert angezeigt wird. Dieses Wissen wird als Szenarioinformation bezeichnet und kann als Text oder Bild vorliegen.

Die zweite Gruppe besteht aus expliziten Informationen, die in externen Dokumenten festgehalten sind. Beispielsweise sind dies Anleitungen oder Veröffentlichungen, die etwa als Pdf- oder Word-Datei vorliegen. Dieses Wissen ist nicht zwingend auf einen bestimmten Prozess zugeschnitten. Es bietet eher ergänzendes Hintergrundwissen für einen interessierten Nutzer. Ein solches Element wird im Folgenden als Wissenskomponente bezeichnet. Eine Wissenskomponente befindet sich außerhalb des Szenarios und des Expertensystemkerns und wird durch das Content Layer verwaltet. An geeigneter

Stelle des Prozesses sollen dem Anwender jedoch die zu diesem Zeitpunkt relevanten externen Informationen angeboten werden. Dies kann über zwei Arten passieren. Zum einen kann direkt auf eine bestimmte Komponente verwiesen werden, wenn dieser Verweis vorher ausdrücklich definiert wurde. Dadurch wird dem Anwender gezielt ein fest vorgegebenes externes Dokument angeboten. Das Expertensystem greift dabei direkt auf das Dokument im Content Layer zu. Dies geschieht typischerweise über einen statischen Verweis, ähnlich einem HTML-Link. Ein zweiter Weg ist die Verwendung der Suchmaschine. In diesem Fall ist ein relevanter Suchausdruck vorgegeben, der an entsprechender Stelle des Szenarios an die Suchmaschine des Search Layer übergeben wird. Anschließend werden die gefundenen Wissenskomponenten dem Benutzer bereitgestellt. Der Vorteil dieser Methode ist, dass gesichert ist, dass auch die neuesten Wissenskomponenten dynamisch in das Szenario eingebunden sind, sobald diese in der Wissensdatenbank eingepflegt sind. Ein Nachteil entsteht hingegen, wenn ein Suchausdruck fehlerhaft oder nicht hinreichend ist und daher relevante Wissenskomponenten ignoriert oder zahlreiche irrelevante Wissenskomponenten angezeigt werden.

Abbildung 5.2 stellt die konzipierten Elemente zur Wissensrepräsentation und deren Zusammenhänge zusammenfassend dar.

5.2 Theorie und Entwurf der Erklärungskomponente

5.2.1 Grundlagen der Erklärungskomponente

Neben der Repräsentation des Wissens ist ein weiterer wesentlicher Erfolgsfaktor für die Konzeption des Expertensystems, dessen Fähigkeit dem Anwender Erläuterungen und Begründungen von getroffenen Entscheidungen und Schlüssen nachvollziehbar und vollständig darzulegen. Für diese Funktion ist in der Theorie eine eigene Komponente des Expertensystems zuständig, die als Erklärungskomponente bezeichnet wird. Die Ziele, die durch diese Erklärung erreicht werden sollen, sind nach [BS84]:

- Erhöhtes Verständnis über den Inhalt der Wissensbasis
- Erleichterte Fehlersuche und -beseitigung
- Schulung des Nutzers
- Erhöhte Akzeptanz des Systems durch den Endanwender
- Überzeugung von der Richtigkeit der Lösung

Aufgrund der Anforderungen an das zu entwerfende Expertensystem, liegt der Fokus bei Konzeption der Erklärungskomponente insbesondere auf der Schulung des Nutzers. Ziel der Erklärungen ist es, dass der Nutzer ein vertieftes Wissen und Verständnis erhält und sowohl zu Grunde liegende konzeptionelle Zusammenhänge erkennt, als auch einen Überblick über die Gesamtstrategie zur Problemlösung erlangt. Da der Mensch Dinge

besser merken und speichern kann, deren Hintergrund er versteht und selbst herleiten kann, erleichtern die gelieferten Erklärungen dem Nutzer auch das Lernen der nötigen Abläufe.

Laut [Wol03] zeigten Studien und Umsetzungen in klassischen Expertensystemen, dass verständliche Erklärungen für den Nutzer in typischen Frageformen formuliert werden können. Wichtige Fragen, die ein Nutzer an das System stellen kann, sind demnach:

- Wie wurde diese Lösung beziehungsweise dieser Schluss gefunden?
- Warum muss ich eine bestimmte Information eingeben?
- Welche weiterführenden Informationen kann ich zu diesem Schluss erhalten?
- Was ist die Bedeutung eines bestimmten Fachbegriffs?
- Was wäre die Lösung, wenn ich eine andere Information angegeben hätte?

Darüber hinausgehende, komplexere Fragestellungen sind außerdem:

- Welche Schlüsse wurden nicht gezogen und warum nicht?
- Wie konsistent sind die Fakten mit anderen Schlüssen?
- Welche Fakten wurden im gezogenen Schluss nicht berücksichtigt?

Gesteigert werden kann die Qualität der gelieferten Antworten darüber hinaus durch die Berücksichtigung des Kenntnisstands des Anwenders. So kann etwa einem erfahrenden Anwender tiefergehende Detailinformationen und Optionen angezeigt werden, die einen absoluten Neuling überfordern würden.

5.2.2 Theoretische Umsetzung der Erklärungskomponente

2003 schreibt [Wol03], dass bis heute keine allgemeine Lösung existiert wie Erklärungskomponenten aufzubauen sind, um die oben genannten Fragen zu beantworten. Insbesondere die folgenden zwei Möglichkeiten zur Erklärung von Regeln haben sich nach [Wol03] herausgebildet:

1. Textuell: Es wird vorbereiteter Text ausgegeben, der die Regeln erklärt und die unterliegenden kausalen Zusammenhänge darstellt. Hierbei besteht jedoch die Gefahr der Inkonsistenz, da Regeln und Texte aufeinander abgestimmt sein müssen.
2. Regelübersetzung: Es wird ein Übersetzungsmodul verwendet, das die Regeln, in eine dem Nutzer verständliche, Form (also eine natürliche Sprache) bringt.

[Wol03] merkt hierzu an, dass der Einsatz des textuellen Ansatzes, in Kombination mit anderen Erklärungsmethoden, sehr häufig sinnvoll ist, jedoch nicht immer hinreichend für ein vollständiges Verständnis.

Die Regelübersetzung ist insbesondere bei Expertensystemen, die zu Schulungszwecken eingesetzt werden sollen, besonderes problematisch, da es sich bei Regeln häufig um „kompiliertes Wissen“ handelt. Diese Regeln sind nicht in der Lage, sich selbst, allein durch die Definition zu erklären. Aus diesen Regeln wird nicht der unterliegende kausale Prozess deutlich und die Rechtfertigung für eine Regel liegt außerhalb der Wissensbasis. Das heißt in der Wissensbasis ist Wissen festgehalten, jedoch nicht die Kenntnis warum dieses Wissen gilt. [Wol03]

Auch die zu Grunde liegenden Zusammenhänge der Lösungsfindungsstrategie sind dem System selbst nicht bekannt, obwohl es nach dieser Strategie handelt. Diese zu kennen, ist für den Nutzer jedoch eine wesentliche Verständnis- und Lernhilfe.

5.2.3 Entwickeltes Konzept der Erklärungskomponente

In der entwickelten Umsetzung der Erklärungskomponente können die beiden zuvor beschriebenen Ansätze zur Erklärung der Schlüsse und Regeln parallel verfolgt werden. Der textuelle Ansatz wird umgesetzt, indem der Experte für Regeln und Prozessschritte einen erklärenden Text mitdefinieren kann. Dieser Text ist eine Information, die fest mit dem auszuführenden Szenario verbunden ist und als Erklärungskommentare bezeichnet wird. Es wird eingesetzt, da manche Regeln nicht selbsterklärend sind. Darüber hinaus können explizit die Zusammenhänge zwischen den einzelnen Prozessschritten des Ruleflows und den getroffenen Entscheidungen hergestellt werden. Dabei handelt es sich vorrangig um Informationen, die in den in Abschnitt 5.2.1 aufgeführten Fragen abgebildet werden können. Diese Informationen werden dem Benutzer nicht automatisch angezeigt, sondern müssen von ihm an entsprechender Stelle vom System angefordert werden. Aufgabe eines Experten und eines Wissensingenieurs ist es, zu erkennen welche Informationen wesentlich für das Verständnis eines Szenarios und welche Informationen lediglich erklärend sind. Diese Unterscheidung erlaubt es, den Endanwender zunächst effektiv und zielgerichtet bei seiner Problemstellung zu unterstützen und nur bei Bedarf Hintergrundinformationen anzubieten.

Der Ansatz des Übersetzungsmoduls wird indirekt verfolgt, indem das Expertensystem die Möglichkeit bereitstellt, eine domänenspezifische Sprache (DSL) zu verwenden. Dazu muss die entsprechende Funktion von JBoss Drools eingebunden werden. Durch die Verwendung dieser Funktion wird die Übersetzung von Regelcode in natürliche Sprache angeboten, was eine erhöhte Verständlichkeit der Regeln für den Endanwender ermöglicht. Diese Übersetzung kann prinzipiell sogar in zwei Richtungen angewendet werden, denn sie stellt dem Anwender nicht nur die Regeln in natürlicher Sprache bereit, sondern erlaubt dem Experten oder Wissensingenieur darüber hinaus auch das Schreiben der Regeln in natürlicher Sprache. Die Übersetzung der Regeln ermöglicht jedoch nicht die Darstellung kausaler Zusammenhänge, somit ist sie nur als Ergänzung zu den Erklärungskommentaren zu verstehen.

Aufbauend auf diesen beiden Ansätzen, bietet das konzipierte Expertensystem dem Nutzer die folgenden Funktionen zur Erklärung an:

- **Gefeuerte Regeln anzeigen:** Es werden alle Regeln, die bis zum aktuellen Zeitpunkt im Szenario abgefeuerte wurden, angezeigt.
  Ziel: Ein Verständnis für die getroffenen Entscheidungen erzeugen, sowie die Überzeugung des Nutzers von der Korrektheit.
- **Gesamten Prozess als Grafik anzeigen:** Dem Anwender soll jederzeit die Möglichkeit geboten werden, den gesamten Prozess visuell zu sehen und sich dadurch zu orientieren (*Wo bin ich gerade? Was war bisher? Was kommt noch?*). Die Grafik soll zunächst nicht interaktiv erzeugt werden, sondern beispielsweise nur ein Screenshot des modellierten Ruleflows sein. Später kann die Grafik dynamisch erzeugt werden und bereits getroffene Entscheidungen und gewählte Wege angezeigt werden.
  Ziel: Das Erzeugen von Verständnis über die Gesamtstrategie zur Lösung des Problems, sowie das Erkennen von kausalen Zusammenhängen.
- **Erklärung zu „Warum werde ich um diese Information gebeten?“:** Zu jedem Zeitpunkt, an dem ein Nutzer aufgefordert wird fallbasierte Fakten einzugeben, wird ihm die Option angeboten, erklärende Informationen darüber zu erhalten, warum ihm eine Frage gestellt wird und warum er bestimmte Informationen eingeben muss. Diese erklärende Information wird vom Experten oder Wissensingenieur definiert und ist Freitext.
  Ziel: Der Anwender soll im Voraus erkennen, welchen Einfluss seine Informationen haben und auf diese Weise lernen, die richtigen Entscheidungen zu treffen.
- **Erklärung zu „Warum werden mir diese Informationen angezeigt?“:** Dem Benutzer wird die Möglichkeit gegeben, Erklärungen zu erfragen, die ihm darlegen, warum ihm die aktuellen Informationen angezeigt werden. So sollen der vorausgehende Prozess und die getroffenen Entscheidungen deutlich werden. Diese erklärende Information wird vom Experten oder Wissensingenieur angelegt und ist Freitext.
  Ziel: Das Verständnis der Lösungsstrategie wird gesteigert, indem die zu Grunde liegenden Regeln und Prozesse dargestellt werden. Der erklärende Text verdeutlicht zudem das unterliegende kausale Wissen, das in einer Regel selbst nicht erfasst ist.
- **Beantwortung der Frage: „Was wäre, wenn andere Fakten eingegeben werden“:** Dem Benutzer wird die Möglichkeit gegeben, verschiedene Eingaben und deren Auswirkungen auf das Szenario auszuprobieren. Durch das Ändern seiner Eingaben, werden gegebenenfalls unterschiedliche Regeln gefeuert und sich unterscheidende Schlüsse gezogen. Zunächst soll dies durch eine Undo-Funktion ermöglicht werden, die es dem Nutzer erlaubt, selbstständig im Szenario zurück-

zugehen und die Auswirkung anderer Eingaben zu testen. Später kann eine sogenannte Sandbox inklusive Vorschaufunktion in die Interviewkomponente eingebaut werden. Hierzu soll ein zweites Working Memory parallel für diese Zwecke verwendet werden. Die nächste Regelgruppe wird anhand der Testfakten des zweiten Working Memorys gefeuert und in einer Vorschau werden die Auswirkungen und Schlüsse angezeigt. Bei Verwendung der Sandbox wird die echte Faktenbasis nicht verändert.
Ziel: Der Benutzer kann die Auswirkungen seiner Fakteneingabe verfolgen. Das Verständnis der Regeln und der Strategie wird erhöht.

- **Beantwortung der Frage: „Welche weiterführenden Informationen gibt es zu diesem gezogenen Schluss?“:** Dem Endanwender werden Verweise auf externe Quellen (Dokumente, Anleitungen, Literatur, geschulte Personen, Webseiten etc.) angeboten. In der prototypischen Realisierung wird dies anhand von vordefinierten Suchausdrücken für das Search Layer umgesetzt. Da diese Suche abhängig vom jeweiligen Zustand des Szenarios ist kann diese auch als kontextsensitive Suche aufgefasst werden.
  Ziel: Der interessierte Anwender erhält gezielt Quellen, die vertiefende Informationen bereitstellen.
- **Freie Suche nach weiteren Informationen.** Über ein Suchinterface, das das Search Layer einbindet, bietet das System dem Endanwender ständig die Option der Suche nach weiteren Informationen in der internen Wissensdatenbank.
  Ziel: Der Nutzer soll vertiefende Informationen finden können, ohne durch die Vorgaben des Wissensingenieurs eingeschränkt zu sein.

5.3 Bereitstellen von Informationen

In dem entwickelten Konzept stellt ein Szenario einen in sich abgeschlossenen Prozess dar, der eine definierte Aufgabenstellung löst. Nach dem Starten des Szenarios wird der Benutzer, auf Basis seiner fallbezogenen Fakten, zum Szenarioziel geführt. Um dies zu ermöglichen müssen zwei wesentliche Funktionen in das Expertensystem integriert sein. Zum einen muss eine vordefinierte Funktion existieren, die es erlaubt dem Anwender Informationen zur Verfügung zu stellen. Mit dieser Funktion befasst sich dieser Abschnitt. Zum anderen muss eine Funktion verfügbar sein, die es erlaubt vom Anwender Informationen zu beziehen. Mit dieser Funktion befasst sich der folgende Abschnitt 5.4.

Um dem Endanwender Informationen bereitzustellen, könnte ihm eine einzige Informationsseite (mit Text, Grafik etc.) angezeigt werden, die alle Lösungsschritte umfasst und keine weitere Interaktion vom Anwender erfordert. Dies ist die simpelste Form, führt jedoch, insbesondere bei umfangreicheren Lösungen, zu einer unübersichtlichen Darstellung, die dem Anwender den zu Grunde liegenden Lösungsprozess nur schlecht

strukturiert darlegt. Darüber hinaus, führt in vielen Anwendungen eine fehlende Interaktion mit dem Anwender dazu, dass dieser ermüdet und seine Konzentration sinkt. In dem zu entwickelnden Expertensystem werden dem Anwender daher die Informationen logisch auf mehrere Informationsseiten aufgeteilt. Der Anwender kann durch die Betätigung eines Knopfs schrittweise zur nächsten Information springen und wird somit aktiv eingebunden, was zu einer erhöhten Aufmerksamkeit führt. Des Weiteren wird es möglich, den Lösungsprozess in einzelne Abschnitte abzubilden, so dass das System situationsbezogen entscheiden kann, welche Informationsseiten in den nächsten Schritten angezeigt werden. Das Expertensystem kann das Szenario aus einzelnen Informationsseiten zusammenbauen. Hierbei entscheiden die Regeln und der Ruleflow anhand der fallbezogenen Fakten, welche Seiten angezeigt werden. Informationsseiten dienen der reinen Darstellung von Wissen und nehmen keine Änderungen am Faktenmodell vor.

Durch diesen Ansatz entsteht für den Wissensingenieur die Aufgabe, bei der Modellierung eines Szenarios die Informationen logisch sinnvoll zu gliedern. Dazu ist eine Analyse des Expertenwissens nötig, die beispielsweise anhand eines BPMN-Flussdiagramms erfolgen kann.

Eine Informationsseite besteht aus den folgenden Elementen:

- **Kurzbeschreibung**: Ein kurzer Text, der den Inhalt der Informationsseite beschreibt.
- **Informationstext:** Dieser Text gibt dem Anwender die grundlegenden Informationen, die für das Erreichen des Szenarioziels notwendig sind
- **Grafik:** Eine grafische Darstellung, die den Informationstext unterstützt. Dieses Element ist optional.
- **Erklärender Text:** Dieser Text teilt dem Endanwender mit, warum diese Informationsseite angezeigt wird. Dieses Element dient dem weiterführenden Verständnis und ist der theoretischen Erklärungskomponente zuzuordnen. Der erklärende Text wird nur auf Verlangen des Anwenders angezeigt.
- **Kontextsensitive Suche:** Externe Wissenskomponenten aus dem Content Layer können anhand einer kontextsensitiven Suche aufgerufen werden, so dass die zu der Informationsseite passenden Informationen angezeigt werden. Dieses Element dient dem weiterführenden Verständnis.

5.4 Erfragen von Fakten auf Basis unterschiedlicher Regeltypen

Da die Lösungsstrategie typischerweise abhängig von fallbezogenen Daten ist, muss in dem Expertensystem eine Funktion verfügbar sein, die dem Endanwender die Eingabe

von Fakten ermöglicht. Auf Basis dieser Fakten operiert die Regelmaschine und wertet die entsprechenden Regeln aus, wodurch die zu verfolgende Lösungsstrategie definiert wird. Für die Eingabe von Informationsseiten werden spezielle Seiten in dem Expertensystem angezeigt, die als Interviewseiten bezeichnet werden. Theoretisch können zwei Ansätze verfolgt werden, um die fallbezogenen Daten vom Endanwender zu erhalten.

Zum einen können zu Beginn des Szenarios direkt alle Fakten über eine einzelne Interviewseite erfragt werden, die für alle Regeln des Szenarios relevant sind. Dies bietet den Vorteil, dass die praktische Umsetzung dieses Ansatzes sehr einfach ist, da sowohl Zeitpunkt als auch Umfang der Anwenderbefragung fest definiert sind. Dieser Ansatz birgt jedoch zwei erhebliche Nachteile, die seinen Einsatz in dem zu entwickelnden System verhindern. Der erste Nachteil ist, dass der Anwender immer alle Informationen eingeben muss, also auch Informationen für Schritte und Verzweigungen, welche gegebenenfalls im Verlauf des Szenarios gar nicht erreicht werden. Dies führt somit zu einem unnötigen Aufwand und ist für den Lernerfolg kontraproduktiv, da der Endanwender die Auswirkungen seiner Informationen nicht klar überschauen kann. Der zweite Nachteil besteht darin, dass es in bestimmten Szenarien dem Endanwender zu Beginn der Ausführung eventuell gar nicht möglich ist alle Informationen bereitzustellen. Dies ist etwa in Szenarien denkbar, in denen der Anwender parallel eine Software bedient und bestimmte Ausgaben dieser Software dem Expertensystem mitteilen soll. Durch die genannten Einschränkungen wird die Verwendung dieses Ansatzes im zu entwickelnden Expertensystem unmöglich.

Der zweite Ansatz zur Erfragung von Fakten besteht darin, die Informationen von Benutzer erst anzufordern, wenn sie direkt von den Regeln benötigt werden. Hierfür stehen jeweils einzelne Interviewseiten zur Verfügung. Dies führt dazu, dass ein Mechanismus in das Expertensystem integriert werden muss, der an den entsprechenden Stellen des Szenarios den Benutzer zur Eingabe weiterer Informationen, anhand der Interviewseiten auffordert. Dieser Mechanismus benötigt genaue Kenntnisse über die Fakten, auf denen die Regeln operieren und den Zeitpunkt, zu dem diese verfügbar sein müssen. Daher ist eine sehr enge Verknüpfung zwischen dem umzusetzenden Mechanismus und den eigentlichen Regeln notwendig. Erschwerend kommt hinzu, dass das Expertensystem einfach erweiterbar sein muss und daher der Mechanismus möglichst simpel anzuwenden sein muss.

Der, in der vorliegenden Arbeit entwickelte Mechanismus, beruht auf der folgenden Erkenntnis. Eine zentral, in das Expertensystem integrierte Funktion ist nicht in der Lage, selbstständig zu erkennen, ob Regeln aktuell zusätzliche Fakten benötigen, da das Working Memory diese Informationen nach außen nicht verfügbar macht. Da somit nur die Regeln beziehungsweise Ruleflow-Gruppen das Wissen besitzen, welche Informationen als nächstes gebraucht werden, ist es nötig, ein Konzept anzubieten, dass es aus dem Working Memory heraus erlaubt, den Faktenbedarf zu ermitteln. Hierzu bietet

sich die Einführung eines besonderen Typs von Regeln an. Deren Aufgabe ist es einzig zu überprüfen, ob alle nötigen Informationen bereits verfügbar sind. Ist dies nicht der Fall, feuert die Regel und aktiviert die entsprechenden Interviewseiten.

Aufgrund des entworfenen Mechanismus verfügt das Expertensystem über Regeln, die sich in zwei Typen einteilen lassen:

- **Faktensammelregeln:** Diese Regeln sind verantwortlich dafür, dass alle nötigen Fakten zur Verfügung stehen und sammelt die fehlenden Informationen vom Benutzer ein. Daher sollte jede Regel dieser Gruppe mindestens einen Wert in der Prämisse haben, der darauf überprüft wird, ob er nicht initialisiert ist. Ist dies der Fall, wird der entsprechende Bedarf mitgeteilt, was durch die Aktivierung von Interviewseiten geschieht. Regeln dieses Typs verfügen über kein direktes Expertenwissen, sondern sind lediglich eine Hilfskonstruktion, um Fakten erst zu dem Zeitpunkt zu erfragen, an dem diese von der Regelgruppe benötigt werden. Die bestmögliche Modellierung dieser Regeln innerhalb des Ruleflows lässt sich nicht generell für alle Anwendungsfälle beantworten und ist daher Aufgabe eines erfahrenen Wissensingenieurs. Dieser muss im Wesentlichen den Zeitpunkt und zugehörige Umfang der Befragung berücksichtigen.
- **Entscheidungsregeln:** Diese Regeln bilden die eigentliche faktenbezogenen Entscheidungen der Lösungsstrategie ab. Sie sind für den weiteren Verlauf des Szenarios verantwortlich und können Informationsseiten aktivieren, externe Funktionen aufrufen oder Fakten verändern. Diese Regeln dürfen nur auf initialisierten Werten operieren, also in der Prämisse nur Überprüfung auf Basis konkreter Werte anstellen. Der Wissensingenieur muss diese Regeln also in der Form erstellen, dass keine Entscheidungen auf Basis des Initialisierungszustandes eines Wertes getroffen werden. Diese Konvention ist nötig, um zu vermeiden, dass Regeln fehlerhaft feuern, nur weil ein Wert bisher noch nicht vom Anwender erfragt wurde. Eine Sonderform stellen Regeln dar, die keine Bedingung besitzen und daher immer erfüllt sind. Diese Regeln können an bestimmten Stellen des Szenarios eingesetzt werden, um Aktionen grundsätzlich und unabhängig von den Fakten auszuführen.

Eine Faktensammelregel besitzt vereinfacht den folgenden Aufbau, der an die Regelstruktur von Drools angelehnt ist. In dem Beispiel aus Listing 5.1 wird überprüft, ob das Fakt, welches das Datenformat einer CAD-Konstruktion definiert, noch uninitialisiert ist. Wenn dies der Fall ist, dann wird die entsprechende Interviewseite für die Aktivierung vorgemerkt.

Listing 5.1: Beispiel einer Faktensammelregel

```
1 rule "Check if CAD format is set"
        ruleflow-group "Convert to Centaur"
3       no-loop
        when
```

```
5                   #cad format is not set
                    cadFacts : CadToCentaurFacts(cadFormat == null)
7                   dS : DroolsSession()
        then
9                   #activate interviewpage to ask user for cad format
                    dS.addInteviewForActivation("cadFormatIsNull");
11                  update( dS );
end
```

Eine auf der gesammelten Information aufbauende Entscheidungsregel würde dementsprechend den folgenden Aufbau besitzen. In dem Beispiel aus Listing 5.2 wird überprüft, ob das CAD-Datenformat in dem zugehörigen Faktenobjekt als Catia definiert ist. Wenn dies zutrifft, wird eine entsprechende Infoseite aktiviert.

Listing 5.2: Beispiel einer Entscheidungsregel

```
rule "Is CAD format Catia?"
2       ruleflow-group "Convert to Centaur"
        no-loop
4       when
                #cad format is catia
6               cadFacts : CadToCentaurFacts(cadFormat soundslike "CATIA")
                dS : DroolsSession()
8       then
                #Activate informationpage to provide infos to the user
10              dS.addInfopageForActivation("CatiaDataFromat");
                update( dS );
12      end
```

Ein Vorteil dieser Lösung besteht darin, dass dieser Mechanismus vollständig durch Regeln umgesetzt ist. Folglich befindet sich der Wissensstand des Mechanismus jederzeit im selben Zustand, wie der Wissensstand der Regelmaschine. Eine korrekte Zustandsüberwachung, der in der Regelmaschine enthaltenen Regeln und Fakten, kann daher gewährleistet werden. Darüber hinaus wird ein leichtes Erlernen dieses Mechanismus durch den Wissensingenieur ermöglicht, da der Mechanismus vollständig über die typische Regelsyntax umgesetzt wird. Es müssen lediglich einige Konventionen eingehalten werden, welche einfach aus Beispielen abgeleitet werden können. Eine simple und flexible Erweiterung des Systems ist an dieser Stelle sichergestellt.

Die durch den beschriebenen Algorithmus ausgelösten Interviewseiten können ein einzelnes Fakt oder mehrere Fakten erfragen, wobei dies der Wissensingenieur abhängig vom jeweiligen Szenario entscheidet.

Die Elemente einer Interviewseite sind die Folgenden:

- **Kurzbeschreibung**: Ein kurzer Text, der den Inhalt der Interviewseite beschreibt.
- **Eingabeelemente**: Anhand der Eingabeelemente, definiert der Endanwender die

nötigen Fakten. Hierbei können verschiedene Eingabeelemente verwendet werden, abhängig vom Faktentyp. Zunächst werden freie Eingabefelder für Text und Zahlen angeboten, sowie Felder für die Auswahl vordefinierter Informationen, wie beispielsweise Checkboxen oder Listen.

- **Erklärender Text:** Anhand dieses Textes wird dem Endanwender erklärt, warum eine bestimmte Information benötigt wird. Dieses Element dient dem weiterführenden Verständnis und ist der theoretischen Erklärungskomponente zuzuordnen. Damit dieser Text angezeigt wird, muss der Endanwender einen Button betätigen.
- **Kontextsensitive Suche:** Externe Wissenskomponenten aus dem Content Layer, können anhand einer kontextsensitiven Suche aufgerufen werden, so dass, die zu der Interviewseite passenden Informationen angezeigt werden. Dieses Element dient dem weiterführenden Verständnis.

5.5 Elemente und Ablauf eines Szenarios

5.5.1 Konzeptionelle Elemente eines Szenarios

Dieser Abschnitt liefert eine Übersicht der konzeptionellen Elemente eines Szenarios.

- **Szenariopaket:** Dies stellt die zusammenfassende Obereinheit dar, die alle anderen Elemente des Szenarios beinhaltet. Ein Szenario besitzt genau ein Szenariopaket. Alle anderen Elemente werden mindestens einem Szenariopaket zugeordnet.
- **Szenariobeschreibung:** Jedes Szenario verfügt über eine Beschreibung. Diese nennt das Problem, welches in dem Szenario gelöst wird und definiert die notwendigen Vorbedingungen, indem der Ausgangspunkt des Szenarios beschrieben wird. Diese Beschreibungen unterstützen den Endanwender bei Auswahl eines geeigneten Szenarios.
- **Informationsseiten:** Diese Seiten enthalten die Szenarioinformationen, die dem Endanwender zur Lösung des Szenarioproblems zur Verfügung gestellt werden. Darüber hinaus können Verweise auf externe Wissenskomponenten eingebaut werden, sowie die erklärenden Informationen der theoretischen Erklärungskomponente. Ein Szenario besteht aus mindestens einer Informationsseite. Typischerweise besitzt ein Szenario jedoch mehrere Informationsseiten, wobei der Wissensingenieur für die geeignete logische Aufteilung der Informationen verantwortlich ist. Das Anzeigen einer Informationsseite bewirkt keine Änderung an der Faktenbasis.
- **Interviewseiten:** Diese Seiten sind Formulare, in denen der Endanwender die notwendigen fallbezogenen Fakten eingeben kann. Hierzu können verschiedene Formularelemente verwendet werden. Um fehlerhafte und ungültige Eingaben zu

vermeiden, werden die eingegebenen Werte überprüft. Auf Basis der Benutzereingaben wird die Faktenbasis aktualisiert. Ein Szenario kann beliebig viele Interviewseiten besitzen. Der Wissensingenieur modelliert den Ablauf der Befragung anhand von Ruleflows und eigens für die Seitenaktivierung erstellten Hilfsregeln. Interviewseiten verfügen über Erklärungen, warum diese Informationen bereitgestellt werden müssen. Eine Interviewseite kann, abhängig vom konkreten Szenario, ein einzelnes Fakt oder mehrere Fakten erfragen.

