Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-95-10

Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON

Volker Ehresmann

Oktober 1995

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON

Volker Ehresmann

DFKI-D-95-10

Diese Arbeit wurde finanziell unterstützt durch das Bundesministerium für Forschung und Technologie (FKZ ITW-9405).

ISSN 0946-0098

Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON

Volker Ehresmann

Oktober 1995

Zusammenfassung:

Im wesentlichen werden bei der wissensbasierten Konfigurierung zwei Ansätze unterschieden: struktur-orientierte und ressourcen-orientierte Konfigurierung. Beim struktur-orientierten Ansatz bezieht man sich auf den strukturellen Aufbau der verwendeten Komponenten(gruppen), wobei für die entsprechende Wissensmodellierung "is-a"- und "has-part"-Beziehungen charakteristisch sind. Im Gegensatz dazu werden beim ressourcen-orientierten Ansatz ausschließlich die funktionalen Eigenschaften der Komponenten betrachtet, welche durch geforderte bzw. bereitgestellte Ressourcen modelliert werden.
Realistische Konfigurierungsaufgaben lassen sich allerdings sehr schwer mit nur einem der beiden Ansätze lösen - vielmehr tauchen in einer Anwendung oft Aspekte beider Paradigmen auf. Deshalb wird in der vorliegenden Arbeit ein Ansatz vorgestellt, der beide Paradigmen in einem Konfigurierungssystem integriert. Dazu wird das struktur-orientierte Konfigurierungssystem TOOCON um ressourcen-orientierte Techniken erweitert. So erhält man einerseits eine erhöhte Ausdrucksmächtigkeit und andererseits eine Verzahnung der beiden unterschiedlichen Inferenz-Prozesse.
Außerdem wird deutlich, daß über die Zwischenstufen "lokale Ressourcen" und "Verbindungskonzept" sogar ein fließender Übergang von ressourcen-orientierter zu struktur-orientierter Konfigurierung möglich ist.

Abstract:

Essentially, one can distinguish two approaches for knowledge-based configuration: structure-oriented and resource-oriented configuration. The structure-oriented approach refers to the structural composition of the (groups of) components used, where "is-a"- and "has-part"-relations are typical for the corresponding knowledge modelling. In contrast, the resource-oriented approach only considers the functional characteristics of a component, which are modelled by required and supplied resources.
But it is hard to solve realistic configuration problems with only one of these approaches - usually there are aspects of both paradigms in an application. Therefore in the present work an approach is introduced that integrates both paradigms in one configuration system. For that purpose the structure-oriented configuration system TOOCON is extended by resource-oriented methods. In this way one gets on the one hand higher expressiveness and on the other hand cooperation of the two different inference mechanisms.
In addition it actually turns out that, via the intermediate stages of "local resources" and "connection-concept", a smooth transition from resource-oriented to structure-oriented configuration is permitted.

Inhaltsverzeichnis

1. Einleitung und Motivation5

1.1 Was ist eine Konfigurierungsaufgabe?5

1.2 Unterstützung eines Experten bei der Lösung einer Konfigurationsaufgabe durch ein wissensbasiertes Konfigurierungssystem6

1.3 Wissen, das für die Konfigurierung eines modular aufgebauten technischen Systems benötigt wird7

1.4 Prinzipielle Vorgehensweise bei der Lösung von Konfigurierungsaufgaben durch ein wissensbasiertes System8

1.5 Verschiedene Arten der Wissensmodellierung10

1.5.1 Struktur-orientierte Konfigurierung10

1.5.2 Ressourcen-orientierte Konfigurierung13

1.5.3 Kurzer Vergleich beider Ansätze14

1.6 Aufgabenstellung dieser Diplomarbeit14

1.7 Gliederung und Aufbau der Arbeit14

2. Der bisherige TOOCON-Werkzeugkasten17

2.1 Die Wissensmodellierung im TOOCON-Werkzeugkasten17

2.1.1 Das terminologische System TAXON17

2.1.2 Erweiterung der Ausdrucksmächtigkeit von TAXON durch Konkrete Bereiche19

2.1.3 Komponenten-Unifikation durch den SKOLEMIZER20

2.1.4 Erzeugung terminaler Konzepte mit dem REALIZER20

2.1.5 Vorwärtsregeln mit dem Tool FORWARD21

2.1.6 Erweiterte Hornklauseln mit dem Tool TAXLOG21

2.1.7 Die Steuerungskomponente CONTROL21

2.1.8 Zusammenfassender Überblick über die einzelnen Tools22

2.2 Die Architektur und das Zusammenspiel der Tools im TOOCON-Werkzeugkasten23

2.3 Der Ablauf einer Konfigurierung mit dem TOOCON-System24

2.4 Verwirklichung des struktur-orientierten Konfigurierungs-Ansatzes durch den TOOCON-Werkzeugkasten26

3. Ressourcen-orientiertes Konfigurieren27

3.1 Der ressourcen-orientierte Konfigurierungsansatz27

3.1.1 Ressourcen-orientierte Wissensmodellierung27

3.1.2 Ressourcen-Bilanzierung28

3.1.3 Formulierung und Überprüfung von ressourcen-orientierten Integritätsbedingungen28

3.1.4 Ressourcen-orientierte Konfigurierung28

3.1.5 Graphische Veranschaulichung29

3.2 Bewertung des ressourcen-orientierten Ansatzes und Vergleich mit dem struktur-orientierten Ansatz30

3.2.1 Vorteile einer ressourcen-orientierten Modellierung30

3.2.2 Probleme bei einer ressourcen-orientierten Modellierung und deren Lösungsansatz31

3.3 Motivation für die Integration beider Ansätze in einem einzigen Konfigurierungssystem32

4. Verschiedene Ansätze zur Integration ressourcen-orientierter Techniken in TOOCON35

4.1 Kompilation35

4.2 Integration ressourcen-orientiertierter Techniken durch die Erweiterung des TOOCON-Systems um ein neues Tool35

4.2.1 Erster Ansatz: TAXRES als ein von TAXON relativ unabhängiges Modul36

4.2.2 Zweiter Ansatz: TAXRES wird als Konkreter Bereich an TAXON angeschlossen36

4.2.3 Interne Aufsplittung von TAXRES in zwei Teile und die entsprechenden Schnittstellen-Erweiterungen38

4.2.4 Die Konkreten Prädikate41

4.2.5 Beseitigung von Ressourcen-Defiziten42

5. Das Implementierungskonzept zur Integration der ressourcen-orientierten Techniken in TOOCON45

5.1 Der neue Konkrete Bereich TAXRES45

5.1.1 Die Syntax der Konkreten Prädikate von TAXRES46

5.1.2 Bemerkungen zur Semantik der Ressource-Variablen48

5.1.3 Watchdogs49

5.2 Erweiterungen für die ressourcen-orientierte Konfigurierung49

5.2.1 Deklarierung der verwendeten Ressourcen49

5.2.2 Das spezielle Konzept ENVIRONMENT50

5.2.3 Die Bilanzierung der Ressourcen51

5.3 Verwendung von Strategien und Heuristiken53

5.3.1 Optimierung der Abarbeitungsreihenfolge der Operatoren auf der Agenda54

5.3.2 Unterstützung der Steuerungseinheit CONTROL bei der Komponenten-Auswahl54

5.3.3 Allgemeine Schnittstellen-Erweiterungen zwischen CONTROL, TAXON und den Konkreten Bereichen55

5.3.4 Die Komponenten-Bewertung durch TAXRES58

6. Anwendungsbeispiel für die ressourcen-orientierte Konfigurierung mit TAXRES61

6.1 Umgangssprachliche Wissensbeschreibung61

6.2 Die Modellierung des Beispiels mit dem TOOCON-Werkzeugkasten unter Verwendung von TAXRES62

7. Erweiterung des Grundkonzeptes um lokale Ressourcen71

7.1 Wie kann man die Komponenten-Zugehörigkeits-Rollenbeziehungen in die A-Box eintragen?72

7.2 Die Definition lokaler Bilanzierungs-Gruppen73

7.3 Das Implementierungskonzept der lokalen Ressourcen74

7.3.1 Die Repräsentation lokaler Ressourcen-Bilanzierungs-Gruppen75

7.3.2 Was passiert durch den Befehl "Def-Local-Resource-Group"?75

7.3.3 Die Bilanzierung lokaler Ressourcen76

8. Vorschläge für Verbesserungen und Erweiterungen79

8.1 Verarbeitung von Ressourcen-Abhängigkeiten79

8.2 Harte und weiche Constraints79

8.3 Übergang von der Konfigurierung zur Konstruktion durch das Verbindungskonzept80

8.3.1 Vergleich zwischen lokaler Ressourcen-Bilanzierung und dem Verbindungskonzept82

8.3.2 Kann man mit der lokalen Ressourcen-Bilanzierung das Verbindungskonzept simulieren bzw. macht das Verbindungskonzept lokale Ressourcen überflüssig?83

9. Zusammenfassung87

Anhang A: Ergänzung zum Anwendungsbeispiel aus Kapitel 691

Anhang B: Literaturverzeichnis105

1. Einleitung und Motivation

Es gibt viele technische Systeme, die nach dem Baukastenprinzip aufgebaut sind. Für den Aufbau eines solchen Systems stehen verschiedene mehr oder weniger komplexe Bausteine (*Komponenten*) zur Verfügung, die aus einem *Komponentenkatalog* ausgewählt und anschließend zum gewünschten Gesamtsystem zusammengesetzt werden. Dieser Synthese-Prozeß wird als *Konfigurierung* und das dabei entstehende Ergebnis als *Konfiguration* bezeichnet.

Die im Katalog zur Verfügung stehenden Komponenten bieten in der Regel sehr vielfältige Einsatz- und Kombinationsmöglichkeiten. Aus wirtschaftlichen Gesichtspunkten bieten solche möglichst universellen Module entscheidende Vorteile:

- Aufgrund der Vielseitigkeit der meisten Baukastenteile kann man mit einer relativ kleinen Komponentenauswahl dennoch für viele Kunden individuelle Lösungen erstellen.
- Außerdem werden durch die hohe Einsatzhäufigkeit der Komponenten die Entwicklungs- und Fertigungskosten günstiger.

Der Entwurf eines modular aufgebauten technischen Systems beschränkt sich hauptsächlich auf das Lösen einer *Konfigurierungsaufgabe*:

1.1 Was ist eine Konfigurierungsaufgabe?

Eine Konfigurationsaufgabe ist eine Aufgabe, bei der folgendes gegeben und gesucht ist:

Gegeben:

- **Komponentenkatalog**:
  Das ist ein Katalog, in dem alle für eine Konfigurierung zur Verfügung stehenden Komponenten und deren Eigenschaften verzeichnet sind.
- **Integritätsbedingungen**:
  Nicht jede theoretisch denkbare Kombination von Komponenten aus dem Komponentenkatalog ergibt eine zulässige bzw. funktionierende Konfiguration. *Integritätsbedingungen* sind (allgemeine) Restriktionen, die dabei unbedingt eingehalten werden müssen.
- **Aufgabenspezifikation**:
  Das sind alle (aktuellen) Anforderungen, die an das zu konfigurierende technische System gestellt werden (aktuelle Problemstellung).

Gesucht:

Gesucht ist eine Konfiguration, d. h. eine Menge von Komponenten aus dem Komponentenkatalog, bei der alle Anforderungen aus der Aufgabenspezifikation und alle Integritätsbedingungen erfüllt sind.

Das Ergebnis bzw. die Lösung einer Konfigurierungsaufgabe ist demnach im einfachsten Fall eine Liste von Komponenten aus dem Komponentenkatalog, die man zum Bau des geplanten technischen Systems benötigt. Hierbei spielt es zunächst noch keine Rolle, wie die einzelnen

Komponenten verbunden werden müssen. Das Ergebnis ist also eine reine Stückliste, die man z.B. verwenden kann, um vorab überschlagsmäßig die Gesamtkosten des geplanten technischen Systems zu bestimmen bzw. abzuschätzen.

Allerdings wäre eine (z. B. die durch Angabe von semantischen Beziehungen zwischen den Komponenten) strukturierte Lösung aus Übersichtlichkeitsgründen wünschenswert. Erstens könnte man dann Fehler leichter erkennen und zweitens würde dadurch in manchen Fällen, insbesondere bei einer sehr großen Komponenten-Anzahl, der nachfolgende praktische Aufbau des technischen Systems erheblich vereinfacht.

Ein weiterer wichtiger Punkt ist, daß bei einer Konfiguration verschiedene *Optimalitätskriterien* erfüllt sein sollten:

- Eine Konfiguration sollte *minimal* sein, d. h. sie sollte keine überflüssigen Komponenten enthalten. Eine Komponente wird hierbei als *überflüssig* bezeichnet, wenn man sie aus einer Konfiguration herausnehmen könnte und trotzdem danach immer noch alle Integritätsbedingungen und alle Anforderungen aus der Aufgabenspezifikation erfüllt wären.
- Noch besser wäre es, wenn die Konfiguration sogar möglichst *optimal* ist. Dazu müßte man aber die Optimalität einer Konfiguration von Fall zu Fall anhand verschiedener Kriterien (und deren Prioritäten) festlegen. Ein mögliches *Optimalitätskriterium* ist z. B., daß eine Konfiguration mit möglichst wenig Kosten, die Anforderungen der Aufgabenspezifikation (unter Berücksichtigung der Integritätsbedingungen) erfüllt.

1.2 Unterstützung eines Experten bei der Lösung einer Konfigurationsaufgabe durch ein wissensbasiertes Konfigurierungssystem

In der Praxis ist die Konfigurierung eines technischen Systems für den Experten oft eine lästige Routinetätigkeit. Dennoch ist diese Tätigkeit zum Teil so komplex, daß einem Menschen dabei entscheidende Fehler unterlaufen können, indem *(Konfigurations-)Anforderungen*, d. h. Anforderungen an die Konfiguration, die sich aus der aktuellen Aufgabenspezifikation oder aus den Integritätsbedingungen ergeben, irrtümlicher Weise nicht beachtet bzw. nicht eingehalten werden.

Die Folge ist, daß dabei z. B. wichtige Teile vergessen werden und so eine unvollständige Konfiguration entsteht. Andererseits kann es z. B. vorkommen, daß einige ausgewählte Teile nicht kompatibel zueinander sind und daß sich dadurch einige Komponenten aus der Konfigurationsmenge beim Aufbau des gewünschten technische System nicht verwenden lassen bzw. daß das mit diesen Teilen aufgebaute technische System in der Praxis nicht funktioniert.

Hierbei ist zwischen einer *unvollständigen* und einer widersprüchlichen (d. h. *inkonsistenten*) Konfiguration zu unterschieden:

- Eine Konfiguration ist *unvollständig*, wenn es Konfigurationsanforderungen gibt, die zwar derzeit nicht erfüllt, aber unter Beibehaltung der bisherigen (partiellen) Konfiguration noch erfüllbar sind, z. B. durch Hinzufügen neuer Komponenten.
- Eine Konfiguration wird als *inkonsistent* bezeichnet, wenn es Konfigurationsanforderungen gibt, die nicht erfüllt und unter Beibehaltung der bisherigen Konfiguration auch nicht mehr erfüllbar sind.[1]

Derartige Fehler werden oft erst relativ spät oder sogar erst bei der Inbetriebnahme bemerkt. Um einerseits dem Experten das routinemäßige Konfigurieren zu erleichtern und andererseits unvollständige oder sogar inkonsistente Lösungen möglichst zu vermeiden, werden *wissensbasierte Konfigurierungssysteme* eingesetzt, die diese Aufgabe weitgehend automatisch lösen sollen.

Damit ein Konfigurierungssystem dies bewerkstelligen kann, muß es zum einen über das dazu notwendige Konfigurierungswissen verfügen und zum anderen in der Lage sein, dieses Wissen auch entsprechend anzuwenden. Dabei ist es wünschenswert, daß ein solches System die Schlußfolgerungen eines menschlichen Experten nachahmt, damit dieser den automatischen Prozeß der Lösungsfindung leichter nachvollziehen kann.

1.3 Wissen, das für die Konfigurierung eines modular aufgebauten technischen Systems benötigt wird

Um eine Konfigurierungsaufgabe erfolgreich lösen zu können, muß ein Experte über unterschiedliche Arten von Wissen verfügen. Dieses Wissen sollte weitgehend auch in einem Konfigurierungssystem repräsentiert sein, und zwar möglichst adäquat und deklarativ (d. h. unabhängig von der Verarbeitung). Dieses *Konfigurierungswissen* kann in Anlehnung an [RSW94] in folgende Kategorien aufgeteilt werden:

Komponentenwissen:

Das ist das Wissen darüber, welche *Komponenten(typen)* für eine Konfiguration verwendet werden können (Komponentenkatalog), welche (funktionalen) Eigenschaften sie haben und/ oder in welcher semantischen Beziehung sie zu anderen Komponenten(typen) stehen.
Hierbei sind mit Komponenten(typen) nicht nur die in der Realität vorkommenden konkreten Bausteine gemeint, sondern auch abstraktere Begriffe (*Komponenten-Oberbegriffe*) zur Beschreibung von Komponenten. Diese abstrakten Begriffe werden oft in einer *Begriffshierarchie* eingeordnet (vgl. Abschnitt 1.5.1).

Integritätsbedingungen:

Das sind Bedingungen, die in einer Konfiguration unbedingt eingehalten werden müssen (vgl. Abschnitt 1.1). Falls eine oder mehrere solcher Bedingungen nicht eingehalten werden, dann ist die generierte Konfiguration *fehlerhaft*, d. h. unvollständig oder sogar inkonsistent (vgl. Abschnitt 1.2):
Bei einer unvollständigen Konfiguration kann der Fehler aufbauend auf der bisherigen Konfiguration nachträglich noch beseitigt (*repariert*) werden. Im Gegensatz dazu kann eine inkonsistente Konfiguration aufgrund von Widersprüchen nicht mehr repariert und daher in der Praxis nicht verwendet werden. In einem solchen Fall besteht nur noch die Möglichkeit, nach einer anderen Konfiguration zu suchen.

---

1. Beispiel:
   Wenn in einer Konfiguration in Form einer (zusätzlichen) semantischen Beziehung angegeben wurde, daß zwei Komponenten K1 und K2 miteinander verbunden werden sollen, obwohl sie aufgrund bestimmter Integritätsbedingungen inkompatibel zueinander sind, dann ist die Konfiguration inkonsistent.

Problemlösungswissen:

Das Problemlösungswissen beschreibt allgemein, wie mit Hilfe des Komponentenwissens und unter Berücksichtigung der Integritätsbedingungen eine Konfigurierungsaufgabe gelöst werden kann. Dieses Wissen besteht normalerweise zu einem großen Teil aus Strategien und Heuristiken.

Für die Lösung einer Konfigurierungsaufgabe müssen verschiedene Arbeitsschritte ausgeführt werden. Dabei gibt es *domänen-unabhängige Arbeitschritte*, die bei jeder Konfigurierung benötigt werden (z. B. Auswahl einer Komponente aus dem Komponentenkatalog). Daneben gibt es andere, die nur in bestimmten Domänen vorkommen. Das Wissen, das solche *domänenabhängigen Arbeitsschritte* bei der Lösung einer Konfigurierungsaufgabe beschreibt, wird als *Aufgabenwissen* bezeichnet. Mit diesem Wissen können z. B. bestimmte Formulierungen in der Aufgabenspezifikation in eine Menge von Teilaufgaben oder in konkrete Komponenten-Anforderungen transformiert werden.
Außerdem enthält das Problemlösungswissen auch noch *Steuerungswissen*, das die Reihenfolge bestimmt, in der die einzelnen Arbeitsschritte abgearbeitet werden.

Aufgabenspezifikation:

(siehe Abschnitt 1.1)

Assertionales Wissen:

In Anlehnung an [RSW94] wird auch in dieser Arbeit das dynamische Wissen über eine aktuelle Problemstellung, das während eines Konfigurierungsprozesses aus dem Komponentenwissen, den Integritätsbedingungen, dem Problemlösungswissen und der Aufgabenspezifikation durch unterschiedliche Inferenzmechanismen generiert wird, als *assertionales Wissen* bezeichnet.

1.4 Prinzipielle Vorgehensweise bei der Lösung von Konfigurierungsaufgaben durch ein wissensbasiertes System

Das Finden einer Konfigurationslösung ist im wesentlichen ein Suchproblem. Jede Menge von Komponenten aus dem Komponentenkatalog ist dabei prinzipiell eine potentielle Lösung. Da eine solche Menge beliebig groß sein kann, hat man hier theoretisch einen unendlich großen Suchraum. Aber selbst wenn man die Anzahl der Komponenten in der Praxis irgendwie beschränkt, hat man aufgrund der *kombinatorischen Explosion* immer noch einen sehr großen Suchraum.

