RecipFinder and Experts for the web of tomorrow

Andreas Rützel

Sulzer Innotec,
Sulzer Markets and Technology AG, Zürcherstr. 14,
CH-8401 Winterthur, Schweiz
E-Mail: andreas.ruetzel@sulzer.com

Paper for Presentation on net.objectcdays 2002

Abstract: In dem Projekt RecipFinder wurde das Auslegungs-Know-how für Kompressoren als Expertensystem hinterlegt. Diese Internetanwendung dient der Verkürzung der Angebotsphase und Wissenssicherung. Für die Definition der Regeln wurde ein einfaches Design gewählt. Es werden verschiedene Arten der Definition gezeigt und Vor- und Nachteil diskutiert. Ein Überblick über nutzbare Expertensystemtools und Agententechnologien wird gegeben.

1 Einleitung

Das Projekt RecipFinder ist eine Entwicklung im Kundenauftrag und wurde als eine webbasierte Expertensystemlösung realisiert. Der Name des Projektes ist zugleich Name des Services und Trademark und setzt sich zusammen aus „Recip“ als Abkürzung für „Reciprocating Compressors“ und „Finder“ als Suchdienst.
RecipFinder soll zwei Aufgaben lösen, die auch für andere Unternehmen zunehmend wichtig werden: Verkürzung der Angebotsphasen und Wissenssicherung.
Die Beweggründe für dieses Projekt halten wir typisch für eine Vielzahl von Unternehmen.
Sulzer Innotec ist deshalb überzeugt von einem weiteren Bedarf an „intelligenten“ Anwendungen. Diese sollen Benutzern Entscheidungen erleichtern, Informationen nach Bedarf liefern und sich einfügen lassen in umfassendere Systeme zur Wissensrepräsentation. Regelbasierte Systeme und Agententechnologien können dabei wertvolle Hilfsmittel sein.
Das Internet beschleunigt den Informations- und Entscheidungsprozess und durch die Interaktion von Anwendungen kann ein zusätzlicher Mehrwert geschaffen werden. Sulzer Innotec möchte diesen Mehrwert im Sinne ihrer Kunden nutzen.
Die Entwicklungen im Bereich Expertensystem und Agententechnologien werden zur Zeit intensiv bei Sulzer Innotec verfolgt und Sulzer Innotec sucht auf Basis dieser Ideen und Tools neue Lösungskonzepte.
Im Folgenden wird über Erfahrungen, Gedanken aus dem Projekt RecipFinder berichtet. Zusätzlich werden verschiedene Arten der Regeldefinition diskutiert, Auswahlkriterien für Architekturen zusammengetragen und ein Überblick über verschiedene Tools gegeben.

Die Entscheidungen über Architekturen und Vorgehen wurden unter pragmatischen Gesichtpunkten getroffen. Diese spiegeln die Kundensicht und die einer Entwicklungsgruppe eines Industrieunternehmens wider und unterscheiden sich von den forschungsorientierten Aspekten im universitären Umfeld. Der geneigte Leser mag aus diesen Entscheidungen Hinweise und Orientierungshilfen für seine Arbeit gewinnen.

2 Sulzer Innotec

Sulzer Innotec ist das Forschungs- und Entwicklungscenter des Sulzerkonzerns. Dies verpflichtet uns zur Orientierung an den besten und neuesten Techniken. Die Gruppe „Knowledge and Information Technology“ ist tätig in der Umsetzung des Kunden Know-hows in Anwendungen. Unsere Kunden verlangen nach Anwendungen, die besondere Anforderungen an die Analyse technischer Probleme und Umsetzung mit modernen SW-Methodiken stellen.

3 Projekt RecipFinder (Anforderungen)

Das Projekt RecipFinder wurde von einer Konzernunternehmung initiiert. Am Anfang stand die Idee unseres Kunden, das Internet als schnelles neues Medium für die Kunden zu nutzen. Dieses Projekt formuliert neben kundenspezifischen auch zwei allgemein typische Anforderungen. Allgemein hat die Bedeutung der Ressource Wissen für die Ausübung von Kernkompetenzen zugenommen. Zugleich hat die Konkurrenzsituation zu einer verstärkten Hinwendung zum Kunden geführt.

3.1 Verkürzung der Angebotsphase und Konsistenz

In Unternehmen besteht Bedarf nach einer schnellen Reaktion auf alle Anfragen. Durch die heutige Ausschreibungspraxis bei komplexen Projekten und die Einschaltung von Kontraktoren und Unterauftragnehmern werden aus einem konkret zu vergebenden Auftrag eine Vielzahl von zu beantwortenden Anfragen generiert. Diese zusätzliche Belastung des Engineering-Personals gilt es abzubauen. Unternehmen in nachfragegesteuerten Märkten haben wenige Möglichkeiten, diese Aufwendungen bei Anfragern in Rechnung zu stellen. Diese Unternehmen suchen daher nach Lösungen, diese Aufwendungen zu reduzieren. Intelligente Internetanwendungen sind eine Möglichkeit dazu.

3.2 Wissenssicherung

Die Konkurrenzsituation bewegt Unternehmen zur Konzentration auf Kerngeschäfte. Dabei sind organisatorische Umstellungen unvermeidbar. Damit einher geht oftmals ein Arbeitsplatzabbau. Diese Verschlankung der Unternehmen gefährdet den Erhalt des Know-hows.

Die Dokumentation des Know-hows ist oftmals unzureichend, personengebunden oder nur implizit vorhanden. Oftmals erschwert Konkurrenzverhalten innerhalb der Firmen die Wissenssicherung.

Zur Sicherung bietet sich als eine Möglichkeit der Aufbau eines Expertensystems an *[KOR02]*.

Das Projekt RecipFinder versucht eine Antwort auf die beiden obengenannten, allgemeinen Anforderungen zu geben. Daneben wurden weitere, teils spezielle, teils allgemeine Anforderungen an das Projekt gestellt.

Projektidee RecipFinder

- Angebot eines Kompressors
- Regel- und modellbasierte Ausarbeitung von Vorschlägen
- Nutzung eines thermodynamischen, mechanischen und Baureihengestützten Berechnungsmodells
- Orientierung an Kunden: Kunde gibt Aufgabenstellung aus seiner Sichtweise vor (Prozessdaten und Gasdaten). Maschinenkenntnis ist nicht erforderlich (Kein Konfiguratoren-System).
- Einfaches Interface: Browser
- Reduktion der notwendigen Eingaben
  Eingaben werden sinnvoll vorgewählt
- Unterstützung des Anwenders durch Plausibilitäts-Check der Eingabedaten und Erläuterung der Lösung
- Anwenderprofilabhängige Funktionen und Präsentation
- Ausgewählter Kreis von Anwendern

Hauptnutzen

- Beschleunigung der Angebotsphase
- Entlastung von Routineaufgaben
- Konsistente Bearbeitung der Anfrage
- Wissenssicherung des Unternehmens

Nebennutzen

- Anpassbarkeit durch Definition der Regelbasis in einer Datenbank
- Statistische Auswertung der Anfragen (Wer, Wie oft, welche Fälle)
- Datenübergabe an ein weiteres Tool zur Berechnung mit erweiterten Möglichkeiten,

4 Projekt RecipFinder (Architektur)

Während der Designphase wurden verschiedene Realisierungsmöglichkeiten einer Expertensystemlösung erwogen und getestet und die Anwendungsmöglichkeit verschiedener Technologien geprüft. Schliessendlich wurde eine einfache objektorientierte Lösung gewählt. Überraschend ist die stürmische Entwicklung im Bereich Expertensysteme. In diesem Umfeld wurden vor allem die zwei Begriffe „Business Rules“ und „Agententechnologie“ als treibende Kräfte ausgemacht.

Im Verlauf des Projektes waren vorrangig folgende Fragen zu klären:

- Welche Lösungen zur Abbildung von Regeln gibt es?

- Welche Möglichkeiten und Risiken bieten die Techniken im Umfeld Business Rules und Agententechnologien?

RecipFinder ist sowohl ein Expertensystem als auch ein Berechnungsprogramm für Kolbenkompressoren. Es ist als 3-Schichten-Architektur aufgebaut und besteht aus Client, Applikations- und Webserver und Datenbankserver.

4.1 Benutzerinterface

Das Benutzerinterface besteht aus dynamisch generierten Html-Seiten. Diese werden serverseitig auf einem MS-Internet-Information-Server (IIS) durch die Active-Server-Page-Erweiterungen (ASP) generiert. Clientseitig werden Javascripts zur Validierung von Eingaben genutzt.