- **Faktenmodell:** Das Faktenmodell bildet die für die Regeln relevanten Fakten des Szenarios ab und bietet die Möglichkeit, die vom Benutzer eingegebenen Werte festzuhalten. Die Fakten sind zunächst uninitialisiert und werden sukzessive mit den Benutzereingaben belegt.
- **Ruleflow:** Der Ruleflow bildet den gesamten Prozess der Lösungsstrategie ab. Die Komplexität des Workflows kann variieren. Für die Repräsentation von Entscheidungen werden Ruleflow-Gruppen an den entsprechenden Stellen eingebaut. Die geeignete Modellierung des Ruleflows ist wesentlich für die Güte des Szenarios. Ein Szenario verfügt über mindestens einen Ruleflow. Dieser kann weitere Subruleflows enthalten.
- **Regeln:** Die Hauptaufgabe der Regeln ist die Abbildung von Entscheidungen, die auf Basis von fallbezogenen Fakten realisiert werden. Durch diese Regeln wird etwa die Darstellung von Informationen angestoßen, komplexe Funktionen aufgerufen oder das weitere Fortschreiten im Ruleflow definiert. Um sicherzustellen, dass die für die Regeln notwendigen Fakten zeitnah vom Endanwender erfragt werden, existieren spezielle Hilfsregeln, welche die Erfragung dieser Fakten anstoßen. Ein Szenario kann beliebig viele Regeln enthalten
- **Ruleflow-Gruppen:** Regeln werden strukturiert und gruppiert, indem diese zu Ruleflow-Gruppen zusammengefasst werden. Der Ausführungszeitpunkt der Ruleflow-Gruppen wird durch den Ruleflow festgelegt.
- **Domain Specific Language:** Es kann für ein Szenario durch den Wissensingenieur eine Domain Specific Language (DSL) definiert werden, die die Formulierung der Regeln in einer natürlichsprachlichen Form erlaubt. Durch eine DSL wird zum einen das Erzeugen von Regeln erleichtert, zum anderen werden die erstellten Regeln leichter verständlich, was die Ziele der Erklärungskomponente unterstützt. Die Verwendung ist optional.
- **Hilfsfunktionen:** Diese Elemente stellen komplexe Funktionen dar, die gegebenenfalls im Rahmen der Lösungsstrategie ausgeführt werden müssen. Dies können etwa Berechnungen sein oder der Aufruf externer Softwareprogramme. Die Verwendung ist optional und abhängig vom Szenario.

5.5.2 Ablauf eines Szenarios

Nachdem im vorherigen Abschnitt die wesentlichen konzeptionellen Komponenten eines Szenarios zusammengefasst wurden, wird im Folgenden der grundsätzliche Ablauf eines Szenarios und das Zusammenspiel der Komponenten vorgestellt. Es ist zu beachten, dass die konkrete Gestaltung eines Szenarios abhängig von der jeweiligen Lösungsstrategie ist und daher stark variieren kann. Es kommt insbesondere der Verwendung von Ruleflows und Ruleflow-Gruppen eine hohe Bedeutung zu, da diese dem Wissensingenieur erlauben, die Regeln in Gruppen zu strukturieren und den Ausführungszeitpunkt zu definieren.

Das Zusammenspiel der Elemente und der Szenarioablauf sind im Folgenden beschrieben. Zusätzlich zeigt Abbildung 5.3 den Ablauf grafisch.

1. **Szenarioauswahl:** Der Anwender wählt das Szenario aus, das ihn bei der Lösung seines Problem unterstützt. Hierzu werden ihm die Szenariobeschreibungen aller verfügbaren Szenarien angeboten.
2. **Starten des Szenarios:** Nach der Auswahl des Szenarios, wird dies gestartet und alle notwendigen Elemente geladen. Gegebenenfalls wird zunächst eine Begrüßungsseite angezeigt, die allgemeine Informationen gibt. Im Hintergrund wird die Regelmaschine initialisiert und der Ruleflow, die Regeln und das Faktenmodell werden in die Regelmaschine eingefügt. Welche Elemente zu einem Szenario gehören, wird durch das Szenariopaket definiert.
3. **Ruleflow wird gestartet:** Die Ausführung des Ruleflows, der den Lösungsprozess darstellt, beginnt am Start-Knoten.
4. **Durchlaufen des Ruleflows:** Der weitere Ablauf, innerhalb des Ruleflows, ist abhängig von der konkreten Modellierung des Ablaufs durch den Wissensingenieur. Generell, wird der Endanwender durch das Anzeigen von Seiten, schrittweise durch das Szenario geführt, wobei er die Geschwindigkeit selbstständig durch die Betätigung einer Weiter-Taste bestimmen kann. Im Hintergrund durchläuft die Regelmaschine die Knotenelemente des Ruleflows. Aus Anwendersicht können bei Betätigung der Weiter-Taste, die folgenden beiden Ereignisse eintreten:
   a) **Informationsseite anzeigen:** Auf Basis der vorhandenen Fakten feuert eine Regel und aktiviert eine Informationsseite. Dem Benutzer werden auf diese Weise, die zu diesem Zeitpunkt relevanten, Informationen angeboten. Es werden keine Änderungen am Faktenmodell vorgenommen.
   b) **Interviewseite anzeigen:** Der Ruleflow aktiviert eine Ruleflow-Gruppe, die ausgewertet werden soll. Wenn die Hilfsregeln dieser Gruppe erkennen, dass zu diesem Zeitpunkt nicht genügend Fakten zur Verfügung stehen, stoßen sie die Aktivierung einer Interviewseite an. Der Benutzer tätigt seine

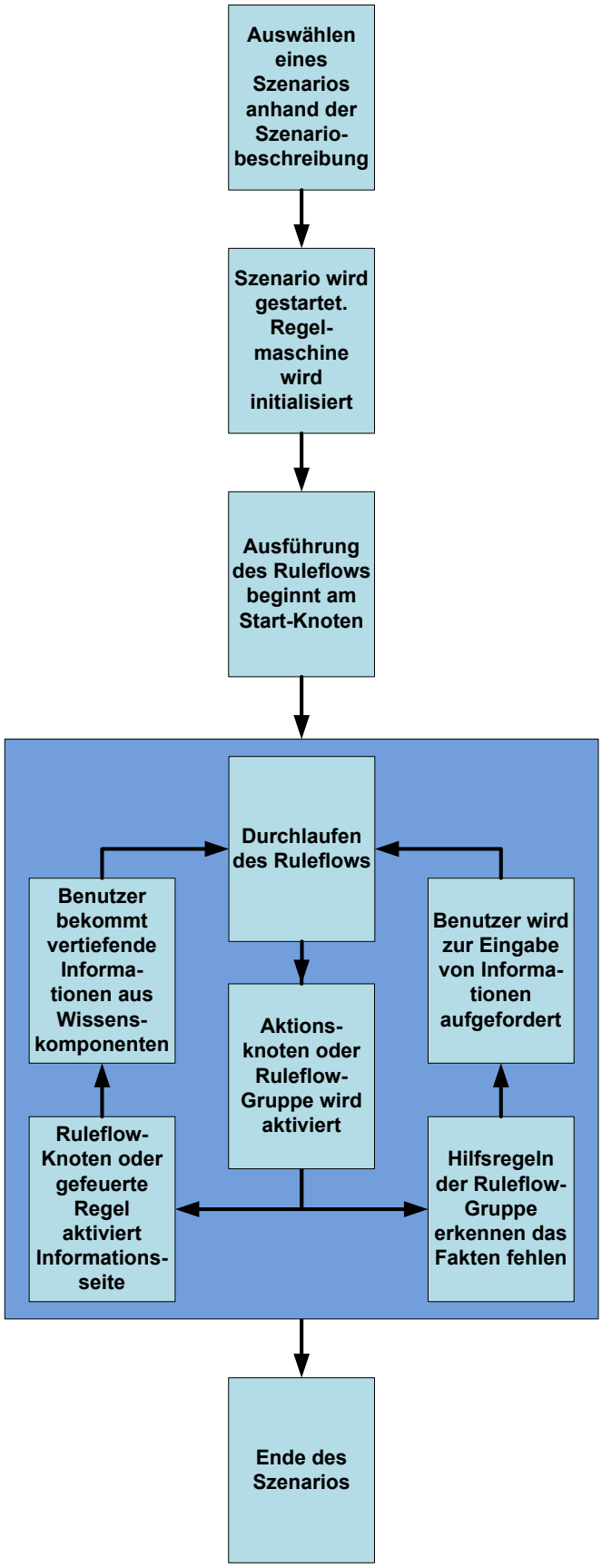

Abbildung 5.3: Der Ablauf eines Szenarios aus Anwendersicht.

Eingaben und sendet diese ab, so dass die Faktenbasis aktualisiert wird.

5. **Ende des Szenarios:** Nach Abarbeitung aller Schritte des Szenarios ist das Ende des Ruleflows erreicht. Dies wird dem Anwender mitgeteilt und das Szenario kann beendet werden.

5.5.3 Aktivierung mehrerer Seiten

Eine Herausforderung bei der Konzeption eines allgemeinen Szenarioablaufs ist die Folgende. Wird JBoss Drools einmalig zum Feuern von Regeln aufgefordert, dann werden alle Regeln, die sich auf der Agenda-Liste befinden, nacheinander aktiviert. Anschließend wird zum nächsten Knotenelement fortgeschritten und dieser abgearbeitet. Die Regelmaschine hält somit nicht automatisch, nach einer gefeuerten Regel oder einem abgearbeiteten Knotenelement, an. In Folge dessen, ist es möglich, dass, während dem Endanwender eine Seite angezeigt wird, im Hintergrund bereits weitere Seiten aktiviert werden. Der Zustand der Regelmaschine und der Zustand dessen, was dem Anwender präsentiert wird, sind somit nicht zwingend synchron. Daher muss in das Expertensystem eine Liste integriert werden, in der Seiten vorgemerkt werden, die von den Regeln aktiviert, aber noch nicht angezeigt wurden. Darüber hinaus ist es erforderlich, eine Konvention einzuführen, die die Reihenfolge festlegt, in der die vorgemerkten Seiten ausgeführt werden. Hierzu ist zunächst, eine Betrachtung der möglichen Aktivierungen nötig.

Bei der Aktivierung mehrerer Seiten können die folgenden drei Fälle auftreten:

1. **Nur Informationsseiten:** Bei der Ausführung des Szenarios können mehrere Informationsseiten aktiviert werden. Dies kann dadurch geschehen, dass mehrere Entscheidungsregeln erfüllt und gefeuert wurden oder dass ein Aktionsknoten des Ruleflows die Aktivierung mehrerer Informationsseiten anstößt.
2. **Nur Interviewseiten:** Der Szenarioablauf kann in der Form modelliert sein, dass mehrere Faktensammelregeln Regeln feuern. Dies führt gegebenenfalls zu mehreren aktivierten Interviewseiten.
3. **Informationsseiten und Interviewseiten:** Bestimmte Situationen sind denkbar, in denen, eine beliebige Kombination von Informationsseiten und Interviewseiten aktiviert wird. Dieser Fall tritt dann ein, wenn Faktensammelregeln erkennen, dass einige Fakten fehlen, gleichzeitig genügend Fakten vorliegen um Entscheidungsregeln zu aktivieren.

Um ein einheitliches und vorhersehbares Verhalten des Expertensystems bei der Präsentation von Seiten zu gewährleisten, ist es notwenig, für die oben genannten Fälle, die Ausführungsreihenfolge festzulegen. Die folgende Konvention wurde definiert:

1. Für den ersten Fall, in dem mehrere Informationsseiten angezeigt werden, wird

Folgendes festgelegt. Die Seiten werden in der Reihenfolge ausgeführt, in der, deren Aktivierung, dem Expertensystem mitgeteilt wird.

Da die Aktivierungsreihenfolge der Regeln grundsätzlich nicht vorhersehbar ist, muss der Wissensingenieur beim Design des Szenarios sicherstellen, dass die, auf den Seiten dargestellten Informatione,n für den Endanwender verständlich bereitgestellt werden. Hierzu kann er entweder ein Design wählen, das den Informationsgehalt unabhängig von der Reihenfolge werden lässt, oder bei der Szenariomodellierung die Reihenfolge explizit festlegen, etwa über die Priorisierung von Regeln durch die Drools Salience Funktion.

Für die Konsistenz der Faktenbasis und der Gültigkeit von bereits aktivierten Regeln, ist die Reihenfolge der angezeigten Informationsseiten gleichgültig, da diese keine Veränderungen an der Faktenbasis vornehmen.

2. Wenn mehrere Interviewseiten angezeigt werden, stellt sich die Herausforderung, dass jede dieser Seiten Änderungen an der Faktenbasis vornimmt. Daher können theoretisch neue Entscheidungsregeln aktiv, oder alte bereits aktivierte Faktensammelregeln ungültig werden. Um dieses Problem zu lösen und die Konsistenz zu sichern, wird ein Ansatz verfolgt, der sich vereinfacht an dem Truth Maintenance Konzept von Drools orientiert (vgl. Abschnitt 3.2.6). Wie bei diesem Konzept, werden in der entwickelten Umsetzung, alle Änderungen, die während des Anzeigeprozesses stattfinden, bis zum Ende dieses Prozesses ignoriert. Dies wird realisiert, indem bis zum Abschluss der Änderungen neue Regeln nur aktiviert, aber nicht gefeuert werden. Daher bleibt, die vor diesem Prozess festgehaltene Menge der aktivierten Seiten unverändert, und die Konsistenz während der Fakteneingabe erhalten. Erst im Anschluss an den Anzeigeprozess, werden die Änderungen an der Faktenbasis berücksichtigt, und es dürfen wieder neue Regeln gefeuert und neue Seiten aktiviert werden. Dieses Verhalten muss von dem Wissensingenieur beachtet werden.

   Die konkrete Reihenfolge, in der die Interviewseiten angezeigt werden, entspricht dem Modell der Informationsseiten. Die Reihenfolge wird also durch den Zeitpunkt der Aktivierung, bei der hierfür zuständige Komponente, bestimmt.

3. Eine weitere Regelung muss bei der gleichzeitigen Aktivierung von Informationsseiten und Interviewseiten getroffen werden. In dieser wird eine Kombination aus den beiden vorherigen Konventionen verwendet. Es wird festgelegt, dass zunächst alle Informationsseiten gemäß der Beschreibung von Punkt Eins angezeigt werden. Erst im Anschluss beginnt die Abarbeitung der Interviewseiten, die wie unter Punkt Zwei beschrieben, erfolgt.

   Diese Regelung wird zum einen durch ihre einfache Nachvollziehbarkeit begründet, wodurch die Modellierung des Szenarios für den Wissensingenieur vereinfacht

wird. Zum anderen ist ein logischerer Informationsfluss gewährleistet, da der Anwender zunächst alle, zu diesem Zeitpunkt, verfügbaren Informationen erhält und erst danach selbst Informationen bereitstellen muss. Es ist also theoretisch gewährleistet, dass der Anwender zum Befragungszeitpunkt das nötige Wissen erhalten hat, um die gestellten Fragen zu verstehen und beantworten zu können.

5.6 Benutzerperspektiven und ihre Funktionen

In der Anforderungsanalyse wurden verschiedene Anwendertypen des Expertensystems ermittelt. Da sich die, von ihnen auszuführenden Aufgaben unterscheiden, und nicht jeder Anwendertyp alle Aufgaben ausführen darf, muss das System in der Lage sein, jedem Anwendertyp, die auf seine Anforderungen zugeschnitten Funktionen zur Verfügung zu stellen. Um dies zu realisieren, wird jedem Anwender eine eigene Perspektive angeboten, die der Definition der Eclipse RCP folgt. Die entwickelte Expertensystemarchitektur verfügt über die folgenden drei Perspektiven. Diese bieten jeweils die Funktionalität, die anhand der Anwendungsfälle in der Anforderungsermittlung (vgl. Abschnitt 4.6) beschrieben wurde.

- **Perspektive für Endanwender:** Zentrales Element dieser Perspektive ist die Möglichkeit, vordefinierte Szenarien auszuführen. Dafür werden dem Endanwender die nötigen Editoren und Views bereitgestellt.
- **Perspektive für Wissensingenieure:** Die Ansicht für den Wissensingenieur wird über eine Kombination von Elementen (Editoren und Views) des JBoss Drools Eclipse Plugins und der standardmäßigen Eclipse IDE realisiert. Hierdurch kann der Wissensingenieur seine zentrale Aufgabe, die Erstellung und Bearbeitung von Szenarien, mit einer möglichst großen Freiheit erledigen, wobei er sich selbstständig an die Konventionen zur Gestaltung von Szenarien halten muss.
- **Perspektive für Release-Manager:** Wie zuvor beschrieben, werden die Aufgaben dieser Perspektive auf Grund des begrenzten Zeitrahmens dieser Arbeit nur am Rande betrachtet und nicht prototypisch implementiert [vgl. 4.3].

6 Technische Realisierung

Dieses Kapitel beschreibt die technische Realisierung des zuvor entwickelten Konzepts. Es werden die wesentlichen Komponenten des Prototyps aufgezeigt, sowie die Entscheidungen, die bei der Umsetzung getroffen wurden, dargelegt und begründet. Darüber hinaus werden die während der Umsetzung entdeckten Probleme genannt. Auf die Erläuterung von typischen und für das Thema dieser Arbeit nicht wesentlichen softwaretechnischen Umsetzungen wird verzichtet.

6.1 Anspruch und Zielsetzung der Realisierung

Bedingt durch den sehr hohen Implementierungsumfang und den beschränkten Zeitrahmen dieser Arbeit, ist es das Ziel der technischen Realisierung einen Prototyp zu erstellen, der die Machbarkeit der wesentlichen Aspekte des entwickelten Konzepts belegt, also einen so genannten Proof-of-Concept erbringt. Hierzu wird zunächst die Kernfunktionalität des Expertensystems umgesetzt und gegebenenfalls damit verbundene Probleme identifiziert. Wichtig sind insbesondere die Integration von JBoss Drools in das Expertensystem, die Fähigkeit zur strukturierten Abbildung von Wissen in Form von Szenarien und die Umsetzung einer einfach erweiterbaren Benutzeroberfläche. Die Einbindung externer Wissenskomponenten aus dem Content Layer, wie etwa Pdf-Dateien, wird nur über die Suchanwendung integriert, da zum Zeitpunkt dieser Arbeit keine andere Schnittstelle zu diesen Daten angeboten wird. Des Weiteren wird die vollständige Umsetzung der Erklärungskomponente vernachlässigt und nur die wesentlichen Funktionen umgesetzt. An den entsprechenden Stellen wird aufgezeigt, dass die Möglichkeit besteht, weitere Funktionen zukünftig in das System zu integrieren, da das Expertensystem auf diese Erweiterungen ausgelegt ist.

6.2 Applikationsarchitektur und Paketmodell

Die Umsetzung der Applikation erfolgt gemäß den ermittelten Anforderungen und verwendet im Wesentlichen die Technologien Java 6, Drools 4.0.7 und Eclipse RCP 3.3. Da die Applikation auf Basis der Eclipse Rich-Client Platform implementiert wird, ist eine entsprechende Plugin-basierte Applikationsarchitektur zu entwerfen. Es werden die verschiedenen Kernkomponenten des Systems in jeweils eigenständige Plugins

integriert, wobei es das Ziel ist, diese Module möglichst unabhängig voneinander zu gestalten, jedoch funktional und logisch zusammenhängende Komponenten zu bündeln. Hierbei wird insbesondere den Benutzertyp-abhängigen Perspektiven Rechnung getragen. Durch die Plugin-basierte Architektur soll eine gute spätere Erweiterbarkeit und Pflege der Applikation erreicht werden.

Unter Berücksichtigung der genannten Überlegungen wird das folgende Plugin-Modell entworfen. Die Namen der Plugins verwenden das für diese Applikation definierte Präfix `de.dlr.xps`, das auch für die Javapakete verwendet wird. Die Plugins `org.drools` und `org.drools.eclipse` umhüllen die externen Drools Pakete und binden diese gebündelt in die Eclipse RCP Infrastruktur ein.

- **de.dlr.xps.app:** Dieses Plugin stellt eine Besonderheit dar, denn es ist der Startpunkt der gesamten Applikation. Jede eigenständige Eclipse RCP-basierte Anwendung benötigt ein Plugin, das diese Sonderrolle übernimmt [ML05]. Hierzu ist es nötig, dass dieses Plugin im Sinne der Eclipse Equinox Runtime ausführbar ist. Seit Eclipse 3.3 wird dies erreicht, indem die Applikation das Interface `org.eclipse.equinox.app.IApplication` implementiert und daher eine `start()`- und eine `stop()`-Methode besitzt, die die Ausführung und Beendigung der Applikation definieren. In diesen Methoden können beispielsweise die weiteren zu ladenden Plugins definiert werden oder etwa ein Splash-Bildschirm inklusive Benutzeranmeldung.

  Um eine klare funktionale Trennung der Zuständigkeiten zu erreichen, wird keine Expertensystem-spezifische Logik in dieses Plugin integriert.

- **de.dlr.xps.core:** Die zentralen Kernelemente des Expertensystems werden in diesem Plugin gebündelt und somit von einem zentralen Punkt aus, anderen Plugins zur Verfügung gestellt. Dieses Plugin enthält insbesondere:
  - Eine Sessionverwaltung für das Expertensystem
  - Das Datenmodell für die Objekte eines Szenarios
  - Einen zentralen Zugriffspunkt auf die Drools Regelmaschine

- **de.dlr.xps.enduser:** In diesem Plugin werden die Komponenten zusammengefasst, die einzig für die Endanwender-Perspektive relevant sind. Es handelt es sich insbesondere um Komponenten, welche die Ausführung von Szenarien steuern und die zugehörigen Klassen zur Definition von Benutzerschnittstellen für die Eingabe und Ausgabe von Informationen.

- **de.dlr.xps.knowledeEngineer:** Entsprechend dem Plugin für den Endanwender, wird hier, die nur für den Wissensingenieur relevante Funktionalität gebündelt. Im Wesentlichen definiert dieses Plugin dazu eine Perspektive, die die relevanten Grafikelemente (Editor und Views) des Drools Eclipse Plugins anordnet.

- **de.dlr.xps.search:** Dieses Plugin stellt Funktionen bereit, die dem Expertensystem einen suchbasierten Zugriff auf die externen Wissenskomponenten des Content Layers ermöglichen. Auf Grund der Schnittstellen zwischen den verschiedenen Layern des Gesamtsystems, wird dieses Plugin gemeinsam im Team entwickelt. Da die Suche eine Funktion darstellt, die außerhalb des Kernsystems des Expertensystems liegt, wird diese als ein OSGi Service angeboten. Hierzu registriert das Search-Plugin seinen Dienst bei der OSGi Service-Registrierung. Im entwickelten Prototypen nimmt das Enduser-Plugin diesen Service in Anspruch und leitet seine Suchausdrücke über die definierte Serviceschnittstelle weiter.
- **org.drools:** Um umfangreiche externe Software-Bibliotheken in einer Eclipse RCP Anwendung zu verwenden, sollten die externen Paket nicht lose importiert werden, sondern durch ein eigenes Plugin zusammengefasst und dieses in die Anwendung integriert werden [SJB08]. Somit entsprechen die externen Komponenten der Plugin-basierten Architektur von Eclipse RCP und der Zugriff kann gemäß dem OSGi-Standard definiert werden. Seit Version 3.3 bietet die Eclipse IDE die Möglichkeit, solch ein Plugin automatisch, auf Basis der einzubindenden externen JAR-Bibliotheken zu erzeugen, wobei gleichzeitig das entsprechende OSGi-Manifest erstellt wird.

  Die beschriebene automatische Möglichkeit zum Erzeugen eines Plugins wird verwendet, um die umfangreiche externe Sammlung von Drools Bibliotheken in das zu entwickelnde Expertensystem einzubinden.
- **org.drools.eclipse:** Dies ist das, vom Drools Team selbst bereitgestellte, Eclipse Plugin, welches den Anwender bei der Erstellung von Drools spezifischen Elementen, wie Regeln, oder Ruleflows, unterstützt. Dazu bietet das Plugin eigene Editoren und Views an, die, in dem entwickelten Prototyp, insbesondere in die Perspektive für Wissensingenieure integriert werden.

Abbildung 6.1 zeigt die Assoziationen zwischen den Paketen der beiden Plugins `de.dlr.xps.core` und `de.dlr.xps.enduser`, welche die umfangreichsten und wesentlichsten Komponenten des implementierten Expertensystems sind. Ergänzend gibt folgende Aufzählung einen Überblick über das Paketmodell aller Plugins.

- `de.dlr.xps.app`
- `de.dlr.xps.core`
  - `de.dlr.xps.core.engine`
  - `de.dlr.xps.core.engine.scenarioObjects`
  - `de.dlr.xps.core.gui`
  - `de.dlr.xps.core.gui.infopage`
  - `de.dlr.xps.core.gui.interview`

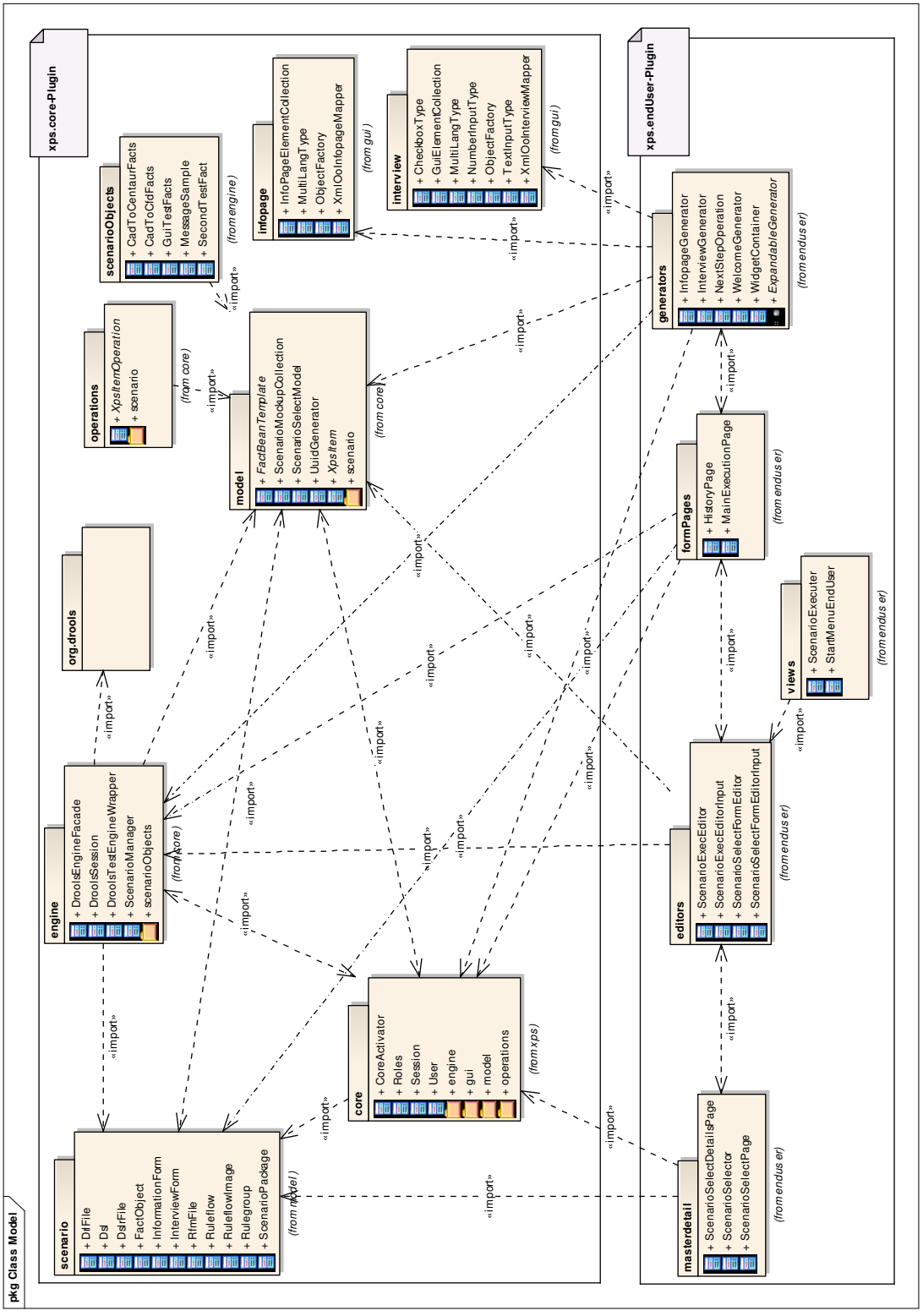

Abbildung 6.1: Darstellung des Paketmodells und der Assoziationen zwischen den Plugins `de.dlr.xps.core` und `de.dlr.xps.enduser`.