Im Prinzip könnte man einfach solange irgendwelche Komponenten-Mengen generieren, bis man eine Konfiguration gefunden hat, die sowohl alle Integritätsbedingungen als auch die vom Benutzer formulierte Aufgabenspezifikation erfüllt. Dies wäre dann eine der möglichen Konfigurationslösungen. Eine solche *"Generate-and-Test"-Methode* ist zwar für viele Suchprobleme anwendbar, aber im allgemeinen für die Problematik der Konfigurierung wegen des zu großen Suchraumes völlig ungeeignet. Statt dessen sollten die Anforderungen an die gewünschte Lösung (d. h. Aufgabenspezifikation und Integritätsbedingungen) schon gleich im Suchprozeß miteinbezogen werden. Bei dem daraus resultierenden Suchverfahren werden die einbezogenen Anforderungen an die gesuchte Lösung als *Constraints* bezeichnet. Durch die

Berücksichtigung der Anforderungen läßt sich die Zahl der Alternativen im Suchraum zum Teil sehr stark einschränken. Aufgrund solcher Einschränkungen (*Restriktionen*) kann es sein, daß weitere alternative Suchwege beschnitten werden können und dadurch wiederum weitere, u.s.w. Diese Fortpflanzung von Einschränkungen nennt man *Propagierung*. Eine solche Vorgehensweise führt zu einem inkrementellen constraint-basierten Konfigurierungsverfahren:

Die Ausgangsbasis bei diesem Verfahren ist eine leere Konfigurationsmenge. Diese erste Teillösung ist zwar konsistent, aber unvollständig[2]. Nun werden Komponenten (Instanzen aus dem Komponentenkatalog) in die Konfigurationsmenge eingetragen, die zum Erfüllen der Aufgabenspezifikation unbedingt notwendig sind. Jeweils nach jeder neuen Komponente werden alle davon abhängigen Integritätsbedingungen (Constraints) durchpropagiert und damit die bisherige partielle Konfiguration durch entsprechende Einträge (z. B. durch neue Komponenten oder durch den Eintrag semantischer Beziehungen zwischen Komponenten) vervollständigt, wodurch der noch vorhandene Suchraum weiter eingeschränkt wird.

Meistens ist es so, daß nach der Propagierung aller bekannten Konfigurationsanforderungen die bisher generierte Konfiguration immer noch nicht vollständig ist. An dieser Stelle ist nun eine *Entscheidung* über die weitere Vorgehensweise zu treffen, d. h. in welcher Reihenfolge die bisher noch offenen Anforderungen wie beseitigt werden sollen. Dabei muß sich das Konfigurierungssystem (basierend auf einer hypothetischen Annahme) zwischen mehreren Alternativen entscheiden (*Wahlpunkt*).

Im Gegensatz zur Propagierung von Anforderungen kann es dabei auch zu *Fehlentscheidungen* kommen, was zu einer späteren Inkonsistenz führt! Außerdem hat jede Entscheidung im Konfigurierungsprozeß, wie z. B. die Auswahl einer bestimmten Komponente aus dem Komponentenkatalog, meistens einen sehr großen Einfluß auf die Effizienz des Verfahrens und die Qualität einer ersten gefundenen Lösung. Daher sollte man hier zur Navigation durch den Suchraum (soweit vorhanden) das entsprechende (heuristische) Problemlösungswissen eines Experten einsetzen (vgl. Abschnitt 1.3).

Sobald eine Entscheidung im Konfigurierungsprozeß getroffen wurde, werden alle Konsequenzen daraus (d. h. alle abhängigen Integritätsbedingungen) durchpropagiert. Dadurch können drei Fälle auftreten:

- Wenn die generierte Konfiguration konsistent und vollständig ist, dann ist der Konfigurierungsprozeß erfolgreich beendet.
- Wenn die bisherige (partielle) Konfiguration zwar konsistent, aber unvollständig ist, dann muß wieder eine Entscheidung über die weitere Vorgehensweise getroffen werden (*Wahlpunkt*).
- Wenn die generierte Konfiguration inkonsistent ist, dann bedeutet das, daß bei der Navigation durch den Suchraum aufgrund einer *Fehlentscheidung* ein falscher Weg eingeschlagen wurde, der nun nicht mehr weiterführt. Solche Inkonsistenzen (d. h. Irrwege im Suchraum) sollten möglichst frühzeitig erkannt werden. Denn in einem solchen Fall müssen durch *Backtracking* solange die letzten Schritte im Konfigurierungsprozeß rückgängig gemacht werden, bis man wieder an einen Punkt (*Wahlpunkt*) gelangt, an dem es zuvor die Auswahl zwischen mehreren Alternativen gab, aber das letzte Mal die falsche Suchrichtung eingeschlagen wurde (indem z. B. bei einer Komponenten-Auswahl eine falsche Komponente gewählt wurde). An dieser Position angelangt, wird nun eine neue Alterna-

2. d. h. die bisherige Teillösung genügt noch nicht allen Anforderungen

tive ausprobiert. Falls dies nicht möglich ist (d. h. falls es keine weitere Alternative mehr gibt), dann wird zu einem früheren Wahlpunkt zurückgegangen und erneut alle verbleibenden Alternativen ausprobiert. Wenn aber kein vorheriger Wahlpunkt vorhanden ist, dann wird an dieser Stelle der Konfigurierungsprozeß erfolglos abgebrochen.

Um die Anzahl der (teuren) Backtracking-Operationen zu reduzieren, sollten bei der Auswahl einer Komponente aus dem Katalog möglichst viele (am besten alle) Anforderungen berücksichtigt werden. Gemäß einer *Least-Commitment-Strategie* sollten solche Entscheidungen daher einerseits weit hinaus geschoben werden und andererseits sich auf möglichst wenig festlegen.

1.5 Verschiedene Arten der Wissensmodellierung

Es gibt einige Konfigurierungssysteme, die für die Wissensmodellierung ihres Komponentenkataloges ausschließlich einen *struktur-orientierten*, und andere, die ausschließlich einen *ressourcen-orientierten* Ansatz verwenden. Entsprechend der unterschiedlichen Art der Wissensmodellierung werden auch unterschiedliche Inferenzmechanismen verwendet. In den nächsten Abschnitten werden beide Ansätze jeweils anhand des Beispiels *Konfigurierung eines PCs* vorgestellt und miteinander verglichen.

1.5.1 Struktur-orientierte Konfigurierung

Bei der Wissensmodellierung und den Inferenzmechanismen orientiert man sich an der Struktur, d. h. am Aufbau des zu konfigurierenden technischen Systems und der dafür zur Verfügung stehenden Komponenten. Hierbei wird die Struktur einer Komponente durch deren semantische Beziehung zu anderen Komponenten beschrieben (siehe Beispiel 1.1).

Beispiel 1.1: Struktur-orientiertes Konfigurieren eines PCs:

Bild 1.1 zeigt die vereinfachte Struktur eines PCs:

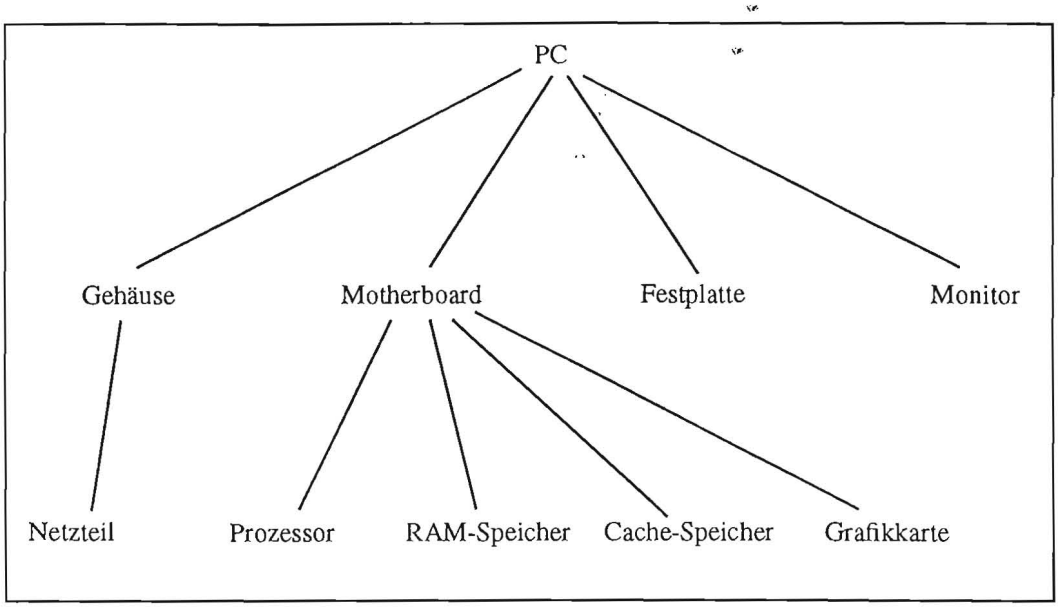


Bild 1.1: Vereinfachte Struktur eines PCs

Hierbei läßt sich z. B. die Beziehung zwischen PC und Motherboard als „Has-Part"-Beziehung charakterisieren, da ein Motherboard ein Teil eines PCs ist. Es gibt aber auch noch andere Struktur-Beziehungen zwischen Komponenten, z. B. die Beziehung zwischen Motherboard und Prozessor. Damit das Motherboard funktionieren kann, muß darin zwar ein Prozessor eingebaut werden, aber streng genommen ist der Prozessor kein Bestandteil vom Motherboard, sondern ein eigenes Bauteil. Aber unabhängig davon, wie man hier diese Komponenten-Beziehungen bezeichnet, hat die Struktur-Beschreibung dieses Beispiels folgende Semantik:

Bei der Konfigurierung eines PCs braucht man ein Gehäuse, ein Motherboard, eine Festplatte und ein Monitor. An der Struktur erkennt man weiter, daß zum Gehäuse ein Netzteil und zum Motherboard ein Prozessor, Cache- und RAM-Speicher und eine Grafikkarte gehören. Diese Teile werden daher ebenfalls benötigt.

Diese Komponentenbezeichnungen sind zunächst nur Oberbegriffe. Im Laufe des Konfigurierungsprozesses müssen alle diese *Komponenten-Oberbegriffe* entsprechend der Aufgabenspezifikation und der Integritätsbedingungen durch Festlegung noch offener Komponenten-Eigenschaften (*Parametrierung*) zu tatsächlichen Komponenten (*Komponenten-Ausprägungen*) verfeinert, d. h. *spezialisiert* werden.

Im Bild 1.2 wird anhand einer „IS-A"-Taxonomie gezeigt, wie man z. B. den Oberbegriff "Festplatte" weiter spezialisieren könnte.

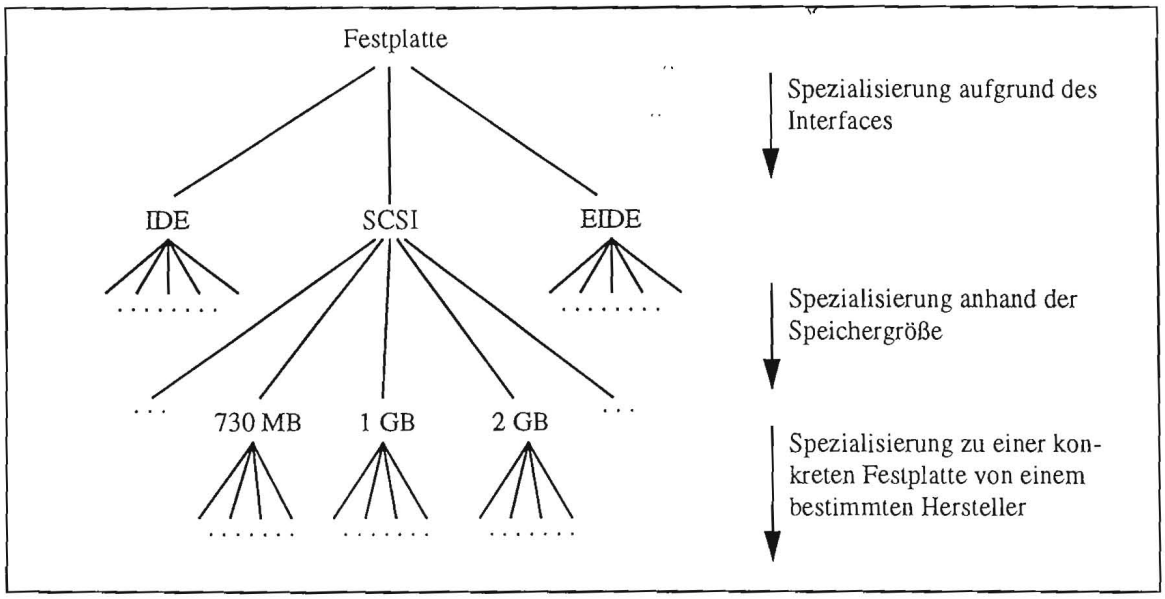


Bild 1.2: Spezialisierungsmöglichkeiten des Oberbegriffs "Festplatte"

Sobald man nur noch Komponenten-Ausprägungen in der Lösungsmenge hat und dennoch alle Integritätsbedingungen erfüllt sind, ist die Aufgabe erfolgreich gelöst und der Konfigurierungsprozeß beendet.

An dem vorherigen Beispiel erkennt man die zentralen Inferenzmechanismen beim strukturorientierten Konfigurieren:

- *Zerlegung* einer Komponente aufgrund ihrer Struktur
  (z. B. besteht ein PC aus Gehäuse, Motherboard, ...)
- Durch *Parametrieren* werden die noch offenen Freiheitsgrade, d. h. die bisher noch nicht spezifizierten Komponenten-Eigenschaften, festgelegt.
  (z. B. kann durch Parametrieren festgelegt werden, daß die Festplatte F ein SCSI-Interface hat)
- *Spezialisierung* einer Komponente anhand der „IS-A"-Taxonomie
  (z. B. kann der Oberbegriff „Festplatte" zu „SCSI-Festplatte" spezialisiert werden)

Der bisherige TOOCON-Werkzeugkasten (siehe Kapitel 2 oder [RSW94]) ist ein Beispiel für ein Konfigurierungssystem, das im wesentlichen auf dem struktur-orientierten Ansatz basiert. Andere Konfigurierungssysteme, die diesem Ansatz verwenden, sind z. B. die Systeme PLAKON [CGS91], KONWERK [Günter95] oder IDAX [DP94, Paulokat95].

1.5.2 Ressourcen-orientierte Konfigurierung

Ausgangspunkt für diesen Ansatz (vgl. [Heinrich93]) ist die Beobachtung, daß es modulare Systeme gibt, bei denen die Aufbau-Struktur nicht so wichtig ist. Bei ihnen kommt es statt dessen hauptsächlich auf die Schnittstellen-Eigenschaften[3] (*Funktionalität*) der Einzelkomponenten an, die zusammen die Funktionalität des Gesamtsystems bestimmen.

Beispiel 1.2: Stark vereinfachte funktionale Beschreibung einer Festplatte:

Eine Festplatte ist eine Komponente, die eine gewisse Menge elektrischer Leistung benötigt und die dafür eine bestimmte Speicherkapazität zur Verfügung stellt.

Bei diesem Modellierungsansatz werden alle Schnittstellen-Eigenschaften (sowohl die der Einzelkomponenten als auch die des Gesamtsystems) mit Hilfe von bereitgestellten bzw. geforderten *Ressourcen* beschrieben. So könnte z. B. die von einer Festplatte geforderte Ressource "elektrische Leistung" durch ein Netzteil bereitgestellt werden.

Während des Konfigurierungsprozesses wird versucht, alle Ressourcen *auszubilanzieren*, d. h. dafür zu sorgen, daß von jeder Ressource mindestens soviel zur Verfügung gestellt wird, wie gefordert ist. Einzelne Ressourcen können z. B. dadurch ausbilanziert werden, daß aufgrund funktionaler Anforderungen neue Komponenten aus dem Katalog ausgewählt und der bisherigen partiellen Konfiguration hinzugefügt werden. Am Ende einer erfolgreichen Konfigurierung sind alle Ressourcen gleichzeitig ausbilanziert und die erwünschte Funktion des Gesamtsystems ist damit erreicht.

Beispiel 1.3: Ressourcen-orientiertes Konfigurieren eines PCs:

Angenommen, man möchte einen PC mit Pentium-Prozessor und (mindestens) 700 MB Festplattenspeicher konfigurieren.

Der zu dieser Aufgabenspezifikation gehörende Konfigurierungsprozeß könnte z. B. folgendermaßen ablaufen:

- Um die Aufgabenspezifikation zu erfüllen, wird ein Pentium-Prozessor und eine mindestens 700 MB große Festplatte aus dem Komponentenkatalog ausgewählt und in die zuvor leere Lösungsmenge eingetragen.
- Da der Prozessor irgendwo angeschlossen werden muß, wird ein dafür geeignetes Motherboard ausgewählt und der bisherigen Lösungsmenge hinzugefügt.
- Sowohl die Festplatte als auch das Motherboard nehmen eine bestimmte Menge elektrische Leistung auf. Um diesen Bedarf zu decken, wird ein Netzteil eingetragen, das mindestens soviel Leistung zur Verfügung stellt, wie die Festplatte und das Motherboard benötigen.

usw.

Bei diesem Prozeß müssen immer wieder unerfüllte funktionale Anforderungen (d. h. *Ressourcen-Defizite*) einzelner Komponenten ausgeglichen werden. Sobald es keine solchen Defizite mehr gibt, ist die Konfigurierungsaufgabe gelöst und dieses Verfahren terminiert.

---

3. Hiermit sind die Schnittstellen eines Bausteins zu anderen Komponenten gemeint.

Eine ausführlichere Betrachtung des ressourcen-orientierten Konfigurierungsansatzes findet man in Kapitel 3.1.

1.5.3 Kurzer Vergleich beider Ansätze

Bei der ressourcen-orientierten Wissensmodellierung braucht man nur anzugeben, wieviel jede einzelne Komponente von welcher Ressource verbraucht bzw. zur Verfügung stellt. Viele Konfigurierungsaufgaben können mit diesem Ansatz relativ einfach und sehr elegant beschrieben und gelöst werden. Allerdings lassen sich mit dieser Modellierungsart nicht alle Aspekte bei Konfigurierungsproblemen adäquat beschreiben, so daß vorhandenes struktur-orientiertes Expertenwissen oft nur sehr schlecht modelliert werden kann (siehe Kapitel 3.2).

Man kann zwar mit beiden Ansätzen Konfigurierungsprobleme lösen (siehe Beispiele 1.1 und 1.3), sie sind aber nicht überall gleich gut geeignet. Oft hat man in der Praxis das Problem, daß in ein und derselben Anwendungsdomäne sowohl Teile vorkommen, die man sinnvoller mit dem struktur-orientierten Ansatz und andere, die man besser mit dem ressourcen-orientierten Ansatz modelliert:

Beispiel 1.4:
Wie soll man mit einer struktur-orientierten Modellierung sicherstellen, daß das Netzteil im PC genügend Leistung zur Verfügung stellt?

Beispiel 1.5:
Wie kann man beim ressourcen-orientierten Ansatz die Aufbaustruktur eines PCs adäquat mit Ressourcen beschreiben?

Deshalb wäre ein Konfigurierungssystem sinnvoll und erwünscht, das beide Modellierungsarten unterstützt. Dann könnte der Wissensingenieur jeweils von Fall zu Fall die seiner Meinung nach adäquateste Methode auswählen (siehe Kapitel 3.3).

1.6 Aufgabenstellung dieser Diplomarbeit

In dieser Diplomarbeit sollten der struktur-orientierte und der ressourcen-orientierte Konfigurierungsansatz untersucht und miteinander verglichen werden. Desweiteren sollte geklärt werden, ob man beide Konfigurierungsansätze miteinander integrieren kann, indem man das bereits vorhandene struktur-orientierte Konfigurierungssystem TOOCON um ressourcen-orientierte Techniken erweitert. Falls dies möglich ist, sollte ein entsprechendes Konzept erstellt und anschließend in das TOOCON-System implementiert werden.

1.7 Gliederung und Aufbau der Arbeit

Das **2. Kapitel** beschreibt den bisherigen struktur-orientierten TOOCON-Werkzeugkasten. Das Kapitel dient zur Abrundung und Vervollständigung dieser Diplomarbeit und ist im wesentlichen eine Zusammenfassung von [RSW94].
Das TOOCON-System ist aus verschiedenen Modulen aufgebaut. TAXON ist dabei das Kernmodul und dient zur Repräsentation des Komponentenkataloges. Es wird deshalb zuerst und auch relativ ausführlich dargestellt. Dabei wird auch intensiv auf die Anbindung Konkreter

Bereiche an TAXON eingegangen. Anschließend folgt eine kurze Beschreibung der Inferenz-Tools SKOLEMIZER, REALIZER, FORWARD und TAXLOG und der Steuerungskomponente CONTROL. Danach folgt eine tabellarische Übersicht über die einzelnen Tools und eine Architektur- und Informationsfluß-Beschreibung des TOOCON-Systems. Zum Abschluß dieses Kapitels wird der prinzipielle Kontrollfluß bei einer Konfiguration erläutert.

In **Kapitel 3** wird zunächst das Grundkonzept der ressourcen-orientierten Konfigurierung erklärt und dann mit dem struktur-orientierten Ansatz verglichen.

Im **4. Kapitel** werden verschiedenen Möglichkeiten diskutiert, wie ressourcen-orientierte Techniken in TOOCON integriert werden können.