4.2 Applikationslogik

Die Applikationslogik ist in einer Komponente auf dem Webserver gekapselt. Es besteht aus ineinander greifenden Teilen von Regeln und Modellen. Diese bilden Bauteilkataloge, „Good Practice"-Regeln und thermodynamische Modelle ab.

Daneben sind folgende Metaregeln zu programmieren:

- Optimierungsstrategien zur Einschränkung des Regelsuchraumes
- Konfliktlösungsstrategien bei widersprechenden Regeln oder Vorgaben
- Entscheidungsregeln bei Wahlfreiheit zwischen Optionen

4.3 Datenbank

Als Backend vorgegeben war eine Oracle8i Datenbank. Diese wurde schon für eine andere Anwendung zur detaillierten Auslegung von Kompressoren benutzt. Die beiden Anwendungen tauschen Daten aus. Die Datenbank ist aufgeteilt in mehrere Schemata, die mit separaten System.dsn angesprochen werden.

Die Architekturentscheidungen sind eher pragmatisch als theoretisch fundiert getroffen und orientieren sich an den Vorgaben des Kunden und den Sachzwängen von beschränkten Ressourcen, Zeit, Geld, Know-how und Risikoabwägungen betreffend die Nutzung von neuen Techniken.

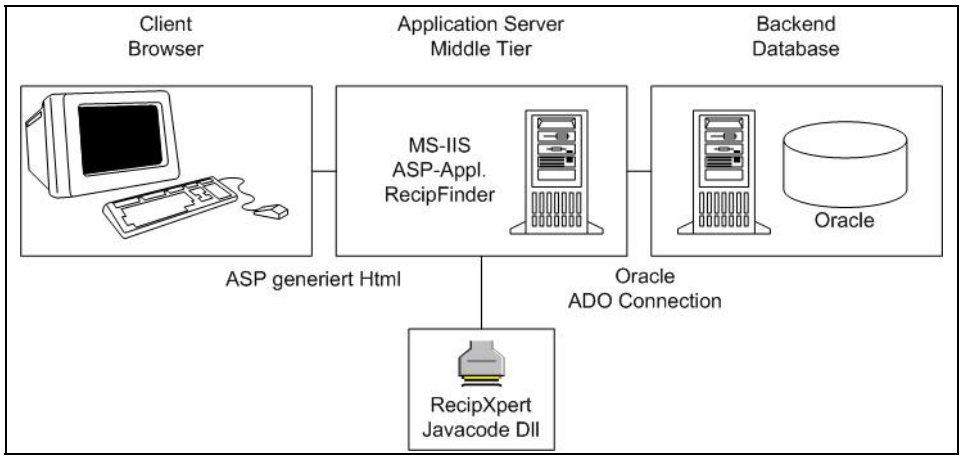


Fig. 1: RecipFinder Architektur

Das Userinterface besteht in der Hauptsache aus drei Seiten zur Spezifikation der Aufgabenstellung. Darüber werden bis zu 23 verschiedene Eingangsgrössen vorgegeben. Diese sind zum grossen Teil optional (Die Gaszusammensetzung wird als ein Parameter gezählt.). Die Definition von userabhängigen Defaulteinstellungen reduziert die zwingend einzugebenden Parameter auf minimal 6 Eingaben. Die Berechnungsergebnisse sind über einen Webbrowser einsehbar (Fig. 3).

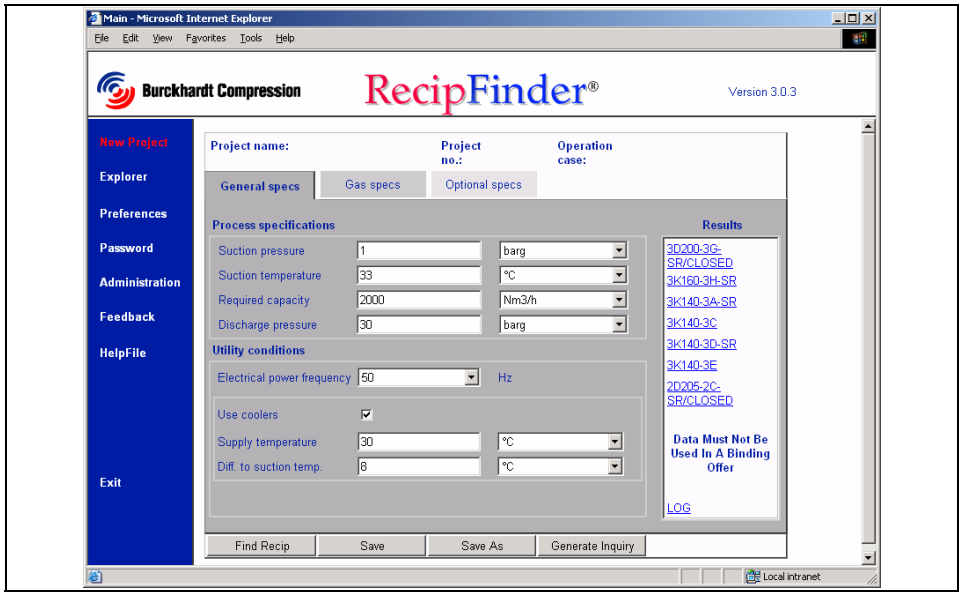


Fig. 2: RecipFinder Hauptseite: Allgemeine Spezifikation

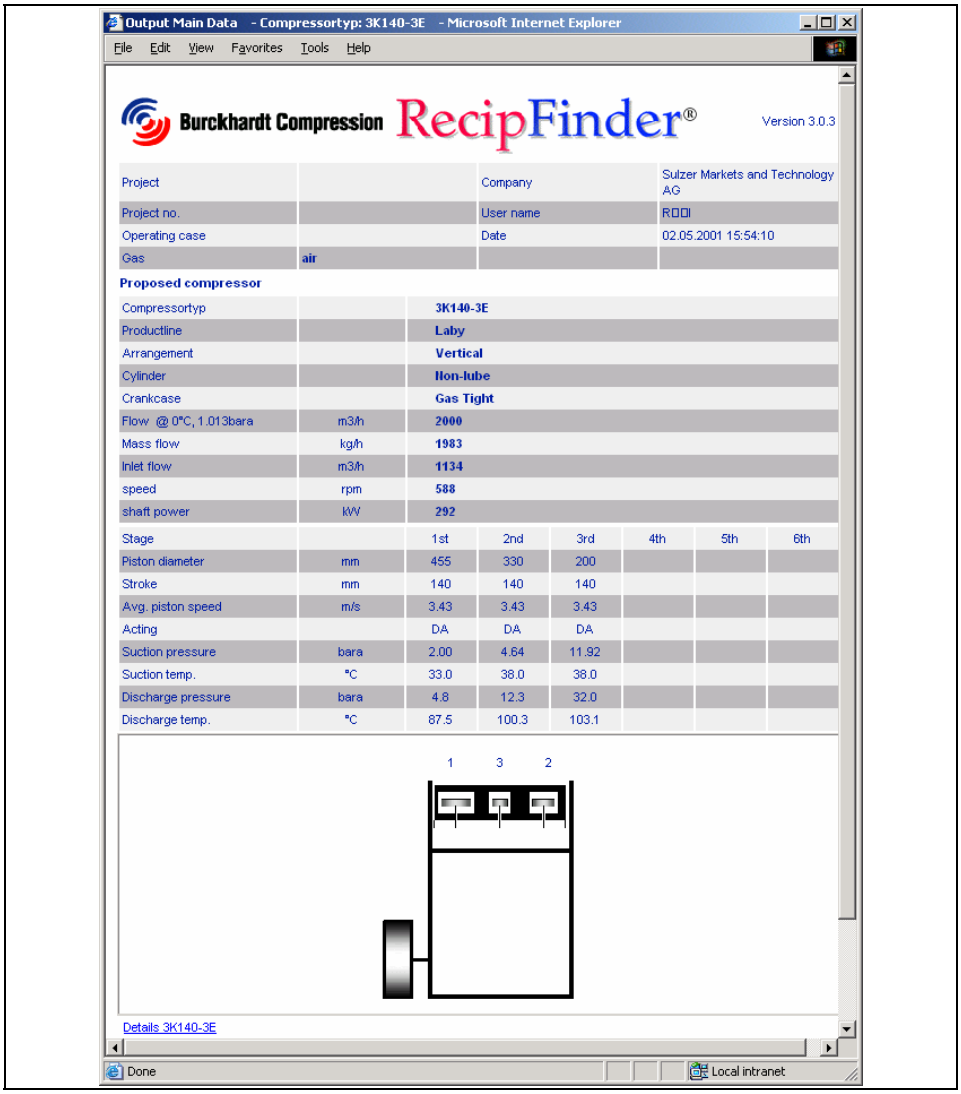


Fig. 3: RecipFinder Ergebnisseite

5 Projekt RecipFinder (Architekturüberlegungen)

5.1 Programmiersprache:

Die Erfahrungshintergründe, was KI-Sprachen angeht, sind generell in Industrieunternehmen eher gering. Die Einarbeitung in Lisp oder Prolog und das

Entwickeln eines Verständnisses für die Sprachkonzepte wird für relativ aufwändig gehalten *[MERR96]*.