  - de.dlr.xps.core.model
  - de.dlr.xps.core.model.scenario
  - de.dlr.xps.core.operations
  - de.dlr.xps.core.operations.scenario
- de.dlr.xps.enduser
  - de.dlr.xps.enduser.editors
  - de.dlr.xps.enduser.formPages
  - de.dlr.xps.enduser.generators
  - de.dlr.xps.enduser.masterdetail
  - de.dlr.xps.enduser.views
- de.dlr.xps.knowledgeEngineer
  - de.dlr.xps.knowledgeEngineer.views
- de.dlr.xps.search

Wie zuvor bereits erwähnt, bindet das Expertensystem darüber hinaus die Plugins org.drools und org.drools ein, die alle Pakete der Drools 4.0.7 Distribution bündeln. Einen Überblick über die dort enthaltenen Pakete bietet die Dokumentation der Drools API [JBo].

6.3 Erweiterbarkeit des Systems um neue Szenarien

Ein Szenario setzt sich aus den, in Abschnitt 5.5.1 beschriebenen, konzeptionellen Elementen zusammen, die wesentlich für die Modellierung und Bereitstellung von Expertenwissen sind. Diese Elemente müssen für die prototypische Implementierung in konkrete Softwarekomponenten und Dateien übertragen werden, die eine Ausführung des Szenarios auf Basis von JBoss Drools und der Eclipse RCP erlauben.

Eine der wesentlichen Anforderungen, an das zu entwickelnde Expertensystem, ist die einfache Erweiterbarkeit durch den Wissensingenieur. Daher muss das prototypisch entwickelte System in der Form umgesetzt werden, dass die folgenden drei Bedingungen erfüllt sind:

- **Wenige verschiedene Softwarekomponenten:** Um das Expertensystem für einen Wissensingenieur nicht zu komplex zu gestalten, muss ein Szenario aus wenigen unterschiedlichen Typen von konkreten Elementen bestehen. Diese Elemente sollten die konzeptionellen Elemente eines Szenarios [vgl. 5.5.1] in konkrete Form, also als Softwarekomponente beziehungsweise Dateien, abbilden. Die jeweilige Aufgaben und Fähigkeiten dieser Komponenten müssen klar definiert sein. Auf Grund dieser Bedingung, wird es dem Wissensingenieur möglich, auch den

Ablauf komplexer Szenarien einfach als Software zu modellieren.

- **Einfach strukturierte Softwarekomponenten:** Der Wissensingenieur muss in der Lage sein, die verschiedenen Szenarioelemente zu implementieren, ohne dabei über umfangreiche Softwareentwicklungskenntnisse zu verfügen. Damit dies möglich ist, müssen die Softwarekomponenten eine einfache interne Struktur aufweisen, die schnell erlernt werden kann. Um ein einzelnes Szenarioelement zu erzeugen, darf es beispielsweise nicht nötig sein, komplexe Javapakete mit verschiedenen Objekten und Abhängigkeiten zu definieren.
- **Konventionen statt Konfigurationen:** Um ein Szenario und seine einzelnen Softwarekomponenten in das Expertensystem zu integrieren, darf es nicht erforderlich sein, umfangreiche Konfigurationen oder gar Änderungen an bestehenden Quellcode vorzunehmen. Stattdessen soll eine Erweiterung des Expertensystems um Szenarien durch das Einhalten von vorgegebenen Konventionen ermöglicht sein. Nur einfache Konfigurationen, beispielsweise die Definition, welche Elemente zu einem Szenario gehören, sollen erforderlich sein.

Dynamisches Einbinden von Szenarien

Um neue Szenarien einzubinden oder alte Szenarien zu entfernen, sollen vom Wissensingenieur keine Änderungen am Quellcode vorgenommen werden. Damit dies möglich ist, dürfen zu keiner Zeit Szenarioklassen oder -dateien fest in den Quellcode des Expertensystems eingebunden werden.

Dies führt dazu, dass sich das Expertensystem zu jedem neuen Start in einem Zustand befindet, in welchem es keine Szenarien kennt. Erst zur Laufzeit werden alle verfügbaren Szenarien und die zugehörigen Objekte dynamisch in das Expertensystem geladen und können anschließend von dem Endanwender verwendet werden. Sowohl Java-Klassen als auch andere Datenformate müssen demnach, bei Bedarf dynamisch über entsprechende Mechanismen in das Expertensystem eingebunden werden. Es kann davon ausgegangen werden, dass diese Klassen kompiliert sind und sich in dem Namensraum des entsprechenden Plugins befinden.

Unter Beachtung des Gesamtsystems [vgl. 4.2], das auch die Layer außerhalb des Expert Core einschließt, wird festgelegt, dass die Szenarien dem Expertensystem durch eine externe Datenbank zur Verfügung gestellt werden. Hierbei wird ein Szenario, mit seinen zugehörigen Dateien nach der Erstellung durch den Wissensingenieur und der Freigabe des Release-Managers in diese Datenbank eingepflegt. Anschließend kann das Szenario durch andere Expertensysteminstanzen als Gesamtpaket abgerufen und lokal gespeichert werden. Da zum Zeitpunkt dieser Arbeit die externe Datenbank noch nicht implementiert ist, wird dieser Aspekt nicht berücksichtigt. Stattdessen liegen die Dateien der Szenarien direkt im lokalen Dateisystem.

6.4 Datenmodell für die Szenarioelemente

6.4.1 Szenariopaket

Ein Szenariopaket fasst alle Elemente eines Szenarios zu einer Einheit zusammen. In der umgesetzten Implementierung ist ein Szenariopaket für alle Komponenten des Expertensystems die zentrale Anlaufstelle, um die zu einem Szenario gehörigen Elemente zu erfragen und aufzufinden. Für die Repräsentation eines Szenariopakets wird daher ein Javaobjekt implementiert, das an das Registry-Muster angelehnt ist und als zentrale Registrierung für die Elemente des Szenarios dient. Hierzu werden in dem Szenariopaket Assoziationen zu den Objekten festgehalten, die die anderen Szenarioelemente repräsentieren. Der Zugriff auf diese Assoziationen ist mittels Abfragemethoden (Getter) und Änderungsmethoden (Setter) möglich.

Die Verwendung dieses Ansatzes erlaubt die dynamische Zusammenstellung von Szenarien zur Laufzeit, ohne dass eine Konfiguration des Expertensystems notwendig ist. Die Komponenten des Expertensystems müssen ihre Funktionen nicht mehr auf ihnen bekannte Objekte ausrichten, sondern können diese zur Laufzeit bei einem zentralen Szenarioobjekt erfragen. Es ist hierzu lediglich erforderlich, dass die entsprechenden Szenarioobjekte, zum Zeitpunkt des Systemstarts, einmalig bei dem, für das jeweilige Szenario zuständige, Szenariopaket registriert werden.

In dem entwickelten Konzept soll für diese einmalige Registrierung, eine automatisierte Funktion zur Zusammenstellung der Szenarien eingebaut werden. Hierzu wird eine zentrale Datenbank verwendet, die, nach der Erstellung durch den Wissensingenieur, alle Szenarien und deren Szenarioelemente speichert. Beim Start bezieht des Expertensystem automatisiert, über eine entsprechende SQL-Anfrage, alle verfügbaren Szenarien und deren zugehörige Szenarioelemente. Anschließend werden diese beim zuständigen Szenariopaketobjekt registriert. Da zum Zeitpunkt dieser Arbeit die externe Datenbank noch nicht implementiert ist, wird dieser dynamische Registrierungsvorgang in dem entwickelten Prototyp nicht umgesetzt. Die Szenariozusammenstellung beruht auf einer fest ausformulierten Registrierung über eine Javamethode und geht davon aus, dass alle Objekte zentral im Dateisystem liegen.

Die Java-Klasse für Szenariopakete ist unter der Bezeichnung `ScenarioPackage` in dem Javapaket `de.dlr.xps.core.model.scenario` angesiedelt.

6.4.2 Szenarioelemente

Alle, in das Expertensystem einzubindende Elemente, liegen zunächst als Dateien vor. Bei diesen muss zwischen zwei Typen unterschieden werden. Zum einen gibt es Elemente, die vollständig als Javaobjekte implementiert sind, etwa Faktenobjekte, die als Java Beans umgesetzt sind. Zum anderen existieren Elemente, die in einem anderen

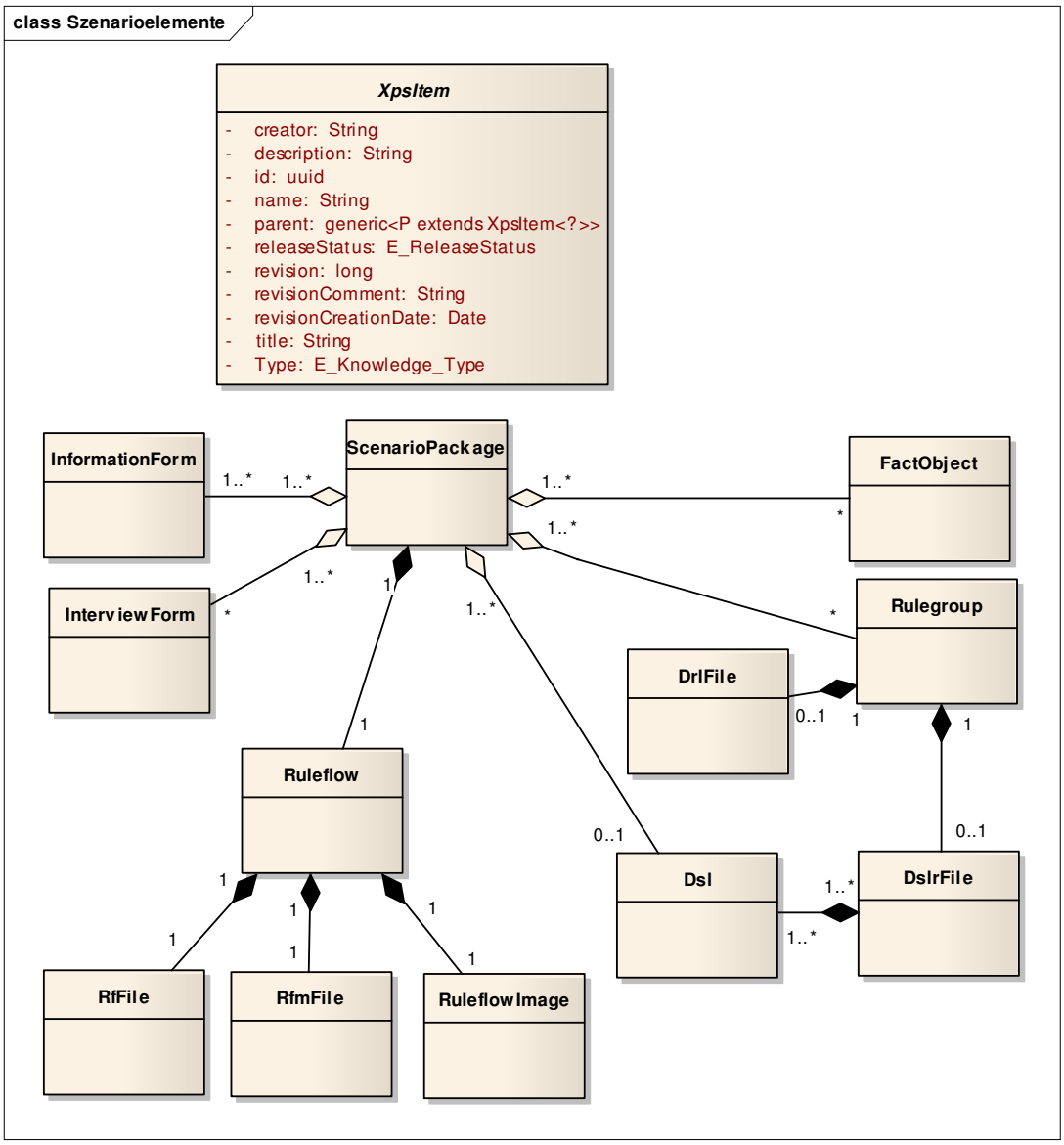

Abbildung 6.2: Klassendiagramm des Datenmodells für Szenarien. Alle Klassen sind Unterklassen von `XPSItem`.

Datenformat vorliegen, etwa als XML-Datei oder als JBoss Drools Regeldatei. Diese Dateien können nicht ohne weiteres dynamisch geladen und direkt als Objekte in dem Szenariopaket registriert werden, da sie zuvor geparst werden müssen. Es besteht für diese Elemente zunächst nur die Möglichkeit, pfadbasierte Verweise auf deren Dateien festzuhalten.

Um dieses Problem abzuschwächen und eine einheitliche Sicht auf alle Szenarioelemente zu gewährleisten, wird ein Datenmodell entwickelt, in dem für jedes Szenarioelement eine umhüllende Klasse erzeugt wird. Diese Klasse verwaltet das Szenarioelement und enthält eine Referenz auf die entsprechende Datei. Sowohl für die Fremdformate, als auch für die Dateien der Javaklassen, wird dieses Konzept angewendet. Dieses Vorgehen bietet darüber hinaus den Vorteil, dass alle Elemente erst zu dem Zeitpunkt in das Expertensystem eingebunden werden müssen, wenn sie gebraucht werden. In der Regel, wenn das entsprechende Szenario gestartet wird. Vor diesem Zeitpunkt können alle notwendigen Operationen auf den umhüllenden Klassen ausgeführt werden und es sind keine komplexen Funktionen für das Parsen beziehungsweise dynamische Einbinden der Szenarioelemente nötig.

Die einheitliche Behandlung aller Szenarioelemente und die strikte Verwendung von umhüllenden Javaklassen, erlaubt eine ausgeprägte objektorientierte Modellierung des Datenmodells. Abbildung 6.2 stellt das entworfene Datenmodell der Szenarioelemente grafisch dar. Die Eigenschaften der konkreten Elemente und Dateien, die durch die Klassen dieses Modells umhüllt sind, werden im weiteren Verlauf der Arbeit dargelegt.

Die folgende Aufzählung gibt einen Überblick über die wesentlichen Dateien, die für die Definition eines Szenarios verwendet werden. Die jeweilige Beschränkung der Anzahl der Dateien, ist eine Konvention, die durch den, aus Sicht des Expertensystems, dynamischen Charakter der Szenarien zu begründen ist. Eine fest definierte Anzahl von Dateien erlaubt einen einfacheren und sicheren Umgang mit den Dateien zur Laufzeit.

- **Ruleflow:** Der Ablauf eines Szenarios wird durch einen einzelnen Ruleflow definiert. Dieser liegt in Form der Drools Ruleflow Dateien mit den Endungen `.rf` und `.rfm` vor.
- **Regelgruppen:** Innerhalb eines Ruleflows werden Regelgruppen aktiviert, die jeweils in einer eigenen Drools-Regel-Datei vom Typ `.drl` vorliegen. Es muss je Regelgruppe genau eine Regeldatei verwendet werden, da dies eine saubere Umsetzung des regelbasierten Mechanismus für die Ermittlung von Fakten erlaubt [vgl. Abschnitt 5.4]. Wird eine domänenspezifische Sprache verwendet, so müssen die Regelgruppen entsprechend, in die hierfür vorgesehenen Dateien vom Typ `.dslr`, vorliegen. Jede `.dslr`-Datei ist mit einer `dsl`-Datei verknüpft, welche die domänenspezifische Sprache für ein oder mehrere Regelgruppen definiert.
- **Fakten:** Die Regelgruppen operieren auf Fakten, die als Java Beans vorliegen. Die

Attribute der Beans bilden die Werte der Fakten ab und werden über die Beans-typischen Getter- und Setter-Methoden befüllt und abgerufen. Jedes Szenario kann über mehrere Faktenklassen verfügen, so dass eine sinnvolle Datenmodellierung umfangreicher Faktendaten möglich ist.

- **Informationsseiten und Interviewseiten:** Diese stellen die individuellen Benutzeroberflächen innerhalb des Szenarios bereit und werden über XML-Dateien definiert, da dies eine einfache Erweiterbarkeit durch den Wissensingenieur ermöglicht. Je Szenario existiert genau eine XML-Datei, die alle Informationsseiten definiert und genau eine für alle Interviewseiten.

Die Pfade zu den lokalen Speicherorten, der zuvor genannten Dateien, werden zunächst in den entsprechenden Objekten des Szenario-Datenmodells festgehalten.

Alle Szenarioelemente sind Unterklassen der abstrakten Klasse `XPSItem`. Diese Vererbung wird aus Gründen der Übersichtlichkeit in der Abbildung nicht dargestellt. Die Klasse `XPSItem` definiert Eigenschaften, die für alle Szenarioelemente verbindlich sind. `XPSItem` vererbt, an die Szenarioelemente, die Verwendung einer eindeutigen Identifizierungskennung, die bei der Konstruktion eines Objekts gesetzt wird. Dies wird mit Hilfe der Java Standardklasse `UUID` implementiert. Anhand dieser Kennung lassen sich alle Szenarioobjekte innerhalb des Expertensystems eindeutig identifizieren. Erzeugt, vergeben und verwaltet werden die Kennungen durch die Singleton-Klasse `UuidGenerator`.

Darüber hinaus bietet die Klasse `XPSItem` ein Konzept an, das die Navigation durch das Datenmodell des Szenarios erleichtert. Es ist möglich, jedem Objekt ein Eltern-Objekt zuzuweisen, das in einem entsprechenden Feld festgehalten wird und durch die Methode `getParent()` abgefragt werden kann. Beispielsweise enthält dieses Feld, für ein RfFile, einen Verweis auf das übergeordnete Ruleflow-Objekt.

Des Weiteren enthält `XPSItem` Felder, die bereits die spätere Umsetzung einer Versionierung der einzelnen Objekte erlauben und einen Release-Prozess unterstützen. Diese Funktionen werden in der vorliegenden Arbeit jedoch nicht weiter betrachtet, da sie Thema der Release-Manager Perspektive sind, die, wie bereits begründet, im Rahmen dieser Arbeit nicht weiter umgesetzt wird.

6.5 Umsetzung des Ruleflow-basierten Szenarioablaufs

6.5.1 Erweitern des org.drools Klassenpfads

Drools ist in einem eigenen Plugin integriert, das alle Java-Bibliotheken von Drools 4.0.7 enthält. Anhand des OSGi-Manifests dieses Plugins ist es möglich, den Zugriff auf diese Bibliotheken sowie deren Klassen und Methoden an Klienten freizugeben. Dies ist ein übliches Vorgehen bei der Entwicklung von Eclipse RCP-Anwendungen und kann entweder über eine Textdatei oder ein spezielles Formular vorgenommen werden.

Diese Freigabe genügt für die Kommunikation, zwischen dem `org.drools`-Plugin und dessen Klienten, dem `dlr.de.xps.core`-Plugin, nicht aus. Drools verwendet, bei der Kompilierung der Regeldateien zu Regelobjekten, Methoden, die es nötig machen, den Klassenpfad des Drools Plugins, um den Klassenpfad des aufrufenden Plugins zu erweitern. Ist dies nicht gegeben, dann findet der Drools Klassenlader die übergebenen Regeldateien nicht. Um die Erweiterung des Klassenpfads vorzunehmen, wird das Eclipse RCP Buddy-Prinzip angewendet, wobei eine manuelle Modifikation der OSGi-Manifeste beider Plugins vorgenommen werden muss [Dau08].

Bei der Umsetzung des Buddy-Prinzips, wird, im Manifest des `org.drools`-Plugins, die Policy definiert, nach der andere Plugins, als so genannte Buddies betracht werden und hierdurch der Namensraum zwischen befreundeten Plugins erweitert werden kann. In dem entwickelten Prototyp wird die Policy „`registered`“ verwendet, so dass der Klassenlader des `org.drools`-Plugin auch Plugins untersucht, die sich bei ihm als Buddy registriert haben. Die notwendige Anweisung lautet: `Eclipse-BuddyPolicy: registered`. Dementsprechend ist es notwendig, dass das OSGi-Manifest des Plugins `dlr.de.xps.core` um die entsprechende Anweisung erweitert wird. Diese lautet: `Eclipse-RegisterBuddy: org.drools`.

6.5.2 Fassade für die Drools Regelmaschine

Um allen Komponenten einen einfachen und zentralen Zugriff auf die, für das Expertensystem notwendigen, Funktionen der Drools Regelmaschine zu bieten, wird das Fassaden-Muster angewendet. Dieses Muster wird in der Klasse `DroolsEngineFacade` des Pakets `de.dlr.xps.core.engine` umgesetzt. Darüber hinaus ist diese Klasse ein Singleton, da global nur eine einzelne Regelmaschinen-Instanz existieren darf und diese über eine einzige Fassade verwaltet werden soll.

Die DroolsEngineFacade dient als zentrale Schnittstelle zwischen dem Expertensystem und der Drools Regelmaschine. Sie erlaubt einen einfachen und spezifischen Zugriff auf dessen Funktionen. Dies wird durch Methoden erzielt, die den Aufgabenstellungen des Expertensystems angepasst sind. Diese Methoden werden auf die komplexere API der DroolsEngine übersetzt. Dabei verwaltet die `DroolsEngineFacade`-Klasse insbesondere die folgenden Aufgaben:

- Initialisierung und Start der Regelmaschine
- Feuern aktivierter Regeln
- Speichern eines Zustands für die Undo- und Speicher-Funktion

Das Klassenmodell der `DroolsEngineFacade` ist in Abbildung 6.3 dargestellt.

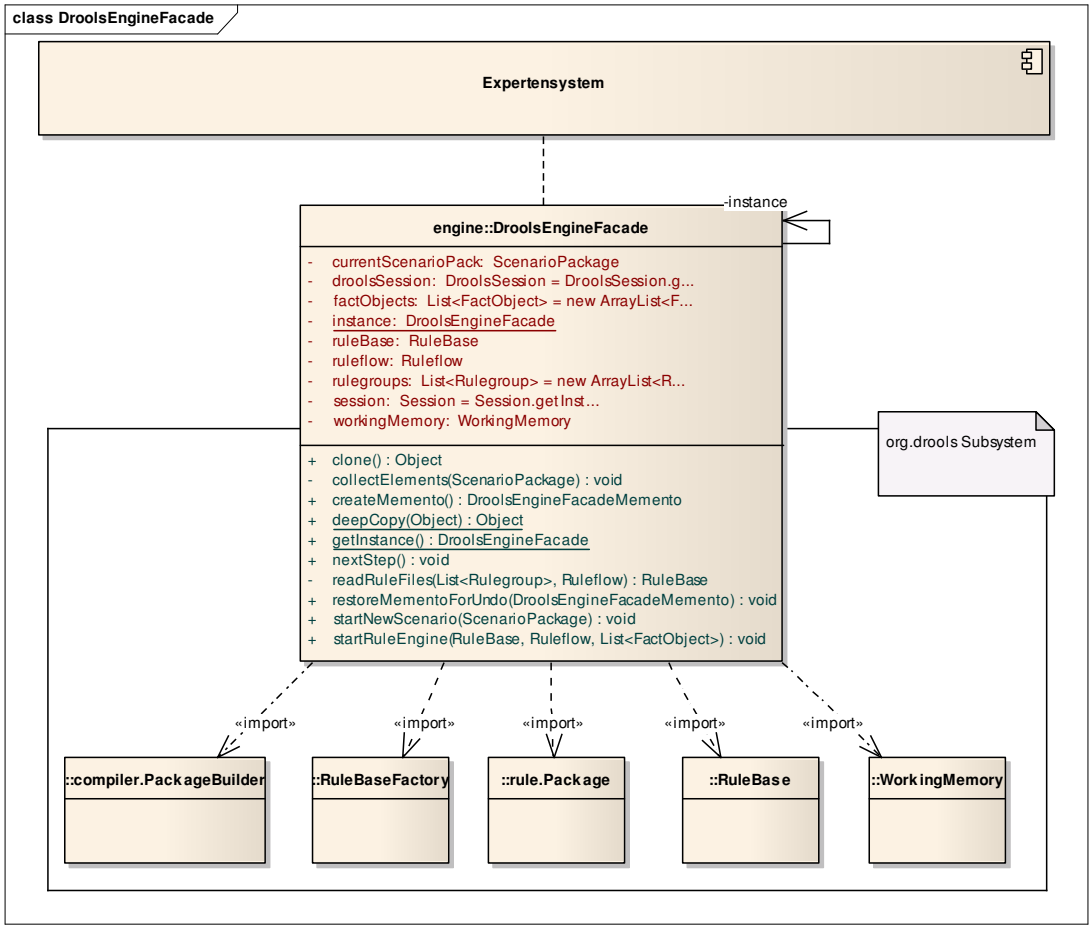

Abbildung 6.3: Das Klassenmodell der `DroolsEngineFacade`, die zentrale Schnittstelle zur Drools Regelmaschine ist.

6.6 Initialisierung der Regelmaschine

Beim Start eines Szenarios werden die Regelmaschine und deren Working Memory initialisiert. Dabei werden die für das Szenario notwendigen Elemente gesammelt, aufbereitet und an die Regelmaschine übergeben. Bei diesen Elementen handelt es sich um den Ruleflow, die Regelgruppen (gegebenenfalls inklusive der DSL-Dateien) und die Faktenobjekte. Eine besondere Herausforderung stellt, an dieser Stelle, der dynamische Charakter dieser Elemente dar. Daher ist es notwendig, die Pfadangaben und Verweise, auf die Dateien aus dem zugehörigen Szenariopaketobjekt, zu ermitteln. Die Initialisierung wird im Prototyp in zwei Schritten umgesetzt.

6.6.1 Schritt 1: Aufbau der Regelbasis

Im ersten Schritt der Initialisierung wird die Regelbasis erzeugt. Dies erfolgt innerhalb der `DroolsEngineFacade` über die Methode `readRuleFiles()`. Listing 6.1 zeigt diese Methode anhand des implementierten Java-Codes. Die Methode erhält als Parameter den Ruleflow sowie eine Liste mit allen, für das Szenario relevanten, Regelgruppen und der zugehörigen DSL (siehe Zeile 1). Diese Liste wird iterativ durchlaufen, wobei ermittelt wird, ob eine DSL existiert. Die Dateien des Ruleflows und die Regelgruppen werden mittels eines `InputStreamReader` eingelesen (Zeile 12, 20, 21 und 28) und an den speziellen Drools-Compiler `PackageBuilder` übergeben (Zeile 13, 22 und 29). Dieser baut die entsprechenden Regelpakete und -objekte (Zeile 32), die anschließend in die neu erzeugte Regelbasis eingefügt werden (Zeile 34 und 35). Vergleiche für die Erzeugung der Regelbasis auch Abschnitt 3.2.2.

Listing 6.1: Initalisierung des Working Memory in der DroolEningeFacadelabel

```
private RuleBase readRuleFiles(List<Rulegroup> myRulegroups, Ruleflow
    myRuleflow){
2   PackageBuilder builder = new PackageBuilder();
  String rulegroupPath = "- warning: no path set -";
4   String ruleflowPath = "- warning: no path set -";
  String pathPrefix = session.getSessionWorkingPath();
6   //Read all ruleGroups
  for(Iterator<Rulegroup> rgIt = myRulegroups.iterator(); rgIt.hasNext();
      ){
8     Rulegroup r = (Rulegroup)rgIt.next();
    //Rulefile without domain specific language
10    if (r.getDrlFile() != null){
      rulegroupPath = pathPrefix + r.getDrlFile().getDrlFilePath();
12      Reader ruleSource = new InputStreamReader (new FileInputStream(
            rulegroupPath));
      builder.addPackageFromDrl(ruleSource);
14    }
    //Rulefile with domain specific language
```

```
16      else if (r.getDslrFile() != null) {
            DslrFile dslr = r.getDslrFile();
18          rulegroupPath = pathPrefix + dslr.getDslrFilePath();
            String dslPath = pathPrefix + dslr.getDsl().getDslFilePath();
20          Reader dsl = new InputStreamReader(new FileInputStream(
                rulegroupPath));
            Reader source = new InputStreamReader(new FileInputStream(dslPath)
                );
22        builder.addPackageFromDrl(source, dsl);
        }
24    }
    //Read ruleflow. Every scenario has to provide a ruleflow
26    if (myRuleflow.getRfFilePath() != null){
      ruleflowPath = pathPrefix + myRuleflow.getRfFilePath();
28    Reader rfSource = new InputStreamReader (new FileInputStream(
          ruleflowPath));
      builder.addRuleFlow(rfSource);
30    }
    //get the compiled package (which is serializable)
32    Package pkg = builder.getPackage();
    //add the package to a rulebase (deploy the rule package).
34    RuleBase ruleBase = RuleBaseFactory.newRuleBase();
    ruleBase.addPackage( pkg );
36    return ruleBase;
  }
```

6.6.2 Schritt 2: Einfügen der Faktenobjekte

Die Übergabe der Faktenobjekte an das Working Memory erfolgt, im Anschluss an den Aufbau der Regelbasis, in einem separaten zweiten Schritt, der mit dem Start des Ruleflow-Prozesses abgeschlossen wird. Diese Aufteilung in zwei Schritte ist damit zu begründen, dass das Einfügen von Fakten in die Faktenbasis bereits das Pattern Matching anstößt und daher eine Trennung von der reinen Initialisierung der Regelmaschine sinnvoll ist.