Kapitel 5 beschreibt die tatsächliche Integration in Form des neuen Konkreten Bereiches TAXRES. Dieses neue Tool ist sowohl für die Verwaltung aller Ressourcen als auch für die Überwachung der Summen-Bilanzen zuständig. Im zweiten Teil dieses Kapitels wird die Reaktion bei Ressourcen-Defiziten behandelt. Dabei wird die Schnittstelle und die Zusammenarbeit zwischen TAXRES und der Steuerungskomponente CONTROL beschrieben. Diesbezüglich wird anschließend noch auf die Verwendung von Strategien und Heuristiken eingegangen.

Das **6. Kapitel** enthält ein etwas ausführlicheres Anwendungsbeispiel für die ressourcen-orientierte Konfigurierung mit TAXRES. Die Wissensmodellierung und der eigentliche Ablauf des Konfigurierungsprozesses werden allerdings in diesem Kapitel nur kurz beschrieben. Die vollständige Wissensmodellierung und einige interessante Stellen während der Konfigurierung sind in Anhang A abgedruckt.

Im **7. Kapitel** wird die Erweiterung des Grundkonzeptes um lokale Ressourcen behandelt. Dabei wird zunächst anhand einiger kleiner Anwendungsbeispiele erklärt, was lokale Ressourcen sind und wofür man sie benötigt. Danach folgt die Beschreibung der dafür notwendigen Erweiterungen von TAXRES.

In **Kapitel 8** werden einige Vorschläge gemacht, wie man den TOOCON-Werkzeugkasten und da insbesondere TAXRES weiter verbessern und erweitern könnte.

Kapitel 9 enthält die Zusammenfassung dieser Diplomarbeit. Insbesondere befindet sich hier ein tabellarischer Vergleich zwischen einem rein struktur- bzw. rein ressourcen-orientierten Konfigurierungssystem und dem durch diese Arbeit erweiterten TOOCON-Werkzeugkasten.

In **Anhang A** steht die vollständige Modellierung des Anwendungsbeispiels aus dem 6. Kapitel. Außerdem sind hier einige markante Zwischenschritte bei der Abarbeitung des Konfigurierungsbeispiels abgedruckt.

Anhang B bildet mit dem Literaturverzeichnis den Abschluß dieser Diplomarbeit.

Diese gesamte Diplomarbeit ist in sich weitgehend abgeschlossen. Da sich diese Arbeit hauptsächlich auf das Konfigurierungssystem *TOOCON* bezieht, werden dessen Aufbau, prinzipielle Funktionsweise und wichtigsten Eigenschaften in zusammengefaßter Form in Kapitel 2 beschrieben. Dabei wird allerdings auf die *Terminologische Wissensrepräsentation* (siehe z. B. [BS85], [Dilger88], [BBHLN92], [Abecker94]) nur recht oberflächlich eingegangen. Ausführlichere Informationen zum TOOCON-Werkzeugkasten, wie sie z. B. in [RSW94] und in

[Wache94] vermittelt werden, sind zwar keine Voraussetzung, können unter Umständen aber an manchen Stellen das Verständnis erleichtern, wie z. B. bei der Befehlssyntax in den Beispielen.

Die Konfigurierungsverfahren, die in dieser Diplomarbeit behandelt werden, sind in sehr vielen technischen Domänen anwendbar. Allerdings stammen die Beispiele in dieser Arbeit hauptsächlich aus einer PC-Domäne, wobei die verwendeten Fachbegriffe aus diesem Bereich als bekannt vorausgesetzt werden. Nähere Informationen dazu findet man z. B. in [SV94].

2. Der bisherige TOOCON-Werkzeugkasten

Der TOOCON[4]-Werkzeugkasten ist ein wissensbasiertes Konfigurierungssystem. Er besteht aus mehreren Tools, die unterschiedliche Wissensrepräsentations-Methoden und Inferenz-Mechanismen[5] zur Verfügung stellen. Sowohl bei der Konzeption der einzelnen Tools als auch bei deren Vereinigung zu einem integrierten Gesamtsystem wurde besonderen Wert auf eine formale Semantik und eine deklarative (d. h. von der Verarbeitung unabhängige) Wissensrepräsentation gelegt. Der TOOCON-Werkzeugkasten bietet dem Wissensingenieur damit ausdrucksstarke Formalismen zum Beschreiben und Lösen von Konfigurierungsproblemen.

In diesem Kapitel wird zunächst die Wissensmodellierung und anschließend der prinzipielle Ablauf einer Konfiguration mit dem TOOCON-Werkzeugkasten behandelt. Allerdings wird hier im wesentlichen nur das beschrieben, was zum Verständnis dieser Diplomarbeit dient. Ausführliche Informationen zu TOOCON findet man in [RSW94] und in [Wache94].

2.1 Die Wissensmodellierung im TOOCON-Werkzeugkasten

Dieses Unterkapitel beschreibt die einzelnen Tools des TOOCON-Systems und deren jeweilige Aufgabe.

2.1.1 Das terminologische System TAXON

Der Kern von TOOCON ist das terminologische System TAXON. Es besteht aus einem terminologischen Teil (*T-Box*) und einem assertionalen Teil (*A-Box*).

Die T-Box:

In der T-Box wird das Komponentenwissen in Form einer Terminologie modelliert. Dadurch erhält man einen strukturieren Komponentenkatalog, in dem sowohl alle zur Verfügung stehenden Komponenten(typen) als auch deren semantischen Abhängigkeiten (*taxonomische Integritätsbedingungen*) untereinander repräsentiert sind.

Alle Komponenten(typen) in der T-Box werden jeweils durch ein *Konzept* beschrieben. Hierbei unterscheidet man zwischen *Oberkonzepten* (Komponenten-Oberbegriffen) und *terminalen Konzepten* (Komponenten-Ausprägungen). Semantische Beziehungen und Abhängigkeiten zwischen den einzelnen Konzepten können mit Hilfe von *Rollen* oder *Attributen* formuliert werden.

Die Definition der einzelnen Komponenten erfolgt in einer intensionalen Beschreibungssprache. Hierbei kann man jede einzelne Komponente entweder in Form eines primitiven oder komplexen Konzeptes definieren. Bei den primitiven Konzepten wird außer dem Namen keine weitere Information angegeben. Der *strukturelle Aufbau* komplexer Konzepte wird durch Konstruktoren mit Hilfe anderer Konzepte der T-Box beschrieben. Dabei stehen folgende Konstruktoren zur Verfügung: AND-, OR-, oder NOT-Konstruktor, Relation mit Exist-Quantor, Relation mit Allquantifizierung, Agreement und Disagreement (siehe [RSW94]).

---

4. Der Name TOOCON steht für *"Tools for Configuration"*.
5. wie z. B. *taxonomisches Reasoning*, *Constraint-Propagierung* und *Regelverarbeitung*

Für die T-Box gibt es einen *Classifier*, der aus den intensionalen Komponentenbeschreibungen automatisch die sich daraus ergebende Taxonomie (Subsumptionsgraph) berechnet.

Die A-Box:

Für die Repräsentation des assertionalen Wissen ist die A-Box zuständig. Hier stehen alle für die konkrete Konfiguration ausgewählten *Individuen* (Menge von Konzept-Instanzen[6] aus der T-Box) und deren semantischen Beziehungen untereinander.

Die wichtigsten Inferenz-Tools, die in der A-Box von TAXON verwendet werden, sind der *Konsistenztester* und der *Realisierer*:

- Der *Konsistenztester* hat die Aufgabe, den Inhalt der A-Box (d. h. die partielle bzw. vollständige Konfiguration) auf logische Korrektheit im Hinblick auf die in der T-Box (Komponentenkatalog) definierte Terminologie, d. h. bezüglich der vorhandenen Konzepte und der taxonomischen Integritätsbedingungen, zu überprüfen.
  Der in TAXON integrierte Konsistenztester ist aber nicht nur dafür zuständig die A-Box zu testen. Er hat zusätzlich die Aufgabe, die A-Box bei Bedarf bezüglich des terminologischen Wissens in der T-Box automatisch zu vervollständigen, indem er Wissen aus der T-Box in die A-Box überträgt.

  Beispiel 2.1:

  Annahme in der T-Box sei durch **(CONC** *PC-Gehäuse* **(SOME** (*has-part*) *Netzteil*)[7] spezifiziert, daß zu jedem PC-Gehäuse ein Netzteil gehört bzw. daß jedem "PC-Gehäuse" über das Attribut "has-part" ein "Netzteil" zugeordnet sein muß.
  Der Einfachheit halber sei angenommen, daß die A-Box zunächst leer ist. Wenn jetzt eine Instanz des Konzeptes "PC-Gehäuse" in die A-Box eingetragen und anschließend der Konsistenztester aufgerufen wird, passiert folgendes: Um die A-Box im Hinblick auf das Wissen in der T-Box zu vervollständigen, wird automatisch ein Netzteil in die A-Box instanziiert und dann der "PC-Gehäuse"-Instanz die neue "Netzteil"-Instanz über das Attribut "has-part" zugeordnet.

- Der *Realisierer* ermittelt für jedes Individuum in der A-Box die Menge der Konzepte aus der T-Box, zu deren Instanz es *spezialisiert*[8] werden kann bzw. muß (*schwache* bzw. *starke Realisierung*).

Überwachung der A-Box durch den Observer:

Im TOOCON-Werkzeugkasten gibt es mehrere Tools, die auf bestimmte Inhalte der A-Box reagieren sollen. Aus Effizienzgründen erfolgt die Überwachung der A-Box zentral im Observer.

---

6. Man beachte:
   Die Komponenten(typen) in der T-Box heißen *Konzepte* und die Komponenten in der A-Box werden als *(Konzept-)Instanz* oder als *Individuum* bezeichnet. Wenn es sich aus dem Zusammenhang klar ergibt, was gemeint ist, wird in dieser Arbeit der Einfachheit halber nicht immer explizit zwischen einem Konzept und einer Konzept-Instanz unterschieden.
7. Syntax und Semantik eines solchen Befehls ist in [Wache94] ausführlich beschrieben.
8. siehe Kapitel 1.5.1

Bei bestimmten Zuständen der A-Box werden vom Observer Aktionen ausgeführt. Bei welchen Zuständen welche Aktionen ausgeführt werden, wird in Form von *Observer-Regeln* festgelegt. TOOCON-Tools, die den Observer als Hilfsmodul benutzen, tragen bei ihm Regeln ein, die beschreiben, was bei welchen A-Box-Eintragungen passieren soll.

2.1.2 Erweiterung der Ausdrucksmächtigkeit von TAXON durch Konkrete Bereiche

Die Ausdrucksmächtigkeit des terminologischen Systems TAXON wurde bewußt eingeschränkt, um entscheidbare Inferenz-Mechanismen zu gewährleisten. Diese eingeschränkte Ausdrucksmächtigkeit ist aber für viele (Konfigurierungs-)Anwendungen zu schwach. In [BH91] wurde gezeigt, wie man die Ausdrucksmächtigkeit eines solchen terminologischen Systems mit Hilfe externer Formalismen (*Konkreter Bereiche*) so erweitern kann, daß die Entscheidbarkeit erhalten bleibt.

Definition eines Konkreten Bereiches:

Laut der formalen Definition besteht ein *Konkreter Bereich*[9] D aus einer Menge dom(D) von *Konkreten Objekten* (der Domäne von D) und aus einer Menge zu D gehörender *Konkreter Prädikate* pred(D). Dabei hat jedes Prädikat $P \in pred(D)$ jeweils eine feste Stelligkeit n und es gilt $P \subseteq dom(D)^n$.

Damit ein Konkreter Bereich *zulässig* ist, muß er folgende drei Voraussetzungen erfüllen:

- Die Zugehörigkeit eines Symbols zur Domäne dom(D) muß über ein Prädikat entscheidbar sein.
- Zu jedem n-stelligen Prädikat $P \in pred(D)$ muß es ein komplementäres Prädikat $\overline{P} \in pred(D)$ mit $\overline{P} := dom(D)^n \setminus P$ geben, d. h. daß die Menge der Konkreten Prädikate pred(D) unter der Negation abgeschlossen sein muß.
- Es muß ein (effizienter) Entscheidungsalgorithmus für das Erfüllbarkeitsproblem beliebiger endlicher Konjunktionen von Prädikaten aus pred(D) existieren.

Die Anbindung Konkreter Bereiche an TAXON:

TAXON arbeitet über einer Menge von abstrakten Objekten und ein Konkreter Bereich arbeitet über einer Menge von *Konkreten Objekten*. Die Schnittstelle zwischen TAXON und einem Konkreten Bereich erfolgt im wesentlichen durch *Konkrete Prädikate*. Diese Konkreten Prädikate können für die Konzept-Beschreibungen in TAXON verwendet werden.

Wenn TAXON in einem Inferenz-Prozeß auf ein Konkretes Prädikat trifft, dann übergibt es dieses an den zugehörigen Konkreten Bereich. Daraufhin wertet der Konkrete Bereich dieses Prädikat aus und überprüft, ob die Konjunktion aller bisher übermittelten Konkreten Prädikate erfüllbar ist. Das Ergebnis dieser Überprüfung wird dann wieder TAXON mitgeteilt.

9. engl.: concrete domain

Anwendungsbeispiele:

Über den Mechanismus der Konkreten Bereiche lassen sich unterschiedliche Verfahren zur Formulierung von zusätzlichen Integritätsbedingungen[10] derart in TAXON integrieren, daß sie beim Konsistenztest in der A-Box mitberücksichtigt werden. Im TOOCON-Werkzeugkasten sind z. B. die drei Konkreten Bereiche CONTAX, EPILYTIS und TWIN vorhanden:

- **CONTAX** ist ein Constraintsolver zur Repräsentation und Verarbeitung von Constraints über endlichen Wertebereichen.
- **EPILYTIS** ist ein Constraintsolver für algebraische Constraints (z. B. P=U*I), dessen Inferenzmechanismus auf der Simplex-Methode basiert.
- Mit dem Modul **TWIN** werden Probleme modelliert, die die räumliche Anordnung von Komponenten betreffen[11].

2.1.3 Komponenten-Unifikation durch den SKOLEMIZER

Um die Anzahl der *Individuen* (Komponenten) in der A-Box zu verringern, versucht der SKOLEMIZER durch Unifikation Komponenten aufgrund ihrer Funktionalitäten zusammenzufassen.

In TAXON unterscheidet man zwei Arten von Individuen in der A-Box: *Komponenten-Variablen* und *Komponenten-Konstanten*. Komponenten-Konstanten repräsentieren Komponenten, die am Ende der Konfigurierung in der Lösungsmenge tatsächlich vorhanden sind. Im Gegensatz dazu ist eine Komponenten-Variable nur ein Platzhalter für eine noch näher zu bestimmende Komponente in der Konfigurationslösung.
Am Ende einer Konfiguration dürfen in der A-Box nur noch Komponenten-Konstanten vorkommen. Um das zu erreichen, soll der SKOLEMIZER alle Komponenten-Variablen nacheinander mit Komponenten-Konstanten unifizieren. Dazu gibt es prinzipiell jeweils zwei verschiedene Möglichkeiten:

a) Der SKOLEMIZER unifiziert eine Komponenten-Variable mit einer bereits vorhandenen Komponenten-Konstanten. Dadurch verringert sich die Komponentenanzahl in der A-Box um eine Komponente.

b) Eine Komponenten-Variablen wird mit einer neuen Komponenten-Konstante unifiziert, wodurch sich die Komponentenanzahl in der A-Box nicht ändert.

Um Komponenten-Variablen und Komponenten-Konstanten symbolisch voneinander unterscheiden zu können, beginnen die Variablen-Namen grundsätzlich mit einem "?" (z. B. ?X) und die Konstanten-Namen (z. B. X) *mit irgendeinem anderen Zeichen.*

2.1.4 Erzeugung terminaler Konzepte mit dem REALIZER

Durch den REALIZER wird jedes Individuum in der A-Box irgendwann im Laufe des Konfigurierungsprozesses zu einer Instanz eines terminalen Konzeptes verfeinert. Er *spezialisiert* also jeden in der A-Box vorhandenen Komponenten-Oberbegriff zu einer Komponenten-Ausprägung (vgl. Abschnitt 2.1.1).

---

10. zusätzlich zu den *taxonomischen* Integritätsbedingungen (vgl. Abschnitt 2.1.1)
11. siehe [Schoeller94]

2.1.5 Vorwärtsregeln mit dem Tool FORWARD

Mit den bisherigen Mechanismen können nur "objekt-zentrierte" Integritätsbedingungen formuliert werden. Um auch Integritätsbedingungen bezüglich der Menge aller in der A-box vorhandenen Komponenten ausdücken zu können (d. h. zu Formulierung von *globalen Integritätsbedingungen*), wurde der TOOCON-Werkzeugkasten um das Tool FORWARD erweitert. Mit FORWARD lassen sich für die gesamte A-Box gültige *Vorwärtsregeln* definieren.

2.1.6 Erweiterte Hornklauseln mit dem Tool TAXLOG

Mit dem Modul TAXLOG lassen sich erweiterte Hornklauseln auf der Basis des Höhfeld-Smolka-Schemas formulieren[12]. Dieses zielgesteuerte Inferenz-Tool wurde in den TOOCON-Werkzeugkasten integriert, um damit Aufgabenwissen zu repräsentieren. Zum Aufgabenwissen zählt z. B. das Wissen darüber, wie eine vom Benutzer angegebenen Aufgabenspezifikation in Teilaufgaben oder in konkrete Komponenten-Anforderungen transformiert werden kann.

2.1.7 Die Steuerungskomponente CONTROL

Die Steuerung des Konfigurierungsprozesses erfolgt durch das Modul CONTROL. Alle Anforderungen bei der Aufgabenspezifikation zu Beginn der Konfigurierung und alle Inferenz-Schritte der *generativen Module* SKOLEMIZER, REALIZER, FORWARD und TAXLOG während der Konfigurierung werden hier in Form von Operatoren eingetragen und auf einer Agenda gespeichert. Bei der Ausführung eines solchen Operators werden entweder die entsprechenden Tools aktiviert oder Eintragungen in die A-Box gemacht.

Durch die Abarbeitung eines Operators kann sich der Inhalt der A-Box ändern. Aufgrund einer veränderten A-Box können dann (aktiviert durch den OBSERVER) die generativen Module erneut Operatoren auf die Agenda eintragen. Wenn allerdings die A-Box nach einem Eintrag inkonsistent ist, muß Backtracking eingeleitet werden.

Die Reihenfolge, in der die Operatoren auf der Agenda nacheinander abgearbeitet werden, ist vom explizit formulierten (heuristischen) Steuerungswissen abhängig. Dieses Steuerungswissen kann in Form von Strategie-Regeln angegeben werden.

Eine Konfigurierungsaufgabe ist erfolgreich gelöst, wenn die Agenda vollständig abgearbeitet und die A-Box immer noch konsistent ist. In der A-Box steht dann die generierte Konfigurationslösung.

12. siehe [Abecker94]

2.1.8 Zusammenfassender Überblick über die einzelnen Tools

Nachfolgende Tabelle gibt zusammenfassend einen tabellarischen Überblick über die einzelnen Tools im bisherigen TOOCON-Werkzeugkasten.