Der Entscheid zugunsten von Java ist aufgrund der grossen Sprachdurchdringung und der vorteilhaften OO-Konzepte gefallen.

Als Entwicklungsumgebung wurde J++ gewählt. Dadurch konnte eine bereits bekannte Entwicklungsumgebung genutzt werden und es fielen keine neuen Lizenzkosten an.

5.2 Webinterface:

Zur Realisierung des Interfaces gibt es eine Reihe von Scriptsprachen. Zur Auswahl standen ASP, JSP und eine Applet-Lösung. Ausgewählt wurde eine ASP-Lösung.

ASP:	+	Strategische Ausrichtung unserer Entwicklung und unserer Kunden auf Microsoftprodukte
	+	Existenz eines Microsoft Foundation Classes (MFC) basierten Prototyps
JSP:	+	Bessere Anbindung von ORACLE mit JDBC und JSP
	+	Einfacheres Deployment auf dem Server
Applet:	-	Kunden schränken durch Security Policies die Nutzbarkeit aktiver Inhalte ein.

5.3 Benutzung einer Rule-Engine

Die hohen Preise für Entwicklungs- und Benutzungslizenzen bei den komfortablen Systemen wirken stark bremsend auf die Nutzung von Frameworks für regelbasierte Systeme. Auf der anderen Seite ist das Vertrauen in den Support, Weiterentwicklung und Komfort Forschungsgetriebener Programme gering. Deshalb wurden die Regeln in RecipFinder in einem objektorientierten Entwurf ausprogrammiert. Dies hat Vorteile für die Überprüfbarkeit der Ergebnisse, aber auch Nachteile bei der Wartung.

Folgende Punkte beeinflussten die softwaremässige Umsetzung der Regeln für RecipFinder:

- Angenommene durchschnittliche Komplexität der Aufgabe
- Auslegungsproblem und Verfahren bekannt
- Änderungsaufwand vergleichsweise gering, da die Auslegungsregeln sich nicht ändern und wesentliche Ergänzungen nur durch neue Baureihen erwartet werden (Diese ersten drei Punkte beruhen auf Annahmen und haben sich nur zum Teil bestätigt. In der Tat wird ein Teil des Wissens erst im Prozess der Regeldefinition abgebildet und zusätzliche Regeln haben Seiteneffekte auf andere Regeln, deren Implementationsaufwand schwer abzuschätzen ist.).
- Existenz eines Prototyps, programmiert mit J++ als Arbeitsplatzversion.
- Kosten einer Rule-Engine für Schulung und Lizenzen

- Lernfähigkeit des Systems war nicht gefordert. So standen neben wirklichen Rule-based Systems auch andere Varianten zur Abbildung des Regeldesign zur Verfügung.

5.4 Regel-Design

Für die Definition von Regeln sind verschiedene Möglichkeiten prozedural, objektorientiert, rule-based *[MERR96], [HASS00]* denkbar. Es gibt eine Fülle von Veröffentlichungen, Büchern und Tools zu dem KI-Thema „Rule based Systems“ und auch zu dem objektorientierten Business-Rules-Ansatz *([DATE00], [HALL02], [ROSS97]*).

Je nach Zahl der Regeln, Komplexität und Anforderungen an die Änderbarkeit sind unterschiedliche Vorgehensweisen zu empfehlen. Dabei sind folgende Punkte individuell zu berücksichtigen:

- **Ist die Regelabfolge im vornherein festlegbar?**
  Wenn dies zutrifft, so ist der Entwurf auch ohne Rule-Engine handhabbar
- **Wer erstellt die Regeln?**
  In den meisten Fällen werden die Regeln von einem Programmierer implementiert. Soll der Experte selbst die Regeln implementieren, so ist ein Regelframework zwingend nötig.
- **Wie oft sind die Regeln zu ändern?**
  Je öfter die Regeln geändert werden, desto grösser der Vorteil einer Rule-Engine, da dort der Pflegeaufwand voraussichtlich geringer ist.
- **Wer soll diese Regeln ändern?**
  Wenn Anpassung durch User gefordert, dann ist eine Expertensystemshell nötig, Wenn nur Parameter der Regeln anpassbar sein müssen, so kann über ein Datenbankinterface auch bei einem OO-Design die Anpassbarkeit gewährleistet werden.
- **Laufzeitverhalten und Stabilität**
  Bei einem OO-Entwurf verlässlicher
- **Voraussagbarkeit und Testbarkeit des Systems**
  OO-System von Vorteil, da besser nachverfolgbar
- **Kosten**
  Die erhöhten Investitionen für das Framework, Einarbeitung und die Lizenzen müssen mit den erwarteten Mehraufwendungen für Entwicklung und Pflege abgewägt werden.

Allgemein gilt: Je komplexer und unbekannter die Aufgabenstellung, desto eher sollte ein Rule-based-System zum Einsatz kommen. Folgend werden die verschiedenen Möglichkeiten der Regeldefinition anhand von Java-Code illustriert und mit Vor- und Nachteilen diskutiert. Dazu dient ein einfaches Beispiel zur Berechnung der volumetrischen Wirkungsgradverluste „deV“ aus dem Projekt RecipFinder.

5.4.1 Prozedurales Modell

Im prozeduralen Modell besteht der Programmcode aus einer Abfolge von Befehlen. Für Berechnungsprogramme mit wenigen Entscheidungsgrössen, die eher datengetrieben sind, ist ein prozeduraler Ansatz nahe liegend. Der Vorteil: Dieses Vorgehen ist bekannt und einfach.

Bei logikintensiven Teilen mit vielen verschachtelten „if-then-else"-Konstrukten und vielen Eingabegrössen wird dieses Vorgehen fraglich. Die Strukturen sind schwer zu überschauen und zu warten. Bei Änderungen und Erweiterungen des Codes sind Überarbeitungen sowohl im aufrufenden Programm als auch in dem aufgerufenen nötig. Dieses Vorgehen bedingt einen relativ grossen Änderungsaufwand und damit verbunden einen grossen Zeitanteil zum Testen und Debuggen.

Jedes Hinzufügen bedingt eine Analyse, welche Regeln betroffen sind, in welcher Reihenfolge welche Regeln abzuarbeiten sind.

```
//Coordinator
switch ( compType ){
case LABY:
dEV[i] = getDeltaEtaVolLaby(ts, ps, pd, m);
break;
case PROC:
dEV[i] = getDeltaEtaVolProc(ts, ps, pd, m);
break;
default:
throw new Exception („undefined Comptype");
break;
}
```

Listing 1: Prozedurales Muster Switchkonstrukt auf Aufrufebene „Coordinator"

5.4.2 Objektorientiertes Modell

Objektorientierte Sprachen wie Java erlauben die Kapselung des unterschiedlichen Verhaltens durch z.B. Vererbung. Logikstrukturen im Programm werden durch Vererbungsstrukturen ersetzt.