Faktenobjekte müssen, nach Vorgabe von JBoss Drools, die Java Bean-Spezifikation erfüllen, wodurch der Regelmaschine ein standardisierter Zugriff auf die Faktenfelder ermöglicht wird. Daher ist diese Vorgabe auch eine Konvention für das hier entwickelte Expertensystem. Darüber hinaus, wird das von Drools angebotene Konzept der Java `PropertyChangeListener` angewendet, so dass der Regelmaschine automatisch Veränderungen an einem Fakt mitgeteilt werden. Die Änderung an einem Fakt führt direkt zu einem neuen Pattern Matching und dem Aktivieren der entsprechenden Regeln. Einzig, das Feuern der aktivierten Regeln, muss der Regelmaschine explizit angewiesen werden.

Das Einfügen der Faktenobjekte in das Working Memory und anschließende Star-

ten der Regelmaschine erfolgt, innerhalb der `DroolsEngineFacade`, über die Methode `startRuleEngine()`, die in Listing 6.2 abgebildet ist. Die Faktenobjekte liegen als kompilierte Java-Klassen-Dateien vor. Um den Zugriff der `DroolsEngineFacade` zu ermöglichen, befinden sich diese der Konvention nach im Namensraum des Plugins `de.dlr.xps.core`, genauer, in dessen Paket `engine.scenarioObjects`. Da diese Klassen dynamisch eingebunden werden sollen, werden diese nicht explizit im Quellcode referenziert. Um sie zur Laufzeit in das Expertensystem zu integrieren, muss die `DroolsEngineFacade` die Klassen zunächst dynamisch laden. Im umgesetzten Prototyp erfolgt dies über die statische Java-Methode `Class.forName()` (Zeile 7). Diese erlaubt, Klassen dynamisch anhand eines Strings, der aus dem Namensraum und dem Namen der Klasse besteht, zu beziehen. Bei der Verwendung, dieser Methode in der Eclipse RCP, muss die folgende Besonderheit beachtet werden. Die Methode `Class.forName()` verwendet standardmäßig den allgemeinen Java-Klassenlader des Hostsystems und nicht den speziellen Klassenlader des aufrufenden Eclipse Plugins. Daher wird eine Klasse, die sich im Namensraum des Plugins befindet, nicht gefunden. Um dieses Problem zu beseitigen, wird beim Methodenaufruf explizit die Verwendung des Plugin-Klassenladers angewiesen. Der Namensraum und der Name der Klassen werden anhand der Informationen des Szenario-Datenmodells bezogen.

Nach dem Laden der Klassen, werden im nächsten Schritt entsprechende Objektinstanzen erzeugt (Zeile 10). Auf Grund des dynamischen Charakters, ist es dem Expertensystem nicht bekannt, welchem Klassentypen das zu erzeugende Objekt entspricht. Daher wäre es im weiteren Verlauf der Applikation, außer durch die Verwendung von Java Reflection, nicht möglich, die speziellen Methoden dieser Klasse zu verwenden. Im vorliegenden Prototyp wird ein anderer Ansatz gewählt, da die Untersuchung von Klasseneigenschaften, mittels Java Reflection, häufig eine umfangreiche und statische Prüfung auf Basis von if - else-Anweisungen, erzwingt. Stattdessen wird die Konvention festgelegt, dass alle Faktenobjekte einer Unterklasse von `FactBeanTemplate` angehören. Die abstrakte `FactBeanTemplate`-Klasse dient somit als Vorlage für alle Faktenklassen. `FactBeanTemplate` definiert sowohl allgemeingültige zu vererbende Methoden, etwa zur Registrierung von `PropertyChangeListenern`, als auch abstrakte Methoden, die von den erbenden Klassen individuell ausimplementiert werden müssen. Bei diesen handelt es sich insbesondere um Methoden, die einen standardisierten Zugriff auf die Fakten erlauben. Durch diesen Ansatz ist es möglich, bei der Instanzierung der dynamischen Faktenobjekte eine explizite Typumwandlung auf die `FactBeanTemplate`-Klasse vorzunehmen. Hierdurch können anschließend, die allgemein bekannte Standardmethoden der `FactBeanTemplate`-Klasse, in allen Komponenten des Expertensystems verwendet werden. Der gesamte notwendige Zugriff auf die Faktenobjekte erfolgt über diese Methoden. Das Zusammenspiel zwischen Regelobjekten und Faktenobjekte stellt eine Ausnahme dar. Aus Sicht der Regelobjekte, besitzen Faktenobjekte keinen dynamischen Charakter, sondern sind, zur Ausführungszeit des Szenarios, zwingend im Working Me-

mory enthalten. Daher können Regelobjekte zusätzlich zu den allgemeinen, geerbten Methoden, auch auf die individuellen Methoden der Faktenobjekte direkt zugreifen. Diese sind insbesondere, die Abfragemethoden (Getter) und Änderungsmethoden (Setter) der Faktenobjekte, die eine Grundlage für die Auswertung und Erfüllung einer Regel bilden.

Nachdem das zu erzeugende Faktenobjekt als `FactBeanTemplate` instanziert wurde, wird es in das Working Memory eingefügt (Zeile 11) und an einem zentralen Ort registriert (Zeile 12), so dass es später wieder gefunden werden kann. Das hierfür zuständige `DroolsSession`-Objekt, das auch in das Working Memory aufgenommen wird, wird in Abschnitt 6.7.2 genauer beleuchtet.

Listing 6.2: Einfügen der Faktenobjekte des Szenarios in das Working Memory

```
1  public void startRuleEngine(RuleBase myRulebase, Ruleflow myRuleflow, List
       <FactObject> myFactObjects){
   workingMemory = myRulebase.newStatefulSession();
3  droolsSession = DroolsSession.getInstance();

5  //Initialize all factObjects
   for (FactObject f : myFactObjects) {
7    //For dynamic loading the whole packagename has to be specified
     Class c = Class.forName(f.getFactObjectPackagePath(), true,
         DroolsEngineFacade.class.getClassLoader());
9    //Create class as FactBeanTemplate
     FactBeanTemplate p = (FactBeanTemplate) c.newInstance();
11   workingMemory.insert(p, true );
     droolsSession.registerInitializedFact(p);
13 }

15 workingMemory.insert(droolsSession, true);

17 workingMemory.startProcess(myRuleflow.getRuleflowID());
}
```

Der Ablauf der Initalisierung der Regelmaschine ist in Abbildung 6.4 zusammenfassend dargestellt.

6.7 Informationsaustausch und Modellierung des Ruleflows

6.7.1 Kommunikation zwischen Expertensystem und Drools

Während der Ausführung des Szenarios, ist ein regelmäßiger Informationsaustausch zwischen dem Expertensystem, das zu dieser Zeit, insbesondere die Kommunikation mit dem Anwender verwaltet, und der Drools Regelmaschine, die den Ablauf des Szenarios anhand des Ruleflows und der Regeln bestimmt, nötig.

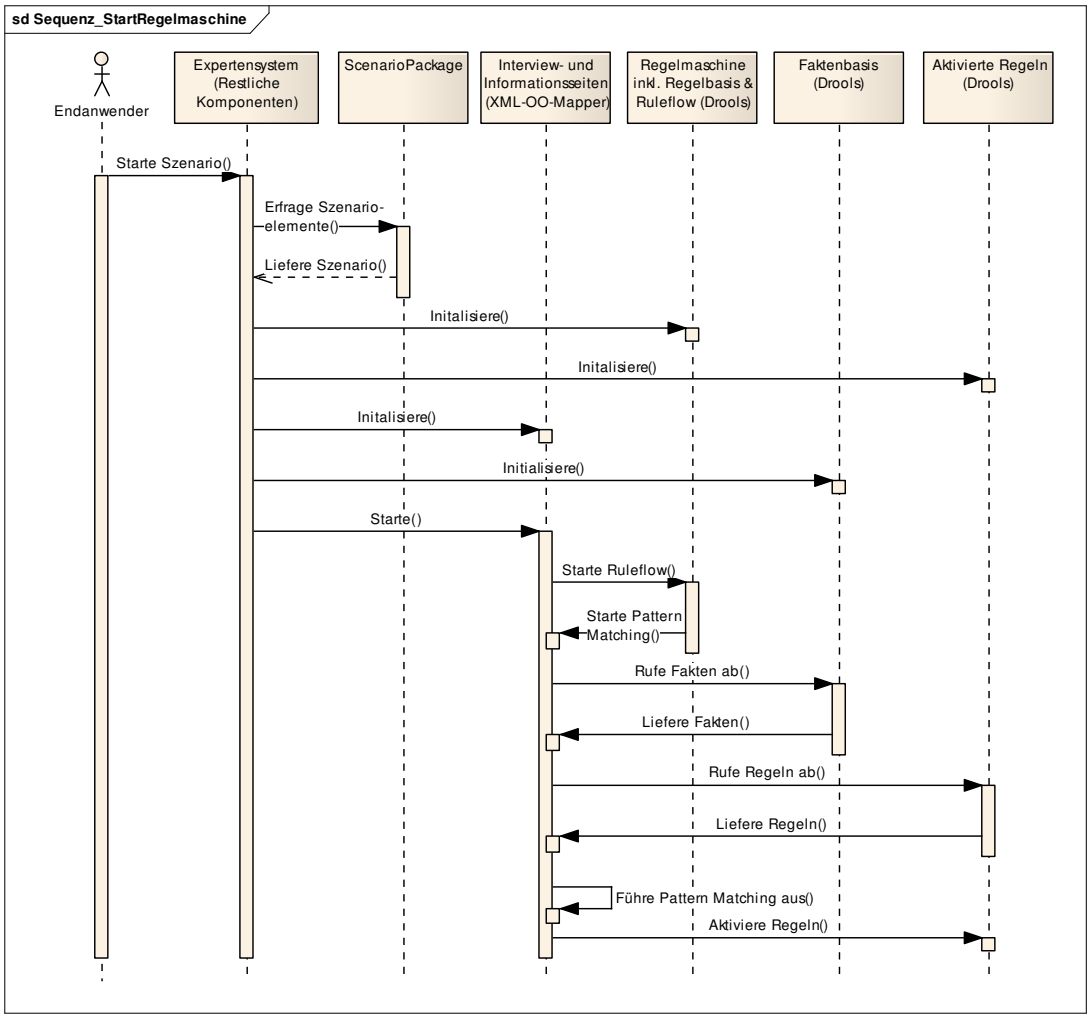

Abbildung 6.4: Sequentieller Ablauf der Regelmaschinen-Initalisierung.

Von dem Expertensystem werden insbesondere fallbasierte Fakten, die der Benutzer eingegeben hat, an das Fakten-Datenmodell weitergereicht, auf dessen Attributen die Drools Regelmaschine operiert. Die Regelmaschine wird durch die `PropertyChangeListener`-Funktion, automatisch über alle Änderung an den Faktenfeldern informiert. Gemäß des entwickelten Konzepts (vgl. 5.3 und 5.4), reicht die Drools Regelmaschine insbesondere, Informationen, über zu aktivierende Interview- und Informationsseiten, an das Expertensystem weiter. Darüber hinaus werden in beide Richtungen gleichermaßen Informationen ausgetauscht, die der Steuerung des Szenarios und des Ruleflows dienen, etwa die Information, dass der Ruleflow weiter fortschreiten soll, da alle Seiten dem Benutzer angezeigt wurden. Für diesen Informationsaustausch ist das folgende Kommunikationskonzept in dem Prototyp implementiert.

Zentraler Austauschpunkt für Informationen, die nicht die Fakten betreffen, ist die Klasse `DroolsSession`, die als Singleton implementiert ist. `DroolsSession` ist an die Idee des Mediator-Muster angelehnt und unterstützt die Endkopplung, bietet jedoch keine Interfaces und Methoden, die den direkten Zugriff zwischen registrierten Objekten delegieren. Stattdessen stellt `DroolsSession` einen zentralen Informationsspeicher dar, in dem die Komponenten der Applikation und die Regelmaschine Informationen ablegen. Hierzu kennen beide die fest definierten Attribute und Methoden der `DroolsSession`. In Richtung der Expertensystemkomponenten wird die Information weitergeleitet, indem die Komponenten, an den notwendigen Stellen, explizit auf die dort abgelegten Informationen zugreifen. Auf diese Weise, erfahren diese beispielsweise, die anzuzeigenden Informationsseiten. In Richtung der Regelmaschine ist das Vorgehen ein Anderes. Die `DroolsSession` ist selbst als ein spezielles Fakt konstruiert und ist eine Unterklasse von `FactBeanTemplate`. Es wird möglich, das `DroolsSession`-Objekt in das Working Memory einzufügen, wodurch es allen Komponenten der Drools Regelmaschine zur Verfügung steht und sowohl den Regelgruppen, als auch dem Ruleflow einen vollständigen Zugriff auf dessen Informationen erlaubt. Durch die Verwendung der `PropertyChangeListener` werden die interessierten Komponenten der Regelmaschine über Änderungen sofort aktiv informiert.

Der beschriebene Informationsaustausch zwischen dem entwickelten Expertensystem und dem Drools-Subsystem ist in Abbildung 6.5 dargestellt.

6.7.2 Modellierung des Ruleflows

Eine Schwierigkeit bei der Abstimmung zwischen Expertensystem und Drools ist es, dass ein typischer Ruleflow, wenn einmal gestartet, immer weiter fortschreitet und nacheinander alle Knoten abarbeitet, bis er den Endknoten erreicht hat. Der Ruleflow wartet währenddessen nicht auf externe Ereignisse. Dieses Verhalten ist dann problematisch, wenn eine Interviewseite angezeigt wird und der Endanwender Fakten eingeben soll. Denn die Regelmaschine hält nicht mit der Ausführung des Ruleflows im Hin-

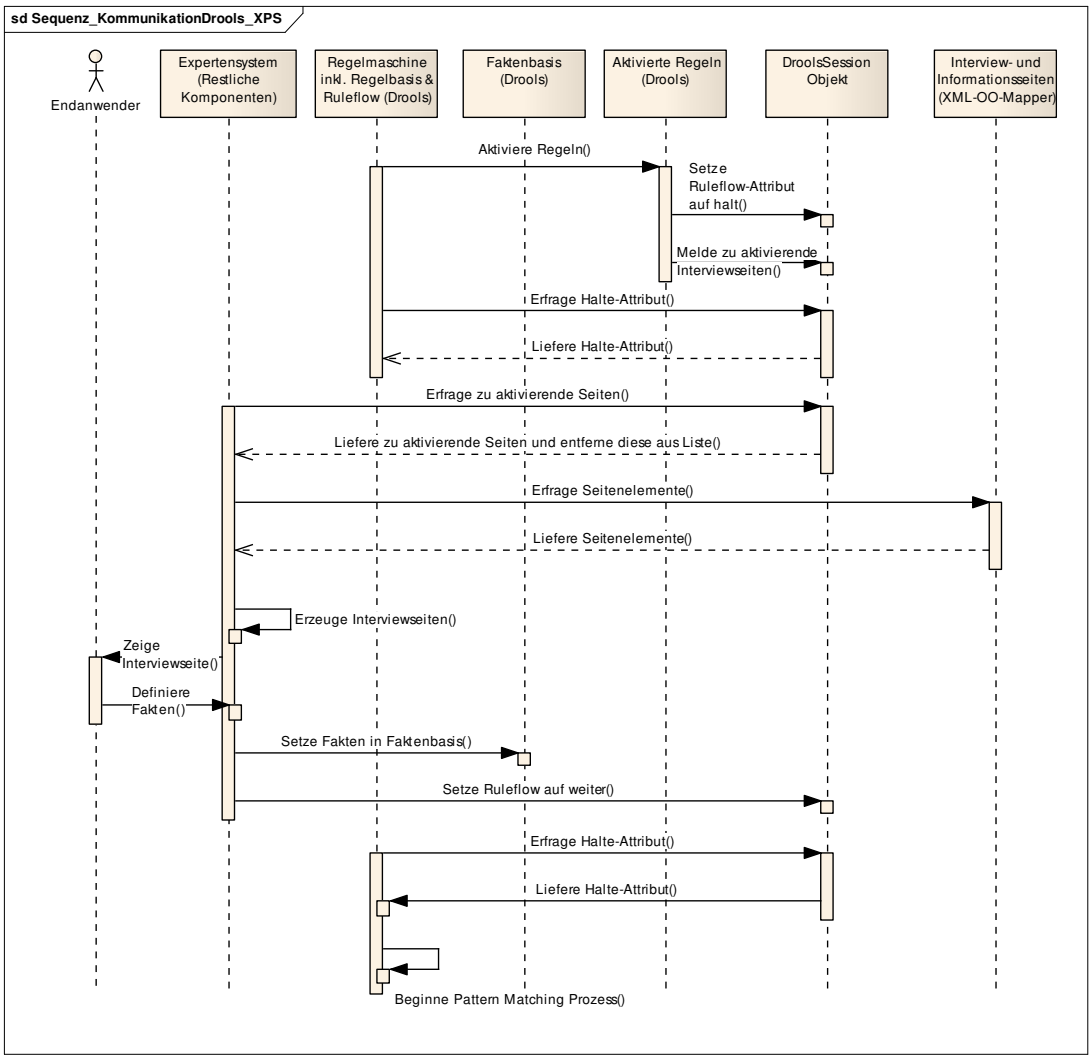

Abbildung 6.5: Sequentieller Ablauf der Kommunikation zwischen dem entwickelten Expertensystem und Drools mittels des `DroolsSession`-Objekts.

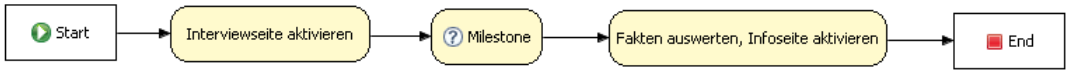

Abbildung 6.6: Darstellung des Milestone-Konzepts. Die Regelmaschine und die Benutzersicht werden synchronisiert. Nach dem Absenden der Interviewseite durch den Anwender wird die Bedingung des Milestones erfüllt und die Ausführung des Ruleflows wird fortgesetzt.

tergrund an, während der Endanwender seine Informationen eingibt. Für die korrekte Ausführung eines Szenarios ist es jedoch in einigen Fällen nötig, dass die Regelmaschine vor dem nächsten Knoten pausiert, bis die Eingaben des Endanwenders in die Faktenbasis eingepflegt sind.

Um das gewünschte Verhalten umzusetzen, wird hinter jedem Knoten, der eine Interviewseite aktiviert, ein Milestone-Knoten in den Ruleflow eingesetzt. Abbildung 6.6 zeigt ein entsprechendes Beispiel. Der Milestone-Knoten hält die Ausführung des Ruleflows an, bis eine bestimmte Bedingung erfüllt ist. Im umgesetzten Prototyp wird hierfür das boolesche Attribut `ruleflowHold` in das `DroolsSession`-Objekt integriert. Die Regeln, die die Aktivierung einer Interviewseite anstoßen, setzen gleichzeitig, dieses Attribut auf `true`. Wird anschließend der Milestone-Knoten erreicht, hält die Ausführung des Ruleflows an. Gleichzeitig werden dem Anwender eine oder mehrere Interviewseiten präsentiert und er kann seine Eingabe tätigen. Nachdem alle Interviewseiten angezeigt und die Faktenbasis aktualisiert wurde, wird das `ruleflowHold`-Attribut im `DroolsSession`-Objekt auf `false` gesetzt. Die Regelmaschine wird durch den `PropertyChangeListener` über diese Änderung informiert und erkennt dadurch, dass die Bedingung des Milestone-Knoten erfüllt ist. Die Ausführung des Ruleflow wird fortgesetzt.

Im Rahmen der Modellierung der Szenarien gibt es Informationsseiten, welche grundsätzlich an einer bestimmten Stelle angezeigt werden sollen, beispielsweise, um dem Anwender direkt nach dem Start einige grundlegende Informationen anzuzeigen. Die Aktivierung dieser Seiten erfolgt nicht auf Basis der Auswertung von fallbasierten Fakten, sondern soll immer erfolgen. Ein nahe liegender Ansatz ist es, die Aktivierung dieser Informationsseiten durch Aktionsknoten an die entsprechende Stelle des Ruleflows zu integrieren. Wobei dieser Knoten eine Aktion auslöst, die die Informationsseite bei dem `DroolsSession`-Objekt registriert. Dieser Ansatz lässt sich jedoch nicht verfolgen, da, in Drools Version 4 Aktionsknoten keinen Zugriff auf die Objekte besitzen die im Working Memory liegen, sondern nur auf globale Objekte. Da das `DroolsSession`-Objekt in dem Working Memory liegt, und die Umsetzung über globale Variablen für den Wissensingenieur umständlich ist, wird der folgende Ansatz verfolgt, der auch vom Drools Entwicklerteam empfohlen wird. Statt Aktionsknoten, werden Regeln verwendet, die

keine Bedingungen besitzen, sondern immer wahr sind, also grundsätzlich feuern. Im umgesetzten Expertensystem ist die einzige, von diesen Regeln ausgeführte, Aktion, das Aktivieren einer Seite über das `DroolsSession`-Objekt. Anstelle eines Aktionsknotens wird die Regelgruppe in den Ruleflow eingefügt, die diese immer erfüllten Regeln umfasst.

6.8 Definition von Benutzeroberflächen und Inhalten

6.8.1 Allgemeine Umsetzung

Die Benutzeroberfläche des Expertensystems wird entsprechend dem allgemeinen Konzept von Eclipse RCP umgesetzt und verwendet Views und Editoren. Dabei unterscheiden sich die dem Anwender zur Verfügung stehenden Elemente je nach Perspektive. In diesem Abschnitt wird die allgemeine Umsetzung der Perspektive für den Endanwender erläutert. Die Perspektive für den Wissensingenieur baut größtenteils auf dem Drools Eclipse Plugin auf und wird in Abschnitt 6.10 beschrieben.

Die Hauptaufgabe der Endanwender-Perspektive ist die Ausführung von vordefinierten Szenarien. Entsprechend der Konvention von Eclipse RCP, werden Tätigkeiten, die direkt der Erfüllung dieser Aufgabe dienen, in einer Editor-Komponente umgesetzt. In dem umgesetzten Prototyp bietet der Editor der Endanwender-Perspektive zwei Hauptfunktionen:

1. **Auswahl eines Szenarios:** Als erster Schritt zur Ausführung eines Szenarios wird die Auswahl eines Szenarios aufgefasst. Hierbei werden dem Endanwender im Editor umfangreiche Informationen zu jedem Szenario angezeigt.
2. **Ausführen eines Szenarios:** Die Ausführung des Szenarios umfasst die Darstellung von Informations- und Interviewseiten, die in dem Editor angezeigt werden.

Neben dem Editor werden unterstützende Views und externe Komponenten verwendet, welche Funktionen anbieten, die nicht der Hauptaufgabe des Expertensystems entsprechen:

1. **Menü-View:** Diese View bietet ein Menü an, das allgemeine Steuerungsfunktionen bereitstellt. Dies sind: „Neues Szenario starten“, „Szenario laden“, „FAQs anzeigen“, „Such-View öffnen“, „Sprache wechseln“.
2. **Such-View:** Die Such-View ermöglicht die freie und die vordefinierte Suche nach externen Informationen innerhalb der Wissenskomponenten des Content Layers. Um diese Funktion anzubieten verwendet das Expertensystem den Service des gemeinsam erstellten Search-Plugins.
3. **Externe Anwendungen:** Zur Darstellung von externen Wissenskomponenten werden entsprechende externe Anwendungen verwendet, welche bei der Auswahl

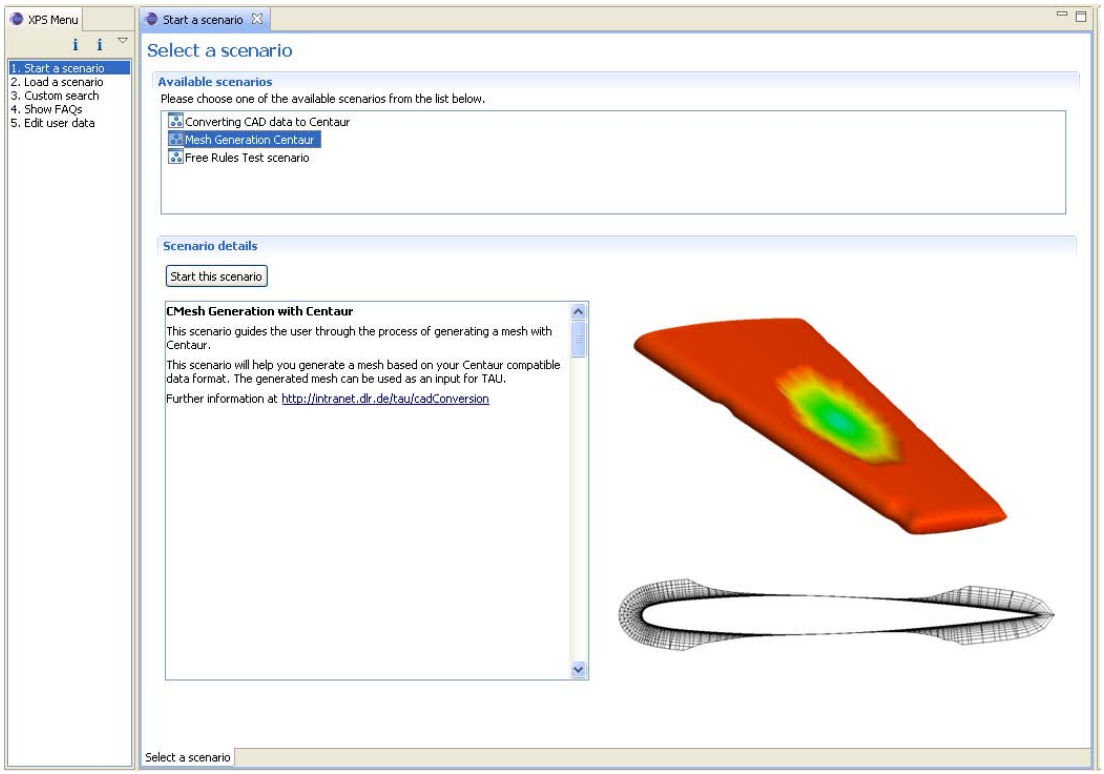

Abbildung 6.7: Bildschirmabbildung der Szenarioauswahl. Diese wurde über das Master-Details Pattern realisiert.

einer Wissenskomponente automatisch gestartet werden. Dies sind beispielsweise HMTL-, Pdf- oder Dokumenten-Betrachter.

Der Editor und die Views werden auf Basis der Bibliotheken SWT, JFace und Eclipse Forms umgesetzt.

Auswahl eines Szenarios

Die Implementierung der Auswahl eines Szenarios in dem Editor erfolgt über das Master-Detail-Muster und der Editor wird in zwei Bereiche eingeteilt. Der Master-Bereich ermöglicht die Auswahl eines Szenarios anhand einer Liste aller verfügbaren Szenarien. Der Details-Bereich beschreibt das im Master-Bereich ausgewählte Szenario auf Basis der vom Wissensingenieur definierten Szenariobeschreibung und erlaubt die Darstellung einer Grafik. Abbildung 6.7 zeigt die umgesetzte Szenarioauswahl.

Die Szenarioauswahl verwendet die Eclipse Forms Bibliothek, die sich an dem Verhalten und dem Aussehen von webbasierten Formularen orientiert und daher dem Benutzer vertraut ist. Eclipse Forms unterstützt die Umsetzung des Master-Detail-Musters indem es die abstrakte Klasse `MasterDetailsBlock` und das Interface `IDetailsPage` anbietet, welche in dem Prototyp verwendet werden. Die Hauptidee ist, dass der Master-Bereich die Auswahlereignisse mittels des JFace `SelectionChangeListener`-Konzepts an den

Details-Bereich sendet. Der Master-Bereich operiert auf einem Datenmodell, das alle verfügbaren Szenarien als Klasse `ScenarioPackage` in einem Array enthält. Ändert sich die Auswahl durch den Benutzer, so wird das ausgewählte `ScenarioPackage`-Objekt an den Details-Bereich gesendet, welches dann die enthaltene Szenariobeschreibung und die Grafik darstellt.

Um das Starten eines Szenarios zu ermöglichen, bietet der Details-Bereich einen entsprechenden Button an. Wird dieser betätigt, so wird das aktuell dargestellte Szenario an ein zentrales Session-Objekt übergeben, das als Singleton umgesetzt ist und allgemeine Informationen zu der aktuellen Session des Expertensystems festhält. Anschließend wird ein neuer Editor geöffnet, der die Ausführung des gewählten Szenarios beginnt.

Darstellung von Interview- und Informationsseiten

Aus Sicht des Endanwenders ist die Darstellung von Interview- und Informationsseiten der zentrale Bestandteil der Szenarienausführung. Diese Seiten werden entsprechend ihrer definierten Reihenfolge in einem Editor angezeigt und der Benutzer kann durch Betätigen eines entsprechenden Buttons vorwärts oder rückwärts navigieren.

Gleich der Seite für die Szenarioauswahl, werden auch die Interview- und Informationsseiten auf Basis von Eclipse Forms implementiert, da dies viele, dem Anwender vertraute, Darstellungsmöglichkeiten bietet.