	Tool		Anwendung	Wissens-Repräsentation	Inferenz-Mechanismus
Kern-System	TAXON	T-Box	Komponentenkatalog, taxonomische Integritätsbedingungen	Konzepte, Attribute, Rollen, Konkrete Prädikate	terminologische Inferenz
		A-Box	assertionales Wissen	Individuen, Relationen, Konkrete Prädikate	
		Observer	Überwachung der A-Box	Observer-Regeln	Regel-Verarbeitung
Konkrete Bereiche	CONTAX		kombinatorische Integritätsbedingungen	Constraints über endlichen Wertebereichen	Constraint-Propagierung
	EPILYTIS			algebraische Constraints	
	TWIN		Modellierung räumlicher Anordnungsprobleme	räumliche Constraints	
	SKOLEMIZER		Minimierung der Konfiguration		
	REALIZER		Auswahl einer Komponentenausprägung		
	FORWARD		globale Integritätsbedingungen	Vorwärtsregeln	Regel-Verarbeitung
	TAXLOG		Aufgabenwissen	erweiterte Hornklauseln	logische Inferenz
Steuerung	CONTROL		Kontrollwissen	Strategieregeln	

Tabelle 2.1: Tabellarischer Überblick über den bisherigen TOOCON-Werkzeugkasten

2.2 Die Architektur und das Zusammenspiel der Tools im TOOCON-Werkzeugkasten

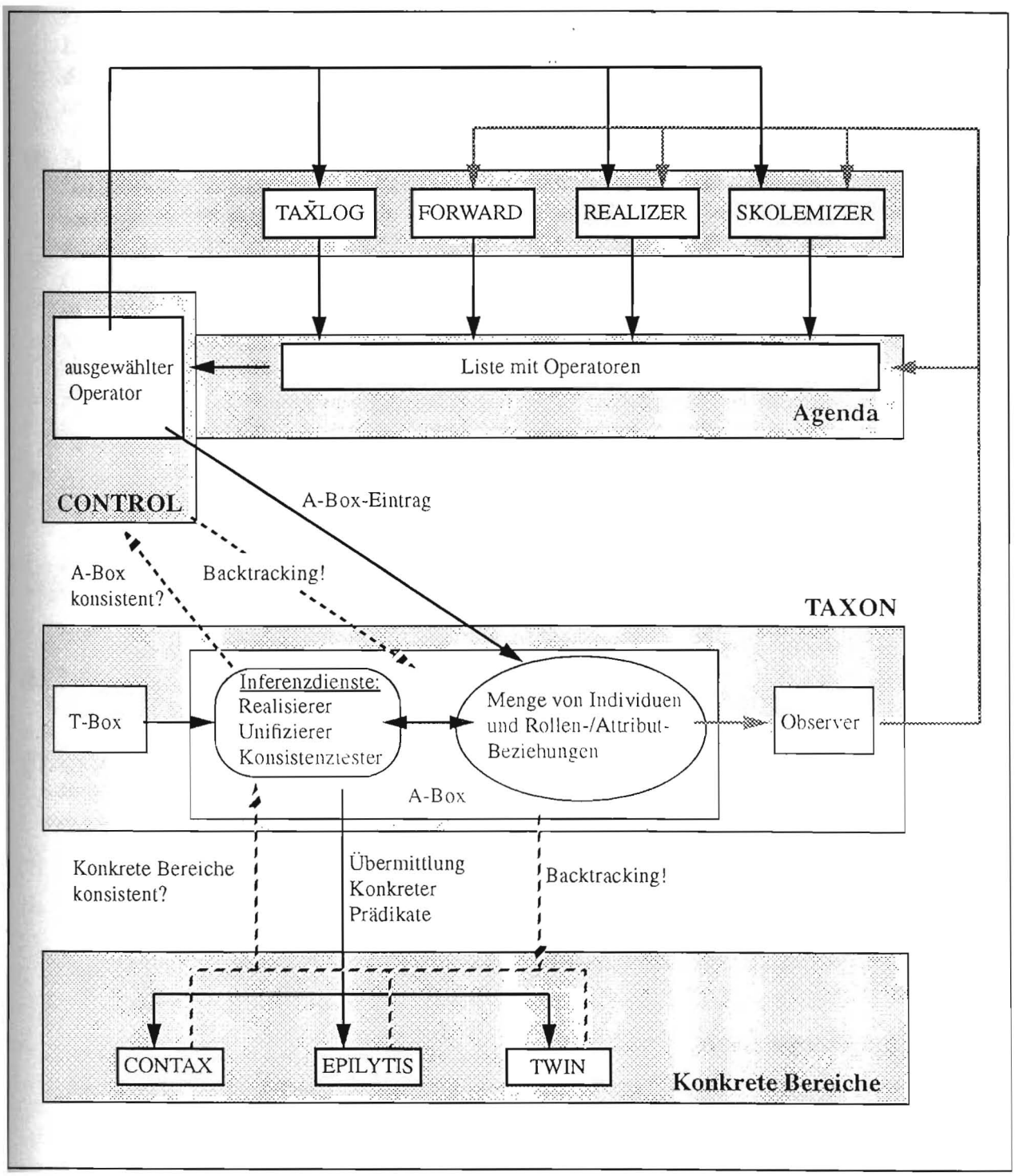


Bild 2.1: Die Architektur und der Informationsfluß im TOOCON-Werkzeugkasten

Bild 2.1 zeigt die Architektur des TOOCON-Werkzeugkastens. Außerdem versucht es die Abhängigkeiten und den Kontrollfluß zwischen den einzelnen Tools zu veranschaulichen.

Die TOOCON-Tools lassen sich in vier Ebenen einteilen. Die zentrale Komponente ist dabei das terminologische System TAXON. Es besteht aus der T-Box (Komponentenkatalog), der A-Box (konkretes Fallwissen) und dem Observer zur Überwachung der A-Box.

Um die Ausdrucksmächtigkeit von TAXON zu erhöhen, lassen sich "darunter" beliebig viele Konkrete Bereiche anschließen. Alle Konkreten Prädikate, die während eines terminologischen Inferenz-Prozesses auftreten, werden einfach direkt an den dafür jeweils zuständigen Konkreten Bereich übergeben. Dieser überprüft, ob die Konjunktion der bisher übermittelten Prädikate erfüllbar ist und teilt dies dem Konsistenztester von TAXON mit. Bei einer Inkonsistenz wird von TAXON Backtracking eingeleitet, wobei auch interne Zustände der Konkreten Bereiche zurückgesetzt werden können.

Über der Ebene von TAXON befindet sich die Steuerkomponente CONTROL und die zugehörige Agenda. CONTROL hat die Aufgabe, alle Operatoren auf der Agenda in einer strategieabhängigen Reihenfolge nacheinander abzuarbeiten. Dabei wird die Anweisung des Operators an das zugehörige Tool übergeben, welches die Anweisung dann durchführt.
Falls nach einem Eintrag in die A-Box diese inkonsistent ist und es TAXON mit einem internen Backtracking-Verfahren nicht gelingt, sie wieder konsistent zu machen, dann muß CONTROL ein globales Backtracking einleiten.

Auf der obersten Ebene sind die Tools TAXLOG, FORWARD, REALIZER und SKOLEMIZER angesiedelt. Diese Tools können entweder von CONTROL aufgrund eines entsprechenden Operators auf der Agenda oder bei bestimmten A-Box-Zuständen durch den Observer aktiviert werden. Nachdem ein solches Tools angestoßen wurde, kann es seinerseits Operatoren auf die Agenda eintragen.

2.3 Der Ablauf einer Konfigurierung mit dem TOOCON-System

Grundprinzip:

Die Spezifikation der gewünschten Konfiguration kann man als eine Menge von Constraints, d. h. Einschränkungen an den Lösungsraum ansehen. Mit Hilfe verschiedener Constraint-Propagierungstechniken, wie z. B. die automatische Realisierung von TAXON oder die Propagierungsverfahren der Constraintsolver, soll der Suchraum immer weiter eingeschränkt werden. Anschließend soll aus der verbleibenden Lösungsmenge eine möglichst optimale Lösung ausgewählt werden.

Algorithmus/Vorgehensweise:

Anhand des nachfolgenden Algorithmus soll die prinzipielle technische Vorgehensweise bei der Lösung einer Konfigurationsaufgabe mit dem TOOCON-Werkzeugkasten verdeutlicht werden:

(1) Aufgabenspezifikation:
Die vom Benutzer formulierte Spezifikation der Konfigurationsaufgabe wird in entsprechende Operatoren transformiert und auf die Agenda eingetragen.

(2) Operator-Auswahl:
Die Steuerungskomponente CONTROL wählt aufgrund verschiedener Kriterien einen Operator aus der Agenda aus und löscht ihn von der Agenda.

(3) Operator-Abarbeitung:
Der ausgewählte Operator wird durch das jeweils dafür zuständige Tool abgearbeitet. Dadurch können z. B. weitere Operatoren auf die Agenda eingetragen werden (z. B. bei der Abarbeitung eines AND-Operators). Es kann aber auch der A-Box-Inhalt verändert oder ein anderes TOOCON-Tool aktiviert werden.

(4) Konsistenz-Überprüfung der A-Box:
Die Konsistenz der (Teil-)Lösung in der A-Box wird durch TAXON und die daran angeschlossenen Konkreten Bereiche überwacht. Falls dabei erkannt wird, daß einige taxonomische Integritätsbedingungen[13] noch nicht erfüllt sind, versucht TAXON die bisherige (Teil-)Lösung in dieser Hinsicht zu vervollständigen. Wenn aber die A-Box inkonsistent ist, dann werden solange durch Backtracking die letzten Inferenzschritte rückgängig gemacht und Alternativen ausprobiert, bis die A-Box entweder wieder konsistent ist oder bis es keine weiteren Alternativen mehr gibt. Im letzteren Fall würde die Suche nach einer Konfigurationslösung erfolglos abgebrochen, die derzeitige A-Box gelöscht und anschließend im Algorithmus zu Schritt (7) gesprungen werden.

(5) Anwendung von Observer-Regeln:
Wenn sich die A-Box geändert hat, werden alle Observer-Regeln ermittelt, die dadurch mit einer neuen Instanziierung anwendbar sind. Die Anweisungen (d. h. die jeweilige Observer-Conclusion) dieser Regeln werden dann ausgeführt, wodurch z. B. andere TOOCON-Tools aktiviert oder neue Operatoren auf die Agenda eingetragen werden können.

(6) Ist die Agenda leer?
Wenn die Agenda noch nicht leer ist, gehe zu Schritt (2).

(7) Ende:
In der A-Box steht jetzt die gefundene Konfigurationslösung. Wenn keine Lösung gefunden wurde, dann ist die A-Box jetzt leer.

13. vgl. Abschnitt 2.1.1

2.4 Verwirklichung des struktur-orientierten Konfigurierungs-Ansatzes durch den TOOCON-Werkzeugkasten

Der Kern von TOOCON, das terminologische System TAXON, repräsentiert sowohl die "IS-A"-Taxonomie der Komponentenbegriffe als auch die Struktur-Beziehungen zwischen den Komponenten. Außerdem verwirklicht es die struktur-orientierten Inferenzen[14] *Zerlegung*, *Spezialisierung* und *Parametrierung*:

- Die *Zerlegung* einer Komponente in Teilkomponenten wird im wesentlichen durch die Vervollständigungs-Fähigkeit des Konsistenztesters in der A-Box erreicht (siehe Kapitel 2.1.1 und da insbesondere Beispiel 2.1). Aber auch die Module FORWARD (siehe Kapitel 2.1.5) und TAXLOG (siehe Kapitel 2.1.6) können eine Komponente in Teilkomponenten zerlegen.
- Für die *Spezialisierung* einer Komponente ist der Realisierer in der A-Box zuständig (siehe Kapitel 2.1.1).
- Die *Parametrierung* von Komponenten erfolgt durch A-Box-Einträge. Solche Eintäge in die A-Box können aus unterschiedlichen Quellen stammen, z. B. aufgrund der Aufgabenspezifikation oder z. B. indem ein Individuum in der A-Box durch den REALIZER (siehe Kapitel 2.1.4) explizit zu einer Komponentenausprägung spezialisiert wird.

Aufgrund dieser Eigenschaften ist der TOOCON-Werkzeugkasten ein struktur-orientiertes Konfigurierungssystem.

14. vgl. Kapitel 1.5.1

3. Ressourcen-orientiertes Konfigurieren

In diesem Kapitel wird die kurze Einführung des ressourcen-orientierten Konfigurierungsansatzes aus dem Einleitungskapitel 1.5.2 weiter vertieft. Anschließend werden die Vor- und Nachteile gegenüber dem struktur-orientierten Ansatz ausführlich diskutiert und eine Integration beider Ansätze motiviert.

3.1 Der ressourcen-orientierte Konfigurierungsansatz

Dieses Unterkapitel beschreibt zunächst die ressourcen-orientierte Wissensmodellierung und die Bilanzierung von Ressourcen. Anschließend wird auf die beiden prinzipiellen Anwendungsmöglichkeiten einer ressourcen-orientierten Wissensmodellierung eingegangen. Zum Abschluß wird eine Methode vorgestellt, wie man sich Komponenten-Ressourcen-Beziehungen graphischen veranschaulichen kann.

3.1.1 Ressourcen-orientierte Wissensmodellierung

Der ressourcen-orientierte Ansatz beruht auf der Annahme, daß es in der zu modellierenden Domäne eine oder mehrere Ressourcen bzw. als Ressourcen modellierbare funktionale Komponenteneigenschaften gibt. Diese Voraussetzung ist bei den meisten Konfigurationsproblemen erfüllt. Mögliche Ressourcen wären z. B. Energie, Leistung, Strom bei einer bestimmten Spannung, Speicherplatz, Geldbudget oder z. B. auch Anzahl der vorhandenen Steckplätze von einem bestimmten Typ.

Ressourcen können verbraucht werden (z. B. Geld) oder gemeinsam genutzt werden (z. B. Software). Die letztere Ressourcen-Art kann, sobald sie einmal zur Verfügung steht, beliebig oft benutzt werden, ohne daß sie weniger wird.
Komponenten in der Domäne, die von einer Ressource eine gewisse Menge verbrauchen bzw. nutzen oder zumindest fordern werden als *Ressourcen-Verbraucher* bezeichnet. Komponenten, die etwas von einer Ressource zur Verfügung stellen, heißen *Ressourcen-Erzeuger*.
In vielen Fällen ist es so, daß ein und dieselbe Komponente gleichzeitig sowohl einige Ressourcen verbraucht als auch einige zur Verfügung stellt. Eine Festplatte z. B. stellt etwas von der Ressource "Speicherplatz" zur Verfügung und fordert dafür im Gegenzug etwas von der Ressource "elektrische Leistung".

Bei der Wissensmodellierung wird für jede einzelne Komponente angegeben, ob und wenn ja, wieviel sie von welcher Ressource verbraucht bzw. nutzt (*Anforderung*) oder zur Verfügung stellt (*Bereitstellung*).
Ressourcen können auch von der Konfigurationsumgebung (*ENVIRONMENT*) zur Verfügung gestellt bzw. gefordert werden. Dadurch kann man eine Art Wechselwirkung zwischen Umgebung und technischem System modellieren. Diese Möglichkeit ist wichtig zur Festlegung von bestimmten Randbedingungen für die Konfiguration, also speziell für die Spezifikation der Schnittstelle zur Außenwelt.
Meistens gibt es in der zu modellierenden Domäne Ressourcen, die von keiner einzigen Komponente aus dem Komponentenkatalog, sondern ausschließlich von der Konfigurationsumgebung zur Verfügung gestellt werden (*Environment-Ressourcen*). Zu diesen Ressourcen gehören z. B. das vorhandene Geldbudget oder der insgesamt maximal zur Verfügung stehende Platz.

3.1.2 Ressourcen-Bilanzierung

Eine Ressource gilt als *ausbilanziert*, wenn von ihr mindestens soviel zur Verfügung gestellt wird, wie gefordert ist, d. h. wenn

$$\sum (\text{Anforderung}) \leq \sum (\text{Bereitstellung})$$

Ansonsten liegt ein *Ressourcen-Defizit* vor. Damit eine Konfiguration korrekt[15] und vollständig[16] sein kann, müssen alle diese impliziten Ressourcen-Bilanzen ausgeglichen sein.

Im wesentlichen handelt sich hierbei um ein *Constraint-Problem*:
Seien $RM^-(K, R)$ und $RM^+(K, R)$ Funktionen, die angeben, wieviel (d. h. welche *Ressource-Menge*) die Komponente K von der Ressource R verbraucht bzw. zur Verfügung stellt. Damit eine Ressource R ausbilanziert ist, muß dann folgendes Constraint erfüllt sein:

$$\sum_{K \in \text{Lösungsmenge}} RM^-(K, R) \stackrel{!}{\leq} \sum_{K \in \text{Lösungsmenge}} RM^+(K, R)$$

Dabei ist zu beachten, daß die Komponenten in der Lösungsmenge erst während der Laufzeit, also während des Inferenzprozesses, festgelegt werden. Dadurch ergibt sich hier ein *dynamisches Constraint-Problem*, bei dem die einzelnen Variablen (d. h. die Komponenten und die zugehörigen Ressource-Mengen) und auch deren Anzahl während der Konfigurierung sich ständig ändern können.

3.1.3 Formulierung und Überprüfung von ressourcen-orientierten Integritätsbedingungen

Die impliziten Bilanzierungs-Constraints kann man als *ressourcen-orientierte Integritätsbedingungen* betrachten. Im einfachsten Fall wird eine ressourcen-orientierte Modellierung nur zum Testen dieser Bedingungen, d. h. zur zusätzlichen Überprüfung einer gegebenen Konfiguration auf Konsistenz und Vollständigkeit verwendet.

3.1.4 Ressourcen-orientierte Konfigurierung

Man kann eine ressourcen-orientierte Modellierung aber auch zum automatischen Generieren einer Konfigurationslösung einsetzen. Ausgehend von der Aufgabenspezifikation wird eine erste Teillösung erzeugt. Anhand der Ressourcen-Defizite soll diese Teillösung nun schrittweise vervollständigt werden, indem die verletzten ressourcen-orientierten Integritätsbedingungen wissensbasiert repariert werden. Dazu werden in einem inkrementellen Prozeß jeweils Anforderung und Bereitstellung miteinander verglichen und daraufhin z. B. Komponenten in die Lösungsmenge hinzugefügt oder durch andere Komponenten ersetzt. Dadurch kann es

15. Bei einem nicht behebbaren Ressource-Defizit ist eine Konfiguration inkonsistent und damit nicht korrekt.
16. Bei einem behebbaren Ressource-Defizit ist eine Konfiguration einfach nur unvollständig.

erneut zu Ressourcenverletzungen kommen, worauf die Lösungsmenge wieder entsprechend verändert oder erweitert werden muß. Erst wenn alle Ressourcen ausbilanziert sind, ist die Konfigurationsaufgabe gelöst.

3.1.5 Graphische Veranschaulichung

Bild 3.1 demonstriert anhand eines abstrakten Beispiels eine Methode, wie man sich eine ressourcen-orientierte Modellierung graphisch veranschaulichen kann.

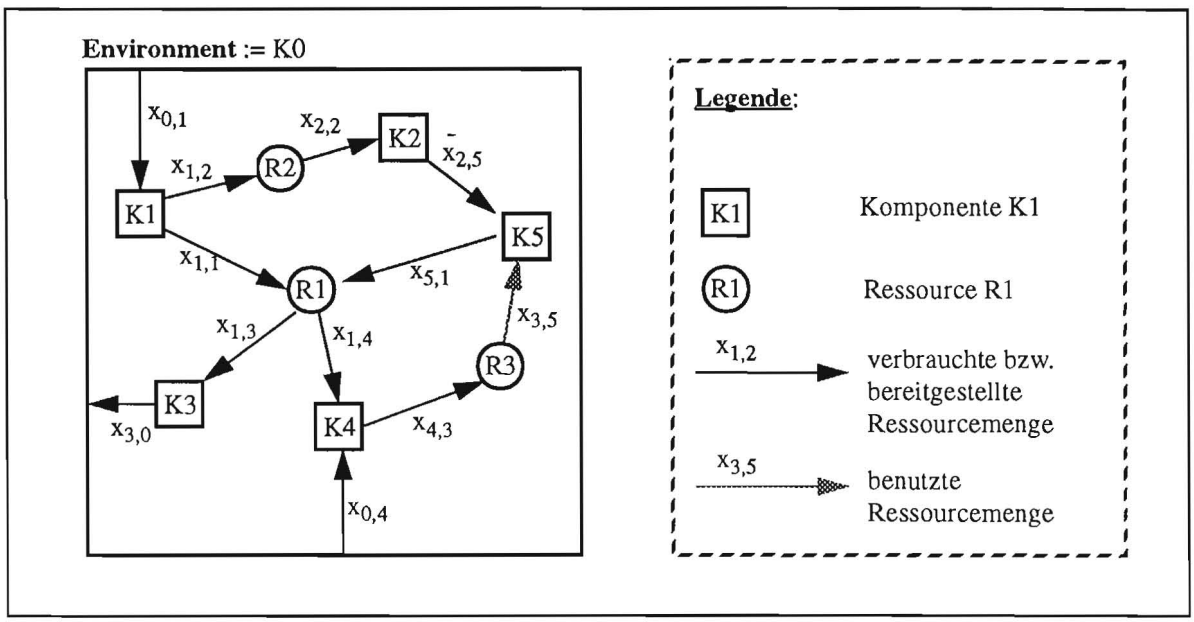
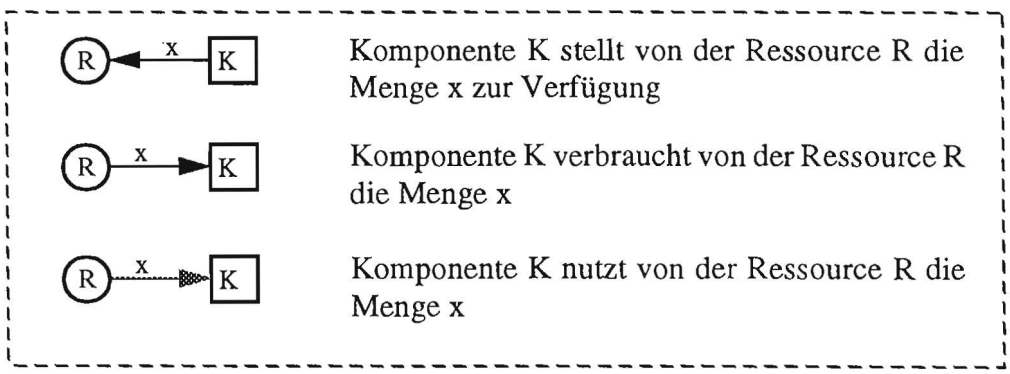


Bild 3.1: Graphische Veranschaulichung einer ressourcen-orientierten Modellierung

In der Domäne dieses Beispiels gibt es die Komponenten K1, K2, ... ,K5 (jeweils symbolisiert durch ein Rechteck) und die Ressourcen R1, R2 und R3 (jeweils symbolisiert durch ein Kreis). Zwischen Komponenten und Ressource gibt es gerichtete Kanten, die angeben, wieviel von einer Ressource gefordert bzw. bereitgestellt wird. Dabei können prinzipiell folgende drei Fälle auftreten:



Die Umgebung des technischen Systems (ENVIRONMENT) kann genau wie jede beliebige Komponente Ressourcen bereitstellen oder fordern. Daher kann sie als eine spezielle Komponente (hier als K0 bezeichnet) angesehen werden.