Bei einfachen Fällen - wie oben - gelingt es, die Unterschiede in der Businesslogik durch Vererbung auf eine untergeordnete Objektebene zu verschieben. Siehe auch *[FOWL00]*, Muster „Replace Conditional with Polymorphism", S.255. Dieses Vorgehen lässt sich auch beibehalten bei unterschiedlicher Parameterzahl durch Überschreiben der Methoden. Im Aufrufprogramm wird keine Fallunterscheidung getroffen. Die Klasse des Objektes „engine" entscheidet über die Auswahl der Berechnungsmethode.

```
//Coordinator
// Im Coordinator keine Fallunterscheidung
dEV = engine.getDeltaEtaVol(ts,ps, pd, m);

//Klassendefinitionen:
Class EngineLaby extends Engine{
….
public double  getDeltaEtaVol(double ts, double ps, double pd, double m){
….
erg = Do_x(ts,ps, pd, m);
return erg;
}}

Class EngineProc extends Engine
….
public double getDeltaEtaVol(double ts, double ps, double pd, double m){
            ….
erg = do_y(ts,ps, pd, m);
return erg;
}}
```

Listing 2: Nutzung der Vererbung

Variiert die Bedeutung, Typ und Zahl der an die Regel übergebenen Parameter, wie im wirklichen Fall (Listing 3), in dem die Methoden mit unterschiedlichen Parametern aufgerufen werden, gelingt eine Abstraktion über einen Callback-ähnlichen Ansatz. Das aufrufende Objekt wird an das anrufende übergeben (Siehe Listing 4 und Sequenzdiagramm.).

```
//delta eta volumetric function of ts, ps, pd, m
dEV = getDeVLaby(ts, ps, pd, m);

//delta eta volumetric function of  cm rho kappa epv
dEV = getDeVProc(cm, rho, kappa, epv);
```

Listing 3: Fallabhängige unterschiedliche Bedeutung der Aufrufparameter

Der Vorteil: Auf der aufrufenden Ebene müssen keine Fallunterscheidungen getroffen werden, die Engine-Objekte wissen, welche Objektdaten für die Ausführung der Funktion relevant sind und holen sich die Daten vom Coordinator-Objekt. Der Nachteil des Callback-Verfahrens: Die Klassen „Coordinator“ und „Engine“ werden aneinander gekoppelt und sich nicht unabhängig voneinander zu ändern.

```
//Coordinator
{//Methodendefinition
…
double get_ts(Coordinator coordinator){
// called by Engine
return coordinator.get_ts();}

//Aufruf der Regel
…
dEV = engine.getDeltaEtaVol(this);
…}

//Klassendefinitionen:
Class EngineLaby extends Engine{
….
public double  getDeltaEtaVol(Coordinator coordinator){
                        …
//Callback to coordinator
ts= get_ts(coordinator);
ps =get_ps(coordinator);
pd = get_pd(coordinator);
m = get_m(coordinator);
erg = do_y(ts,ps, pd, m);
return erg;
// oder kurz
//return do_yvar3(get_ts(coordinator), get_ps(coordinator),
get_pd(coordinator),get_m(coordinator));
}
//Methodendefinition
Class EngineProcc extends Engine{

public double  getDeltaEtaVol(Coordinator coordinator){
…
//Callback to coordinator
mc= get_cm(coordinator);
rho =get_rho(coordinator);
kappa= get_kappa(coordinator);
epv = get_epv(coordinator);
erg = do_y(mc,rho, kappa, epv);
return erg;}}
```

Listing 4: Callback an Coordinator-Objekt

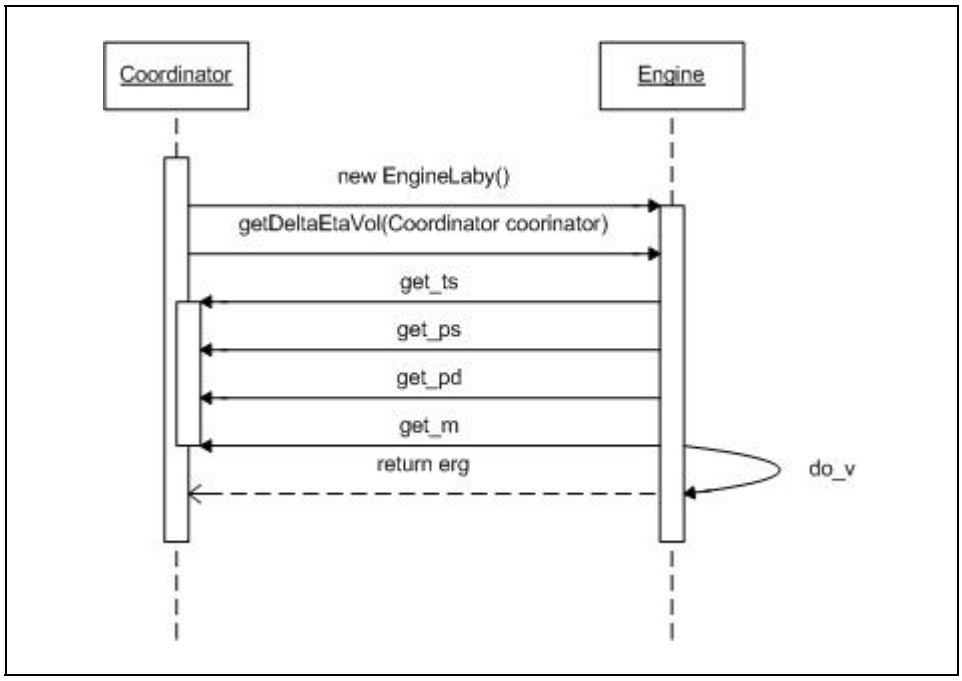


Fig. 4: Sequenzdiagramm für Callbackmechanismus

Das Design kann noch stärker objektorientiert gestaltet werden. Der Wert „dEV“ wird als privates Attribut zur Klasse „Engine“ deklariert. Der Funktionsaufruf „dEV = engine.getDeltaEtaVol(this); wird ersetzt durch zwei Aufrufe:

```
Class EngineLaby extends Engine{
private Coordinator _coordinator= null;
private double deV = DEV_START;
Public EngineLaby(Coordinator coordinator){
_coordinator = coordinator;}
public void setDeV(){
deV =  do_yvar3(get_ts(_coordinator), get_ps(_coordinator),
get_pd(_coordinator),get_m(_coordinator));
        }
public double getDeV(return deV;}}
```

Listing 5: Refactoring: Benutzung eines privaten Attributes

Allgemein verbessern Objekte die Modularität und damit die Wartbarkeit des Codes.

In diesem Beispiel wurde Vererbung genutzt, um die Logik in den Objekten zu kapseln und den Ablauf übersichtlicher zu machen. Wendet man dieses Verfahren an, so wird jeder Zweig in einer „if-then-else“-Regel ersetzt durch unterschiedliche

Methodenrümpfe in Unterklassen. Es entsteht eine Hierarchie wie in Fig. 5 Vererbungshierarchie gezeigt.

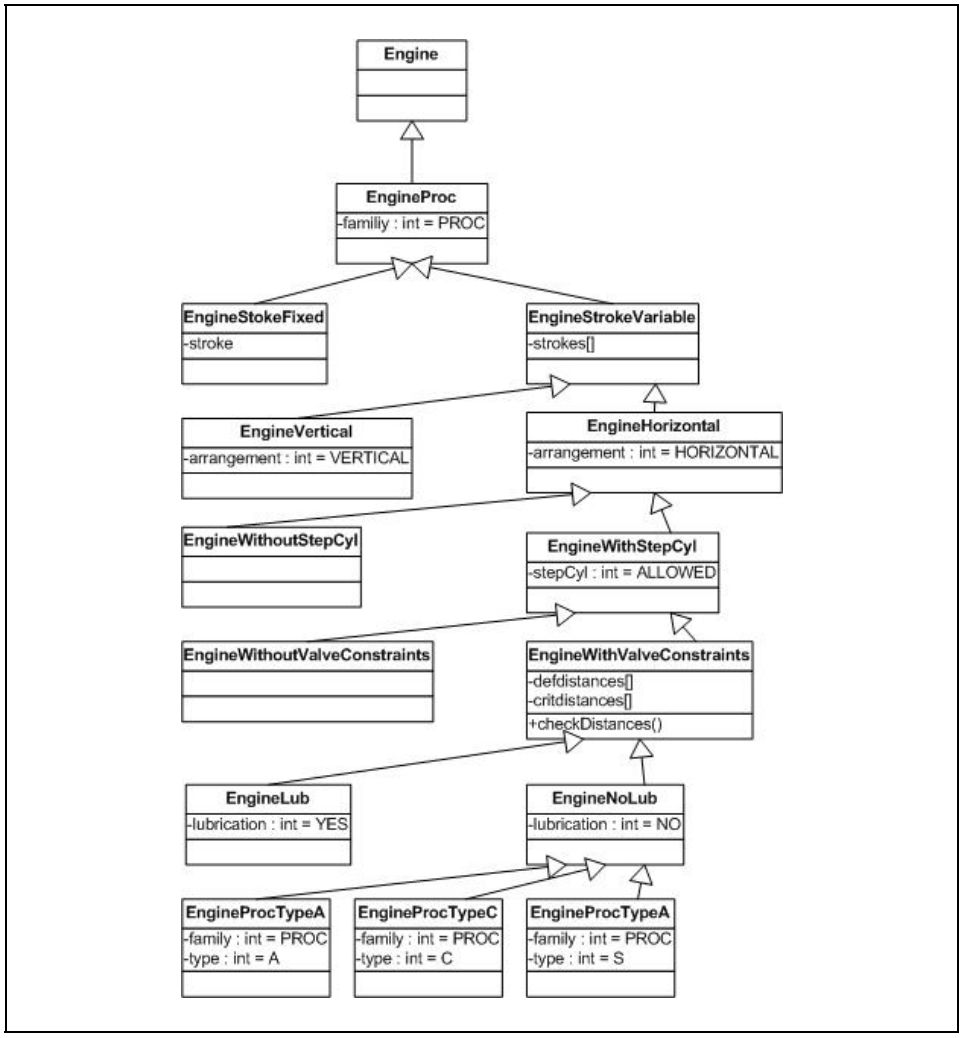


Fig. 5: Vererbungshierarchie für Kompressoren im Projekt RecipFinder

(Es sind nicht alle Vererbungspfeile gezeigt. Jede Unterkategorie ist mit den darüber liegenden kombinierbar.)