Ausgangspunkt für die Darstellung ist die Klasse `MainExecutionPage`, die eine Unterklasse der abstrakten `FormPage`-Klasse ist und die Eclipse Form Konzepte verwenden kann. Diese Klasse stellt ein Seitengerüst innerhalb des Editors dar und verfügt über dauerhaft gültige Informationen, wie den Szenarionamen und eine Kurzbeschreibung des Szenarios. Darüber hinaus besitzt diese Klasse einen Platzhalter, der ein Forms-Widget vom Typ `ExpandableComposite` ist. Dieser Platzhalter wird im Lauf des Szenarios mit den situationsabhängigen Elementen der Interview- und Informationsseiten befüllt. Diese Seiten sind somit keine vollständigen eigenen Editorseiten, sondern lediglich ein Teil dieser. Genauer betrachtet sind diese Elemente ein `ExpandableComposite`, welches in das Gesamtgerüst der Seite eingefügt wird. Eine `ExpandableComposite` besteht wiederum aus einzelnen grafischen Elementen, die zusammen die Interview- oder Informationsseite bilden.

Die Erstellung dieser `ExpandableComposites` erfolgt über das Strategie-Muster, wobei die Klasse ExpandableGenerator das Interface der Strategie definiert. Im umgesetzten Prototyp besteht der Algorithmus der Strategie aus der Methode `generateForm()`, die ein `ExpandableComposite` zurückliefert. Implementiert wird die Strategie konkret in den Klassen `InfopageGenerator`, `InterviewGenerator` und `WelcomeGenerator`, die die entsprechenden `ExpandableComposite`-Elemente für Informations-, Interview- und Willkommensseiten erzeugen. Die Willkommensseite wird nur nach dem Start des Sze-

narios angezeigt und besitzt keine Erklärungen und keinen Zurück-Button. Die Auswahl der beiden anderen Strategieklassen erfolgt in der speziellen Klasse `NextStepOperation` und ist abhängig von der Liste der zu aktivierenden Seiten. Das Vorgehen richtet sich nach den in Abschnitt 5.5.3 beschriebenen Konventionen für die Ausführungsreihenfolge von Interview- und Informationsseiten.

Die Verwendung des Strategie-Musters erlaubt eine einfache spätere Erweiterbarkeit des Prototyps und des Szenariokonzepts, so dass während der Ausführung eines Szenarios andere Seitentypen angezeigt werden können. Es müssen lediglich weitere Klassen integriert werden, die die Strategie konkret implementieren und ein `ExpandableComposite` in der gewünschten Form erstellen.

6.8.2 Editierbarkeit von Interview- und Informationsseiten

Vorteile des Einsatzes von XML

Für die sinnvolle Ausführung eines Szenarios müssen von dem Wissensingenieur individuelle Interview- und Informationsseiten definiert werden, die an der entsprechenden Stelle von der Regelmaschine aktiviert und anschließend in dem Editor dargestellt werden. Die Definition dieser Seiten muss für den Wissensingenieur möglichst einfach sein, so dass dieser zielgerichtet und zügig neues Wissen einpflegen kann. Darüber hinaus muss die Definition spezifisch genug sein, um diese Seiten dynamisch in das System einzupflegen und auf Basis dieser Informationen konkrete Seitenelemente darzustellen. Im Fall der Interviewseite müssen die Eingabefelder darüber hinaus den entsprechenden Fakten zugeordnet werden können.

Im Rahmen dieser Arbeit wird das folgende Konzept, welches die beschriebenen Anforderungen erfüllt, implementiert.

Um eine möglichst einfache und korrekte Erstellung von Seiten zu ermöglichen, werden diese anhand von XML-Dokumenten definiert. Diese Entscheidung ist damit zu begründen, dass XML ein weit verbreiteter Standard ist, der bereits vielen Wissensingenieuren bekannt ist beziehungsweise auf Grund des klaren Aufbaus und der umfangreichen Dokumentationen, einfach erlernt werden kann. Darüber hinaus existieren zahlreiche XML-Editoren, die den Wissensingenieur unterstützen. Auch für die Eclipse IDE sind ausgereifte XML-Plugins verfügbar, die in die Wissensingenieur-Perspektive eingefügt werden können.

Ein weiteres Argument für den Einsatz von XML liefert die Möglichkeit XML-Schema einzusetzen. Dies erlaubt es, die gültige Struktur und den gültigen Inhalt von XML-Dokumenten einzuschränken. Durch die Verwendung von XML-Schema ist es folglich möglich, dem Wissensingenieur bereits während der Erstellung der Seitendefinition verschiedene Fehler anzuzeigen, etwa die Ungültigkeit von Attributen oder die Inkorrekt-

heit der Definitionsstruktur. Folglich ist zu erwarten, dass die vom Wissensingenieur erstellten und in das System eingebundenen Seitendefinitionen eine geringe Fehlerrate aufweisen.

XML-Standards zur Definition von Benutzeroberflächen

Bei der Entwicklung einer Lösung auf Basis von XML, welche die Seiten des Szenarios definiert, ist zu beachten, dass bereits standardisierte Lösungen existieren, die die Spezifikation von allgemeinen Benutzerelementen anhand von XML-Dokumenten erlauben. Daher wurden die beiden am weitesten verbreiteten Standards, XUL [Moz] und XAML [Mic], untersucht und auf ihre Einsatztauglichkeit geprüft.

- **XUL:** Der von der Mozilla Foundation entwickelte Standard wird in zahlreichen Mozilla-Anwendung, wie etwa Firefox oder Thunderbird, verwendet um Benutzeroberflächen zu beschreiben. Der Hauptfokus von XUL ist nicht die Definition von Webseiten, sondern von Benutzeroberflächen für plattformübergreifende Applikationen. Daher liegt der Fokus von XUL auf der Definition von Elementen, die typisch für diese Applikationen sind. Die in XUL beschriebenen Elemente können mit einem entsprechenden Konverter in die Zielbibliothek übersetzt werden. Für SWT existieren verschiedene Konverter (bspw. Luxor XUL), die Eclipse Form Bibliothek wird hingegen nicht unterstützt.
- **XAML:** Dies ist ein von Microsoft entwickelter Standard, der die Beschreibung von Benutzeroberflächen ermöglicht. Der Fokus liegt auf Windows-basierten Grafikbibliotheken im .NET-Umfeld, wobei Werkzeuge für die Konvertierung auf andere Plattformen und Bibliotheken existieren. Für SWT existiert der Konverter eFace von Soyatec. Die Übersetzung zu Eclipse Form wird nicht angeboten.

Neben den genannten Standards, die den plattformunabhängigen Einsatz mit verschiedenen Grafikbibliotheken anstreben, existieren spezielle XML-Beschreibungssprachen für die SWT Bibliothek, etwa SWTCook oder Jelly-SWT. [Dau08] bietet einen Überblick über diese speziellen Beschreibungssprachen.

Bei der Betrachtung aller zuvor genannten Ansätze wird deutlich, dass es deren Ziele ist, allgemeine Lösungen für verschiedenste Einsatzszenarien anzubieten. Es werden umfangreiche und mächtige Sprachen bereitgestellt, die sehr viele verschiedene grafische Elemente unterstützen. Diese Komplexität erschwert die Einarbeitung für einen Wissensingenieur, der einen solchen Standard verwenden soll. Durch die zahlreichen Elemente ist es darüber hinaus schwierig, die Szenarioseiten dynamisch in das Expertensystem einzubinden und mit dem Faktenmodell zu verknüpfen. Außerdem kann die Gültigkeit einer erstellten Seite nur schwer geprüft werden. Da anzunehmen ist, dass nur eine begrenzte Zahl verschiedener grafischer Elemente nötig ist, um die Informations- und Interviewseiten vollständig darzustellen, ist die von den Standardlösungen gebotene

Komplexität für den Prototyp unnötig und widerspricht den Anforderungen. Trotz der hohen Anzahl unterstützter Elemente, werden die in dem entwickelten Prototyp verwendeten Elemente der Eclipse Form-Bibliothek von keiner betrachteten Standardlösung unterstützt.

Darüber hinaus muss bei der Verwendung einer externen XML-Beschreibungssprache eine weitere Technologie in die Applikation integriert werden, was zu einer langfristigen Abhängigkeit von einer weiteren Technologie führt. Dies kann bei späteren Entwicklungen des Expertensystems gegebenenfalls Probleme erzeugen, etwa wenn diese Technologie die angestrebte Entwicklung nicht unterstützt oder neue Versionen der Bibliotheken ein ungünstiges Lizenzmodell verwenden. Daher wird die Verwendung externer Lösungen auch aus strategischen Gründen nicht empfohlen.

Auf Basis der genannten Aspekte wird die Entscheidung getroffen, eine individuelle Lösung für die XML-Definition von Interview- und Informationsseiten zu entwickeln.

6.8.3 Entwickelte und umgesetzte Individuallösung zur Seitendefinition

Für die Definition von Informations- und Interviewseiten wird eine individuelle Lösung umgesetzt, die es dem Wissensingenieur erlaubt, Interview- und Informationsseiten vollständig in XML zu definieren. Anschließend werden diese, durch selbst entwickelte Konverterklassen, während der Ausführung des Szenarios in konkrete grafische Elemente umgesetzt. Um die Komplexität der Seiten zu verringern, so dass diese dynamisch in das Expertensystem integriert werden können, wird vorgegeben, welche Elemente auf einer Seite verfügbar sein dürfen und müssen. Diese Spezifikation ist jeweils für Interview- und Informationsseiten anhand von XML-Schema festgelegt. Dadurch kann der Wissensingenieur, bereits während der Erstellung einer Seite, deren strukturelle Gültigkeit sicher stellen. Dies ist durch die Verwendung von XML-Editoren möglich, die eine Prüfung auf Basis von XML-Schema anbieten.

Die Verwendung von selbst entwickelten Konvertern, erlaubt es in den XML-Dateien individuelle Informationen mit den GUI-Elementen zu verknüpfen, welche in Standardlösungen nur schwer einzubauen sind.

Informationsseiten

Eine Informationsseite muss die in der Konzeption (vgl. Abschnitt 5.3) festgelegten Elemente enthalten. Daher wird ein XML-Schema entworfen, das diese Elemente enthält. Listing 6.3 stellt dieses Schema dar. Um eine weitere Fragmentierung des Szenarios zu vermeiden, wird die Entscheidung getroffen, dass die Informationstexte direkt in die XML-Datei integriert werden. Eine Informationsseite besteht somit aus der Definition von Grafikelementen, die angezeigt werden sollen und dem Inhalt, der in diesen Feldern dargestellt werden soll. Darüber hinaus wird zur weiteren Vermeidung der Fragmen-

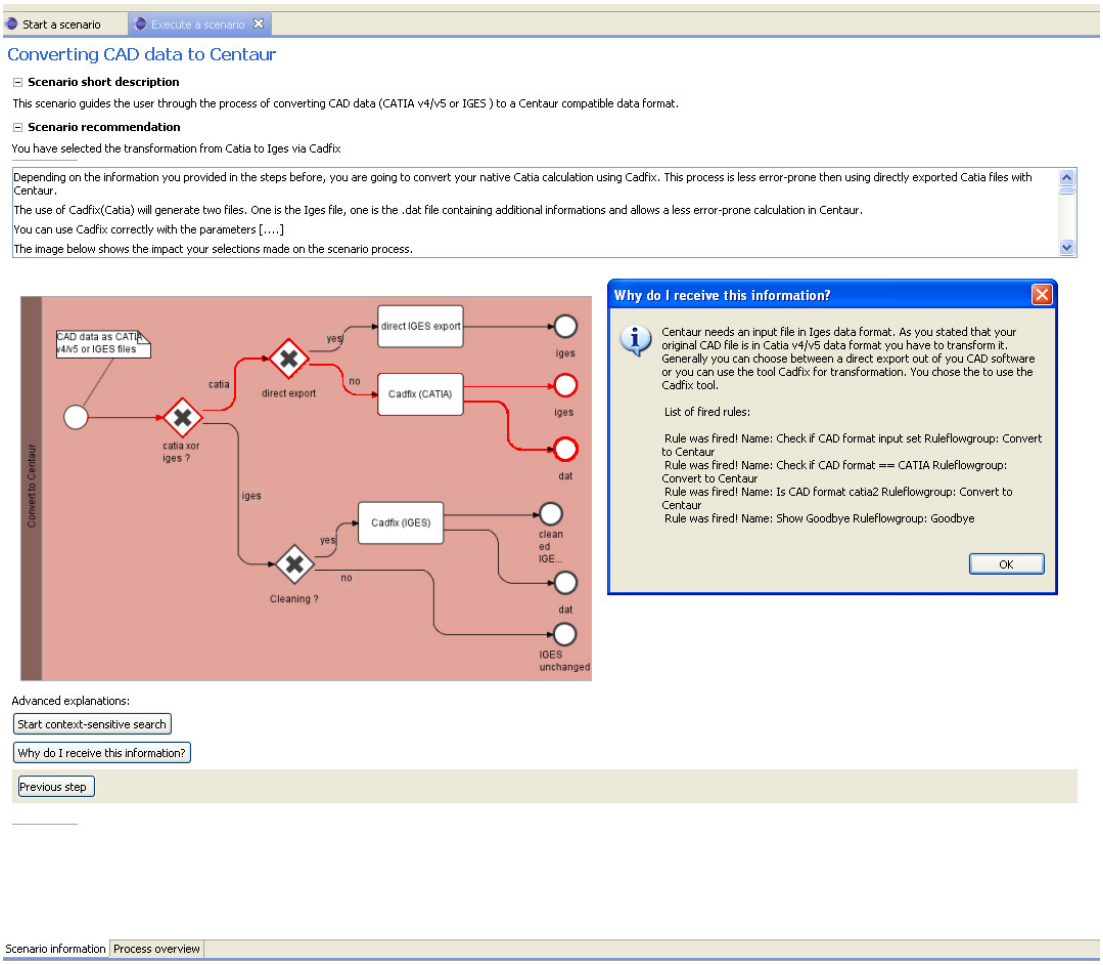

Abbildung 6.8: Beispiel einer Informationsseite, die auf Basis des XML-Schemas erstellt wurde. Es wird ein Erklärung angezeigt, welche auch die bisher gefeuerten Regeln nennt.

tierung, die Konvention aufgestellt, dass nur eine XML-Datei je Szenario verwendet wird. Diese Datei enthält alle Informationsseiten des Szenarios, die jeweils über ein Schlüsselwort (`elementId`) identifiziert werden können. Dies vereinfacht das dynamische Einbinden dieser Seiten. Um Informationstext formatieren zu können, beispielsweise um bestimmten Text fett zu formatieren, wird die XML-basierte Markup-Sprache des `FormText`-Elements verwendet. Dieses Element ist Teil der Eclipse Form Bibliothek. Folglich müssen alle definierten Textelemente in das `FormText`-Element konvertiert werden. Die Anforderung der Mehrsprachigkeit wird erfüllt, indem Texte in Deutsch und Englisch definiert werden können, wobei hierzu eigene XML-Tags und -Datentypen verfügbar sind (Zeile 32 - 37). Um neben statischem Text, auch Informationen dynamisch zur Laufzeit anzeigen zu können, wird das Hilfskonstrukt `dynamicBeanValueType` verwendet (Zeile 24). Dieses Konstrukt implementiert eine Funktion, die es erlaubt Attribute der Faktenklassen anzuzeigen, wie beispielsweise die Ergebnisse von Berechnungen. Eine genaue Beleuchtung wird im folgenden Abschnitt zu Interviewseiten geboten.

Für Umsetzung der Erklärungskomponente ist es möglich, einen Erklärungstext anzugeben, der dem Endanwender darlegt, warum ihm die aktuelle Informationsseite angezeigt wird (Zeile 13). Es ist Aufgabe des Wissensingenieurs diese Erklärung in der Form zu formulieren und mit dem Szenarioablauf zu verknüpfen, dass sie stets eine sinnvolle Erklärung liefert. Im umgesetzten Prototyp kann außerdem ein kompletter Suchausdruck (Zeile 15) oder einzelne Suchbegriffe (Zeile 16) definiert werden, die als Vorgaben der kontext-sensitiven Suche verwendet werden. Abbildung 6.8 zeigt eine beispielhafte Informationsseite.

Listing 6.3: XML-Schema für Informationsseiten

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jaxb="http://
    java.sun.com/xml/ns/jaxb" jaxb:version="1.0">
 <xs:element name="InfoPageElementCollection">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="InfoPageElements" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="elementId" type="xs:string"/>
       <xs:element name="pageDescription" type="multiLangType" minOccurs="
           0"/>
       <xs:element name="informationText" type="multiLangType"/>
       <xs:element name="dynamicBeanValue" type="dynamicBeanValueType"
           minOccurs="0" />
       <xs:element name="pictureLink" type="xs:string" minOccurs="0"
           maxOccurs="unbounded"/>
       <xs:element name="explanationText" type="multiLangType" minOccurs="
           0"/>
       <xs:choice>
        <xs:element name="searchStatement" type="multiLangType" minOccurs=
            "0"/>
        <xs:element name="searchPhrase" type="multiLangType" minOccurs="0"
             maxOccurs="unbounded"/>
       </xs:choice>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="dynamicBeanValueType">
  <xs:sequence>
   <xs:element name="idName" type="xs:string"/>
   <xs:element name="description" type="multiLangType" minOccurs="0"/>
   <xs:element name="factClass" type="xs:string"/>
   <xs:element name="factKeyword" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
```

```
32  <xs:complexType name="multiLangType">
     <xs:sequence>
34    <xs:element name="german" type="xs:string" minOccurs="0"/>
      <xs:element name="english" type="xs:string" minOccurs="0"/>
36   </xs:sequence>
    </xs:complexType>
38 </xs:schema>
```

Interviewseiten

Die Erstellung einer XML-Definition von Interviewseiten folgt der in Abschnitt 5.4 dargelegten Konzeption. Das XML-Schema für Interviewseiten ist in Listing 6.5 enthalten. Dieses ist im Grundaufbau an das Schema für Informationsseiten angelehnt, erweitert es aber um Eingabeelemente.

Bei der Betrachtung des Szenarioablaufs ist erkennbar, dass der dynamische Charakter der Szenarien für die Definition von Interviewseiten eine besondere Herausforderung darstellt. Während Informationsseiten nur an der korrekten Stelle angezeigt werden müssen, ist es bei den Interviewseiten notwendig, diese mit dem Faktenmodell zu verknüpfen. Es darf aber nicht erforderlich sein, dass der Wissensingenieur umfangreichen Java-Code erstellt. Um dies zu ermöglichen wird die Konvention getroffen, dass jedes Eingabeelement zusätzliche Informationen enthalten muss, die es erlauben, dieses Element automatisch auf ein konkretes Attribut im Faktenmodell abzubilden. Diese Information besteht aus zwei Werten. Zum einen wird die Faktenklasse angegeben, in der sich das zu verändernde Fakt befindet und zum anderen ein Schlüsselwort (siehe bspw. Zeile 44 und 45 Listing 6.5). Diese Verwendung von Schlüsselworten ist nötig, da die individuellen Methoden, einer dynamisch eingebundenen Faktenklasse, dem Expertensystem nicht direkt bekannt sind und Java Reflections vermieden werden soll. Daher besitzt jede Faktenklassen eine generische Methode, die es anderen Klassen ermöglicht Werte zusammen mit Schlüsselworten zu übergeben. Hierzu wird eine HashMap verwendet. Anhand der Schlüsselworte ist die Faktenklasse in der Lage, ihre eigenen Faktenfelder zu ändern. Damit dies möglich ist, muss der Wissensingenieur die von `FactBeanTemplate` geerbte Methode `updateFacts` ausimplementieren. In dieser Methode wird die HashMap nach vordefinierten Schlüsselworten durchsucht. Wird ein Schlüsselwort gefunden, dann wird der zugehörige Wert, der aus der Benutzereingabe stammt, dem entsprechenden Attribut der Faktenklasse zugewiesen. Die XML-Datei und die Faktenklassen müssen für dieses Vorgehen lediglich ein gemeinsames Schlüsselwort besitzen. Durch dieses Vorgehen ist das interne Wissen über den Aufbau des Faktenmodells vor dem Expertensystem verborgen. Trotzdem können die Eingabeelemente und Faktenklassen dynamische integriert werden. Durch das Schlüsselwort besitzen beide genügend Informationen, um zur Laufzeit mittels der `updateFacts`-Methode aneinander gekoppelt zu

werden. Das folgende Listing zeigt eine beispielhafte `updateFacts`-Methode innerhalb einer Faktenklasse. Setter- und Getter-Methoden werden nicht dargestellt.

Listing 6.4: Beispiel der updateFacts-Methode die für Faktenklassen verpflichtend ist.

```
public class CadToCentaurFacts extends FactBeanTemplate {
  // Factmodel specific data
  private String cadFormat;
  private Boolean directExport;
  private Boolean cleaning;