3.2 Bewertung des ressourcen-orientierten Ansatzes und Vergleich mit dem struktur-orientierten Ansatz

Der ressourcen-orientierte Ansatz liefert ein allgemeines, anwendungsunabhängiges Modell zur Konfiguration technischer Systeme. Hierbei werden alle Komponentenschnittstellen und damit auch deren Haupteigenschaften einheitlich in Form von Ressourcen modelliert.

3.2.1 Vorteile einer ressourcen-orientierten Modellierung

Dieser Ansatz hebt sich durch seine einfache Wissensmodellierung und relativ einfachen Inferenzprozeß hervor. Bei den einzelnen Komponenten braucht nämlich nur angegeben zu werden, wieviel sie von welcher Ressource benötigen bzw. bereitstellen.

Die einzelnen Komponenten stehen in keiner direkten Objektbeziehung, sondern sind nur indirekt über den Austausch von Ressourcen miteinander verbunden. Sie stehen über die Ressourcen somit in einer n:m-Beziehung:

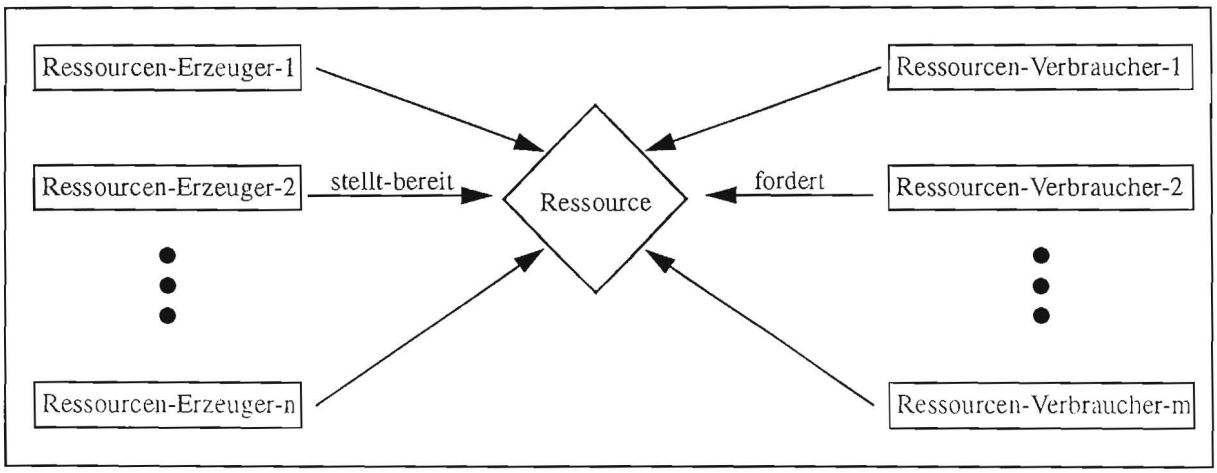


Bild 3.2: Die n:m-Beziehung zwischen Komponenten, die Ressourcen bereitstellen und Komponenten, die Ressourcen benötigen

Durch diese Abstraktion wird das Wissens über die tatsächliche Beziehung zwischen den einzelnen Komponenten durch die Ressourcen entkoppelt. Dadurch kann die systematische Strukturierung des Wissens (z. B. in Taxonomien) entfallen und der Erstellungsaufwand wird erheblich reduziert.

Im Gegensatz zum struktur-orientierten Ansatz kann hier das relativ umfangreiche Katalogwissen vom eigentlichen Expertenwissen besser getrennt werden. Denn im Katalog werden in einer deklarativen Form nur die funktionalen Komponenten-Eigenschaften beschrieben. Dabei muß nicht angegeben werden, in welcher (direkten) Beziehung die einzelnen Komponenten zueinander stehen bzw. wie sie in einer Konfiguration miteinander verbunden werden.

Außerdem wird die Wartung der Wissensbasis erleichtert, da z. B. keine Abhängigkeiten beim Löschen oder Hinzufügen von Objekten berücksichtigt werden müssen. Das bedeutet, daß eine lokale Änderung der Wissensbasis möglich ist, ohne die anderen Komponenten überarbeiten zu müssen. Dies bewirkt eine Reduzierung des Wissenspflegeaufwandes.

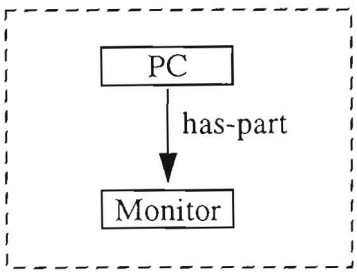
Der ressourcen-orientierte Ansatz hat noch einen weiteren Vorteil, der besonders für die Praxis relevant sein dürfte: Prinzipiell ist es relativ einfach möglich, daß mehrere Konfigurierungssysteme über einheitliche Ressourcen-Schnittstellen zusammenarbeiten. In dem Fall würde man die (lokale) Umgebung eines Systems als eine komplexe Komponente eines größeren technischen Systems betrachten.

3.2.2 Probleme bei einer ressourcen-orientierten Modellierung und deren Lösungsansatz

Für ressourcen-orientiertes Konfigurieren braucht man "nur" die verwendeten Komponenten und deren Ressourcen-Schnittstellen anzugeben. Auf der anderen Seite kann man aber sonstiges vorhandenes Expertenwissen oft nur sehr schlecht bzw. unnatürlich repräsentieren.

Modellierung einer Aufbau-Struktur:

Man betrachte z. B. die "Has-Part"-Beziehung bei der Komposition von Objekten. Angenommen, in einer zu modellierenden Rechner-Domäne würde zu jedem PC als Bestandteil genau ein Monitor gehören.



Um diese Beziehung rein ressourcen-orientiert zu modellieren, müßte man für sie eine eigene Ressource deklarieren, z. B. Monitor-Ressource, die man sonst nirgends verwenden könnte. Außerdem wäre weder beim PC noch beim Monitor zu erkennen, daß beide Objekte in einer direkten "Has-Part"-Beziehung zueinander stehen. Solche Probleme hat man oft z. B. bei der ressourcen-orientierten Modellierung von 1:1-Beziehungen. Beim struktur-orientierten Ansatz auf der Basis eines terminologischen Systems könnte man solche Beziehungen adäquater als Komponenten-Relation (Attribut oder Rolle) repräsentieren.
Anhand dieses Beispiels ist zu erkennen, daß es prinzipiell möglich ist, Strukturen durch Ressourcen zu simulieren, indem man z. B. jede einzelne Komponenten-Beziehung durch eine eigene Ressource ausdrückt. Allerdings wäre eine solche Modellierung in der Regel weder adäquat noch effizient!

Flache Konfigurierung:

Als Ergebnis bekommt man beim ressourcen-orientierten Konfigurieren eine ungeordnete und unstrukturierte Liste der benötigten Komponenten. Man spricht hier daher von einer *flachen Konfigurierung*. Wenn man aber eine sehr große Lösungsmenge mit vielen Objekten hat, wäre eine strukturierte Lösung wünschenswert, allein schon deshalb, weil bei einer übersichtlichen Lösungspräsentation Fehler leichter zu erkennen sind. Um dieses Problem etwas zu verklei-

nern, könnte man *lokale Ressourcen* verwenden. Dadurch würde man die gesamte Lösungsmenge in einzelne Teilmengen untergliedern.[17]
Die struktur-orientierte Wissensmodellierung mit einem terminologischen Basissystem ist im allgemeinen aufwendiger als die ressourcen-orientierte Modellierung. Dafür hat man dann aber die Möglichkeit, eine strukturierte Lösung zu generieren.

Ressourcen-Abhängigkeiten:

Normalerweise geht man bei der ressourcen-orientierten Konfigurierung davon aus, daß die einzelnen Ressourcen prinzipiell unabhängig voneinander sind. Dies ist aber in der Praxis leider nicht immer zutreffend. Man könnte z. B. in einer elektrotechnischen Domäne sowohl die elektrische Leistung P als auch den elektrischen Strom I (bei jeweils einer festen Spannung U) als Ressourcen modellieren. Diese Ressourcen wären dann über das Gesetz P=U*I direkt voneinander abhängig. Um ein solches algebraisches Constraint über Ressourcen-Variablen modellieren zu können, wäre eine Konzepterweiterung notwendig.[18]

Optimalität der generierten Lösung:

Die ressourcen-orientierte Konfigurierung liefert am Ende höchstens eine Lösung. Aufgrund von ihrem relativ einfachen Inferenz-Mechanismus kann sie nur irgendeine, aber nicht unbedingt eine optimale Lösung finden. Hier werden nacheinander schrittweise alle Ressourcen ausbilanziert und jeweils bei Ressourcen-Verletzungen alternative Komponentenmengen erzeugt. Im Gegensatz dazu werden bei einer constraint-basierten Konfiguration mehrere konsistente Lösungen gleichzeitig betrachtet und nach der Propagierung aller Anforderungen die möglichst optimalste Lösung ausgewählt. Dieses Vorgehen ist hier aber nicht möglich, da man dazu alle Ressourcen gleichzeitig betrachten und ausbilanzieren müßte.
Statt dessen könnte man eine Steuerungskomponente integrieren, die aufgrund von benutzerdefinierten Heuristiken und Strategien eine möglichst optimale Reihenfolge bei der Ressourcen-Bilanzierung bestimmt und bei der Komponentenauswahl die Entscheidung aufgrund mehrerer Ressourcen-Bilanzierungs-Constraints trifft. Dadurch würde man versuchen, eine Konfigurationslösung zu finden, die auf lokalen Optimalitätsentscheidungen beruht. Es ist allerdings fraglich, inwieweit solches Steuerungswissen vom Experten akquiriert werden kann.

3.3 Motivation für die Integration beider Ansätze in einem einzigen Konfigurierungssystem

Die ressourcen-orientierte Konfiguration hat einige interessante Vorteile und ist für viele Anwendungsbereiche gut geeignet. Viele Probleme lassen sich mit Ressourcen elegant beschreiben und lösen. Es gibt aber auch einige Nachteile, die man nicht vernachlässigen sollte. In bestimmten Anwendungsbereichen kann eine ressourcen-orientierte Konfigurierung inadäquat oder ineffizient sein (z. B. bei der "Has-Part"-Relation). In solchen Fällen müßte

17. Lokale Ressourcen werden ausführlich in Kapitel 7 behandelt.
18. Man könnte versuchen, dieses Problem zu umgehen, indem man bei jeder Komponente den Strom in die zugehörige Leistung umrechnet. Aber selbst das würde nicht immer funktionieren.
Beispiel: Netzteil, das unterschiedliche Spannungen zur Verfügung stellt, bei dem aber sowohl die maximale Leistung P_{max} als auch der maximale Strom I_{max} begrenzt sind, aber nicht $P_{max} = U_{max} * I_{max}$ gilt.
(Nebenbemerkung: Man beachte, daß es sich hier nicht um ein rein akademisches Beispiel handelt, da solche Netzteile in der Praxis sogar recht häufig vorkommen!)

man auf andere Methoden (z. B. auf struktur-orientierte Komponenten-Repräsentation) zugreifen können! Daher wäre eine Erweiterung oder Integration mit anderen Repräsentationsarten und Inferenzmethoden sehr sinnvoll. Neben der adäquateren Modellierung ist dabei eine Steigerung der Effizienz zu erwarten, da es für diese Repräsentationsmechanismen spezialisiertere und daher effizientere Inferenz-Algorithmen gibt. Dadurch hätte der Wissensingenieur die Möglichkeit, jeweils die adäquateste Repräsentations- und Inferenzmethode auszuwählen.

Durch eine Integration mit dem struktur-orientierten Ansatz hat man aber nicht nur jeweils die Auswahl zwischen mehreren Mechanismen, sondern gewinnt noch zusätzliche Vorteile. So können beide Ansätze durch eine Integration profitieren:

- Bei Ressourcen-Verletzungen hat man oft die Auswahl zwischen mehreren Alternativen. Normalerweise muß man sich dabei für irgendeine konkrete Alternative entscheiden. Dadurch kann es zu späteren Konflikten kommen.

  Idee:
  Statt einer konkreten Alternative gibt man das Oberkonzept aller möglichen Alternativen an. Spätere Propagierungen von Ressourcen-Forderungen führen dann im Idealfall nur zu Einschränkungen der vorhandenen Oberkonzepte. Dadurch können im Konfigurierungsprozeß unter Umständen viele teure Backtracking-Operationen vermieden werden.[19]

- Ressourcen-Informationen können bei einem struktur-orientierten Konfigurierungssystem zusätzlich zum Aufbau der Taxonomie und zum Formulieren und Prüfen ressourcen-orientierter Integritätsbedingungen verwendet werden.

19. Auf diese Idee wird in Kapitel 4.2.5 nocheinmal näher eingegangen, und in Kapitel 5.2.3 ist beschrieben, wie sie in die Praxis umgesetzt werden kann.

4. Verschiedene Ansätze zur Integration ressourcen-orientierter Techniken in TOOCON

Der TOOCON-Werkzeugkasten basiert im wesentlichen auf dem struktur-orientierten Ansatz (siehe Kapitel 2.4). Eine ressourcen-orientierte Konfiguration war in diesem System bisher aber nicht vorgesehen. Die Frage ist jetzt, ob der ressourcen-orientierte Ansatz grundsätzlich ein Widerspruch zum TOOCON-System ist oder ob ressourcen-orientierte Techniken darin irgendwie integriert werden können. Zur Beantwortung dieser Frage, werden in diesem Kapitel verschiedene Integrationsansätze diskutiert[20].

4.1 Kompilation

Bei diesem Verfahren versucht man die ressourcen-orientierten Methoden auf andere Tools des TOOCON-Werkzeugkastens abzubilden, d. h. sie mit anderen Mechanismen zu simulieren.

Da die Ressourcen-Bilanzierung im wesentlichen ein *dynamisches Constraint-Problem* darstellt (siehe Kapitel 3.1.2), könnte man (von den bereits vorhandenen Tools des TOOCON-Systems) für eine Kompilation den algebraischen Constraint-Solver EPILYTIS verwenden. CONTAX ist zwar ebenfalls ein Constraint-Solver, aber nur für Constraints über endlichen Wertebereichen (siehe Kapitel 2.1.2) und daher hier nicht geeignet.

Um das dynamische Constraint-Problem der Ressourcen-Bilanzierung verarbeiten zu können, müßte EPILYTIS in der Lage sein, eine Summe über eine nicht festgelegte, aber beliebig große Anzahl von Termen bzw. eine entsprechende inkrementelle Summenformel der Art $X := X + Y$ berechnen zu können. Denn während der gesamten Konfigurierung können immer wieder neue Komponenten in die A-Box eingetragen werden, die von einigen Ressourcen eine bestimmte Menge bereitstellen oder fordern, wodurch sich die entsprechenden Summenbilanzen ständig ändern. Die Verarbeitung derartiger dynamischer Summen ist aber in diesem algebraischen Constraintsolver von CLP (R) [JMSY87], wenn überhaupt, nur mit einem sehr großen Änderungsaufwand möglich.

Das bedeutet, daß man ressourcen-orientierte Techniken besser über ein neues Tool in das TOOCON-System integrieren sollte.

4.2 Integration ressourcen-orientiertierter Techniken durch die Erweiterung des TOOCON-Systems um ein neues Tool

Wenn man das TOOCON-System zur Integration ressourcen-orientierter Techniken um ein neues Tool erweitert, dann muß dieses Tool zwangsläufig mit TAXON eng zusammenarbeiten, da in TAXON sowohl das Komponentenwissen als auch das assertionale Wissen repräsentiert wird. Zur leichteren Referenzierung wird dieses (geplante) neue Tool im folgenden Text mit dem Namen *TAXRES* bezeichnet, da es eine Verbindung zwischen einem taxonomischen System (TAXON) und Ressourcen herstellen soll.

20. vgl. [Wache93]

Für eine ressourcen-orientierte Modellierung muß man für jede einzelne Komponente angeben können, wieviel sie jeweils von welcher Ressource bereitstellt bzw. verbraucht. Hierbei gelten die Komponenten als *Abstrakte Objekte*. Daher können sie in der T-Box von TAXON als *Konzepte* repräsentiert werden. Im Gegensatz dazu werden Zahlenwerte, wie sie z. B. zur Beschreibung der bereitgestellten bzw. verbrauchten Menge einer Ressource (*Ressource-Menge*) benötigt werden, als *Konkrete Objekte* angesehen. Auf Konkrete Objekte kann im terminologischen System TAXON aber nur indirekt in Form von *Konkreten Prädikaten* zugegriffen werden, die jeweils zu einem bestimmten *Konkreten Bereich* gehören[21].

Das bedeutet, daß man die Ressourcen-Informationen für jede einzelne Komponente entweder komplett in TAXRES repräsentieren muß oder daß TAXRES als *zulässiger Konkreter Bereich* an TAXON angeschlossen sein muß. Im ersten Fall wäre TAXRES ein relativ unabhängiges Modul, das, um die einzelnen Ressourcen-Bilanzen überwachen zu können, nur wissen müßte, welche *Individuen* (d. h. welche *Komponenten-Instanzen*) in der A-Box vorhanden sind. Im zweiten Fall wäre TAXRES stark von TAXON abhängig, da dann die Information, welche Komponente wieviel von welcher Ressource fordert bzw. bereitstellt, nur in der T-Box von TAXON gespeichert wäre.

4.2.1 Erster Ansatz: TAXRES als ein von TAXON relativ unabhängiges Modul

Bei diesem Ansatz ist TAXRES ein eigenständiges Modul, das alle Ressourcen-Informationen verwaltet und die Ressourcen-Bilanzen überprüft. Die für die Bilanzierung notwendige Information, welche Komponenten in der A-Box vorhanden sind, könnte man TAXRES z. B. mit Hilfe von Vorwärts- bzw. Observer-Regel mitteilen.

Diese erste Variante hat aber den entscheidenden Nachteil, daß TAXON dabei keinen Zugriff auf die Ressourcen-Informationen hat und daher diese weder in der T-Box beim Aufbau der Taxonomie noch in der A-Box beim Konsistenztest oder für die Realisierung verwenden kann. Daher sollte man hier die zweite Variante, die im nächsten Abschnitt beschrieben wird, nach Möglichkeit vorziehen.

4.2.2 Zweiter Ansatz: TAXRES wird als Konkreter Bereich an TAXON angeschlossen

Hierbei soll TAXRES an das terminologische System TAXON als *Konkreter Bereich*[22] angekoppelt werden. Die Schnittstelle zwischen TAXON und TAXRES besteht dann ausschließlich aus *Konkreten Prädikaten*. Bild 4.1 soll das sich daraus ergebende Kommunikationsprotokoll zwischen TAXON und TAXRES verdeutlichen:

---

21. Konkrete Objekte können in einem puren terminologischen System überhaupt nicht verwendet werden. Aus diesem Grund wurde das terminologische System TAXON um die Möglichkeit zur Anbindung Konkreter Bereiche erweitert (siehe Kapitel 2.1.2).
22. Die formale Definition eines zulässigen Konkreten Bereichs ist in Kapitel 2.1.2 beschrieben. Damit das neue Tool TAXRES vom terminologischen System TAXON als zulässiger Konkreter Bereich akzeptiert wird, muß es alle diese Anforderungen erfüllen und die entsprechenden Schnittstellenfunktionen zur Verfügung stellen.

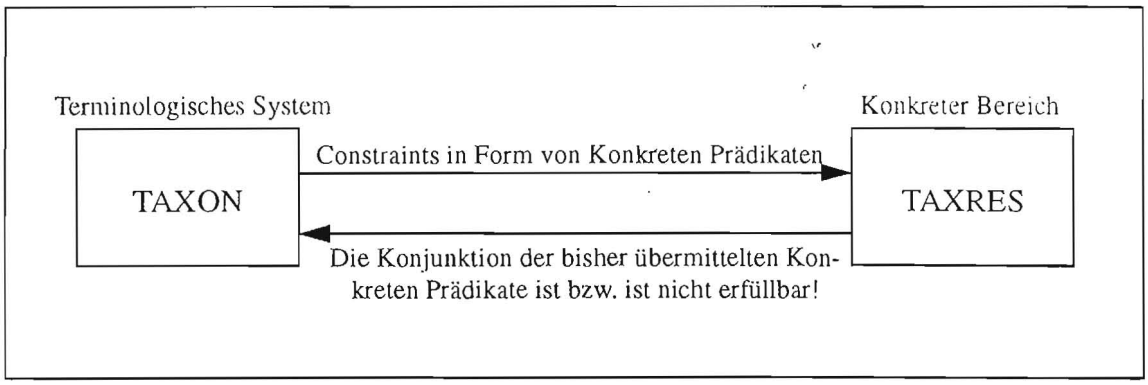


Bild 4.1: Die Konkrete-Bereiche-Schnittstelle zwischen TAXON und TAXRES

Konkrete Prädikate kann man als *Constraints*[23] an den zugehörigen Konkreten Bereich ansehen. Wenn TAXON in einem Inferenz-Prozeß ein Konkretes Prädikat von TAXRES auszuwerten hat, dann übergibt es dieses an TAXRES. Daraufhin überprüft TAXRES, ob die Konjunktion aller bisher übermittelten Konkreten Prädikate (Constraints) erfüllbar ist oder nicht. Das Ergebnis dieser Überprüfung wird TAXON mitgeteilt, was dann daraus entsprechende Konsequenzen[24] ziehen kann.