Der Nachteil dabei: Die Unterklassen werden an die Eigenschaften und Methoden der Oberklasse gebunden; das Hinzufügen neuer Regeln in Superklassen beeinflusst die nachgeordneten Klassen. Es entsteht eine grosse Vielfalt an Klassen, da jede Kombination auftreten kann. Im Projekt RecipFinder gibt es aufgrund der unterschiedlichen Baureihen und Eigenschaften eine Vielzahl von Unterklassen.

Es ist möglich und wünschenswert, die Menge an Klassenhierarchien zu reduzieren. Dazu gibt es zwei Möglichkeiten.

5.4.2.1 Nutzung des Wissens um die Regelabhängigkeiten:

Nicht jede Regel muss eine neue Hierarchiestufe deklarieren. In einem Regelwerk stehen nicht alle Regeln im Zusammenhang mit anderen. Bei Regeln, die andere Fakten nicht beeinflussen, können die Unterschiede durch Attribute in einem Feld mit zulässigen Möglichkeiten beschrieben werden. Bei RecipFinder ist dieses Verfahren angewendet worden, um die Klassenzahl zu begrenzen. Wenn alle unterschiedlichen Regeln eine neue Klasse definierten, so entstünde ein unnatürliches Objektmodell.

Int allowLubOptions[] = {LUB, NONLUB};

RecipFinder ist ein Programm, das zulässige und beste Lösungen findet. Sind beide Optionen für den Kompressortyp zulässig und nicht durch Regeln im Regelwerk für Gasdaten oder Uservorgaben im vornherein ausgeschlossen, entsteht Wahlfreiheit. Es ist dann zusätzlich nötig, eine Strategie festzulegen, welche Option genommen wird.

Der Nachteil bei dieser Art der Reduzierung von Klassenhierarchien ist: Im Falle des Hinzufügens neuer Baureihen und Regeln können die getroffenen Annahmen verletzt werden. Es sind wieder aufwändige Änderungen nötig.

5.4.2.2 Nutzung von Komposition statt Vererbung

Statt Vererbung kann Komposition verwendet werden. Dies ist ein patternbasiertes Vorgehen zur Vermeidung der Nachteile der Vererbungshierarchien. Bei Bloch werden die Vorteile und Nachteile diskutiert *[BLOCH01, Thema 14]*. Zur Vermeidung der Klassenvielfalt bietet sich z.B. das Dekoratormuster an *[GoF95,* S.175*]*. Dies bedeutet, dass das zugrundeliegende Objekt mit verschiedenen Eigenschaften und Methoden eingepackt „dekoriert“ wird. Auch dieses Vorgehen ist nicht unbeschränkt nutzbar.

Das Dekoratormuster eignet sich nicht für Callbackarchitekturen *[LIEB86*], zitiert *in [BLOCH01]*. Die Regelparameter müssen deshalb in einem anderen allgemeinen Lösungsansatz an die Funktionen übergeben werden. Dies gelingt durch die Übergabe als Feld von Strings oder durch eine Collection. Die Voraussetzungen des Dekoratormusters im Hinblick auf die Nutzung des Musters in einem Framework werden bei *[CINN96]* erläutert.

Wenn - wie hier erforderlich - mehrfach Dekoriermuster und diese in beliebiger Folge anwendbar sind, so ist im Code auch ein hoher Aufwand für die Weiterleitung aller Methoden erforderlich.

Ein anderer Ansatz mit dem „Type Object Pattern“ [*JOHN97]* erlaubt darüber hinaus zur Laufzeit das Erzeugen neuer Klassen.

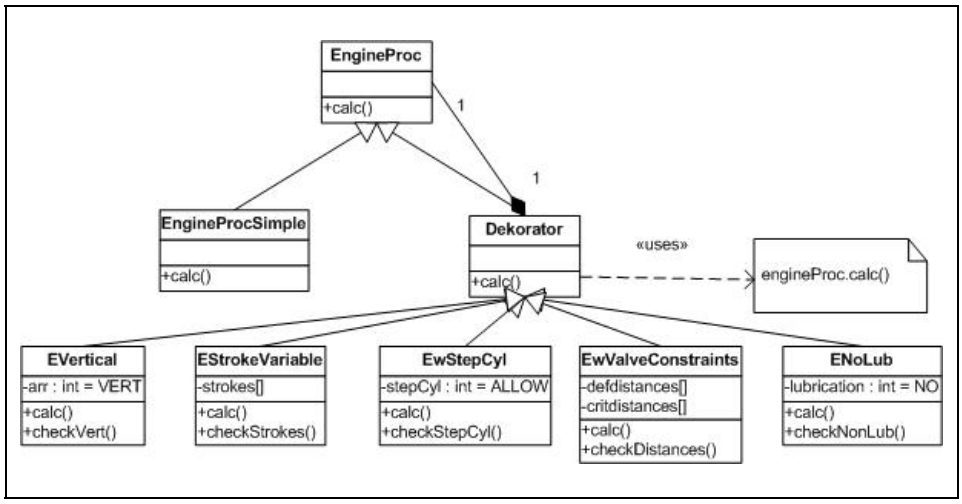


Fig. 6: Benutzung des Decorator Patterns

Wenn statt der Konstruktoren Factory-Methoden benutzt werden, bietet sich eine elegante Benutzung des Reflection-Api zum Ersatz der Switches dort an. Siehe *[FOWL00]* "Replace Constructor with Factory Method", S.306.

```
static Engine create(int type){
switch ( type){
case LABY:
return new EngineLaby();
case PROC:
return new EngineProc();
default:
throw new Exception("Unknown Engine Type");
}}

static Engine create(String  type){
try{
return (Engine) Class.forName(type).newInstance();
}
catch (Exception e){
throw new Exception("Unknown Engine Type");
}}
```

Listing 6: Ersatz der Fallunterscheidung bei Nutzung der Reflection API

5.4.3 Business-Rules-Ansatz

Der Business-Rules-Ansatz verfolgt die Ausgliederung der Regeln aus den Businessobjekten. Dazu werden diese Regeln durch ein Standard-Schema beschrieben. Die Regeln bestehen demnach aus beliebig vielen Bedingungsteilen und Aktionsteilen. Dies lässt sich gut in relationalen Strukturen speichern.

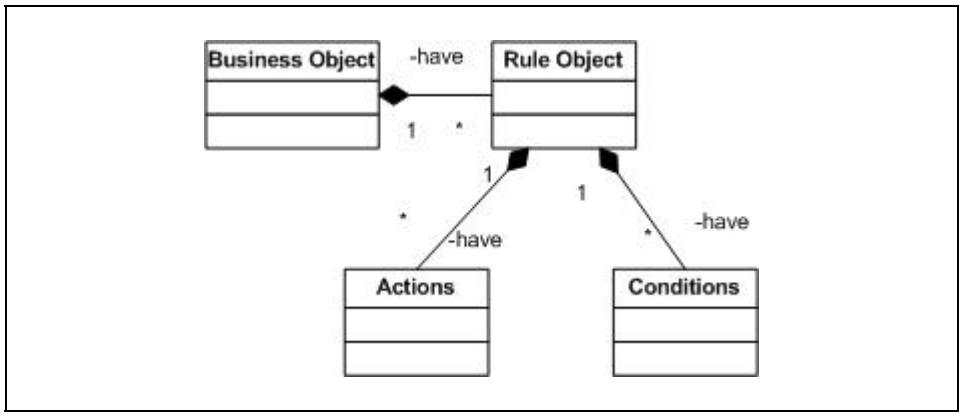


Fig. 7: Auskopplung der Business Rules aus dem Business Objekt

Zur weiterführenden Beschäftigung zur Umsetzung von Regeln mit Objektstrukturen siehe auch *[COR00], [ARS01]*. Assani benutzt zusätzlich noch das Entwurfsmuster „Komposition“ *[GoF95]* zur Verknüpfung der einzelnen Regeln zu Regelfolgen. Dieses Vorgehen vermeidet die vielgestaffelte Vererbungshierarchie und die Schwierigkeiten mit der Callback-Architektur bei dem Dekoratormuster. Die Regeln werden durch die nötigen Objekte „Rule Objects“, „Actions“ und „Conditions“ zergliedert. Um einen Überblick zu halten, bedarf es eines Frameworks.