  // Check if key/value pair is included, then update the facts
  @Override
  public void updateFacts(HashMap factValues) {
    try{
      if (factValues.containsKey("cadFormat")){
        setCadFormat((String) factValues.get("cadFormat"));
      }
      if (factValues.containsKey("directExport")) {
        setDirectExport((Boolean) factValues.get("directExport"));
      }
      if (factValues.containsKey("cleaning")) {
        setCleaning((Boolean) factValues.get("cleaning"));
      }
    }catch(java.lang.ClassCastException e){
      System.err.println(e);
     }
   }
}
```

Ein ähnliches Konzept wird sowohl in den Interviewseiten als auch in den Informationsseiten verwendet, um aktuelle Informationen aus der Faktenklasse dynamisch in der Seite anzuzeigen. Hierzu wird das Hilfskonstrukt `dynamicBeanValueType` verwendet (siehe Zeile 28 - 35 Listing 6.5). Auch in diesem Fall enthält die XML-Datei Informationen über die Faktenklasse und das Schlüsselwort des anzuzeigenden Werts. Beim erzeugen der Seiten wird diese Information verwendet, um aus der Faktenklasse die relevante Information zu erhalten. Die Faktenklassen besitzt dazu eine Funktion, welche alle internen Fakten in Form einer HashMap liefert. Die Generatorklassen ermitteln anhand des Schlüsselworts den relevanten Wert und fügen diesen, zusammen mit einem in der XML-Datei definierten Text, in die Seite ein. Auf diese Weise können dem Anwender dynamische und aktuelle Informationen zur Laufzeit angeboten werden. Diese Informationen können beispielsweise auf einer Berechnungsfunktion oder einer aktivierten Regel basieren.

Im Rahmen der prototypischen Implementierung, welche das Ziel hat ein Machbarkeitsnachweis zu erbringen, werden zunächst nur die, für die Erfassung der Fakten, notwendigen grafischen Eingabeelemente unterstützt. Anhand der Analyse von typischen

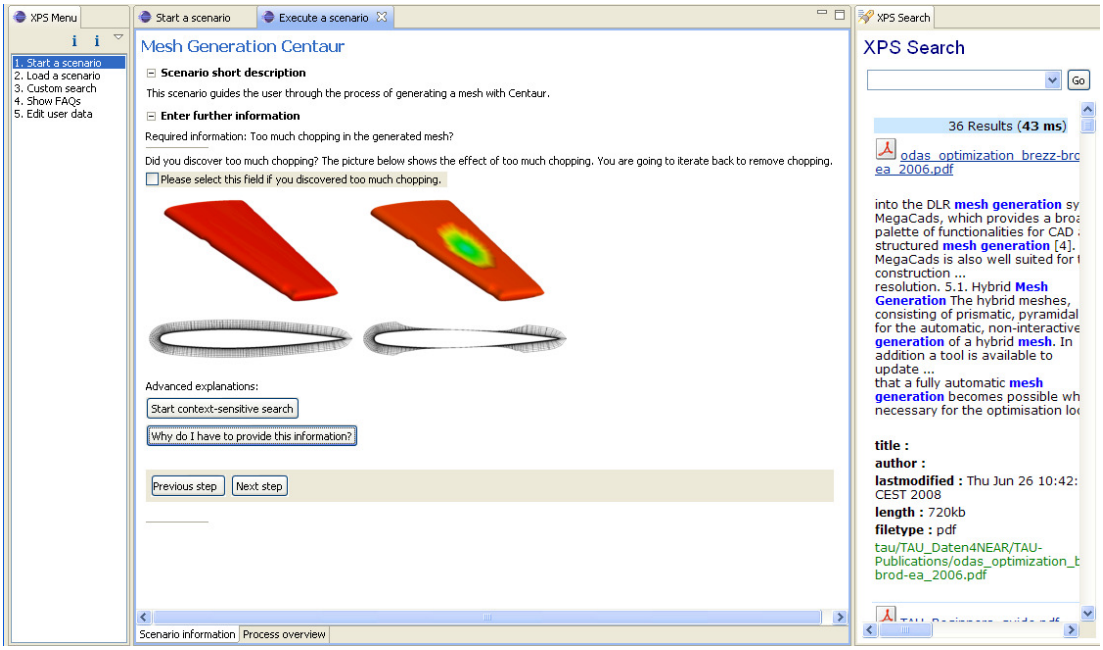

Abbildung 6.9: Beispiel einer Interviewseite, die über das XML-Schema definiert wurde und eine Checkbox verwendet. Rechts werden die kontextsensitiven Suchergebnisse angezeigt, die auf den Suchbegriffvorgaben des Wissensingenieurs beruhen.

Faktenmodellen kann ermittelt werden, dass nur Eingabefelder nötig sind für Texte, Zahlen, boolesche Werte, sowie die Auswahl aus einer Menge vorgegebener Werte, etwa über eine Liste. Die folgenden grafischen Eingabeelemente werden in der implementierten Umsetzung unterstützt.

1. **Eingabefeld für Text:** Ein Feld, das dem Benutzer die freie Eingabe von Informationen erlaubt. Vom Wissensingenieur können vorab die gültigen Eingaben definiert werden, so dass diese bereits bei der Eingabe überprüft werden können. (Zeile 37 - 47 Listing 6.5)
2. **Eingabefeld für Zahlen:** Der Benutzer kann frei in dieses Feld Zahlen eingeben. Der gültige Wertebereich kann durch eine obere und untere Grenze bestimmt werden. (Zeile 48 - 60 Listing 6.5)
3. **Checkbox:** Diese ermöglicht dem Benutzer die Eingabe von booleschen Werten. Das heißt, er kann durch das Ankreuzen der Checkbox das vorgegebene Attribut als wahr markieren. Kreuzt er die Checkbox nicht an, dann wird das vorgegebene Attribut automatisch als falsch eingestuft. (Zeile 61 - 71 Listing 6.5)
4. **Combo:** Dieses Eingabeelement stellt eine Dropdown-Liste dar und ermöglicht die exklusive Auswahl eines Wertes aus mehreren vordefinierten Werten. Dieses Element erlaubt es beispielsweise ein „Wähle 1 aus 5“ anzubieten. (Zeile 72 - 83 Listing 6.5)

Diese Auswahl kann bei Bedarf später auf andere Eingabeelemente und - eigenschaften erweitert werden. Dazu muss das XML-Schema angepasst werden und eine Umsetzung auf die entsprechenden grafischen Elemente in das Expertensystem integriert werden.

Um eine Fragmentierung der Szenarien zu vermeiden, besteht, wie bei der Definition von Informationsseiten, auch hier die Konvention, dass alle Interviewseiten eines Szenarios gemeinsam in einer Datei festgehalten werden und durch ein Schlüsselwort identifiziert werden. Die Abbildung 6.9 zeigt eine beispielhafte Interviewseite.

Listing 6.5: XML-Schema für Interviewseiten

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jaxb="http://
    java.sun.com/xml/ns/jaxb" jaxb:version="1.0">
 <xs:element name="GuiElementCollection">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="interviewElements" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="elementId" type="xs:string"/>
       <xs:element name="formDescription" type="multiLangType" minOccurs="
           0"/>
       <xs:element name="dynamicBeanValue" type="dynamicBeanValueType"
           minOccurs="0" />
       <xs:element name="textInput" type="textInputType" minOccurs="0"
           maxOccurs="unbounded"/>
       <xs:element name="numberInput" type="numberInputType" minOccurs="0"
            maxOccurs="unbounded"/>
       <xs:element name="checkbox" type="checkboxType" minOccurs="0"
           maxOccurs="unbounded"/>
       <xs:element name="combo" type="comboType" minOccurs="0" maxOccurs="
           unbounded"/>
       <xs:element name="pictureLink" type="xs:string" minOccurs="0"
           maxOccurs="1"/>
       <xs:element name="explanationText" type="multiLangType" minOccurs="
           0"/>
       <xs:choice>
        <xs:element name="searchStatement" type="multiLangType" minOccurs=
            "0"/>
        <xs:element name="searchPhrase" type="multiLangType" minOccurs="0"
             maxOccurs="unbounded"/>
       </xs:choice>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="dynamicBeanValueType">
  <xs:sequence>
```

```
    <xs:element name="idName" type="xs:string"/>
30  <xs:element name="description" type="multiLangType" minOccurs="0"/>
    <xs:element name="factClass" type="xs:string"/>
32  <xs:element name="factKeyword" type="xs:string"/>
   </xs:sequence>
34 </xs:complexType>
   <xs:complexType name="textInputType">
36  <xs:sequence>
    <xs:element name="idName" type="xs:string"/>
38  <xs:element name="description" type="multiLangType" minOccurs="0"/>
    <xs:element name="toolTip" type="multiLangType" minOccurs="0"/>
40  <xs:element name="label" type="multiLangType"/>
    <xs:element name="allowedValue" type="xs:string" minOccurs="0"
        maxOccurs="unbounded"/>
42  <xs:element name="factClass" type="xs:string"/>
    <xs:element name="factKeyword" type="xs:string"/>
44  </xs:sequence>
   </xs:complexType>
46 <xs:complexType name="numberInputType">
   <xs:sequence>
48  <xs:element name="idName" type="xs:string"/>
    <xs:element name="description" type="multiLangType" minOccurs="0"/>
50  <xs:element name="toolTip" type="multiLangType" minOccurs="0"/>
    <xs:element name="label" type="multiLangType"/>
52  <xs:element name="lowerRange" type="xs:double" minOccurs="0"/>
    <xs:element name="upperRange" type="xs:double" minOccurs="0"/>
54  <xs:element name="factClass" type="xs:string"/>
    <xs:element name="factKeyword" type="xs:string"/>
56  <xs:element name="absolutePosition" type="xs:integer" minOccurs="0"/>
   </xs:sequence>
58 </xs:complexType>
   <xs:complexType name="checkboxType">
60  <xs:sequence>
    <xs:element name="idName" type="xs:string"/>
62  <xs:element name="description" type="multiLangType" minOccurs="0"/>
    <xs:element name="toolTip" type="multiLangType" minOccurs="0"/>
64  <xs:element name="label" type="multiLangType"/>
    <xs:element name="factClass" type="xs:string"/>
66  <xs:element name="factKeyword" type="xs:string"/>
    <xs:element name="absolutePosition" type="xs:integer" minOccurs="0"/>
68  </xs:sequence>
   </xs:complexType>
70 <xs:complexType name="comboType">
   <xs:sequence>
72  <xs:element name="idName" type="xs:string"/>
    <xs:element name="description" type="multiLangType" minOccurs="0"/>
74  <xs:element name="toolTip" type="multiLangType" minOccurs="0"/>
    <xs:element name="label" type="multiLangType"/>
76  <xs:element name="values" type="xs:string" minOccurs="0"  maxOccurs="
```

```
        unbounded" />
    <xs:element name="factClass" type="xs:string" />
    <xs:element name="factKeyword" type="xs:string" />
    <xs:element name="absolutePosition" type="xs:integer" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="multiLangType">
   <xs:sequence>
    <xs:element name="german" type="xs:string" minOccurs="0" />
    <xs:element name="english" type="xs:string" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Einbinden der Seiten in das System

Die Regelmaschine bestimmt den Zeitpunkt, an dem die Seiten in das Expertensystem eingebunden werden. Dabei übermitteln Regeln den entsprechenden Bedarf an eine zentrale Stelle. In dem entwickelten Expertensystem ist diese zentrale Stelle die Singleton-Klasse `DroolsSession`, die Methoden anbietet, um zu aktivierende Seiten zu registrieren und abzurufen. Die Bestimmung der Seiten basiert auf einem Schlüsselwort, das eindeutig für eine Seite steht und zwischen aktivierender Regel, beziehungsweise Aktionsknoten, und dem entsprechenden Element in dem XML-Dokument abgestimmt sein muss. Die Klasse `NextStepOperation` verwaltet den Einbindungsprozess von Seiten und setzt die in Abschnitt 5.5.3 beschriebene Aktivierungsreihenfolge um.

Nachdem die zu aktivierende Seite ermittelt wurde, wird diese erstellt. Diese Aufgabe übernehmen, abhängig vom Seitentyp, die Klassen `InterviewGenerator` und `InfopageGenerator`. Diese implementieren die Strategie der Klasse `ExpandableGenerator` aus und erzeugen ein `ExpandableComposite`, das in das Seitengerüst des Editors eingebunden wird.

Um die XML-Definition der Seiten einfach auslesen und verwalten zu können, verwendet der Prototyp die externe Bibliothek JAXB [Sunb] von Sun Microsystems, welche in der Version 2.0 in dem Java SE Development Kit (JDK) 6 enthalten ist. JAXB erlaubt den Zugriff auf XML-Daten und deren Bindung an Java-Objekte, ohne hierfür XML-Paser wie SAX oder DOM direkt verwenden zu müssen. Ein JAXB-Compiler erzeugt einmalig einen Baum von Java-Klassen auf Basis des Schemas der XML-Dokumente. Diese Klassen können in eine Applikation eingebunden werden und verfügen über Attribute, die dem definierten XML-Schema entsprechen. Ein Vorgang der als Unmarshalling bezeichnet wird, erlaubt, über die JAXB API, zur Laufzeit das einfache Binden eines XML-Dokuments an die erzeugten Klassen. Durch komfortable Abfragemethoden können die Daten des Dokuments abgerufen und in anderen Komponenten des Exper-

tensystems verwendet werden. Im Prototyp sind die Klassen `XmlOOInfopageMapper` und `XmlOoInterviewMapper` für das Auslesen und Sammeln der Daten auf Basis der JAXB API zentral verantwortlich und stellen alle Elemente der XML-Datei gesammelt als Liste zur Verfügung. Die Klassen `InfopageGenerator` und `InterviewGenerator` verwenden die Elemente dieser Listen und setzen diese in die entsprechenden grafischen Elemente um. Im Fall der Infoseiten wird ein vordefiniertes Skelett mit den vorhandenen Daten befüllt und alles gemeinsam als `ExpandableComposite` in die Editorseite integriert.

Die Erzeugung von Interviewseiten stellt sich komplexer dar, da die Elemente der Seite vom Wissensingenieur frei definiert werden können, die Eingaben auf Gültigkeit überprüft und darüber hinaus an das Faktenmodell gebunden werden müssen. Das Erzeugen einer Interviewseite besteht aus drei Teilen:

- **Erzeugen der Eingabeelemente:** Die vom `XmlOoInterviewMapper` übergebene Liste wird durchlaufen und abhängig vom enthaltenen Elementtyp werden die entsprechenden Eingabeelemente erzeugt und auf der Seite platziert. Des Weiteren wird jedes Element, zusammen mit allen Zusatzinformationen (beispielsweise erlaubte Eingabewerte, zugehörige Faktenklasse und Fakten-Schlüsselwort), in die spezielle Klasse `WidgetContainer` eingebettet und in eine Liste, die alle für diese Seite erzeugten `WidgetContainer` enthält, abgelegt.
- **Umsetzung der Gültigkeitsprüfung:** Die Gültigkeitsprüfung wird umgesetzt, indem der „Weiter“-Button der Seite mit einem Ereignis verknüpft wird, der die Prüfung auslöst. Hierzu wird eine Methode integriert, welche die Prüfung auf Basis der Informationen des `WidgetContainer` vornimmt. Nicht gültige Eingaben werden dem Anwender direkt im Formular angezeigt. Der Funktionsumfang der Prüfung kann in späteren Versionen des Systems leicht innerhalb der zuständigen Komponente erweitert werden.
- **Umsetzung der Bindung von Eingaben an das Faktenmodell:** Die Übergabe von Eingaben an die Faktenklassen wird beim Betätigen des „Weiter“-Buttons ausgelöst. Die zugehörige Faktenklasse wird anhand der Informationen des `WidgetContainer` ermittelt, wobei diese Informationen mit einer Liste abgeglichen wird, die alle Faktenobjekt enthält. Stimmen beide überein und die zugehörige Faktenklasse ist ermittelt, dann werden dieser die zugehörigen eingegebenen Werte weitergereicht. Da auf Grund des dynamischen Charakters, die Methoden der Faktenklasse dem System nicht bekannt sind, wird die bereits zuvor beschriebene generische Methode `updateFacts` verwendet. Diese Methode kann aufgerufen werden, da sie von der Superklasse `FactBeanTemplate` geerbt wird und daher in allen `FactBeans` vorhanden ist. In dieser Methode werden die übergebenen Fakten anhand der Schlüsselwerte den Attributen zugewiesen und die vom Anwender eingegebenen Werte sind in das Faktenmodell übertragen.

6.9 Weitere Aspekte der technischen Realisierung

6.9.1 Speicher- und Undo-Funktion

In dem entwickelten Prototyp basiert die Umsetzung der Speicher- und Undo-Funktion auf demselben Konzept. Dabei wird die Idee verfolgt, an einen bestimmten Zeitpunkt die internen Informationen der Komponenten festzuhalten, welche für den aktuellen Zustand eines Szenarios relevant sind. Durch das spätere Abrufen und Widerherstellen der gespeicherten Informationen ist das System in der Lage, den eigenen Original-Zustand zu rekonstruieren. Ein Vorteil dieses Ansatzes ist es, dass derselbe Grundmechanismus sowohl für die Speicher-Funktion als auch für die Undo-Funktion verwendet werden kann, was die Komplexität des Systems verringert und die Wartbarkeit erhöht.

Ein anderer Ansatz die Undo-Funktion umzusetzen wäre, die in einem Schritt ausgeführten Methoden und Operationen festzuhalten und lediglich diese, durch entsprechende zugehörige Undo-Operationen, wieder rückgängig zu machen. Es wird nicht der Zustand gespeichert, sondern es werden Operationen angeboten, die durch verknüpfte Undo-Operationen aktiv rückgängig gemacht werden können. Dieser Ansatz ist insbesondere in einem regelbasierten System problematisch, da es äußerst schwer ist, die durch die aktivierten Regeln ausgelösten Operationen vollständig nachzuvollziehen und darüber hinaus entsprechende Undo-Operationen anzubieten, die alle Änderungen aktiv rückgängig machen. Das von Drools angebotene Truth Maintenance hilft hierbei nur beschränkt und erlaubt es maximal den Zustand der Agenda-Liste und der Faktenbasis wiederherzustellen.

Für die Umsetzung der Speicher- und Undo-Funktion wird im Prototyp ein Ansatz verwendet, der sich an der Idee des Memento-Musters orientiert. Hierbei erstellt ein Urheber-Objekt, welches seinen internen Zustand speichern möchte, ein spezielles Memento-Objekt, in das es seine, für den aktuellen Zustand relevante, internen Attribute kopiert. Dieses Memento-Objekt ist gegen den Zugriff durch andere Objekte, als dem Urheber-Objekt, geschützt. Ein Aufbewahrer nimmt das Memento-Objekt entgegen und verwaltet dies, ohne die internen Werte des Memento-Objekts zu verändern. Soll der vorherige Zustand wiederhergestellt werden, wird das Memento-Objekt an das Urheber-Objekt übergeben. Dieses stellt seinen internen Zustand selbstständig wieder her, indem es seine eigenen Attribute mit den Attributen des Memento-Objekts überschreibt.

Im Prototyp ist die Klasse `DroolsEngineFacade` das Urheber-Objekt, da diese, als alleinige Schnittstelle zu dem Drools System, als Einzige Zugriff auf alle zustandsrelevanten Komponenten besitzt. `DroolsEngineFacade` verfügt über eine private innere Memento-Klasse, die vor dem Zugriff durch andere Klassen geschützt ist. Die Klasse `NextStepOperation`, als Verwalter des Szenarioablaufs, übernimmt die Funktion des Aufbewahrers. Diese Klasse fordert, vor dem Auslösen des Fortschreitens im Szenario,

ein Memento-Objekt von der `DroolsEngineFacade` an und speichert dies.

Um den Zustand eines Szenarios vollständig festhalten zu können, müssen die Informationen identifiziert werden, die den Zustand beschreiben. In dem Prototyp werden die folgenden Komponenten bestimmt, die dies gewährleisten:

- **Drools Working Memory:** Das Drools Working Memory-Objekt bietet Zugriff auf das interne Subsystem der Drools Regelmaschine inklusive des Ruleflows und der Agenda-Liste. Der Zustand dieses Objekts und der referenzierten Objekte muss festgehalten werden, da diese den Zustand der gesamten Regelmaschine bündeln. Auf die Komponenten des Subsystems kann nicht einzeln zugegriffen werden, so dass das Working Memory-Objekt, inklusive seiner referenzierten Klassen vollständig kopiert und gespeichert werden muss.
- **Faktenklassen:** Faktenklassen enthalten die vom Benutzer eingegebenen Informationen auf denen die Regelmaschine operiert. Der Zustand aller Faktenklassen muss gespeichert werden. Um das Memento-Muster vollständig umzusetzen, müsste jede Faktenklasse wiederum eine eigene innere Memento-Klasse besitzen, in welche diese ihre Attribute kopiert. Dies führt zu einem stark erhöhten Aufwand bei der Erstellung der Java Faktenklassen und widerspricht der Anforderung, dass Szenarios einfach durch einen Wissensingenieur erstellt werden sollen. Daher werden Kopien der Faktenobjekte selbst gespeichert und nicht deren Memento-Objekt. Diese Kopien werden in dem Memento-Objekt der `DroolsEngineFacade` festgehalten.
- **DroolsSession:** Diese Klasse enthält insbesondere Steuerungsinformationen für den weiteren Verlauf des Szenarios, so dass dessen Zustand gespeichert werden muss. Da es sich bei dieser Klasse um einen Singleton handelt, darf die Klasse nicht einfach kopiert werden. Stattdessen werden deren Attribute in ein spezielles `DroolsSessionMemento`-Objekt kopiert, das eine innere Klasse von `DroolsSession` ist. Dieses `DroolsSessionMemento`-Objekt wird in dem Memento-Objekt der `DroolsEngineFacade` festgehalten.

Abbildung 6.10 veranschaulicht das im Prototyp implementierte Memento-Muster.

Da die Faktenklassen und das Working Memory vollständig kopiert werden müssen, also inklusive der referenzierten Objekte, die diese enthalten (so genanntes Deep-Copying), eignet sich insbesondere das Kopieren über die Serialisierungsfunktion von Java [HC08]. Hierbei können Objekte und deren Referenzen kopiert beziehungsweise geklont werden, ohne dass deren `clone()`-Methode überschrieben wird. Stattdessen wird das Originalobjekt in ein Bytearray geschrieben und anschließend in das Zielobjekt (die Kopie) zurück überführt. Damit dieser Ansatz umsetzbar ist, muss das Originalobjekt das `Serializable`-Interface implementieren. In dem Prototyp wird diese Bedingung in dem `FactBeanTemplate` erfüllt und an alle Faktenklassen vererbt. Da das `DroolsSession`-