Ansatz zur Definition einer Semantik für die zugehörigen Konkreten Prädikate:

Man könnte mit Hilfe Konkreter Prädikate überprüfen, ob die ressourcen-orientierten Integritätsbedingungen erfüllt sind oder nicht. Dazu könnte man sich evtl. für jede Ressource R ein Konkretes Prädikat KP_R definieren, das genau dann erfüllt ist, wenn die implizite *Ressourcen-Bilanzierungs-Bedingung* (siehe Kapitel 3.1.2) erfüllt ist:

$$KP_R(...) \text{ ist erfüllt} \quad :\Leftrightarrow \quad \left(\sum_{K \in \text{A-Box}} RM^-(K, R) \quad \leq \quad \sum_{K \in \text{A-Box}} RM^+(K, R) \right)$$

wobei $RM^-(K, R)$ und $RM^+(K, R)$ Funktionen sind, die angeben, wieviel (d. h. welche *Ressource-Menge*) die Komponente K von der Ressource R verbraucht bzw. zur Verfügung stellt.

Hierbei müßten die Argumente des Konkreten Prädikates KP_R die Information enthalten, welche Komponente in der A-Box wieviel von der Ressource R fordert bzw. bereitstellt. Da sich die Anzahl der Komponenten in der A-Box aber ständig ändert, ist es unklar, wie man die Argumente von KP_R spezifizieren könnte. Bei diesem Ansatz würde aber noch ein viel größeres Problem entstehen:

23. vgl. Kapitel 1.4
24. z. B. Einleitung von Backtracking

Hauptproblem bei diesem Ansatz:

Verletzungen bei den ressourcen-orientierten Integritätsbedingungen sind unter Umständen reparierbar, indem neue Komponenten in die A-Box eingetragen werden (→ ressourcen-orientiertes Konfigurieren). Daher ist hier zwischen folgenden drei Fällen zu unterscheiden:

(1) Alle Ressourcen sind ausbilanziert, d. h. alle *Bilänzierungs-Bedingungen sind erfüllt.*

(2) Es gibt zwar Ressourcen-Defizite, die aber beseitigt werden können, indem z. B. die Komponenten K_1, K_2, ..., K_n in die A-Box instanziiert werden. In diesem Fall ist die A-Box einfach nur *unvollständig*, aber die *Bilanzierungs-Bedingungen sind noch erfüllbar.*

(3) In der A-Box gibt es Ressourcen-Defizite, die aus irgendeinem Grund unter Beibehaltung der bisherigen Konfiguration mit Sicherheit nicht beseitigt werden können[25], d. h. die A-Box *ist inkonsistent*! In diesem Fall muß unbedingt Backtracking eingeleitet werden, da ansonsten die *Bilanzierungs-Bedingungen nicht mehr erfüllbar* wären.

Da aber ein Konkreter Bereich TAXON nur mitteilen kann, ob die Konjunktion aller Konkreten Prädikate erfüllbar ist oder nicht, reicht die reine Konkrete-Bereiche-Schnittstelle für eine ressourcen-orientierte Konfigurierung nicht aus! Diese Feststellung führt zu einer Aufsplittung der zu übertragenden Informationen und zu einer erweiterten Schnittstelle von TAXRES (siehe Abschnitt 4.2.3) und außerdem zu einer anderen Semantik-Definition für die Konkreten Prädikate (siehe Abschnitt 4.2.4). Zum Abschluß wird in Abschnitt 4.2.5 noch auf zwei prinzipielle Möglichkeiten eingegangen, wie bei einer unvollständigen Konfiguration reparierbare Ressourcen-Bilanzierungs-Bedingungen erfüllt werden können.

4.2.3 Interne Aufsplittung von TAXRES in zwei Teile und die entsprechenden Schnittstellen-Erweiterungen

Auf der einen Seite soll TAXON in der Lage sein, die Ressourcen-Informationen zur Berechnung der Taxonomie in der T-Box und für den Realisierungsprozeß in der A-Box verwenden zu können. Dazu braucht es die Unterstützung eines Konkreten Bereiches, da TAXON ansonsten die Konkreten Zahlenwerte bei den Ressourcen-Angaben nicht verarbeiten könnte.

Auf der anderen Seite muß die Steuerungseinheit CONTROL wissen, ob aufgrund von Fehlentscheidungen und dadurch resultierenden nicht erfüllbaren Ressourcen-Bilanzierungs-Bedingungen im Konfigurierungsprozeß Backtracking eingeleitet werden muß oder nicht. Falls eine oder mehrere Bilanzierungs-Bedingungen zwar derzeit nicht erfüllt, aber noch erfüllbar sind, dann muß CONTROL mitgeteilt werden, wie es die A-Box entsprechend den Ressourcen-Defiziten vervollständigen kann.

Daher bietet es sich an, TAXRES in zwei Teile aufzusplitten. Der eine Teil verwaltet alle in Form von Konkreten Prädikaten gemachten Ressourcen-Angaben. Der andere Teil ist für die Bilanzierung der Ressourcen zuständig. Die entsprechenden prinzipiellen Schnittstellen von TAXRES sind in Bild 4.2 dargestellt. Das darauf folgende Bild 4.3 zeigt in einer detaillierteren Form, wie über diese Schnittstellen der Informationsaustausch zwischen TAXRES, TAXON und CONTROL erfolgen könnte[26].

---

25. Ein Defizit bei der Environment-Ressource Geld kann z. B. nicht beseitigt werden.
26. vgl. mit Bild 2.1

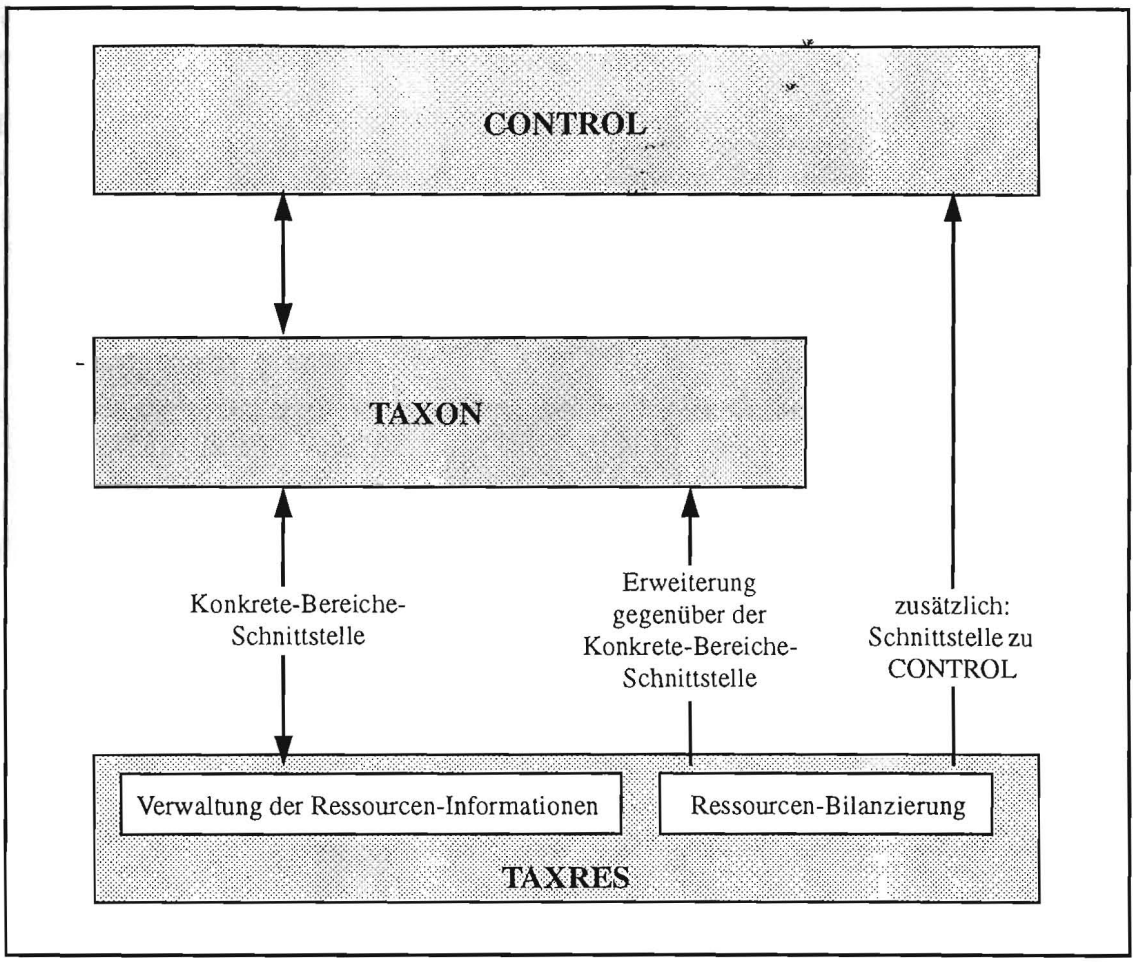


Bild 4.2: Die prinzipiellen Schnittstellen zwischen TAXRES, TAXON und CONTROL

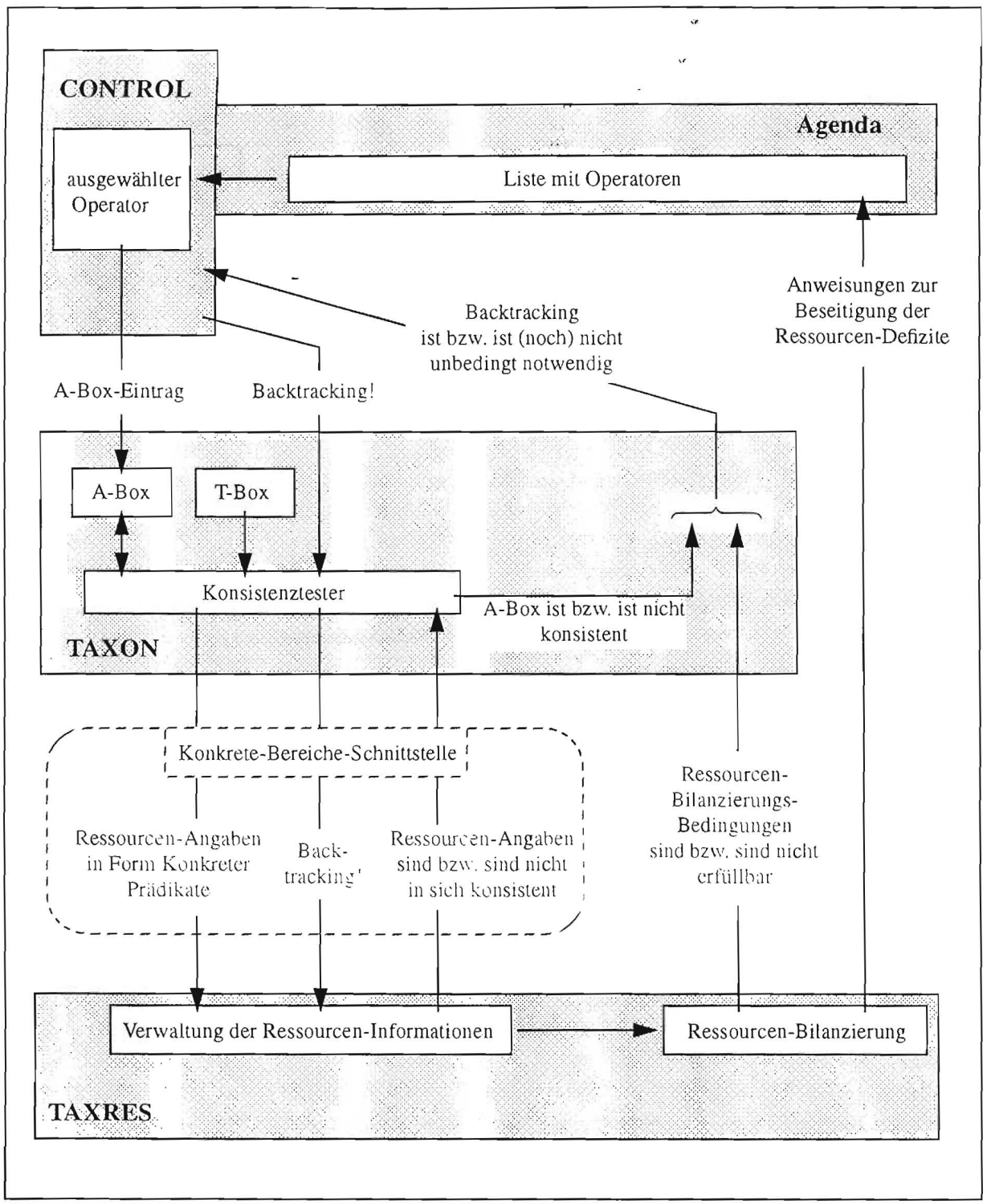


Bild 4.3: Der Informationsfluß zwischen TAXRES, TAXON und CONTROL

4.2.4 Die Konkreten Prädikate

Über die Schnittstelle der Konkreten Prädikate soll TAXRES mitgeteilt werden, von welcher Ressource wieviel verbraucht bzw. zur Verfügung gestellt wird.

Problem:

Laut der formalen Definition eines zulässigen Konkreten Bereichs in Kapitel 2.1.2 muß zu jedem erlaubten Konkreten Prädikat auch die zugehörige Negation existieren.
Wie löst man dieses Problem der Negierung? Welche Bedeutung steckt z. B. hinter der Aussage, daß von einer Ressource eine bestimmte Menge nicht verbraucht wird?

Erster Lösungsansatz:

Man modifiziert die Konkrete-Bereiche-Schnittstelle derart, daß negative Prädikate nicht mehr unbedingt benötigt werden. Dafür bräuchte man dann aber ein formal wohl fundiertes Modell. Das ist hier aber nicht das einzige Problem:

Aufgrund der fehlenden Negation, können die Ressourcen-Informationen nicht zur Berechnung der Taxonomie (→ *Subsumptionsgraph*) verwendet werden. Denn laut Definition wird ein Konzept C von einem Konzept D *subsumiert*, wenn kein Modell für $C \wedge (\neg D)$ existiert.

Je mehr Informationen über ein Individuum in der A-Box bekannt sind, desto spezielleren Konzepten kann es zugeordnet werden. Auf diese Art erfolgt ein allmähliches Einsinken eines Individuums im Subsumptionsgraphen (*Realisierung*). Wenn aber die Ressourcen-Informationen nicht im Subsumptionsgraphen enthalten sind, können sie auch nicht bei der Realisierung mit einfließen. Es ist aber zu erwarten, daß sich einzelne Individuen zum Teil nur aufgrund unterschiedlicher Ressourcen-Angaben unterscheiden.

Bei einer Konfigurierung ist die Realisierung für eine ständige Einschränkung des verbleibenden Lösungsraumes äußerst wichtig (vgl. Kapitel 1.4). Daher sollte bei den Konkreten Prädikaten nach Möglichkeit nicht auf die Negation verzichtet werden.

Zweiter Lösungsansatz:

Bei diesem Ansatz wird die Syntax und Semantik der Konkreten Prädikate so gewählt, daß sich die Negation darin auf natürliche Weise integrieren läßt:

Wenn eine Komponente K von einer Ressource R eine gewisse Menge (*Ressource-Menge*) benötigt bzw. zur Verfügung stellt, dann wird diese Ressource-Menge durch eine Variable $RV_{K,R}$ symbolisiert. Eine solche Variable wird als *Ressource-Variable* bezeichnet. Um den *Wertebereich* dieser Variablen zu beschreiben, werden Konkrete Prädikate verwendet, die im wesentlichen den mathematischen Prädikaten $<$, $\leq$, $=$, $\neq$, $\geq$ und $>$ entsprechen. Hierbei ist zu beachten, daß zu jedem dieser Prädikate auch die Negation vorhanden ist, wodurch das Negations-Problem gelöst ist.

Beispiel 4.1:

Wenn man fordert, daß das Prädikat ($RV_{K,R} = 5$) gültig ist, würde man den Wertebereich der Ressource-Variable $RV_{K,R}$ auf den Wert 5 einschränken. Damit könnte man dann angeben, daß die Komponente K von der Ressource R die Menge 5 fordert bzw. zur Verfügung

stellt.
Den gleichen Sachverhalt könnte man auch z. B. durch eine der folgenden Konjunktionen[27] ausdrücken: $(RV_{K,R} \leq 5) \wedge (RV_{K,R} \geq 5)$ oder $\neg(RV_{K,R} > 5) \wedge \neg(RV_{K,R} < 5)$.

Der Wertebereich einer Ressourcen-Variable muß aber nicht unbedingt auf genau einen Wert eingeschränkt werden, wie das nächste Beispiel zeigt:

Beispiel 4.2:

In einer PC-Domäne könnte man z. B. ein Oberkonzept für alle IDE-Festplatten modellieren. Dieses Oberkonzept hätte dann für die Ressource Festplatten-Speicher einen Wertebereich zwischen 0 und 528 MB[28]. Wenn eine Instanz von diesem Oberkonzept in die A-Box eingetragen wird, symbolisiert dieses Individuum dann eine IDE-Festplatte mit einer bestimmten Speicherkapazität X, wobei X zwar eine feste, aber zunächst noch unbekannte Zahl aus dem Intervall (Wertebereich) $0 < X \leq 528$ ist.

Nach diesem Ansatz kann TAXRES prinzipiell als vollständiger Konkreter Bereich an TAXON angeschlossen werden. Dadurch, daß auch die Negation vorhanden ist, können Ressourcen-Informationen sowohl zum Aufbau des Subsumptionsgraphen als auch für das taxonomische Reasoning in der A-Box verwendet werden.

Jetzt bleibt noch zu klären, wie prinzipiell eine partielle Konfiguration durch Beseitigung von behebbaren Ressourcen-Defiziten vervollständigt werden kann.

4.2.5 Beseitigung von Ressourcen-Defiziten

Erster Ansatz zur Beseitigung von Ressourcen-Defiziten:

Sobald bei einer Ressource ein Defizit erkannt wird, werden alle Möglichkeiten (Alternativen) bestimmt, die dieses Defizit beheben. Hierbei besteht jede einzelne Alternative aus einer beliebigen Menge von Komponenten-Ausprägungen (d. h. Instanzen terminaler Konzepte). Bei einem Speicherdefizit könnte eine Alternative z. B. aus zwei Festplatten von einem ganz bestimmten Typ bestehen. Aufgrund irgend eines Bewertungsmaßstabes wird dann aus der Menge aller Alternativen eine möglichst gute Alternative ausgewählt und in die A-Box instanziiert[29].

---

27. Ein Konkreter Bereich überprüft automatisch, ob die Konjunktion aller übermittelten Konkreten Prädikate erfüllbar ist oder nicht. Daher braucht man eine Konjunktion von Prädikaten nicht explizit zu formulieren.
28. Randbemerkung:
DOS kann bei IDE-Festplatten nur max. 1024 Zylinder, max. 16 Köpfe und max 63 Sektoren á 512 Byte ansprechen. Daraus ergibt sich diese Maximalkapazität von 1024 * 16 * 63 * 512 Byte =528 MB.
29. Rein ressourcen-orientierte Konfigurierungssysteme (wie z. B. das COSMOS-System) arbeiten nach diesem oder einem ähnlichen Grundprinzip.

Problem bei diesem ersten Ansatz:

Zur Beseitigung eines Ressource-Defizites muß man sich hier jeweils ohne Wenn und Aber für eine ganz konkrete Alternative entscheiden. Dadurch muß man man sich während der Konfigurierung oft unnötigerweise zu früh auf bestimmte Komponenten festlegen. Mit "zu früh festlegen" ist hier gemeint, daß Komponenten aus dem Katalog zu einem Zeitpunkt ausgewählt werden, zu dem noch nicht alle Anforderungen an die auszuwählende Komponente bekannt sind. Die Folge ist, daß es häufig aufgrund dieser Unwissenheit zu Fehlentscheidungen kommt, die später durch teures Backtracking wieder revidiert werden müssen.

Leider ist es aber bei einer Konfigurierung nicht immer möglich, mit jeder Entscheidung solange zu warten, bis man wirklich alle dafür notwendigen Informationen zur Verfügung hat. Dennoch kann man gemäß einer *Least-Commitment-Strategie* (vgl. Kapitel 1.4) versuchen, wenn man schon eine Entscheidungen treffen muß, sich dabei dann unter Ausnutzung des terminologischen Basissystems möglichst wenig festzulegen. Wie sich diese Strategie verwirklichen läßt, wird im nächsten Abschnitt erklärt.

Alternativer Ansatz:

Sobald ein Ressource-Defizit erkannt wird, wird das Oberkonzept aller Konzepte, die irgendetwas von dieser fehlenden Ressource zur Verfügung stellen, instanziiert. Ein solches Oberkonzept wird im folgenden als *Ressource-Oberkonzept* bezeichnet. Da es theoretisch beliebig viel von der Ressource zur Verfügung stellen kann, ist das Ressource-Defizit damit zunächst beseitigt. Im Laufe des Konfigurierungsprozesses werden alle Instanzen von Oberkonzepten (durch den Realisierungsprozeß in der A-Box) allmählich immer weiter eingeschränkt (wodurch sich die Zahl der noch möglichen Komponenten-Ausprägungen ständig reduziert) bis schließlich nur noch Instanzen terminaler Konzepte vorhanden sind. Durch die Einschränkung eines Ressource-Oberkonzeptes können wiederum neue Ressourcen-Defizite entstehen, die dann wieder nach dem gleichen Verfahren beseitigt werden.