5.5 Objektspeicherung im Persistenz-Layer

Die Businesslogik ist bei dem objektorientierten Vorgehensmodell an die Middle Tier gebunden. Zudem sind die Regeln im gesamten Quellcode verteilt. Jedes Hinzufügen einer Regel bedingt Anpassungen am Quellcode. Dieser Programmcode ist nicht direkt zugänglich. Eine Zugriffsmöglichkeit auf die Regeln ist aber oft wünschenswert.

Möchte man auf diese Logik zugreifen und einzelne Anpassungen über ein Datenbankinterface vornehmen, ist es denkbar, den Java–Code mit Hilfe von JAXP aus XML zu erzeugen *[OHL_01]* und in einer Datenbank zu speichern. XML gestützte oder orientierte Datenbanken wie z.B. Oracle 9i *[JANS02],* Cache, Jasmine oder Tamino sollten dazu eine gute Wahl sein.

Manchmal ist es hinreichend, den Zugriff nur auf wichtige Regelparameter zuzulassen. Dies reduziert das Problem der Speicherung der Daten in der Datenbank.

Die Ausgliederung der Regeln mit dem Business Rules Ansatz verbessert durch die Normalisierung der Regeln in Regel, Bedingungsteil und Aktionsteil die Möglich-

keiten zur Speicherung in dem relationalen Datenbankmodell. So kann auch die komplette Regel in der Datenbank editiert werden.

5.6 Rule based Model

Mit zunehmender Problemkomplexität werden regelbasierte Systeme interessant. Diese unterscheiden sich von herkömmlichen Systemen durch mehrere Komponenten:

- Inference-Engine (Entscheidet über die Abarbeitung der Regel)
- Rule-Editor (Grafisches Interface zur Regeleingabe)
- Erklärungskomponente (Generierte Lösungen werden mit Lösungsweg aufgezeigt)
- Truth-Maintenance (Fakten werden auf Widerspruchsfreiheit getestet.)

Hier sei nur auf die Inference-Engine eingegangen: Die Inference-Engine bestimmt den Ablauf der Regelabarbeitung.

Das Verhalten des Systems ist dadurch nicht im vornherein festgelegt, es kann durch Reihenfolgeeinflüsse vom gewünschten Verhalten abweichen. In diesem Fall kann der Programmierer zusätzlich Metaregeln definieren. Diese steuern den Ablauf von Regeln.

Vorteil:

- Der deklarative Ansatz entlastet den Programmierer in der frühen Phase des Projektes von der Festlegung der Regelabfolge. Im Sinne eines Rapid-Prototypings kann man sich auf die Aufnahme der Regeln konzentrieren.
- Geringe Wartungs- und Änderungskosten

Nachteil:

- Falls die Regelabarbeitung nicht das gewünschte Ergebnis erzielt, ist ein Verfolgen der Regelabarbeitung im Debugging allerdings mühevoll.
- Vorhersage der Laufzeitverhaltens schwierig
- Kosten

Die Syntax von verfügbaren Systemen orientiert sich an vorhandenen Sprachen der KI wie LISP und Prolog.

Die Definition der Regeln besteht aus Namen, Bedingungsteil und Aktionsteil:

(defrule Name (Conditionteil)(Actionsteil))

Die Abarbeitung der Regeln ist der zeitbestimmende Faktor bei der Nutzung eines Expertensystems.

Ein besonderes Augenmerk liegt deshalb auf einem effizienten Regelabarbeitungs-algorithmus Ein besonders effizienter Algorithmus ist der Rete-Algorithmus. Siehe dazu auch *[FORG82]*.

6 Zusammenfassung der Erfahrungen im Projekt RecipFinder

Es konnten und sollen hier nicht alle Varianten des Regeldesigns vorgestellt werden. Es ist möglich, einfache Expertensysteme mit einfachen Ansätzen zu realisieren. Bei komplexen Systemen macht die Ausgliederung der Regeln aus den Businessobjekten und die Nutzung eines Frameworks Sinn. Die Gesamtkosten werden von Lizenzkosten und Wartungskosten bestimmt. Je nach Änderungsaufwand sind verschiedene Entscheidungen vertretbar. Die Erfahrungen mit dem Projekt RecipFinder zeigen:

- Trotz intensiver Diskussion werden nicht alle Regeln in der Designphase erkannt. Vielmehr wird ein Teil des Expertenwissens erst während der Testphase explorativ erschlossen. Die nur teilweise Aufdeckung des Expertenwissens erfordert eine vermehrte Anpassung im Bereich der Regeln. Bei neuen Regeln müssen diese in das bereits vorhandene Konzept eingegliedert werden (Veränderungen an Abfolge, Logik und Parameterübergabe).
- Dies führt zu einem fortwährenden Entwicklungsbedarf im Verlauf der Aufdeckung des Experten-Know-hows. Der Anteil für Tests und Überprüfung der Ergebnisse ist relativ hoch.
- Ein Satz an überprüfbaren Aufgabenstellungen und gewünschten Lösungen ist wichtig
- Die Regelwerke sind zudem für den Programmierer nicht im vornherein mit allen Konsequenzen überschaubar.
- Die Menge an Regeln wird von Experten unterschätzt
- Die Verwendung eines regelbasiert arbeitenden Framework führt wohlmöglich zu einer Verkürzung der Projektdauer geführt

7 Ausblick auf Expertenlösungen im zukünftigem Web

Während des Designkonzeptes für das Projekt RecipFinder versuchten wir uns ein Bild über den bestehenden Softwaremarkt im Bereich Expertensysteme zu machen. Nach der Ernüchterungsphase der 80iger Jahre haben wir damit gerechnet, dass die Entwicklung in diesem Gebiet stagniert. Tatsächlich hat sich der Focus verschoben. Die Methoden sind erweitert worden durch Datamining und neuronale Strukturen, genetische Algorithmen und Fuzzy-Sets *[LUG02]*.

Das Spektrum von zur Verfügung stehenden Anwendungen reicht von sehr umfangreichen Anwendungen zur Einbindung von Businesslogik in Webapplikationen wie z.B. Neugents, Ilog, Blaze Advisor bis zu Applets oder OCX-Komponenten (Haley).

Eine grosse Bandbreite an Sprachen (Java, Prolog, Lisp, C, VB) steht zur Verfügung. Auffällig die Häufung im Bereich Java. Die Javaplattform ist oftmals Grundlage für KI-Systeme, die mit Lisp oder Prolog ähnlicher Syntax arbeiten (z.B. Jess).

Im Anhang sind einige Techniken zusammengetragen, die von Innotec beurteilt und getestet wurden. Aufgrund der rasanten Entwicklung und der wenigen veröffentlichten Vergleiche *[LAMM01], [LAUR02]* gelingt es schwer, einen rechten Überblick zu halten und so kann diese Liste nicht vollständig sein.

Als Triebkraft für die Weiterentwicklung regelbasierter Systeme wurden vor allem die beiden Begriffe Business Rules und Agententechnologien ausgemacht. Eine Begründung für diese Begriffe und eine Abgrenzung von anderen Begriffen ist nicht offensichtlich und wird im weitern Verlauf erläutert.

Der Begriff Business Rules (BR) *[ROSS97], [HALL02], [DATE00]* beschreibt den Teil eines Business Objektes, der sich erfahrungsgemäss relativ häufig ändert. Die beteiligten Businessobjekte mit ihren Methoden und Werten werden im Vergleich dazu relativ wenig geändert.
Softwaremässig unterscheiden sich Business Rules (BR) und Rules nicht. Sie sind gleichermassen definiert durch Konditionsteil und Aktionsteil.

Vor allem der Wunsch nach schneller und sicherer Anpassbarkeit der Regeln, gerade auch in Internetanwendungen, treibt die Branche an. Ist schnelle und wartungsarme Änderbarkeit gefordert, spielen regelbasierte Systeme ihre spezifischen Vorteile aus.

Expertensysteme deklarieren Wissen. Das Wissen kann über das Web zur Verfügung stehen. Es existieren dazu formalisierte Sprachkonzepte wie KQML FIPA-ACL und DAML+OIL, *[FINN92]* die den Wissensaustausch standardisieren. Das heutige Problem besteht darin, diese Information zu finden.