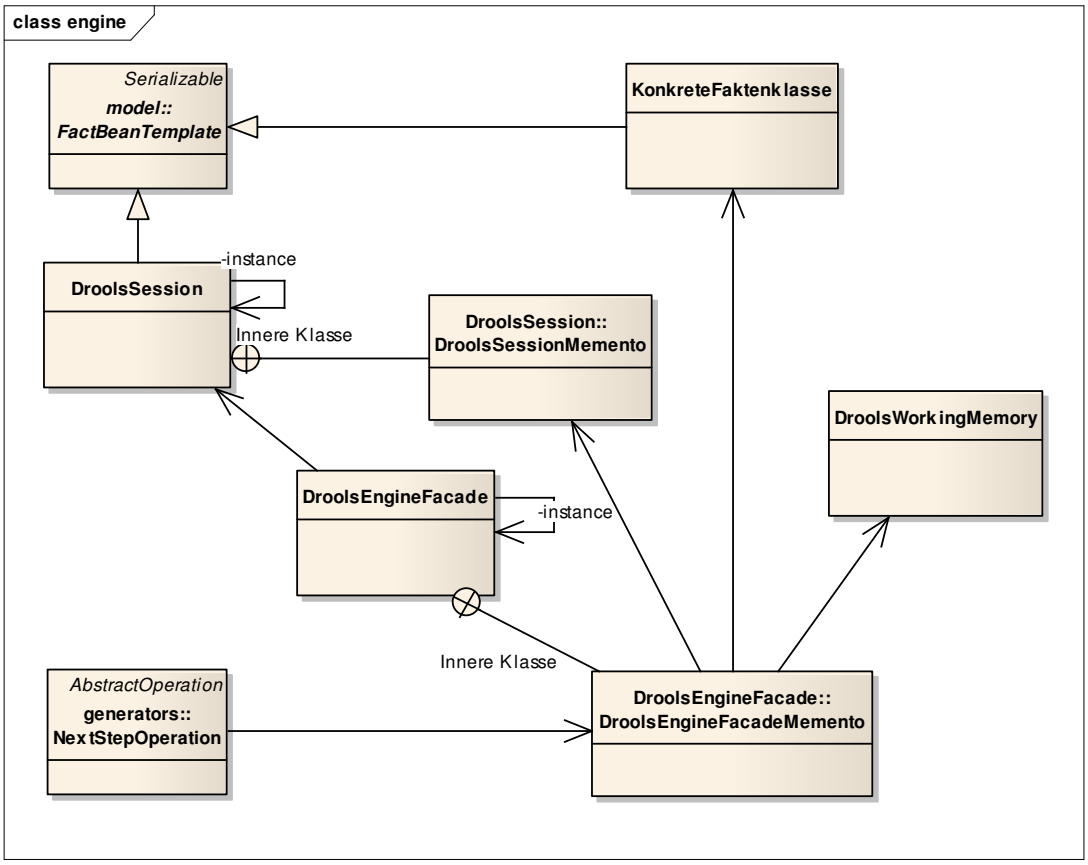

Abbildung 6.10: Das Klassenmodell des umgesetzten Memento-Musters, welches das Speichern eines Zustands ermöglicht.

Objekt ein Singleton ist, wird für dieses, statt der Serialisierung, die bereits beschriebene eigene Memento-Klasse verwendet, so dass das Singleton-Muster nicht verletzt wird.

Eine Schwierigkeit stellt die Verwendung der Serialisierung für das Drools Working Memory-Objekt dar, denn die dort referenzierte Ruleflow-Komponente bietet, entgegen der Aussage der Drools Anleitung [PNF+], zunächst keine Serialisierung. Dies bedeutet, dass das Kopieren der Working Memory-Komponente mit einer entsprechenden Fehlermeldung abbricht. Im Rahmen dieser Arbeit wurde das Drools Entwickler-Team über dieses Problem informiert und es wurde von diesem ein überarbeitetes und serialisierbares Ruleflow-Paket bereitgestellt. Die Verwendung dieses Pakets erlaubt dem Prototyp das Serialisieren des gesamten Working Memorys, inklusive aller zugehörigen Referenzen, ohne das Fehlermeldungen auftreten.

Bei der Integration der Undo-Funktion in den Prototyp wurde jedoch ein weiterer, im Folgenden beschriebener, Fehler in Verbindung mit der Ruleflow-Komponente identifiziert. Es zeigt sich, dass das entwickelte Konzept zur Speicherung des Zustands funktioniert und über das Memento-Muster ein alter Zustand eingespielt werden kann. Anhand

dieser Informationen sind sowohl die Drools Regelmaschine, als auch das Expertensystem in der Lage, den vorherigen Zustand wiederherzustellen. Es wird die Faktenbasis, die Agenda-Liste und die, dem Anwender angezeigte, Seite korrekt wiederhergestellt. Der Ruleflow befindet sich hingegen nur dann in dem richtigen Zustand, der einzig über den aktuellen Knoten bestimmt wird, wenn dieser Knoten sich seit dem Speichern nicht geändert hat. Dies bedeutet, dass es nicht möglich ist, zu einem vorherigen Knoten innerhalb des Ruleflows zurückzukehren. Das Drools Entwicklerteam wurde über diese Eigenschaft informiert, woraufhin diese sowohl das Problem bestätigten, als auch das, in den entwickelten Prototypen umgesetzte, Konzept als grundsätzlich korrekt einstuften. Für die Lösung des Problems wird vom Drools Team auf Drools Version 5 verwiesen, welches erstmalig eine vollständige Programmierschnittstelle für das Speichern und Wiederherstellen des Zustands aller Drools Working Memory-Komponenten anbieten soll. Da zum Zeitpunkt der Erstellung dieser Arbeit Drools Version 5 noch nicht erschienen ist, kann die Undo- und Speicherfunktion nicht vollständig umgesetzt werden und bleibt Aufgabe zukünftiger Arbeiten. Diese können sich auf das bereits implementierte Memento-Muster stützen und müssen die bestehende Implementierung der geänderten Programmierschnittstelle von Drools anpassen.

6.9.2 Realisierung der Erklärungskomponente

Die Erklärungskomponente wird nicht, wie im theoretischen Modell, über ein eigenes Modul repräsentiert, sondern ist, wie zuvor bereits beschrieben, an verschiedenen Stellen integriert. Dies ist mit dem unterschiedlichen Charakter der angebotenen Informationen zu begründen, die aus unterschiedlichen Quellen stammen und daher auf sehr verschiedene Art eingebunden werden müssen. Diese Quellen sind insbesondere die Drools Regelmaschine, die Suchmaschine und die XML-Dateien, welche die Informations- und Interviewseiten beschreiben. Diese Eigenschaft führt dazu, dass eine zentrale Erklärungskomponente sehr eng mit vielen anderen Modulen verwoben sein muss, was die Komplexität und die Abhängigkeiten des Systems erhöhen und eine spätere Erweiterbarkeit sowie Pflege erschweren würde. Stattdessen werden die Erklärungsfunktionen in die Komponenten eingebaut, denen die zu Grunde liegenden Informationsquellen zuzuordnen sind.

Auf Grund des beschränkten zeitlichen Rahmens dieser Arbeit werden nicht alle entworfenen Erklärungsfunktionen (vgl. 5.2.3) umgesetzt. Die folgende Aufzählung gibt einen Überblick über die implementierte Funktionalität:

- **Gefeuerte Regeln anzeigen:** Dem Benutzer wird der Name und die Regelgruppe der gefeuerten Regeln angezeigt, wenn er den Button „Warum werde ich um diese Information gebeten?“ beziehungsweise „Warum wird mir diese Information angezeigt?“ betätigt. Die Umsetzung erfolgt über die spezielle Listener-Klasse `FiredRulesListener`, die als Unterklasse der `DefaultAgendaEventListener`-

Klasse von JBoss Drools implementiert ist. Diese Klasse wird, beim Drools Working Memory registriert, und mittels Events, automatisch über das Feuern von Regeln informiert. Der `FiredRulesListener` ermittelt den Namen und die Regelgruppe der gefeuerten Regel und speichert diesen in einer Liste des `DroolsSession`-Objekts. Die Generatoren der Interview- und Informationsseiten bauen die Informationen dieser Liste in die Seiten ein. Der `FiredRulesListener` kann anhand des Events nicht die textuelle Regeldefinition ermitteln, da diese nicht durch die Regelmaschine zur Verfügung gestellt wird. Die Regelmaschine selbst, enthält die Regeln nur in kompilierter Form, nicht als natürlichsprachlichen Text. Spätere Versionen des Expertensystems könnten dem Anwender den Regeltext zur Verfügung stellen, indem eine Funktion eingebaut wird, die die original Regeldatei parst und den aktuell anzuzeigenden Regeltext anhand des Regelnamens ermittelt.

- **Gesamten Prozess als Grafik anzeigen:** Diese Grafik kann durch den Wissensingenieur in den umgesetzten Prototypen als statische Bilddatei eingebunden und im Rahmen einer Informationsseite angezeigt werden. Hierfür muss diese Grafik in der XML-Datei definiert werden. Bei dieser Grafik kann es sich beispielsweise um einen Screenshot des Ruleflows handeln. Spätere Versionen des Expertensystems könnten eine Funktion implementieren, um die Grafik dynamisch zu erzeugen und parallel in einem View anzuzeigen. Nach Aussage von [Dau08] erscheint für diese Aufgabe insbesondere der Einsatz des Graphical Editing Frameworks (GEF) sinnvoll, welches ein Eclipse-Projekt ist und die Entwicklung von Diagrammeditoren und -viewern für Eclipse RCP unterstützt.
- **Erklärung zu „Warum werde ich um diese Information gebeten?“ beziehungsweise „Warum wird mir diese Information angezeigt?“:** Diese Information wird vom Experten definiert und in die XML-Datei, welche die Interview- oder Informationsseite definiert, integriert. Anschließend wird dieser Text zusammen mit den anderen Elementen von der Generatorklasse in die Seite integriert. Der Endanwender kann diese durch einen Button abrufen. Der Wissensingenieur muss die Informationen in der Form verfassen, dass diese für jeden möglichen Ablauf des Szenarios gültig ist. Bei der gegenwärtigen Struktur des Expertenwissens ist dies problemlos möglich. Sollte später Expertenwissen in einer anderen Struktur vorliegen, so dass diese statische Definition der Erklärungen nicht möglich ist, muss eine Funktion eingebaut werden, die es Ruleflow-Knoten oder den Regeln erlaubt, diese Informationen dynamisch, abhängig vom Szenarioverlauf, Erklärungen festzulegen.
- **„Was wäre, wenn andere Fakten eingegeben werden“ - Funktion:** Diese Funktion wird auf Basis der Undo-Funktion umgesetzt, anhand derer der Endanwender die Auswirkungen verschiedener Entscheidungen ändern kann. Die

Anwendung einer so genannten Sandbox, die eine Vorschau, ohne Änderung der Fakten, bietet, kann später durch eine zweite Regelmaschinen-Instanz integriert werden.

- **Beantwortung der Frage: „Welche weiterführenden Informationen gibt es zu diesem gezogenen Schluss?“:** Durch die Verwendung von FormText-Elementen auf den Interview- und Informationsseiten kann der Wissensingenieur Verweise auf externe Quellen, ähnlich des HTML-Standards, einbauen. Darüber hinaus kann er einzelne Suchwörter und komplette Suchausdrücke definieren, so dass der Endanwender weiterführende Informationen angeboten bekommt. Diese Informationen liegen insbesondere innerhalb des Content Layers.
- **Freie Suche nach weiteren Informationen:** Über eine spezielle View, die auf dem Search-Plugin basiert, kann der Endanwender nach Informationen des Content Layers suchen.

6.10 Perspektive für den Wissensingenieur

Um den Wissensingenieur optimal bei seinen Aufgaben zu unterstützen, wird für diesen Anwendertyp eine eigene Perspektive in das Expertensystem integriert. Diese Perspektive baut auf dem Drools Eclipse-Plugin auf und verwendet dessen spezielle Editoren. Dies erlaubt die bestmögliche Unterstützung bei der Erstellung von Ruleflows, Regeln und Domänenspezifische Sprachen, da die von dem Drools Plugin gebotenen Hilfsfunktionen vollständig zur Verfügung stehen. Es kann beispielsweise die syntaktische Korrektheit von Regeln im Editor geprüft werden. Für die Definition von Faktenklassen steht dem Wissensingenieur der typische Eclipse Java-Editor zur Verfügung, der die Erzeugung von Java Beans durch die automatische Generierung von Getter- und Setter-Methoden vereinfacht und darüber hinaus weitere nützliche Hilfsfunktionen bietet. Die auf XML-Dateien basierenden Informations- und Interviewseiten können mit Hilfe des integrierten Eclipse XML-Editors erstellt werden. Dieser bietet die Möglichkeit eine XML-Dateien auf Grundlage eines XML-Schemas zu erstellen, wobei die Korrektheit der Datei geprüft wird. Der Editor unterstützt den Wissensingenieur beispielsweise durch eine Funktion, die es ermöglicht die gültigen XML-Elemente automatisch zu integrieren. Der Editor unterstützt sowohl eine XML-basierte Ansicht der XML-Informationen, als auch eine Baum-basierte Tabelle der XML-Struktur.

Das Drools Plugin wird integriert, indem es in die sogenannte Target Platform der Eclipse RCP-Anwendung aufgenommen wird. Die im `de.dlr.xps.knowledgeEngenieer`-Plugin definierte Perspektive baut die Views und Editoren zusammen. Standardmäßig registriert das Drools Plugin, wenn es in die Target Platform aufgenommen wird, verschiedene Views und Aktionen (Buttons) in die Eclipse Workbench und bietet diese in allen Perspektiven an. Da der Endanwender keinen Zugriff auf diese Funktionen besitzen

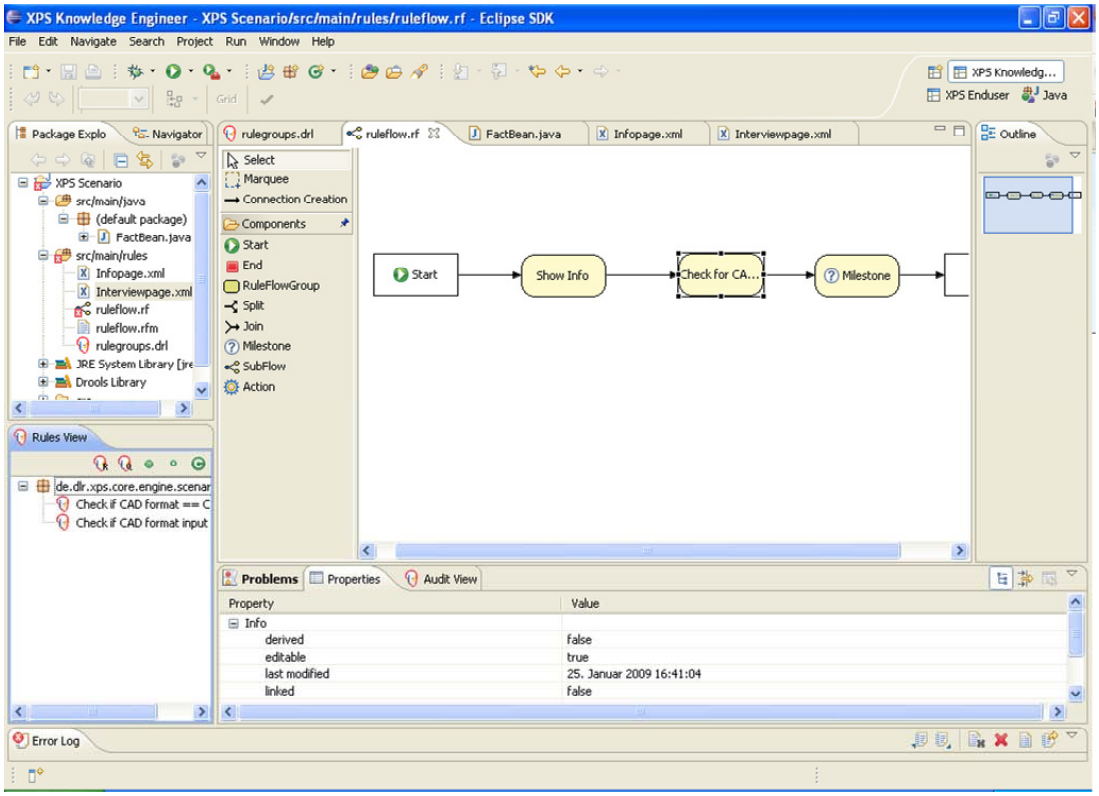

Abbildung 6.11: Die in das Expertensystem integrierte Perspektive für Wissensingenieure. Um Szenarien zu erstellen, bietet diese Perspektive Editoren mit unterstützenden Funktionen.

soll, wurde die Plugin.xml Datei des Drools Plugins abgeändert und die entsprechenden Extensions entfernt.

Abbildung 6.11 zeigt die Perspektive für Wissensingenieure. Diese Perspektive kann später einfach durch die Eclipse-RCP typischen Funktionen erweitert werden. Denkbar ist insbesondere eine, speziell auf die Konventionen des Expertensystems ausgerichtete Unterstützungen bei der Erstellung von Szenarien, die den Wissensingenieur schrittweise führt. Dies kann beispielsweise über sogenannte Wizards umgesetzt werden. Diese arbeiten Formular-basiert und werden dem Anwender der Eclipse IDE, beispielsweise bei der Erstellung neuer Java-Projekte angeboten. Eine weitere zukünftige Aufgabe ist die Anbindung der Perspektive an den Release-Prozess, so dass fertige Szenarien automatisch in eine zentrale Datenbank aufgenommen und an den Release-Manager weitergereicht werden.

7 Evaluation

Um die Korrektheit der entwickelten Konzeption und der prototypischen Implementierung sicherzustellen, wurden, begleitend zu diesen Arbeiten, Szenarien zu Testzwecken in den Prototypen integriert. Diese Testszenarien bilden Expertenwissen von unterschiedlichem Umfang und von unterschiedlicher Wissensstruktur ab. Die Testszenarien wurden in der Wissensingenieur-Perspektive des Expertensystems erstellt. Die gesammelten Erfahrungen flossen zurück in die Konzeptions- und Implementierungsphase und erlaubten eine iterative Verbesserung und Verfeinerung der Expertensystemarchitektur. Das vorliegende Kapitel gibt einen Überblick über die Testszenarien. Anschließend wird das erstellte System, in Hinblick auf die gestellten Anforderungen bewertet. Die Dateien, die die Testszenarien abbilden, befinden sich auf der beiliegenden CD.

7.1 Testszenarien

7.1.1 Von CAD zu Centaur

Dieses Szenario umfasst den Ablauf, der in Abbildung 8.1.(unten) enthalten ist und repräsentiert einen simplen Fall, in dem wenig Regel- und Ablaufwissen abgebildet werden muss. Die Erstellung dieses Testszenarios zeigt, dass bereits bei wenig umfangreichen Szenarien unterschiedliche Möglichkeiten der Wissensrepräsentation bestehen. Die gewählte Modellierung ist in Abbildung 7.1 zu sehen. Es wird die gesamte Logik des Szenarios in einer einzelnen Ruleflow-Gruppe festgehalten, die als „*Convert to Centaur*“ bezeichnet ist. Diese Regeln operieren auf drei fallbezogenen Fakten:

- `String cadFormat`
- `Boolean directExport`
- `Boolean cleaning`

Die Regelgruppe wird in einer Schleife durchlaufen, bis eine der Regeln über das `DroolsSession`-Objekt mitteilt, dass diese Regelgruppe beendet ist. In der Regelgruppe sind drei Faktensammelregeln enthalten, die automatisch die, in der jeweiligen Situation notwendigen, fallbezogene Fakten anhand von Interviewseiten erfragen. Um die faktenbezogenen Informationsseiten anzuzeigen sind dieser Regelgruppe vier Entscheidungsregeln zugeordnet. Es ist bei dem Erfragen von Fakten notwendig, dass die Ausführung der Schleife anhält, bis die Interviewseite vom Endanwender ausgefüllt und

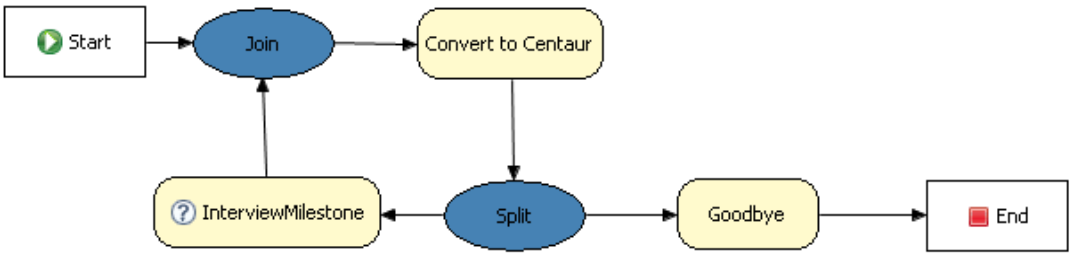

Abbildung 7.1: Ruleflow des Szenarios „Von CAD zu Centaur“

abgesendet wurde. Hierzu wird ein Milestone-Knoten verwendet, der die Ausführung solange anhält, bis ein entsprechendes Stop-Attribut in dem `DroolsSession`-Objekt den gewünschten Wert annimmt. Dieser Wert wird, nach Abarbeiten aller aktivierten Interviewseiten, automatisch vom Expertensystem gesetzt. Die „*Goodbye*“-Regelgruppe besteht aus nur einer Regel, die keine Prämisse besitzt, also immer erfüllt wird und lediglich dazu dient vor dem Ende des Szenarios eine verabschiedende Informationsseite anzuzeigen. Insgesamt besitzt das Szenario sechs Informationsseiten und drei Interviewseiten.

Es kann festgehalten werden, dass die Abbildung des Expertenwissens ohne Probleme und ohne unverhältnismäßigen Aufwand möglich ist. Alternativ zu der gewählten Modellierung ist eine Aufteilung in mehrere Regelgruppen umsetzbar, so dass die grafische Darstellung des Ruleflows eher der Originalabbildung 8.1.(unten) entspricht.

7.1.2 Netzgenerierung mit Centaur

Ein weiteres Testszenario setzt den, in Abbildung 8.2 beschriebenen, Ablauf um. Es wird die Möglichkeit zur Abbildung von solchen Szenarien geprüft, die eine Mischung aus mehreren prozessorientierten Schritten und Regelwissen besitzen. Darüber hinaus enthält der Ablauf Verzweigungen, die zurück zu vorherigen Schritten führen.

Der modellierte Workflow wird anhand der Abbildung 7.2 erklärt. Das Expertenwissen dieses Szenarios enthält einige Informationen, die nicht auf fallbasierten Fakten beruhen, sondern grundsätzlich bei dem Erreichen von bestimmten Stellen des Szenarios angezeigt werden müssen. Auf Grund der in Abschnitt 6.7.2 angesprochenen Problematik bei dem Einsatz von Aktionsknoten werden, für das Anzeigen von nicht fallbasierten Informationen, Regelgruppen verwendet, die jeweils nur aus einer einzigen Regel bestehen. Diese Regel besitzt keine Bedingung und wird daher immer erfüllt. Beim Feuern aktiviert diese Regel eine Informationsseite. In dem Szenario sind diese Regelgruppen: „*Import CAD*“, „*Cleaning*“, „*SetConstraints*“, „*SetSources*“, „*GenSurfaceMesh*“, „*Prism Mesh Param*“, „*Prism Mesh Gen*“, „*Tetra Mesh Gen*“ und „*Good-*

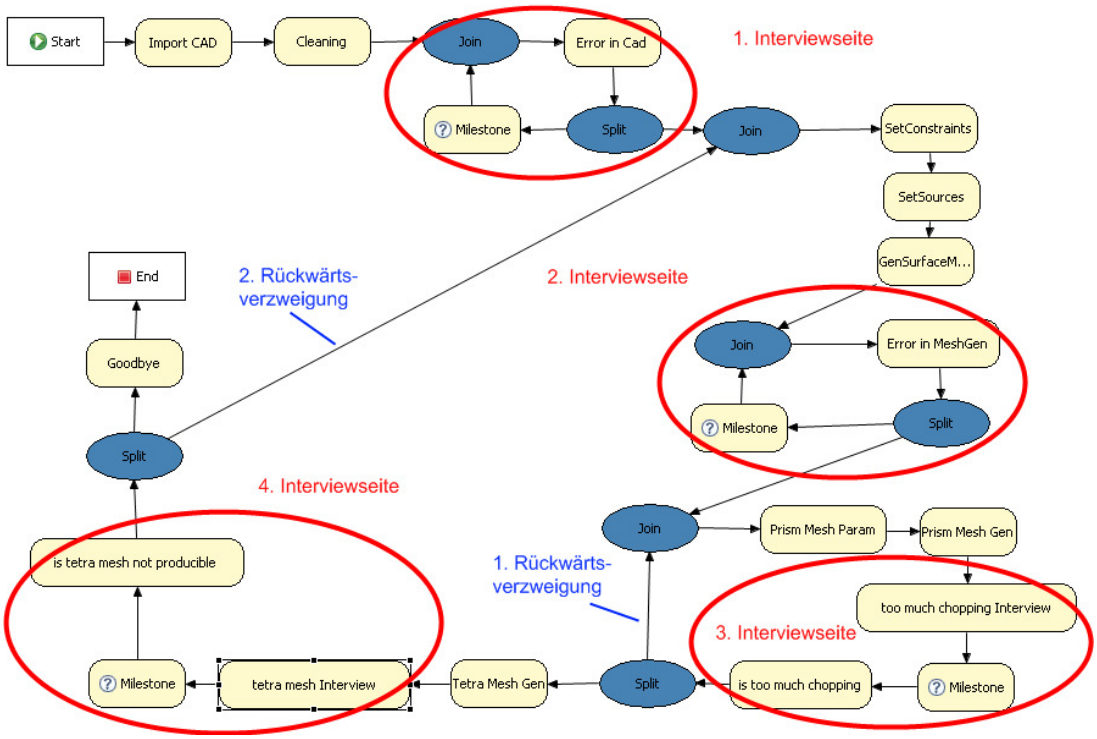

Abbildung 7.2: Ruleflow des Szenarios „Netzgenerierung mit Centaur“

bye“. Darüber hinaus besitzt das Szenario Stellen, an denen Aktionen auf Grundlage von fallbezogenen Fakten getroffen werden. In dem vorliegenden Beispiel sind dies Informationen darüber, ob bestimmte Fehler aufgetreten sind. Diese Fakten werden in dem Faktenobjekt durch vier boolesche Werte definiert. Die von diesen Werten abhängigen Stellen sind in der Abbildung rot umkreist. Die erste und zweite fallbasierte Stelle wurde entsprechend, dem in Abschnitt 7.1.1 beschriebenen Vorgehen, modelliert. Diese verwenden somit eine Schleife, da sich Faktensammelregeln und Entscheidungsregeln in einer gemeinsamen Regelgruppe befinden. Zu Demonstrationszwecken ist die dritte und vierte fallbasierte Stelle alternativ modelliert. Dort befinden sich die Faktensammelregeln und die Entscheidungsregeln in getrennten Regelgruppen und auf den Einsatz einer Schleife wird verzichtet. Die Faktensammelregeln werden stattdessen explizit vor die Entscheidungsregeln platziert. Für das vorliegende Beispiel funktioniert dieser Ansatz problemlos, so dass der Wissensingenieur in Szenarien dieses Charakters zwischen den beiden Modellierungsformen frei wählen kann. Bei Szenarien mit umfangreicheren Regelsammlungen ist jedoch zu erwarten, dass eine Anordnung beider Regeltypen in einer gemeinsamen Regelgruppe vorteilhaft ist. Denn dieser Ansatz erlaubt es, dass die Faktensammelregeln auf Änderungen an der Faktenbasis reagieren können, die von den Entscheidungsregeln vorgenommen werden. Es findet somit ein direkter Informationsaustausch zwischen beiden Regeltypen statt, während dies beim Hintereinanderschalten nicht der Fall ist.

Das Szenario demonstriert an zwei Stellen die Möglichkeit zu einer vorherigen Stelle zurückzukehren. Dies lässt sich problemlos durch Split-Knoten modellieren.

Dieses Szenario umfasst vierzehn Informationsseiten und vier Interviewseiten, sowie vier Informationssammelregeln und acht Entscheidungsregeln. Für die gezielte Aktivierung von Informationsseiten werden neun Regeln eingesetzt, welche keine Bedingungen besitzen.

Durch dieses Testszenario kann gezeigt werden, dass das Expertensystem in der Lage ist, komplexer gestaltete Abläufe, die Rückwärtsverzweigungen besitzen, zu modellieren. Darüber hinaus werden in diesem Szenario, zwei Möglichkeiten zur Modellierung von Benutzerbefragungen, erfolgreich demonstriert.

7.1.3 Testszenario zur Prüfung der Erweiterbarkeit der Domäne

Das in diesem Szenario abgebildete Wissen ist frei erfunden. Dieses Szenario dient der Prüfung, ob das Expertensystem in der Lage ist, zukünftig Expertenwissen abzubilden, das eine andere Charakteristik aufweist, als das bereitgestellte Beispielwissen. Das erstellte Testwissen erweitert den Workflow-orientierten Ansatz und verwendet Entscheidungsregeln, die auf den Ergebnissen von Berechnungsfunktionen operieren. In diesem Szenario ist es wahrscheinlich, dass mehrere Regeln gleichzeitig feuern und Seiten aktivieren. Daher eignet sich dieses Szenario für die Überprüfung der definierten Ausführungsreihenfolge der Seiten. Die berechneten Werte werden dynamisch in die Seiten eingebunden und dem Anwender präsentiert. Dies ermöglicht die Validierung dieser Funktion, die über den `DynamicBeanValue`-Typen der XML-Schema umgesetzt ist.