Über diesen Ansatz lassen sich die Wissensmodellierung und die zugehörigen Inferenzen des ressourcen-orientierten und des struktur-orientierten Konfigurierungsansatzes elegant miteinander kombinieren!

5. Das Implementierungskonzept zur Integration der ressourcen-orientierten Techniken in TOOCON

Zur Integration ressourcen-orientierter Techniken in den TOOCON-Werkzeugkasten wird dieser um das neue Tool *TAXRES* erweitert. TAXRES besteht aus zwei Teilen:

- Auf der einen Seite ist es ein Konkreter Bereich[30], der für die Verwaltung und Konsistenz-Überprüfung[31] aller verwendeten Ressourcen zuständig ist.
  Dieser Teil von TAXRES basiert auf dem Ansatz aus Kapitel 4.2 und wird in Abschnitt 5.1 in ausführlicher Form beschrieben.
- Auf der anderen Seite hat TAXRES zusätzlich die Aufgabe, *behebbare Ressourcen-Defizite* durch Überprüfung aller ressourcen-orientierter Integritätsbedingungen zur erkennen und sie in Zusammenarbeit mit dem Steuerungsmodul CONTROL zu beseitigen (→ ressourcen-orientiertes Konfigurieren).
  Die Erweiterungen, die dazu gegenüber der Konkrete-Bereiche-Schnittstelle notwendig sind, werden in Abschnitt 5.2 behandelt.

5.1 Der neue Konkrete Bereich TAXRES

Jede von einer Komponente bereitgestellte bzw. geforderte Ressource-Menge wird in TAXRES jeweils durch eine eigene *Ressource-Variable* repräsentiert (vgl. Kapitel 4.2.4). Diese Variablen arbeiten über dem "Concrete Domain" der reellen Zahlen.

TAXRES wird an TAXON als Konkreter Bereich angeschlossen. Daher kann der Informationsaustausch zwischen einer Komponente aus TAXON und einer zugehörigen Ressource-Variable in TAXRES nicht direkt, sondern nur indirekt über Konkrete Prädikate erfolgen (siehe Bild 5.1).

30. siehe Kapitel 2.1.2
31. An dieser Stelle wird nur überprüft, ob die Ressourcen-Angaben in sich konsistent sind (aber noch nicht die ressourcen-orientierten Integritätsbedingungen!).
    Eine Festplatte kann z. B. nicht gleichzeitig als Wert für die maximale Speicherkapazität 500 MB und 700 MB haben. Solche Ressourcen-Angaben wären in sich inkonsistent.

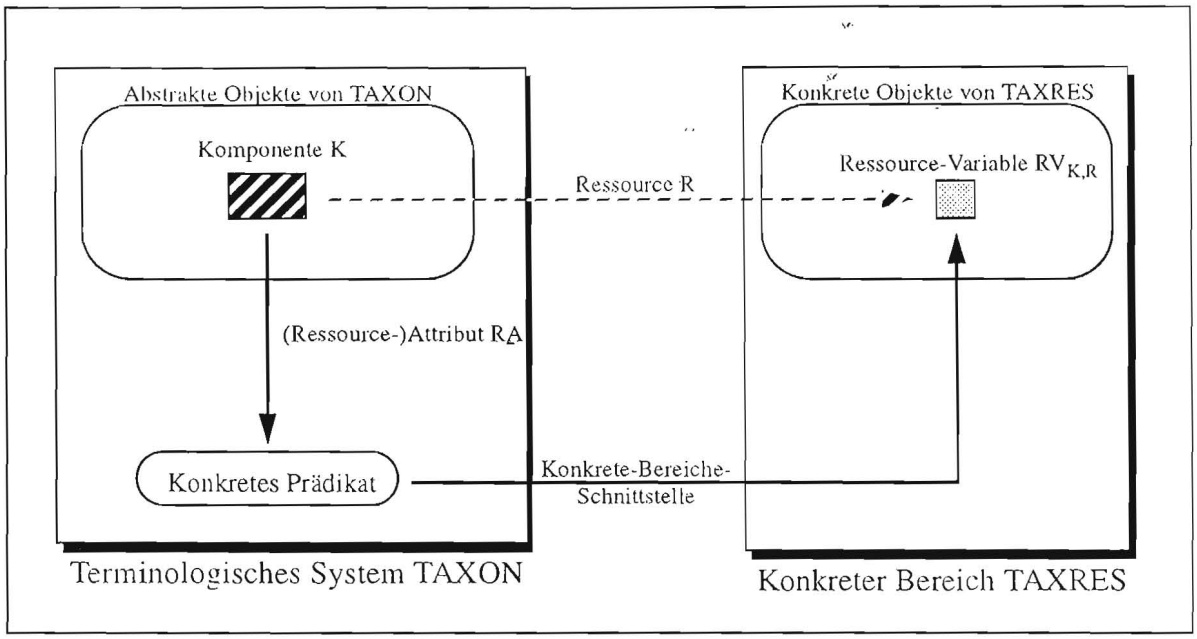


Bild 5.1: Veranschaulichung, wie einer Komponente aus TAXON indirekt über Konkrete Prädikate eine Ressource-Menge aus TAXRES zugewiesen wird

Um auszudrücken, daß ein Konzept K (aus TAXON) von einer Ressource R eine gewisse Menge zur Verfügung stellt (bzw. fordert), wird K mit Hilfe eines Attributes RA (*Ressource-Attribut*) einem Konkreten Prädikat aus TAXRES zugeordnet. Dieses Konkrete Prädikat beschreibt als Constraint den Wertebereich der entsprechenden Ressource-Variable $RV_{K,R}$ in TAXRES[32].

Defaultmäßig wird jeder Ressource-Variable bei ihrer Initialisierung als Wertebereich das (reelle) Zahlen-Intervall von $-\infty$ bis $+\infty$ zugeordnet. Mit Hilfe Konkreter Prädikate, die im wesentlichen den mathematischen Prädikaten $<$, $\leq$, $=$, $\neq$, $\geq$ und $>$ entsprechen, kann TAXON den Wertebereich von Ressource-Variablen einschränken oder sogar auf einen Wert festlegen[33].

Solange jede einzelne Ressource-Variable in TAXRES noch mindestens eines Wert annehmen kann (also keinen leeren Wertebereich hat), ist die Konjunktion aller bisher von TAXON übermittelten Konkreten Prädikate erfüllbar. Sobald das nicht mehr zutrifft, wird das als Inkonsistenz angesehen und TAXON aufgefordert, Backtracking einzuleiten.

5.1.1 Die Syntax der Konkreten Prädikate von TAXRES

Die Konkreten Prädikate werden als Schnittstelle zwischen TAXON und TAXRES verwendet. Damit TAXRES die einzelnen Ressourcen korrekt verwalten kann, benötigt es für jede einzelne Ressource-Variable neben dem Wertebereich noch weitere Informationen. Zum einen muß es den zugehörigen Ressource-Namen kennen und zum anderen muß es wissen, ob es sich

32. siehe Bild 4.1
33. siehe Beispiel 4.1

hierbei um einen Ressourcen-Verbraucher (*resource-consumer*) oder um einen Ressourcen-Erzeuger (*resource-supplier*) handelt. Diese notwendigen Zusatzinformationen werden ebenfalls über die Schnittstelle der Konkreten Prädikate übermittelt.

Zur Unterscheidung von Prädikaten aus anderen Konkreten Bereichen (wie z. B. EPILYTIS) werden die Prädikat-Symbole von TAXRES im Vergleich zu den mathematischen Prädikat-Symbolen grundsätzlich mit einem nachgestellten Ausrufezeichen (Suffix) notiert. Zusätzlich wird die Formulierung von Intervallen unterstützt.

Die Konkreten Prädikate von TAXRES werden nach folgenden Syntaxregeln formuliert:

(cpred ⟨*Prädikat-Name*⟩ **(:taxres** **(** ⟨*Ressource-Variable*⟩**)**
(⟨*Ressource-Name*⟩
⟨*Supplier/Consumer*⟩
⟨*Prädikat-Ausdruck*⟩ **)))** , mit

⟨*Supplier/Consumer*⟩ ::= **Supplier** | **Consumer** | **NIL**

⟨*Prädikat-Ausdruck*⟩ ::= **(not** ⟨*Prädikat-Ausdruck*⟩ **)** |
(⟨*Ressource-Variable*⟩ *(***>!** | **>=!** | **=!** | **<=!** | **<!***)* ⟨*Zahl*⟩**)** |
(⟨*Ressource-Variable*⟩ **:in** ⟨*Intervall-Ausdruck*⟩**)**

⟨*Intervall-Ausdruck*⟩ ::= **(**⟨*untere Intervall-Grenze*⟩ ⟨*obere Intervall-Grenze*⟩**)**

⟨*untere Intervall-Grenze*⟩ ::= **(***(***>!** | **>=!***)* ⟨*Zahl*⟩**)** | **NIL**

⟨*obere Intervall-Grenze*⟩ ::= **(***(***<=!** | **<!***)* ⟨*Zahl*⟩**)** | **NIL**

Zusatzbemerkungen zu diesen Syntaxregeln:

- Bei der Angabe von ⟨*Supplier/Consumer*⟩ kann auch das Symbol *NIL* stehen[34]. In diesem Fall wird es (zunächst noch) offen gelassen, ob es sich (bei der zugehörigen Komponente) um einen Ressourcen-Erzeuger oder um einen Ressourcen-Verbraucher handelt. Dabei ist aber zu beachten, daß eine Ressource-Variable im Bilanzierungsprozeß erst ab dem Zeitpunkt berücksichtigt werden kann, ab dem diese für die Bilanzierung notwendige Information bekannt ist.
- Wenn bei einer Intervall-Grenze das Symbol *NIL* steht, dann heißt das, daß auf dieser Seite vom Intervall keine Grenze vorhanden ist. Das ist gleichbedeutend mit -∞ (bei einer unteren Intervall-Grenze) bzw. +∞ (bei einer oberen Intervall-Grenze).

Im folgenden werden obige Syntax-Regeln anhand einiger Beispiele verdeutlicht:

Beispiel 5.1:

(cpred Strom-Minus-ge-10 (:taxres (?X) Strom Consumer (?X >=! 10)))

Dieses Konkrete Prädikat steht für alle Strom-Verbraucher, die als Minimum 10 Einheiten der Ressource Strom benötigen.

34. Der Spezialfall, daß man bei ⟨Supplier/Consumer⟩ auch das Symbol NIL verwenden kann, wird hier nur der Vollständigkeit halber erwähnt. Diese Möglichkeit wird z. B. bei der automatischen Definition des speziellen Konzeptes ENVIRONMENT (siehe Abschnitt 5.2.2) benötigt.

Beispiel 5.2:

(cpred HD-Speicher-IDE (:taxres (?X) HD-Speicher Supplier (?X :in ((> 0) (<= 528)))))

Da jede IDE-Festplatte von der Ressource HD-Speicher zwischen 0 und 528 MB zur Verfügung stellt, könnte man dieses Konkrete Prädikat für die Modellierung eines Oberkonzeptes aller IDE-Festplatten verwenden.

Beispiel 5.3:

(cpred Strom-Minus-ge-10-le-nil (:taxres (?X) Strom Consumer (?X :in ((>=! 10) NIL)))))

Dieses Konkrete Prädikat ist nur eine andere Darstellung von dem in Beispiel 5.1, indem hier die obere Intervall-Grenze des Wertebereichs explizit hingeschrieben wurde.

5.1.2 Bemerkungen zur Semantik der Ressource-Variablen

Der Wertebereich einer Ressource-Variablen RV gibt an, wieviel von dieser Ressource durch die zugehörige Komponente K (also dem Konzept aus TAXON) bereitgestellt bzw. gefordert werden kann. Prinzipiell würde es ausreichen, wenn eine Ressource-Variable nicht einen ganzen Wertebereich, sondern nur einen bestimmten Wert hätte. Um zu modellieren, daß eine Komponente von einer bestimmten Ressource *als Minimum* die Menge X fordert oder *maximal* die Menge X zur Verfügung stellt, würde man den Wertebereich der zugehörigen Ressource-Variablen RV auf einen festen Wert einschränken: (RV =! X) , wobei X irgendeine feste Zahl ist.

Im Gegensatz zu herkömmlichen reinen ressourcen-orientierten Konfigurierungssystemen hat man aber hier durch die Integration in den TOOCON-Werkzeugkasten ein terminologisches System (TAXON) zur Verfügung. Das bedeutet, daß man im Komponentenkatalog (T-Box) sowohl Komponenten-Oberbegriffe (Oberkonzepte) als auch Komponenten-Ausprägungen (terminale Konzepte) definieren kann. Diese Option bietet zwei große Vorteile: Erstens kann man damit den Komponentenkatalog besser strukturieren. Zweites hat man bei der Generierung der Konfigurationslösung die Möglichkeit, zunächst einmal nur Oberkonzepte in die A-Box einzusetzen und diese dann später weiter einzuschränken. Dadurch braucht man sich nicht zu früh auf ganz bestimmte Komponenten festzulegen und kann so im Konfigurationsprozeß viele aufwendige Backtracking-Operationen vermeiden. Aus diesem Grund gibt es hier bei der Angabe von Ressource-Mengen die Möglichkeit, Wertebereiche zu verwenden[35].

Prinzipiell kann man bei jeder Ressource-Mengenangabe einen Mindest-Wert (z. B. ?Strom ≥! 5), einen Maximal-Wert (z. B. ?Strom ≤! 10) oder einen exakten Wert (z. B. ?Strom =! 7) angeben. Bei Ressourcen-Verbrauchern sollte man aber auf jeden Fall immer entweder einen Mindest-Wert (bei der Modellierung von Oberkonzepten) oder der exakte Wert (bei terminalen Konzepten) verwenden. Dagegen sollte bei Ressourcen-Erzeugern stets entweder ein Maximal-Wert (bei der Modellierung von Oberkonzepten) oder den exakten Wert (bei terminalen Konzepten) angegeben werden. Denn bei der Beurteilung, ob eine Ressource ausbilanziert ist oder nicht, werden nur die minimal geforderten Mengen mit den maximal zur Verfügung gestellten Mengen verglichen[36]. Dennoch kann man natürlich zur besseren Strukturierung der T-Box, auch bei Ressourcen-Verbrauchern Maximal-Werte bzw. bei Ressourcen-Erzeugern Minimal-Werte (zusätzlich) angeben.

---

35. siehe Beispiel 4.2
36. Die Bilanzierung der Ressourcen wird in Abschnitt 5.2.3 behandelt.

5.1.3 Watchdogs

Im TOOCON-Werkzeugkasten müssen Konkrete Prädikate nicht unbedingt als Constraints verwendet werden. Mit Hilfe von *Watchdogs*[37] besteht auch die Möglichkeit, Konkrete Prädikate einfach nur zu überprüfen, ohne aber deren Erfüllbarkeit explizit zu fordern. Daher wird hierbei (im Gegensatz zu dem oben beschriebenen Verfahren) der Wertebereich von Ressource-Variablen nicht verändert.

Watchdogs können beispielsweise zum Überprüfen von Bedingungen in Vorwärtsregeln eingesetzt werden: Wenn feststeht, daß ein Konkretes Prädikat mit Sicherheit gültig bzw. ungültig ist, dann kann die Regel feuern. Da der Wertebereich von Ressource-Variablen später noch weiter eingeschränkt werden könnte, kann es vorkommen, daß die Gültigkeit eines Konkreten Prädikates nicht direkt entscheidbar ist. In solchen Fällen wird der Watchdog einfach nur in TAXRES eingetragen. Sobald dann TAXRES in der Lage ist, eine Entscheidung zu treffen, führt es die entsprechende im Watchdog angegebene Reaktionsanweisung aus.

5.2 Erweiterungen für die ressourcen-orientierte Konfigurierung

Um ressourcen-orientiertes Konfigurieren zu ermöglichen, benötigt TAXRES Eigenschaften und Fähigkeiten, die über die eines Konkreten Bereiches hinausgehen. Wieso welche Erweiterung der Konkrete-Bereiche-Schnittstelle benötigt wird, wird in nachfolgenden Unterkapiteln näher beschrieben.

5.2.1 Deklarierung der verwendeten Ressourcen

TAXRES unterscheidet zwei Arten von Ressourcen: *Standard-Ressourcen* und *Environment-Ressourcen* (siehe Kapitel 3.1.1).

Für eine *Standard-Ressource* gibt es in der T-Box Komponenten, die eine gewisse Ressourcen-Menge davon zur Verfügung stellen. Wenn von einer solchen Ressource irgendwann im Laufe des Konfigurierungsprozesses eine gewisse Menge fehlt, dann kann man ein solches Defizit immer beseitigen, indem man z. B. einfach in die A-Box eine genügend große Anzahl von Komponenten einträgt, die diese Ressource bereitstellen.

Im Gegensatz dazu werden die *Environment-Ressourcen* ausschließlich in einer bestimmten festen Menge von außen zur Verfügung gestellt. Bei diesen Ressourcen gibt es daher keine Möglichkeit, ein einmal entstandenes Ressourcen-Defizit irgendwie auszugleichen. Deshalb müßte in einem solchen Fall von der Steuerungskomponente CONTROL unbedingt Backtracking eingeleitet werden.

37. siehe Spezifikation der Schnittstelle zwischen TAXON und Konkreten Bereichen in [JW93]

Damit TAXRES weiß, von welchem Typ jede einzelne Ressource ist, müssen alle verwendeten Ressourcen vor ihrem Gebrauch deklariert. Dazu werden die beiden folgenden Befehle verwendet:

```
(Declare-Standard-Resources {⟨Ressource-Name⟩}+)
(Declare-Environment-Resources {⟨Ressource-Name⟩}+)
```

Automatisch generierte Ressource-Attribute:

Um einer Komponente aus TAXON eine Ressource-Menge zuordnen zu können, wird stets ein (Ressource-)Attribut benötigt (siehe Bild 5.1). Damit der Benutzer ein solches Attribut nicht selbst zu deklarieren braucht, wird als Nebeneffekt bei der Ressourcen-Deklaration automatisch zu jeder Ressource ein gleichnamiges Ressource-Attribut deklariert.

5.2.2 Das spezielle Konzept ENVIRONMENT

Zur Angabe von Environment-Ressourcen wird das spezielle mit dem Befehl

```
(DEFINE-ENVIRONMENT-CONC)
```

generierte Konzept *Environment* zusammen mit den entsprechenden Ressource-Attributen verwendet. Diese Angaben gehören zur Aufgabenspezifikation und können durch den *SOLVE*- oder durch den *MASSE*-Befehl[38] gemacht werden.

Beispiel 5.4:

```
(solve ( (Env :in Environment)
         (Env = Geld => ?Geld)
         (Geld :Supplier (?Geld =! 1000))
         ....
         ) )
```

In dieser Beschreibung einer Konfigurierungsaufgabe wird u. a. spezifiziert, daß die Umgebung des technischen Systems von der Ressource *Geld* die Menge 1000 zur Verfügung stellt.

Angaben zum Verbrauch einer Environment-Ressource erfolgen ausschließlich als statisches Wissen bei den einzelnen Komponenten in der T-Box. Im Gegensatz dazu, werden die bereitgestellten Environment-Ressourcen nur bei der Spezifikation der jeweiligen Konfigurierungsaufgabe angegeben. Damit der Benutzer bei der Formulierung einer Aufgabenspezifikation Angaben zu Environment-Ressourcen auch weglassen kann, geht TAXRES defaultmäßig davon aus, daß von solchen Ressourcen genügend viel zur Verfügung gestellt wird.

38. Die Befehle SOLVE und MASSE sind in [Wache94] beschrieben.

5.2.3 Die Bilanzierung der Ressourcen

Immer wenn die Konsistenz der A-Box durch das Steuerungsmodul CONTROL überprüft wird, wird in TAXRES ein *Resource-Check* durchgeführt. Dabei wird jeweils überprüft, ob die folgende implizite Bilanzierungs-Bedingung für jede einzelne Ressource R erfüllt ist oder nicht:

> Die *insgesamt maximal zur Verfügung stehende Menge* (Max-Plus) der Ressource R muß größer oder gleich sein als die *insgesamt als Minimum geforderte Menge* (Min-Minus) von der Ressource R.

Hierbei wird Max-Plus berechnet als die Summe der Maximal-Werte der Wertebereiche aller Ressource-Variablen, die eine Ressource-Bereitstellung von der Ressource R symbolisieren. Analog dazu ist Min-Minus die Summe der Minimal-Werte der Wertebereiche aller Ressource-Variablen, die eine Ressource-Forderung von der Ressource R repräsentieren.

Falls dabei jeweils (Min-Minus ≤ Max-Plus) ist, dann ist alles in Ordnung und die Ressource wird als ausbilanziert betrachtet. In dem Fall ist es nämlich theoretisch noch möglich, die noch vorhandenen Oberkonzepte so weiter einzuschränken, daß am Ende der Konfiguration[39] tatsächlich mindestens soviel von der Ressource bereitgestellt wird, wie gefordert ist. Ansonsten liegt ein Ressource-Defizit vor und TAXRES muß entsprechende Gegenmaßnahmen einleiten.