Der zweite neu auftauchende Begriff Agent steht in diesem Zusammenhang. Es gibt zwei Auslegungen des Begriffes Agenten: Die schwache Form bietet autonomes Verhalten, Sozialverhalten, Reaktivität, Proaktivität und zielgerichtetes Handeln. Die starke Form fordert zusätzlich Mobilität. Die Bedeutung der Begriffe wird z.B. bei Hermans erklärt *[HERM97]*. Das Agentenkonzept bietet viele Anwendungsmöglichkeiten: Die Personalisierung von Inhalten und Informationsfilterung, Robotersteuerung, natursprachliche Kommunikation. *Ein Hauptimpuls für das* Agentenkonzept ist das Finden und Filtern von Informationen im Netz. Als weiterer Impulsgeber für die Agententechnologie wirkt der Wünsch nach autonomen Verhalten von Robotern. Ferber beschreibt dazu verschiedene Organisationsformen und Architekturen *[FERB01]* S.127, S.148.

Tools wie RecipFinder deklarieren Wissen, z.B. zur Auslegung von Kolbenkompressoren. Ein Ansatzpunkt zur Benutzung der Agententechnologie ist die verteilte Speicherung von Wissensgebieten. Ähnlich wie die Komponententechnik gewisse Funktionalität in Komponenten speichert, können Domain Agents das Wissen zur Verfügung stellen und von Broker Agents angesprochen werden.

Für die Kompressorberechnung bietet sich die Aufsplittung in folgende Wissensgebiete an:

- Einheitenumrechnung
- Gasdynamik im Kompressor
- Schmierung
- Gasdatenbank (Gift/ Korrosion/ Entzündbarkeit/Feststoffe)
- Materialeigenschaften
- Mechanische Auslegung
- Baureihen Katalog
- Anwendbarkeitsgrenzen Thermodynamischer Gasgesetze

- Thermodynamik von Gasen

Denkbar ist auch eine Einbindung in einen grösseren Kontext der Anlagenplanung. Anlagenplanung ist ein iterativer Prozess. Die Einzelkomponenten wirken zurück auf den Gesamtprozess und können zu entscheidenden Änderungen führen.

Was verhindert eine Agentenlösung? *[GUTT97], [ABB02], [NWA99], [JENN00]* Kommunikationsprotokolle und Tools zur Erstellung von Agenten stehen zur Verfügung *[ODE00]*. Wirkliche Agentenlösungen sind dagegen eher rar gesät *[TVEIT01*], *[NWA99]*. Bedarf es erst einer Erweiterung des Web *[BERN01]?* Durch Tags mit semantischem Inhalt wäre eine computerverständliche Suche nach Themengebieten möglich *[SMW3C].* Diese Tags beschreiben nicht wie in der Html-Sprachdefinition die Formate, sondern die Inhalte. Es ist vorstellbar, dass dies der Agententechnologie entscheidenden Auftrieb zu geben vermag.

Die erwähnten Eigenschaften wie Proaktivität sind in einem unbekannten Umfeld wie dem Internet von Vorteil. Denkt man aber an eine B2B-Anwendung, so lässt sich eine Anwendung wie RecipFinder auch als Zusammenschluss von Webservices definieren, die nur auf Anfragen reagieren.

Seit 1995 *[WOOL95]* ist zum Thema Agenten viel geschehen. Die nähere Zukunft wird zeigen, ob sich dieses Konzept durchsetzen kann und wieweit ein semantisches Netz dazu Vorbedingung ist.

Anhang

8 Übersicht KI-Produkte und relevante Technologien

Folgende Produkte sind im Umfeld der KI Expertensystemtechnik und Agenten-technologie analysiert worden.

Produkt	***Hersteller***	***Link***
Übersichten Klassische Expertensysteme: http://www.pcai.com/web/ai_info/expert_systems.html http://www.cs.berkeley.edu/~russell/ai.html		
Assistum	*Assistum*	*http://www.assistum.com (VBA)*
Brokat	Advisor rule service	http://www.blaze-advisor.net/brokat_advisor.htm
ProKappa	IntelliCorp	http://www.intellicorp.com
Cyc	Cycorp	http://www.cyc.com/cycl.html
eGain Knowledge Agent	Inference Ltd	http://www.egain.com/docs/products/egain_knowledge_agent.pdf
Firepond	Brightware	http://www.brightware.com
GENSYM	Gensym Corporat	http://www.gensym.com
XpertRule Knowledge Builder	Attar Software products	http://www.attar.com/pages/latest.htm
Java Expertensysteme		
Jess	Sandia National Laboratories	http://herzberg.ca.sandia.gov/jess/ (Rete-Algorithmus)
Jess Rulebase Translator	ontogenics	http://www.ontogenics.com/research/default.htm
Jeops	Carlos Figueira Filho	http://www.di.ufpe.br/~jeps/
JProlog	Bart Demoen and Paul Tarau	http://www.cs.kuleuven.ac.be/~bmd/PrologInJava/README
Amzi	Amzi	http://www.amzi.com/
New Business Oriented Expertensysteme		
Ilog JRules	Ilog	http://www.ilog.com/products/rules/engines/jrules/ (Rete-Algorithmus)
Neugent Technology	Computer Associates	http://www.ca.com/Solution
Hugin Decision Engine	Hugin	http://www.hugin.com/products/software/Feature_List/Feature_List_for_the_HDE.article

Ecclipse	Haley	http://www.haley.com (Rete-Algorithmus)
Übersichten Agententechnologien: http://agents.umbc.edu/technology/frameworks.shtml http://www.ida.liu.se/~flagr/Agents.html http://www.ida.liu.se/labs/iislab/courses/Agents/2002/presentations/ail.ppt. http://www.ece.arizona.edu/~rinda/compareagents.html#OAA http://www.agentbuilder.com/AgentTools/academic.php http://www.multiagent.com/Software/Commercial_Products/index.html http://www.csc.liv.ac.uk/~mjw/links/ http://www.ida.liu.se/~flagr/Agents.html		
Open Agent Architecture OAA	SRI International Artificial Intelligence Center	http://www.ai.sri.com/~oaa/ http://www.openagent.com/
JAFMAS & JIVE	University of Cincinnati	http://www.ececs.uc.edu/~abaker/JAFMAS/ (JIVE) JAFMAS integrated Visual Environment http://www.ececs.uc.edu/%7eabaker/JiVE/index.html
Java Agent Template	Stanford University Rob Frost	http://java.stanford.edu/java_agent/html/ (Overseeded by JatLlite)
JatLite	Stanford University	http://www-cdr.stanford.edu/ABE/JavaAgent.html
Agent-K	Stanford University	ftp://ftp.csd.abdn.ac.uk/pub/pedwards/PUBS/cikm94.ps.Z Integration von AOP und KQML
Cougaar:	Cougaar.org	http://www.cougaar.org
Aglets	IBM	http://www.trl.ibm.com/aglets/index.html
Übersicht Repräsentation von Wissen: Tutorial on Knowledge Markup Techniques http://www.dfki.uni-kl.de/km/knowmark/		
UMBC KQML	Lab for Advanced Information Technology by Tim Finin	http://www.csee.umbc.edu/kqml/ http://www.cs.umbc.edu/kqml/kqml-applications.html
KRSL	DARPA/Rome Laboratory Planning and	Knowledge Representation Specification Language (KRSL)

	Scheduling Initiative	
Rule Markup Language	DFKI GmbH	http://www.dfki.uni-kl.de/ruleml/
DAML+OIL	DARPA Agent Markup Language (DAML) Program	http://www.daml.org/2001/03/daml+oil-index.html http://www.daml.org/2001/03/daml+oil-walkthru.html#xml-guide
XML RDF	W3C	http://www.csee.umbc.edu/kqml/ http://www.w3.org/RDF/Overview.html
The ICS-FORTH RDFSuite		(Parser Supporting RQL)
IsaViz:		Emmanuel Pietriga of W3C and Xerox Research Centre Europe (Visual authoring tool for browsing and authoring RDF models represented as graphs.)
Jena Java RDF API and toolkit	W3C	Brian McBride, Jeremy Carroll, Andy Seaborne, Dave Reynolds and Ian Dickinson, HP Labs (API for manipulating RDF models including statement the RDQL query language, See the jena discussion list for more information.)
Lalo	Computer Research center In Montreal	(Sprache und Framework für KQML)
Datenbanken		
Webgain Toplink:		http://www.webgain.com/products/toplink/
Oracle 9i:		http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_92twp.pdf
Cache:		http://cache.intersys.com/cache/edbms.html
Jasmine:		http://www.cavo.com/solutions/jasmine.asp
Tamino:		http://www.softwareag.com/tamino/
Andere relevante Technologien		
WSDL	W3C	Web Services Description Language http://www.w3.org/TR/wsdl
UDDI	Uddi.org	Universal Description, Discovery and Integration http://www.uddi.org/
SOAP	W3C	Simple Object Access Protocol http://www.w3.org/TR/SOAP/

9 Literaturnachweis

9.1 BusinessRules

[DATE00]
C. J. Date: What, Not How: The Business Rules Approach to Application Development, Addison-Wesley, 2000.