Die Testberechnungen setzen verschiedene Eigenschaften (Länge, Spannweite, Gewicht) eines Flugzeugs ins Verhältnis und geben, abhängig vom gewählten Flugzeugtyp, an, ob die jeweiligen Verhältnisse innerhalb eines bestimmten Wertebereichs liegen. Ist dies nicht der Fall, wird der Benutzer zur Überarbeitung der Fakten aufgefordert. Die Berechnungen und Regeln sind vollständig frei erfunden und dienen lediglich dem Testzweck. Sie beruhen nicht auf physikalischen Tatsachen.

Der modellierte Ruleflow ist in Grafik 7.3 dargestellt. Das Testszenario besitzt elf Regeln, wovon sechs die berechneten Werte daraufhin überprüfen, ob diese innerhalb eines bestimmten Wertebereichs liegen. Es gibt sechs Informationsseiten, von denen drei Seiten die berechneten Werte dynamisch anzeigen. Für die Ermittlung der Eigenschaften existiert eine Interviewseite, die diese gesammelt erfragt.

Dieses Testszenario zeigt, dass das Expertensystem in der Lage ist, die weiterführende Funktionalität von JBoss Drools zu integrieren und beispielsweise externe Funktionen anzustoßen und Ergebnisse dynamisch in die Seiten einzubinden. Dies erlaubt sehr viel Freiheit bei der Modellierung von Szenarien und lässt darauf schließen, dass mit dem

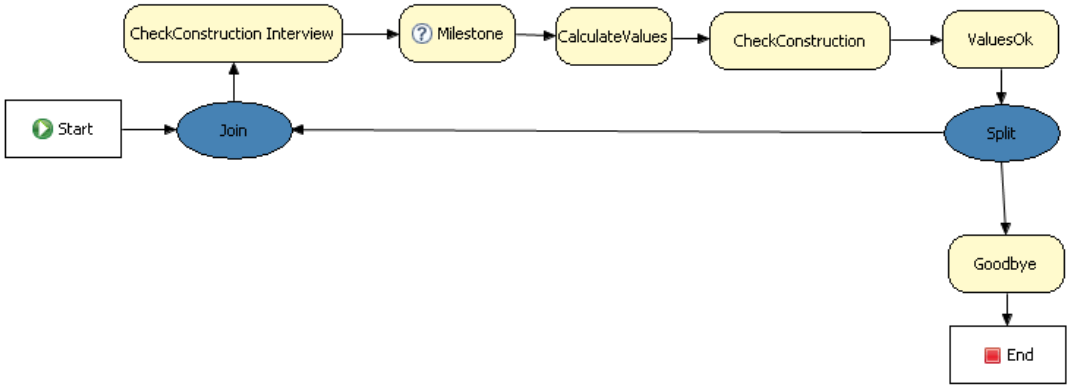

Abbildung 7.3: Ruleflow des Testszenarios zur Prüfung der Erweiterbarkeit der Domäne

Expertensystem zukünftig Wissen aus anderen Domänen abgebildet werden kann. Auf Basis der verwendbaren Funktionalität von Drools ist zu erwarten, dass die Fähigkeit der Wissensrepräsentation ähnlich mächtig ist, wie die von JBoss Drools alleine.

7.2 Bewertung

Um eine angemessene Bewertung des entwickelten Systems zu ermöglichen, wurden die Testszenarien zusammen mit verschiedenen Stakeholdern analysiert. Während dessen wurde deutlich, dass dem Endanwender bei der Ausführung aller Testszenarien ein logisch korrekter Ablauf präsentiert wird und die Interview- und Informationsseiten in einer sinnvollen Reihenfolge angezeigt werden. Dies bestätigt die getroffene Entscheidung, Ruleflows und Regelgruppen für die Repräsentation der Lösungsstrategie zu verwenden. Der Inhalt der einzelnen Seiten kann, anhand der XML-Definitionen, durch den Wissensingenieur sehr frei gestaltet werden. Auf Grund dessen, können dem Endanwender Seitenelemente und Informationen angezeigt werden, die in der jeweiligen Situation angemessen sind. Darüber hinaus werden dem Endanwender, auf dessen Wunsch, hilfreiche Erklärungen bereitgestellt. Zur gezielten Wissensvertiefung liefert die kontextsensitive Suchfunktion nützliche Dokumente und Ergebnisse. Es lässt sich an dieser Stelle schlussfolgern, dass das Expertensystem in der Lage ist, das Expertenwissen über flugphysikalischen Simulationen des DLRs vollständig abzubilden und dem ungeschulten Anwender in nützlicher Form bereitzustellen. Eine zukünftige Anwendung des Expertensystems in anderen Wissensdomänen ist möglich. Dies wird durch das dritte Testszenario belegt. Das Expertensystem erfüllt daher alle Anforderungen, die an die Abbildung und Bereitstellung von Wissen gestellt sind.

Die umgesetzte Funktion zur Erweiterung und Pflege von Wissen wurde von den Stakeholdern als durchaus angemessen bezeichnet. Die verwendeten Konventionen, sowie die verschiedenen Modellierungselemente (besonderes XML-Dateien, Faktenklas-

sen, Regeln und Ruleflows), sind auch durch Wissensingenieure, die keine tiefgehenden Programmierkenntnisse besitzen, zu definieren. Es wurde festgestellt, dass die Unterstützung durch die Wissensingenieur-Perspektive sehr hilfreich ist und diese eine zügige Erstellung neuer Szenarien erlaubt. Bereits vorhandene Szenarien können als Vorlage verwendet werden, wobei oftmals einzelne Elemente mit nur geringer Anpassung erneut genutzt werden können. Folglich erfüllt das Expertensystem auch in dieser Hinsicht die gestellten Anforderungen.

Für die, im Rahmen dieser Arbeit, identifizierten Probleme wurden bereits Lösungsansätze entwickelt und aufgezeigt. Diese zielen im Wesentlichen auf die zukünftige Verwendung der Version 5 von JBoss Drools ab. Auf Grund der zeitlichen Beschränkung dieser Arbeit konnten einige Komponenten und Funktionen nicht vollständig implementiert werden. Hierbei handelt es sich nicht um Kernelemente des Systems, sondern um Erweiterungen, die auf Basis der Kernkomponenten aufgebaut werden können. Dies ist etwa die Perspektive für Release-Manager oder erweiterte Erklärungsfunktionen. Sowohl die zukünftige Lösung der Probleme, als auch die Erweiterung der Funktionalität wurde bei der Konzeption und Implementierung berücksichtigt. Daher ist insgesamt eine problemlose zukünftige Erweiterung und Anpassung des Systems zu erwarten.

Auf Basis der Analyse und unter Berücksichtigung der theoretischen Anforderungen (vgl. Abschnitt 4.4), kann die zusammenfassende Bewertung erteilt werden, dass die entwickelte Architektur, die im Rahmen dieser Arbeit zu erwartenden Ziele vollständig erfüllt. Die Machbarkeit der praktischen Umsetzung wurde anhand der prototypischen Implementierung gezeigt. Durch unterschiedlich charakterisierte Testszenarien wurden Korrektheit und Qualität der Architektur sichergestellt.

8 Fazit und Ausblick

8.1 Fazit

Die in dieser Arbeit konzipierte und implementierte Architektur für Expertensysteme kann zusammenfassend betrachtet als durchaus erfolgreich bezeichnet werden. Das System ist in der Lage das Expertenwissen im Gebiet der flugphysikalischen Simulationen vollständig und strukturiert abzubilden und dem Endanwender auf sehr geeignete Weise zur Verfügung zu stellen. Das System erlaubt es Wissen sowohl regelbasiert, als auch Workflow-basiert festzuhalten. Dies lässt eine sehr gute Erweiterbarkeit auf andere Wissensdomänen erwarten.

Die Evaluation der Architektur zeigt, dass neues Wissen einfach, schnell und flexibel durch einen geschulten Wissensingenieur in das Expertensystem eingepflegt werden kann. Voraussetzung ist, dass dieser sich an vordefinierte Konventionen hält. Es werden keine umfangreichen Programmierkenntnisse benötigt. Bei der Definition neuen Wissens wird der Wissensingenieur durch die Hilfsfunktionen spezieller Editoren unterstützt. Bereits vorhandene Szenarien können als hilfreiche Vorlagen verwendet werden.

Die Architektur wurde prototypisch implementiert, wodurch die Machbarkeit der entwickelten Konzepte aufgezeigt werden konnte. Es wurde dabei deutlich, dass die entwickelte Architektur in der Lage ist, die ermittelten Anforderungen und Ziele vollständig zu erfüllen.

Die entwickelte Architektur kann als äußerst nützliches Werkzeug für einen Wissensingenieur aufgefasst werden. Die Qualität des enthaltenen Expertenwissens und der Nutzen für Endanwender, sind aber im Wesentlichen abhängig von zwei externen Faktoren. Zum einen ist dies die Güte der Modellierung durch den Wissensingenieur. Zum anderen der Umfang und die Qualität des Expertenwissens, das dem Wissensingenieur zur Verfügung steht.

Im Anschluss an diese Arbeit ist der Einsatz des entwickelten Expertensystems in verschiedenen Instituten des Deutschen Zentrums für Luft- und Raumfahrt geplant. Die Hauptaufgaben werden dort die dauerhafte Sicherung von Wissen, die Schulung von Personal, sowie die Qualitätssicherung sein.

8.2 Ausblick

Im Rahmen dieser Arbeit konnten verschiedene Erweiterungen des Expertensystems identifiziert werden, die auf Grund des zeitlich begrenzten Rahmens nicht umgesetzt wurden, aber im Folgenden aufgezeigt werden.

Um seine volle Leistungsfähigkeit entfalten zu können, muss das entwickelte Expertensystem vollständig in die Gesamtarchitektur eingebunden werden, die das Search Layer, das Content Layer und das Tool Layer umfasst. Eine Aufgabe ist dabei die Anbindung einer externen Datenbank, die sich im Content Layer befinden wird. In diese können die Szenarien zentral abgelegt werden. Die Struktur der Datenbank kann sich an dem Szenario-Datenmodell orientieren. Da die Anforderung, das Expertensystem in die Gesamtarchitektur einzubinden, in der vorliegenden Arbeit berücksichtigt wurde, ist eine problemlose Integration zu erwarten.

Aufbauend auf die Datenbank kann ein vollständiger Release-Prozess entwickelt werden, der bei der Erstellung neuer Szenarien verbindlich angewendet wird. Hierzu sollte ein Release-Manager Perspektive entwickelt und in das Expertensystem integriert werden. Erstrebenswert ist in diesem Zusammenhang die Fähigkeit des Systems zur Versionierung von Szenariodaten. Aufbauend auf dem Release-Prozess und der Datenbank kann eine Rechteverwaltung erstellt werden, so dass das jeweilige Expertenwissen nur bestimmten Personen zugänglich gemacht werden kann.

Um alle Funktionen des Expertensystems vollständig nutzen zu können ist darüber hinaus eine Migration auf die, zum Zeitpunkt dieser Arbeit noch nicht verfügbare, Version 5 von JBoss Drools erstrebenswert. Laut Aussage des Drools-Teams löst dies die identifizierten Probleme im Zusammenhang mit der Speicher- und Undo-Funktion. Zentraler Punkt für die Migration ist die Klasse DroolsEngineFacade.

Die genannten Tätigkeiten werden im direkten Anschluss an diese Arbeit in der Einrichtung „Simulations- und Softwaretechnik“ erbracht. Diese Erweiterungen führen zu einer vollständigen Erfüllung aller ermittelten Anforderungen und Ziele.

Darüber hinaus existieren mehrere Möglichkeiten, die bereits implementierten Funktionen des Systems zukünftig zu erweitern. Die vorliegende Arbeit gibt insbesondere in Kapitel 5 und 6 Vorschläge hierzu. Beispielsweise kann die Erklärungskomponente durch eine Funktion erweitert werden, die das schrittweise Fortschreiten des Szenarios durch einen Graphen abbildet und die gefeuerten Regeln in aufbereiteter Form anzeigt. Darüber hinaus ist es denkenswert weitere Typen von Darstellungsseiten zu integrieren. Dies kann auf einfache Weise unter Berücksichtigung des implementierten Strategie-Musters geschehen, so wie unter Verwendung neuer XML-Schemata, die über JAXB eingebunden werden.

Ein langfristiges Ziel könnte die Entwicklung der Architektur zu einem allgemeinen

Framework für JBoss Drools-basierte Anwendungen sein, so dass das Einsatzgebiet über Expertensysteme hinausgeht. Eine erweiterte Serviceorientierung der Architektur wäre in diesem Fall erstrebenswert. Dieses Framework könnte insbesondere eine allgemeine Definition für den Kontrollfluss, die Benutzerschnittstellen und das Datenmanagement bieten und in verschiedensten Anwendungsbereichen eingesetzt werden.

Anhang A: Modelliertes Expertenwissen

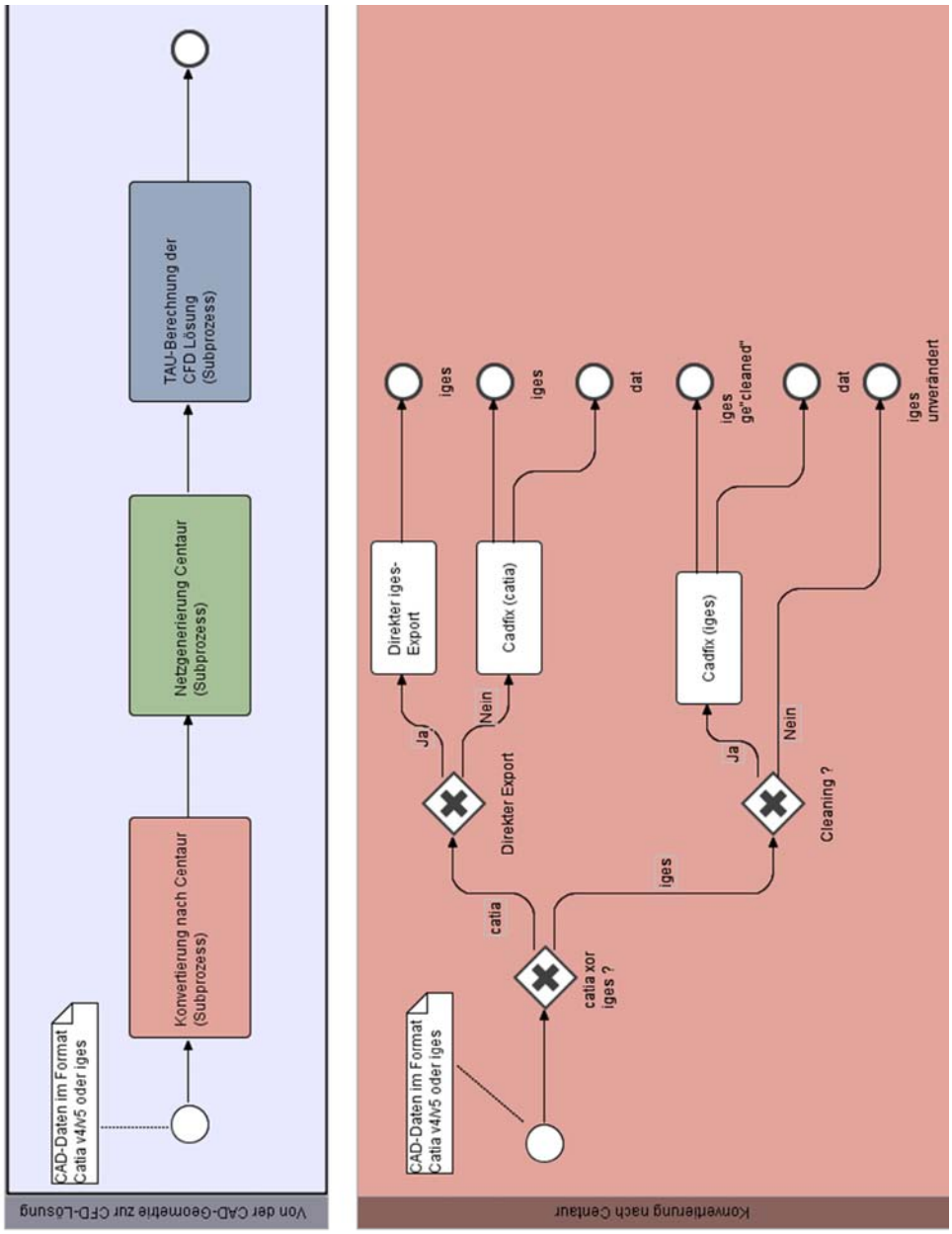

Abbildung 8.1: Von der CAD-Geometrie zur CFD-Lösung (oben / blau) / Konvertierung nach Centaur (unten / rot)

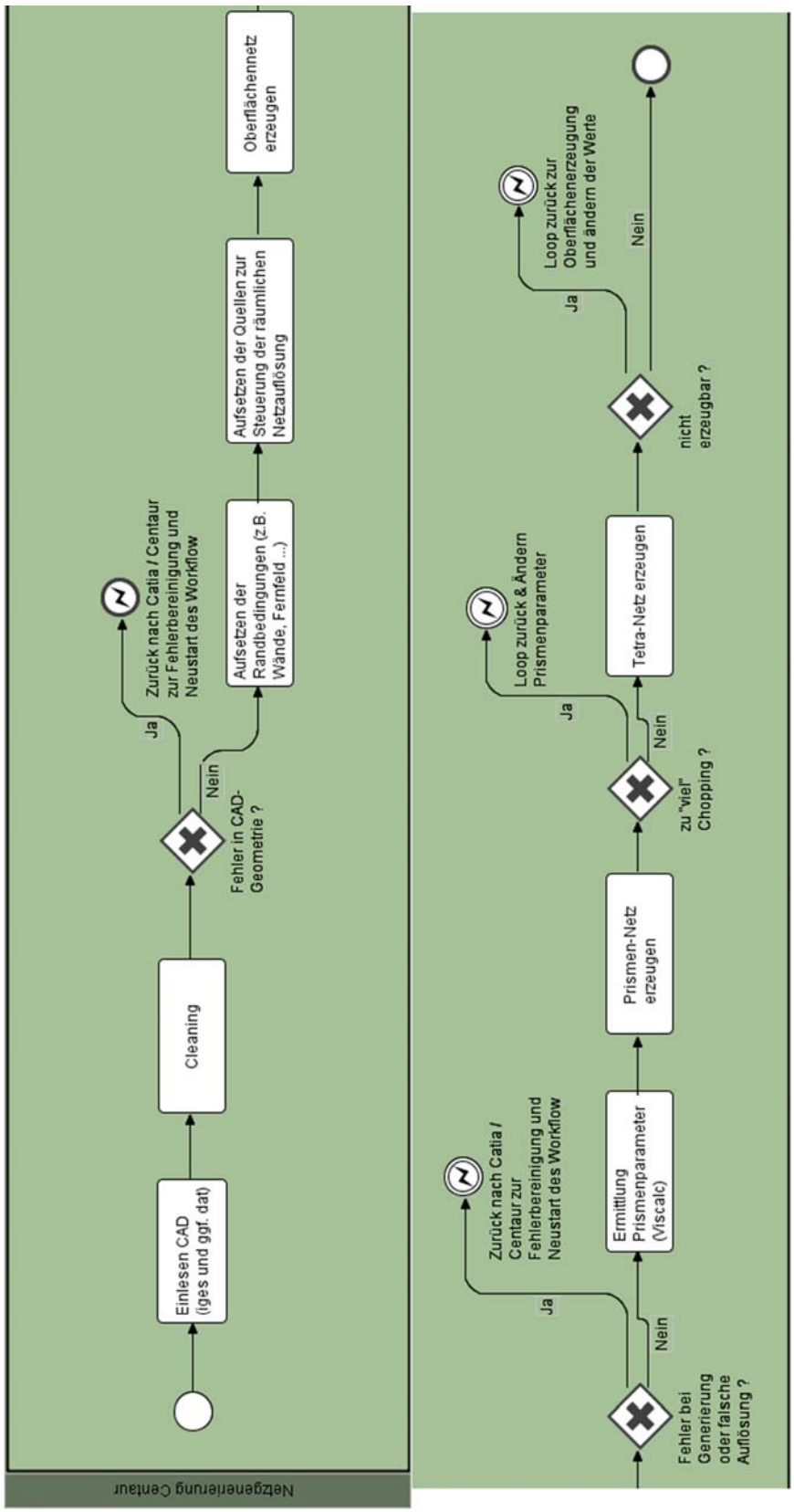

Abbildung 8.2: Netzgenerierung mit Centaur

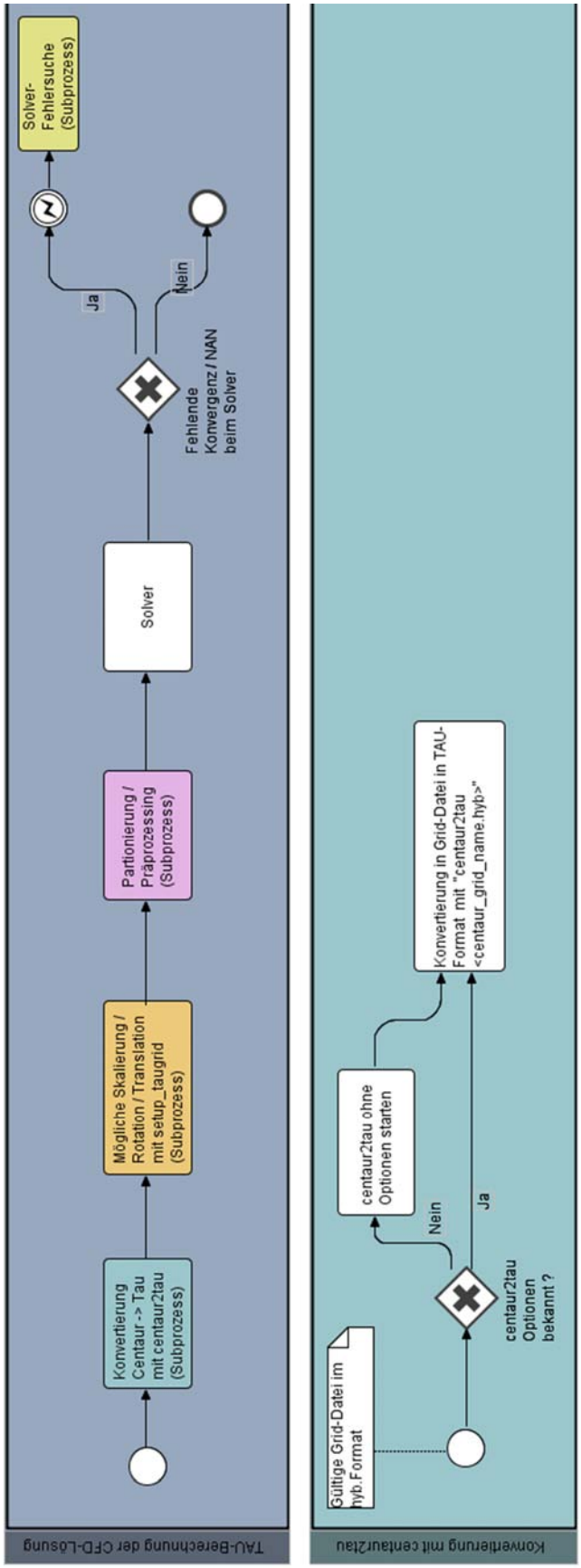

Abbildung 8.3: TAU-Berechnung der CFD-Lösung (oben / dunkelblau) / Konvertierung mit centaur2tau (unten / hellblau)

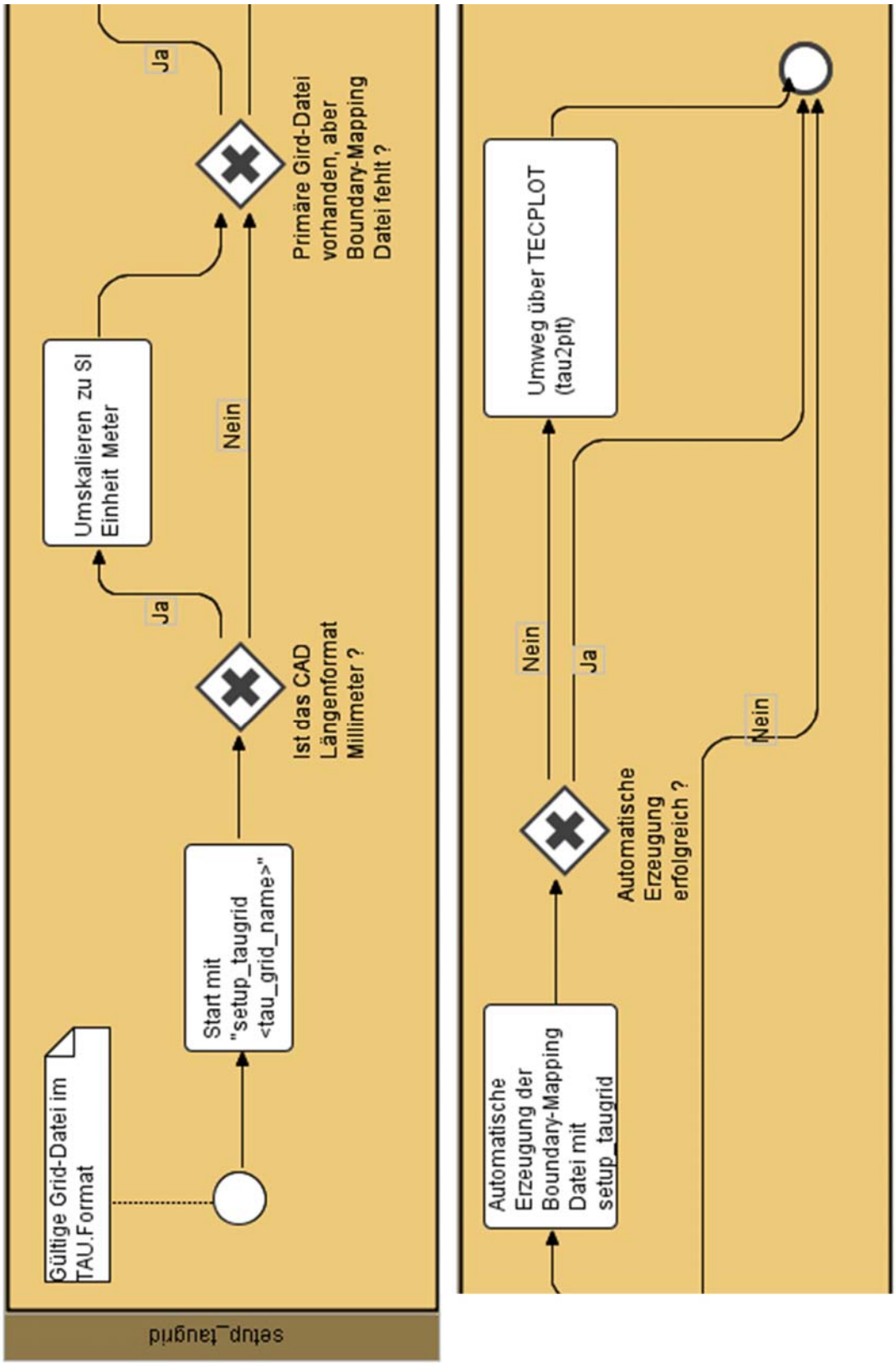

Abbildung 8.4: Grid Setup

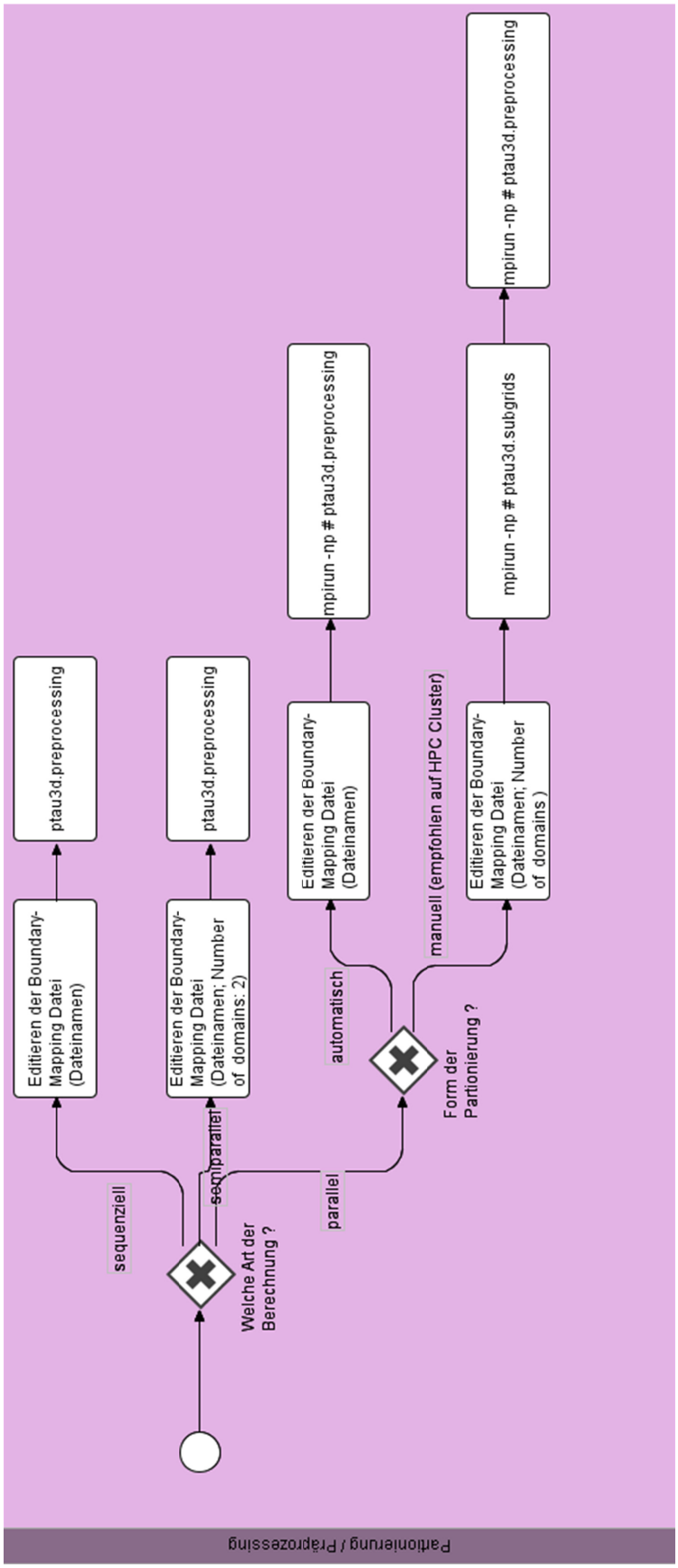

Abbildung 8.5: Partitionierung, Präprozessing und Start des Solvers

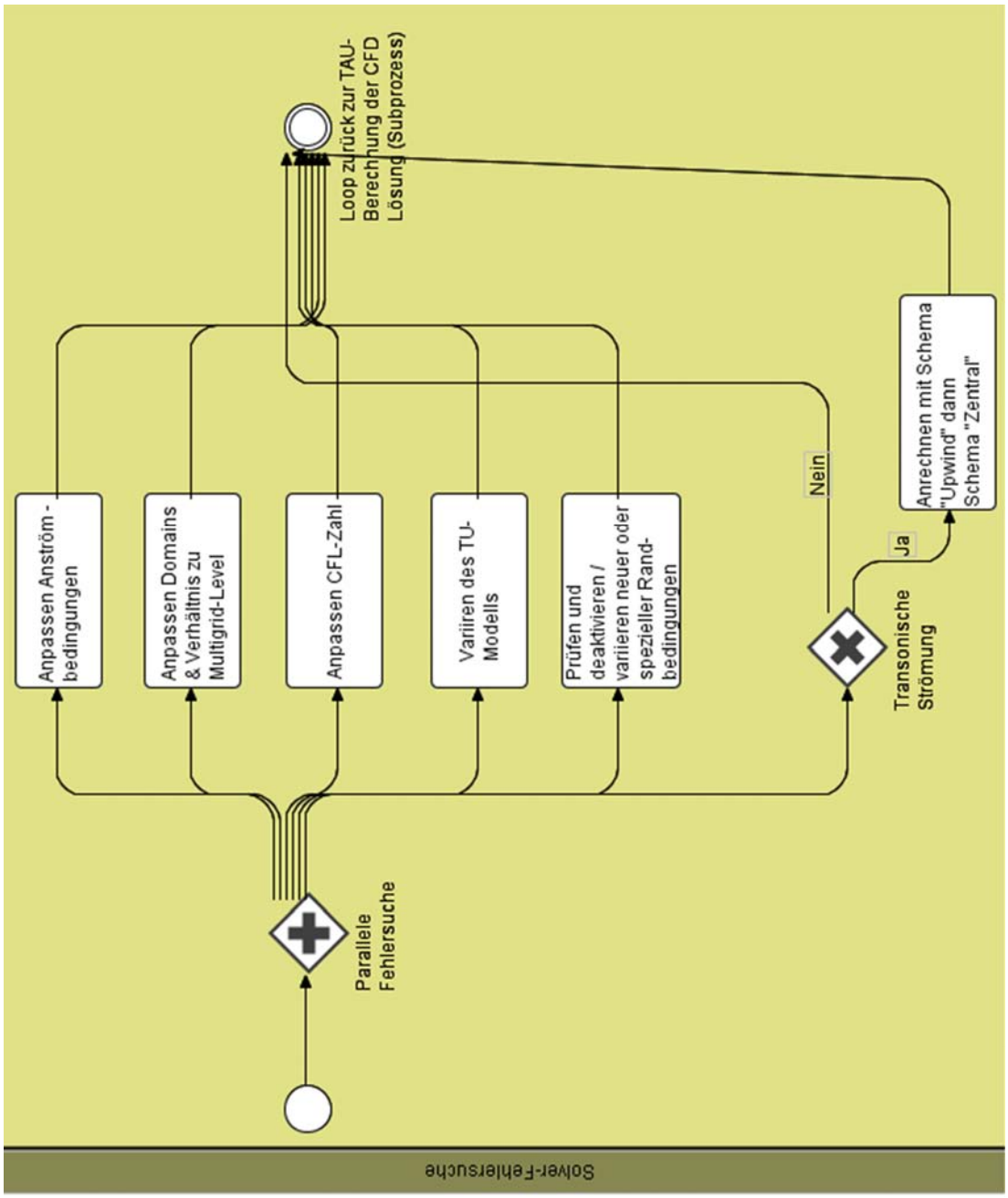

Abbildung 8.6: Solver-Fehlersuche

Literaturverzeichnis

[AF88] Albert, L. ; Fage, F.: Average case complexity analysis of the RETE multi-pattern match algorithm. In: *Lecture Notes in Computer Science: Automata, Languages and Programming* Bd. 317. 1988, S. 18–37

[ant] antlr.org (Hrsg.): *ANTLR v3*. http://www.antlr.org/, Abruf: 5. 12. 2008. – Homepage des ANTLR Parser Generator Projekts

[Apa] Apache Software Foundation, The (Hrsg.): *Apache Lucene - Overview*. http://lucene.apache.org/java/docs, Abruf: 9. 12. 2008. – Homepage des Apache Lucene Projekts

[Bec08] Becker, K.: Hochleistungsrechnen für Airbus: Ein Weg zu besseren Flugzeugen: Einweihung des High Performance Computing Clustersystems. Braunschweig, 2008. – $C^2A^2S^2E$ - Einweihung: http://www.dlr.de/as/Portaldata/5/Resources/dokumente/abteilungen/abt_ca/cluster_einweihung/08_05_13_Gesamtpraesentation_C_A_S_E_Web.pdf, Abruf am 07.09.2008

[BS84] Buchanan, B. (Hrsg.) ; Shortliffe, E. (Hrsg.): *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA : Addison–Wesley, 1984

[Coda] Codehaus, The (Hrsg.): *MVEL*. http://mvel.codehaus.org/, Abruf: 5. 12. 2008. – Homepage des MVEL Projekts

[Codb] Codehaus, The and Proctor, M. and McWhirter, B. (Hrsg.): *Codehaus Drools Documentation*. http://docs.codehaus.org/display/DROOLS/ReteOO, Abruf: 5. 12. 2008. – Dokumentation des Codehaus Drools Projekts

[CR06] Clayberg, E. ; Rubel, D.: *Eclipse: Building commercial–quality plug–ins*. 2. ed. Upper Saddle River, NJ : Addison–Wesley, 2006

[Dau08] Daum, B.: *Rich–Client–Entwicklung mit Eclipse 3.3*. 3. Aufl. Heidelberg : dpunkt–Verl., 2008

[Deu] Deutsches Zentrum für Luft- und Raumfahrt (Hrsg.): *DataFinder – Organize your data.* http://www.dlr.de/DataFinder, Abruf: 9. 12. 2008. – Homepage des DataFinder Projekts

[Doo95] Doorenbos, R.: *Production Matching for Large Learning Systems*, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Diss., 1995

[Ecla] Eclipse Foundation, The (Hrsg.): *Eclipse documentation.* http://help.eclipse.org/help33/index.jsp, Abruf: 9. 12. 2008. – Offizielle Dokumentation und Hilfe zu Eclipse 3.3

[Eclb] Eclipse Foundation, The (Hrsg.): *The Eclipse Project.* http://www.eclipse.org, Abruf: 12. 11. 2008. – Offizielle Homepage des Eclipse Projekts

[Eclc] Eclipse Foundation, The (Hrsg.): *Eclipse Rich Client Platform.* http://wiki.eclipse.org/index.php/Rich_Client_Platform, Abruf: 8. 12. 2008. – Homepage des Eclipse Rich Client Platform Projekts

[FH03] Friedman-Hill, E.: *Jess in action: Rule–based systems in Java.* Greenwich, Conn. : Manning, 2003

[For79] Forgy, C.: *On the Efficient Implementation of Production Systems*, Carnegie-Mellon University, Diss., 1979

[For82] Forgy, C.: Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. In: *Artificial Intelligence 19* Bd. 1. 1982, S. 17–37

[FS90] Friedrich, G. ; Stumptner, M.: Einführung. In: Gottlob, G. (Hrsg.) ; Frühwirth, T. (Hrsg.) ; Horn, W. (Hrsg.): *Expertensysteme.* Berlin, Heidelberg : Springer Verlag, 1990, S. 1–19

[GB04] Gamma, E. ; Beck, K.: *Eclipse erweitern.* München : Addison–Wesley, 2004

[GR04] Giarratano, J. ; Riley, G.: *Expert systems: Principles and programming.* 4. ed. Boston, Mass. : Thomson, 2004

[GRS03] Görz, G. (Hrsg.) ; Rollinger, C.-R. (Hrsg.) ; Schneeberger, J. (Hrsg.): *Handbuch der Künstlichen Intelligenz.* 4., korrigierte Auflage. München, Wien : Oldenbourg Verlag, 2003

[Gup86] Gupta, A.: *Parallelism in production systems*, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Diss., 1986

[HC08] Horstmann, C. ; Cornell, G.: *Core Java Volume II: Advanced Features.* 8th ed. Upper Saddle River, NJ : Sun Microsystems Press, 2008

[Ins07] Institut für Aerodynamik und Strömungstechnik ; Deutsches Zentrum für Luft- und Raumfahrt (Hrsg.): *TAU Beginners Guide.* 2007

[Jac99] Jackson, P.: *Introduction to expert systems.* 3. ed. Harlow : Pearson Addison–Wesley, 1999

[JBo] JBoss Drools (Hrsg.): *JBoss Drools Website.* http://jboss.org/drools/, Abruf: 5. 12. 2008. – Homepage des JBoss Drools Projekts

[Lig06] Ligêza, A.: *Logical foundations for rule–based systems.* 2. ed. Berlin, Heidelberg : Springer, 2006

[Lug01] Luger, G.: *Künstliche Intelligenz: Strategien zur Lösung komplexer Probleme.* München : Pearson Studium, 2001

[Mad03] Madden, N.: Optimising Rete for low-memory, multiagent systems. In: *Proceedings of Game-On 2003: 4th International Conference on Intelligent Games and Simulation.* 2003, S. 77–81

[Mic] Microsoft Corporation (Hrsg.): *MSDN XAML.* http://msdn.microsoft.com/en-us/library/ms747122.aspx, Abruf: 22. 11. 2008. – Offizielle Dokumentation des XAML Projektes

[ML05] McAffer, J. ; Lemieux, J.-M.: *Eclipse rich client platform: Designing, coding, and packaging Java applications.* Upper Saddle River, NJ : Addison–Wesley, 2005

[Moz] Mozilla Foundation (Hrsg.): *XUL (XML User Interface Language).* https://developer.mozilla.org/en/XUL, Abruf: 22. 11. 2008. – Homepage des Mozilla XUL Projektes

[NS63] Newell, A. ; Simon, H.: GPS, a program that simulates human thought. In: Feigenbaum, E. (Hrsg.) ; Feldmann, J. (Hrsg.): *Computers and Thought.* Cambridge, MA : AAAI Press, 1963, S. 279–296

[Obj] Object Management Group (Hrsg.): *Business Process Modeling Notation, V1.1 - Specification.* http://www.bpmn.org/Documents/BPMN1-1Specification.pdf, Abruf: 27. 12. 2008. – Offizielle Spezifikation der Business Process Modeling Notation (BPMN)

[OSG] OSGi Alliance (Hrsg.): *OSGi - The Dynamic Module System for Java.* http://www.osgi.org, Abruf: 9. 12. 2008. – Homepage des OSGi Projekts

[PNF+] Proctor, M. ; Neale, M. ; Frandsen, M. ; Griffith Jr., S. ; Tirelli, E. ; Meyer, F. ; Verlaenen, K. ; JBoss Inc. (Hrsg.): *Drools Documentation.* http://www.jboss.org/drools/documentation.html

[RN04] Russell, S. ; Norvig, P.: *Künstliche Intelligenz: Ein moderner Ansatz*. 2. Aufl. München : Pearson Studium, 2004

[Sch05] Schöning, U.: *Logik für Informatiker*. 5. Aufl.,. Heidelberg : Spektrum Akad. Verl., 2005

[SGH06] Schwamborn, D. ; Gerhold, T. ; Heinrich, R.: The DLR TAU–Code: Recent applications in research and industry. Delft (NL), 2006. – ECCOMAS CFD 2006 Conference: http://www.numerical.rl.ac.uk/people/marioli/Archives/ECCOMAS-CFD-2006/documents/619.pdf, Abruf am 07.09.2008

[SJB08] Sippel, H. (Hrsg.) ; Jastram, M. (Hrsg.) ; Bendisposto, J. (Hrsg.): *Die Eclipse Rich Client Platform: Entwicklung von erweiterbaren Anwendungen mit RCP*. Frankfurt am Main : Entwickler Press, 2008

[Suna] Sun Microsystems (Hrsg.): *Java Rule Engine API*. http://jcp.org/en/jsr/detail?id=94, Abruf: 5. 12. 2008. – Homepage des Java Specification Requests (JSR) 94: Java Rule Engine API

[Sunb] Sun Microsystems (Hrsg.): *JAXB Reference Implementation Project*. https://jaxb.dev.java.net/, Abruf: 14. 12. 2008. – Homepage des JAXB Referenzprojektes

[WHKL08] Wütherich, G. ; Hartmann, N. ; Kolb, B. ; Lübken, M.: *Die OSGi Service Platform*. Heidelberg : dpunkt Verlag, 2008

[Wol03] Wollny, S.: *Erklärungsfähigkeit kooperierender regelbasierter Expertensysteme zum diagnostischen Problemlösen*, Fakultät Elektrotechnik und Informatik, Technischen Universität Berlin, Diss., 2003

[Wun06] Wunderlich, L.: *Java Rules Engines*. Frankfurt am Main : Entwickler Press, 2006

Erklärung

Carsten Bochner
Vondelstraße 56, 50677 Köln

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Köln, den 10. Februar 2009