[HALL02]
Barbara von Halle: Business Rules Applied: Building Better Systems Using the Business Rules Approach; John Wiley & Sons, 2002

[ROSS97]
Ronald G. Ross: The Business Rule Book: Classifying, Defining and Modeling Rules, Version 4.0, Oasis Press, 1997

9.2 Wissenssicherung

[KOR02]
Stefan Koruna, Andri Fry: Wenn Experten gehen; New Management, Nr. 4, S.32-39, 2002

9.3 Softwareengineering

[GoF95]
E.Gamma, R.Helm, R. Johnson, J.Vlissides: Design Patterns, Addison-Wesley, 1995

[FOWL00]
Martin Fowler: Refactoring, Improving the design of existing Code; Addison-Wesley, 2000

[BLOCH01]
Joshua Bloch: Effective Java Programming, Addison –Wesley, 2001

[JOHN97]
Ralph Johnson, Bobby Woolf: The Type Object Pattern, 1997, Knowledge Systems Corporation, http://www.ksccary.com/article3.htm

[LIEB86]
Henry Lieberman: Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems; 1986; SIGPLAN Notices; http://citeseer.nj.nec.com/lieberman86using.html

[CINN96]
Mel O' Cinne'ide: Towards Automating the Introduction of the Decorator Pattern to Avoid Subclass explosion, Department of Computer Science University College Dublin Ireland., 1996, www.cs.ucd.ie/staff/meloc/home/papers/OOPSLA96.ps

9.4 Rules

[ARS01]
Ali Arsanjani:, Rule Object 2001: A Pattern Language for Adaptive ans Scalable Business Rule Construction, http://www.arsanjani.org/public/RPL2001.pdf

[CARL98]
Dave Carlson: Rules and Web-Object Systems, Object Magazine, June 1998 http://www.ontogenics.com/research/papers/ObjectMag/June98.htm

[COR00]
Paul Corazza: Using the If-then-else-framework, Java World, 2000, http://homepages.kdsi.net/~pcorazza/javaArticles.html

[Forg82]
Charles L. Forgy: Rete: A fast Algorithm for the many pattern/many object pattern match problem, AI 19, S.17-37, 1982

[HASS00]
Mehrdad Hassani: A Component-Based Methodology for Real-Time Decision-Making in Embedded Systems, Ph.D. Dissertation, Electrical and Computer Engineering Dept., University of Maryland, College Park. 2000, Chapter 2 „Decision-Making Software

[LAMM01]
E. Lamma, L. Maestrami, P. Mello, F. Riguzzi and S. Storari: Rule-based Programming for Building Expert Systems: a Comparison in the Microbiological Data Validation and Surveillance Domain; http://www.elsevier.com/gej-ng/31/29/23/89/33/36/59.4.010.pdf

[LAUR02]
Guido Laures: Regelgerecht Anpassbare Geschäftsregeln durch Rules Engines; iX 3/02, Seite 110-117, 2002

[LUG02]
George F. Luger: Artificial Intelligence; Pearson Education Limited, 4th Ed., 2002

[MERR96]
Dennis Merritt: PC AI magazine, Volume 10, Number 2, Mar/Apr, 1996; http://www.amzi.com/articles/rule_engines.htm

9.5 Mapping Object to Databases

[AMBL00]
Scott W. Ambler: Mapping Object to relational Database, 2000, http://www.ambysoft.com/mappingObjects.pdf

[FOWL_01]
Mapping to a Relational Database, ThoughtWorks, http://martinfowler.com/isa/OR-mapping.html

[JANS02]
Rudolf Jansen, JavaMagazin Grosse Koalition Integration von XML-Technologien in Oracle 9i, XML-Magazin, 4.2002; S.88-93

9.6 Semantic Web

[BERN01]
Tim Berners-Lee, James Hendler, Ora Lassil: The Semantic Web, Scientific American, May 2001; http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2

[SMW3C]
W3C: Semantic Web: http://www.w3.org/2001/sw/

9.7 Agenten

[ABB02]
Michael Abbott: xml webservices one conference, 2002, http://www.xmlconference.com/sanjose/speakers2.asp?faculty_code=1372

[FERB01]
Ferber Jacques: Multiagentensystem: Eine Einführung in die verteilte künstliche Intelligenz, Addison-Wesley, 2001 dt. Übersetzung. Original 1995 Les Systèmes Multi-Agents, InterEditions, Paris, 1995

[FINN92]
Finin, T., D. McKay, and R.Fritzson (eds.): An overview over KQML A Knowledge Query and Manipulation Language KQML Advisory Group, March 1992

[GUTT97]
Rob. Guttmann: Software Agents and Electronic Commerce Beyond the Hype Rob Guttman, Software Agents Group, MIT Media Lab, 1997; http://web.media.mit.edu/~guttman/research/commerce/talk6/slide1.html

[HERM97]
Björn Hermans, Intelligent Software Agents on the Internet, http://www.firstmonday.dk/issues/issue2_3/ch_123/ ; 1997

[JENN00]
Nicholas R. Jennings: On agent-based software engineering, Artificial Intelligence, 2000

[ODE00]
James Odell: Agents (Part 2): Complex Systems," Executive Report, Volume 3, Number 6, Cutter Consortium, Arlington, MA, 2000; http://www.jamesodell.com/Cutter_Agent_Reports.zip

[NWA99]
Hyacinth S. Nwana and DivineT. Ndumu: A Perspective on Software Agents;

Research Applied Research & Technology Department British Telecommunications; The Knowledge Engineering Review; 1999; http://agents.umbc.edu/introduction/hn-dn-ker99.pdf.

[TVEIT01]
Amund Tveit: A survey of Agent-Oriented Software Engineering. NTNU Computer Science Graduate Student Conference. Norwegian University of Science and Technology, May 2001. Kap. 5.1.; Agents in the real world http://csgsc.idi.ntnu.no/2001/pages/papers/atveit.pdf

[WOOL95]
M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115--152, 1995

10 Bildnachweis

Fig. 1: RecipFinder Architektur 5
Fig. 2: RecipFinder Hauptseite: Allgemeine Spezifikation 5
Fig. 3: RecipFinder Ergebnisseite 6
Fig. 4: Sequenzdiagramm für Callbackmechanismus 12
Fig. 5: Vererbungshierarchie für Kompressoren im Projekt RecipFinder 13
Fig. 6: Benutzung des Decorator Patterns 15
Fig. 7: Auskopplung der Business Rules aus dem Business Objekt 16

Listing 1: Prozedurales Muster Switchkonstrukt auf Aufrufebene Coordinator 9
Listing 2: Nutzung der Vererbung 10
Listing 3: Fallabhängige unterschiedliche Bedeutung der Aufrufparameter 10
Listing 4: Callback an Coordinatorobjekt 11
Listing 5: Refactoring: Benutzung eines privaten Attributes 12
Listing 6: Ersatz der Fallunterscheidung bei Nutzung der Reflection API 15

Inhaltsregister

1 Einleitung 1
2 Sulzer Innotec 2
3 Projekt RecipFinder (Anforderungen) 2
3.1 Verkürzung der Angebotsphase und Konsistenz 2
3.2 Wissenssicherung 2
4 Projekt RecipFinder (Architektur) 3
4.1 Benutzerinterface 4
4.2 Applikationslogik 4
4.3 Datenbank 4
5 Projekt RecipFinder (Architekturüberlegungen) 6
5.1 Programmiersprache: 6
5.2 Webinterface: 7
5.3 Benutzung einer Rule-Engine 7
5.4 Regel-Design 8
5.4.1 Prozedurales Modell 9
5.4.2 Objektorientiertes Modell 9
5.4.3 Business-Rules-Ansatz 16
5.5 Objektspeicherung im Persistenz-Layer 16
5.6 Rule based Model 17
6 Zusammenfassung der Erfahrungen im Projekt RecipFinder 18
7 Ausblick auf Expertenlösungen im zukünftigem Web 18
8 Übersicht KI-Produkte und relevante Technologien 21
9 Literaturnachweis 24
9.1 BusinessRules 24
9.2 Wissensicherung 24
9.3 Softwareengineering 24
9.4 Rules 25
9.5 Mapping Object to Databases 25
9.6 Semantic Web 26
9.7 Agenten 26
10 Bildnachweis 28