Bei einem Ressource-Check werden zuerst werden alle **Environment-Ressourcen** der Reihe nach untersucht. Falls hierbei festgestellt wird, daß von einer solchen Ressource noch eine gewisse Menge fehlt, wird CONTROL aufgefordert, Backtracking einzuleiten. In der T-Box gibt es nämlich per Definition keine Komponenten, die eine Environment-Ressource zur Verfügung stellen können.

Anschließend werden alle **Standard-Ressourcen** überprüft. Falls dabei von der Ressource R ein Defizit erkannt wird, dann wird bei CONTROL nachgefragt, ob auf der Agenda (aufgrund eines früheren Ressource-Checks) bereits ein Operator zur Beseitigung dieses Ressourcen-Defizites vorhanden ist und nur noch nicht abgearbeitet wurde. Falls dem nicht so ist, dann muß TAXRES geeignete Maßnahmen einleiten, um dieses Ressource-Defizit zu beseitigen. Dazu gibt es prinzipiell folgende Möglichkeiten:

a) Einschränkung eines geeigneten bereits in der A-Box vorhandenen Oberkonzeptes

b) Eintragung einer oder mehrerer neue Komponenten in die A-Box

Beispiel 5.5:

In der A-Box sei eine Instanz des Oberkonzeptes Motherboard. Desweiteren sei angenommen, daß beim Ressourcen-Bilanzieren festgestellt wurde, daß ein SCSI-Anschluß (d. h. die Menge 1 von der Ressource SCSI-Anschlüsse) fehlt. Um dieses Defizit zu beseitigen, gibt es hier die beiden folgenden Varianten:

---

39. Am Ende der Konfiguration stehen in der A-Box nur noch terminale Konzepte, also nur noch Komponenten-Ausprägungen und keine Komponenten-Oberbegriffe.

a) Man könnte das Oberkonzept "Motherboard" zu einem Oberkonzept "Motherboard-mit-integriertem-SCSI-Controller" einschränken.

b) Alternativ dazu könnte man einen separaten SCSI-Controller der A-Box hinzufügen.

Welche der beiden Varianten für die jeweilige Konfiguration besser ist, kann man für den allgemeinen Fall nicht sagen. Aus dem Grund läßt sich TAXRES beide Möglichkeiten offen:

Vorgehensweise von TAXRES, um ein Defizit bei einer Standard-Ressource zu beseitigen:

TAXRES fordert durch den Eintrag eines entsprechenden Operators auf die Agenda das Steuerungsmodul CONTROL auf, eine neue Komponenten-Variable ?K in die A-Box einzutragen, die folgende Bedingungen erfüllt:

(1) Sie muß eine gewisse Menge von der fehlenden Ressource R zur Verfügung stellen.
Zu diesem Zweck wird für jede deklarierte Standard-Ressource *R* automatisch ein Konzept mit dem Namen ⟨*R*⟩-*Supplier-Object* generiert, das Oberkonzept aller Konzepte ist, die irgendeine Menge von der Ressource *R* zur Verfügung stellen.
Um zu gewährleisten, daß ?K eine gewisse Menge von R zur Verfügung stellt, wird jetzt in diesem Agenda-Operator gefordert, daß ?K eine Instanz von ⟨R⟩-Supplier-Object sein muß.

(2) Sie muß ungleich aller zum Zeitpunkt der Bilanzierung in der A-Box vorhandenen Konzepte sein, die etwas von der Ressource R bereitstellen. Das bedeutet, daß die zugehörige Ressource-Variable (bzgl. der Ressource R) zu den entsprechenden bisher eingetragenen Ressource-Erzeuger-Variablen ungleich sein muß.

Solche von TAXRES auf die Agenda eingetragenen Operatoren weisen prinzipiell folgende Struktur auf:

```
(AND   (?K :in R-Supplier-Object)
       (?K = R => ?RV)
       (?RV :unequal ?RV1) (?RV :unequal ?RV2) ... (?RV :unequal ?RVn)
```

Dabei ist ?K eine neue Komponenten-Variable, R die Ressource mit dem Defizit, ?RV eine neue Ressource-Variable. $?RV_1$, $?RV_2$, ... , $?RV_n$ sind die Ressource-Variablen der Konzepte, die bisher eine jeweils gewisse Menge von der Ressource R zur Verfügung gestellt haben.

Diese neue Komponenten-Variable wird irgendwann von CONTROL in die A-Box eingetragen und im Laufe des weiteren Konfigurierungsprozesses durch den SKOLEMIZER (siehe Abschnitt 2.1.3) mit einer Komponenten-Konstanten unifiziert. Bei dieser Unifikation wird die

Entscheidung[40] gefällt, ob die Komponenten-Variable mit einem bereits vorhandenen Konzept unifiziert (und dieses dadurch evtl. eingeschränkt) wird (Fall a) oder ob aus der Komponenten-Variable eine neue Komponenten-Konstante gemacht wird (Fall b).

Das in die A-Box eingetragene Ressource-Oberkonzept stellt zunächst beliebig viel von der betreffenden Ressource zur Verfügung, wodurch diese automatisch ausbilanziert ist. Infolge verschiedener Realisierungsprozesse während der Konfiguration werden alle Oberkonzepte immer mehr eingeschränkt und schließlich zu einem terminalen Konzept gemacht. Dadurch können erneut Ressourcen-Verletzungen auftreten und das Bilanzierungs-Verfahren beginnt von vorne.

5.3 Verwendung von Strategien und Heuristiken

Während der Konfigurierung hat CONTROL wichtige Entscheidungen zu treffen, die einen sehr großen Einfluß auf die Dauer des Konfigurierungsprozesses (→ Effizienz) und die Qualität der gefundenen Lösung haben:

(1) In welcher Reihenfolge werden die Operatoren auf der Agenda abgearbeitet?

(2) Der SKOLEMIZER ist dafür verantwortlich, daß im Laufe des Konfigurierungsprozesses aus jeder Komponenten-Variable eine Komponenten-Konstante gemacht wird. Um eine Komponenten-Variable zu skolemisieren, ermittelt er alle für sie in Frage kommenden Komponenten-Konstanten. Anschließend fordert er das Steuerungsmodul CONTROL (durch den Eintrag eines entsprechenden OR-Operators in die Agenda) auf, die Komponenten-Variable mit einer dieser Komponenten-Konstanten zu unifizieren. Dabei muß dann CONTROL nach irgendeinem Kriterium genau eine Komponenten-Konstante auswählen.

(3) Alle Komponenten-Oberbegriffe (Oberkonzepte) müssen irgendwann zu Komponenten-Ausprägungen (terminale Konzepte) realisiert werden. Für diese Aufgabe ist primär der REALIZER zuständig. Um ein Oberkonzept zu realisieren, bestimmt er von diesem alle noch möglichen Komponenten-Ausprägungen. Danach schreibt er einen OR-Operator auf die Agenda, der CONTROL auffordert, das Oberkonzept zu einem dieser Terminal-Konzepte zu realisieren. Ähnlich wie bei der Skolemisierung muß sich CONTROL auch hier für genau eine Komponente entscheiden.

Um diese Entscheidungen möglichst intelligent treffen zu können, benötigt CONTROL entsprechendes Wissen. Dieses Wissen besteht hauptsächlich aus Heuristiken. Wissen zur Optimierung der Abarbeitungsreihenfolge der Operatoren auf der Agenda wird in Abschnitt 5.3.1 behandelt. Neben solchem statischen Strategie-Wissen wäre es sehr sinnvoll, wenn CONTROL während der Konfigurierung auf interne Informationen bestimmter Konkreter Bereiche zugreifen könnte. Denn woher soll CONTROL z. B. bei der Realisierung wissen, welche Komponenten-Ausprägung bezüglich der Ressourcen-Bilanzen besser oder schlechter geeignet ist. Wie CONTROL bei der Komponenten-Auswahl durch Konkrete Bereiche unterstützt werden kann, wird in Abschnitt 5.3.2 beschrieben.

---

40. Diese Entscheidung wird letztlich durch CONTROL in Abhängigkeit vom Steuerungswissen getroffen.

5.3.1 Optimierung der Abarbeitungsreihenfolge der Operatoren auf der Agenda

Es ist sehr wichtig, Wissen über die Bearbeitungsreihenfolge bzw. Wissen zur Bewertung der Wichtigkeit der einzelnen Agenda-Operatoren einzusetzen, um den Schlußfolgerungsvorgang etwas zu steuern. Dadurch können in der Regel viele Irrwege während der Navigation durch den Suchraum vermieden werden.
Dieses Wissen wird hauptsächlich in Form von Strategie-Regeln bei CONTROL repräsentiert. In wesentlich geringerem Umfang läßt sich die Abarbeitungsreihenfolge auch durch die Formulierung der Aufgabenspezifikation (z. B. durch deren Reihenfolge) beeinflussen.

Beispiel 5.6:

Operatoren, die angeben, wieviel von welcher Environment-Ressource durch die Umgebung des technischen Systems zur Verfügung gestellt wird (siehe Unterkapitel 5.2.2), sollten vorrangig abgearbeitet werden. Weil Defizite bei Environment-Ressourcen nicht behoben werden können und in solchen Fällen immer Backtracking eingeleitet werden muß, sollten solche Defizite möglichst frühzeitig erkannt werden.

Beispiel 5.7:

Es ist meistens sinnvoll, alle Agenda-Operatoren, die sich unmittelbar aus der Aufgabenspezifikation ergeben, zuerst abzuarbeiten. Denn die Bedingungen, die durch solche Operatoren repräsentiert werden, müssen in einer Konfigurationslösung auf jeden Fall erfüllt sein.[41]

5.3.2 Unterstützung der Steuerungseinheit CONTROL bei der Komponenten-Auswahl

Bei der Komponenten-Auswahl für die Skolemisierung bzw. Realisierung möchte man vermeiden, daß CONTROL diese unter Umständen folgenschwere Entscheidung völlig willkürlich trifft und im schlimmsten Fall einfach alle Möglichkeiten stumpfsinnig mit teuren Backtracking-Operationen durchprobiert. Um hier eine gute Entscheidung treffen zu können, müßte CONTROL alle möglichen Komponenten aufgrund des Kosten-Nutzen-Verhältnisses und unter überschlagsmäßiger Berücksichtigung der Folgekosten bewerten. Für eine solche Bewertung bräuchte CONTROL aber teilweise Informationen, die nur in bestimmten Konkreten Bereichen als internes Wissen vorhanden sind. Zu diesen Informationen gehören z. B. die (internen) Ressourcen-Bilanzen im Konkreten Bereich TAXRES.

Gerade beim ressourcen-orientierten Konfigurieren müßte CONTROL für eine möglichst intelligente Komponenten-Auswahl unbedingt wissen, wieviel jeweils von welcher Ressource fehlt. Für dieses ressourcen-spezifische Problem gibt es z. B. folgende Lösungsansätze:

a) Wenn TAXRES aufgrund eines Ressourcen-Defizites R_Δ einen Operator auf die Agenda einträgt (siehe Abschnitt 5.2.3), dann könnte es in diesem Operator von der neu in die A-Box einzutragenden Komponenten-Variable fordern, daß sie mindestens die fehlende Ressource-Menge R_Δ zur Verfügung stellt. Dieses Vorgehen würde aber den Suchraum zu

41. Zu diesem Zweck wurde der bisherige SOLVE-Befehl (siehe [Wache94]) um den zusätzlichen Parameter **:weight** erweitert. Mit diesem Parameter kann man den Operatoren, die unmittelbar aufgrund der Aufgabenspezifikation eingetragen werden, ein bestimmtes Gewicht zuordnen (siehe Anwendungsbeispiel in Kapitel 6).

stark einschränken, da der Ressource-Bedarf auch durch mehrere Komponenten gedeckt werden könnte.
Deshalb wäre es besser, wenn TAXRES statt dessen einen OR-Operator eintragen würde, bei dem zuerst eine Komponente gesucht wird, die direkt mindestens das Ressource-Defizit ausgleicht. Falls diese erste Suche scheitern sollte, würde der zweite Teil des OR-Operators sich auch mit Komponenten zufrieden geben, die weniger von der fehlenden Ressource zur Verfügung stellen. In diesem zweiten Fall könnte dann das Ressource-Defizit nur verringert werden und es müßte anschließend noch mindestens eine weitere Komponente eingetragen werden.
Prinzipiell würde dieses heuristische Verfahren zwar seinen Zweck erfüllen und ließe sich auch relativ einfach implementieren, hätte dafür aber einige schwerwiegende Nachteile:

- Es wäre zu unflexibel, da der Benutzer die dahinterstehende Heuristik nicht verändern könnte.
- Das zum Zeitpunkt der Bilanzierung festgestellte und im Operator implizit gespeicherte Ressource-Defizit könnte sich bis zum Zeitpunkt, an dem der Operator abgearbeitet wird, schon wieder geändert haben.
- Dieses Verfahren würde sich nur auf Komponenten-Variablen auswirken, die TAXRES explizit angefordert hat.

b) Man könnte CONTROL um eine ressourcen-orientierte Bewertungsfunktion für die Komponenten-Auswahl bei der Skolemisierung bzw. Realisierung erweitern, die irgendwie mit TAXRES zusammenarbeiten müßte.
Diese Methode würde die in a) aufgeführten Nachteile vermeiden. Insbesondere würde die ressourcen-orientierte Bewertung erst unmittelbar vor der Abarbeitung des entsprechenden Operators erfolgen. Außerdem könnte man bei der Bewertung theoretisch alle Ressourcen gleichzeitig berücksichtigen.
Problem: CONTROL kann nicht direkt auf Informationen aus TAXRES zugreifen, da nach der TOOCON-Philosophie für CONTROL noch nicht einmal bekannt ist, ob TAXRES überhaupt vorhanden ist oder nicht.[42]

Ein Vergleich beider Lösungsansätze zeigt, daß Verfahren b) zwar aufwendiger zu realisieren, aber ansonsten für die ressourcen-orientierte Konfigurierung viel besser geeignet ist. Um Verfahren b) zu implementieren, müssen aber einige Schnittstellen-Erweiterungen vorgenommen werden.

5.3.3 Allgemeine Schnittstellen-Erweiterungen zwischen CONTROL, TAXON und den Konkreten Bereichen

Aufgrund der Architektur des TOOCON-Werkzeugkastens (siehe Bild 2.1) kann CONTROL weder auf Informationen von TAXRES noch auf die eines anderen Konkreten Bereichs direkt zugreifen. Die Verbindung zwischen CONTROL und TAXRES könnte aber über erweiterte Schnittstellen zwischen CONTROL und TAXON und zwischen TAXON und den Konkreten Bereichen erfolgen. Um CONTROL bei der Komponenten-Auswahl möglichst optimal unterstützen zu können, werden diese Schnittstellen gemäß folgendem Szenario erweitert:

42. Für eine Konfigurierung mit dem TOOCON-Werkzeugkasten brauchen nämlich nicht unbedingt alle verfügbaren Tools geladen zu werden.

Wenn CONTROL aus einem Oberkonzept ein Terminal-Konzept machen soll, dann kann es im Zweifelsfall das betreffende Oberkonzept zusammen mit der Liste aller noch möglichen Komponenten-Ausprägungen zur Bewertung an TAXON übergeben. TAXON leitet seinerseits diese Informationen an alle Konkreten Bereiche weiter, die über diese erweiterte Schnittstelle verfügen[43]. Jeder dieser Konkreten Bereiche ordnet dann jeder einzelnen Komponenten-Ausprägung eine Bewertungszahl zu. Je größer diese Zahl ist, desto besser ist die jeweilige Komponenten-Ausprägungen aus der Sicht dieses Konkreten Bereiches. Die bewerteten Komponenten-Listen werden jeweils an TAXON zurückgegeben, welches diese Listen direkt an CONTROL weiterleitet. Mit Hilfe dieser neuen Informationen und den gespeicherten Heuristiken fällt CONTROL schließlich die Realisierungs-Entscheidung (siehe Bild 5.2).

Wenn CONTROL entscheiden muß, welche Komponenten-Variable mit welcher Komponenten-Konstanten unifiziert wird, wird analog dazu vorgegangen.

43. Derzeit ist das nur TAXRES. Später können aber noch weitere Konkrete Bereiche folgen.

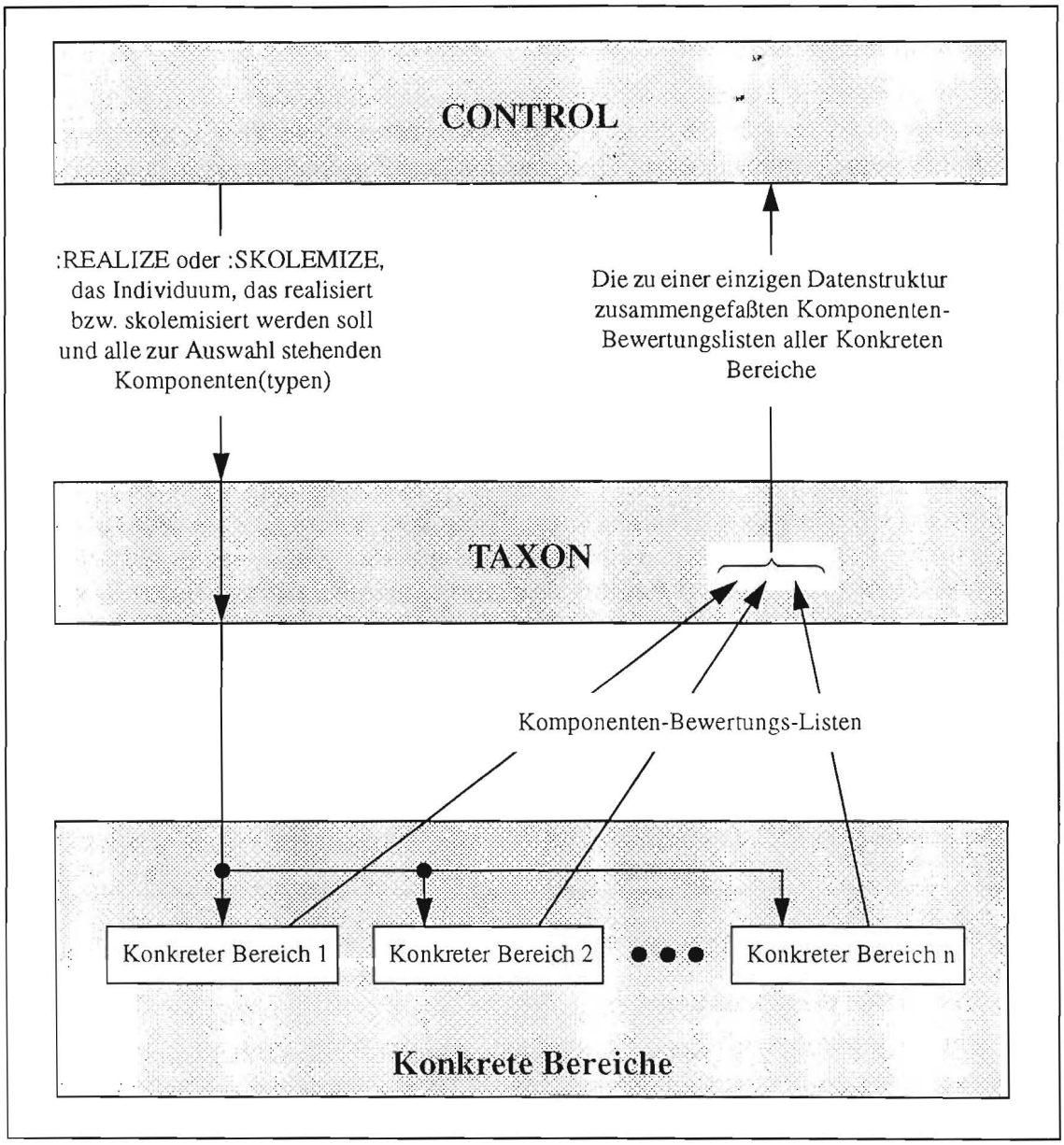


Bild 5.2: Die Schnittstellen-Erweiterungen um CONTROL bei der Komponenten-Auswahl zu unterstützen

Bemerkung:

Es ist hier wichtig, daß CONTROL alle zur Auswahl stehenden Komponenten gleichzeitig übergibt, da es im allgemeinen einfacher ist, eine in Frage kommende Komponente im Vergleich zu anderen Komponenten als sie einzeln zu bewerten[44].

44. vgl. z. B. die Art und Weise wie ein Lehrer seinen Schülern Noten vergibt

5.3.4 Die Komponenten-Bewertung durch TAXRES

Pre-Compilation der T-Box:

Damit TAXRES mehrere zur Auswahl stehende Komponenten miteinander vergleichen und ressourcen-orientiert bewerten kann, muß es von allen Konzepten der T-Box wissen, wieviel sie von welcher Ressource minimal fordern bzw. maximal zur Verfügung stellen. Diese Angaben werden am Ende aller Konzept-Definitionen mit dem Befehl

```
(Init-TAXRES)
```

durch eine Art Pre-Compilation des Komponentenwissens einmalig berechnet und in TAXRES gespeichert.

Definition von Ressource-Bewertungszahlen:

Da die einzelnen Ressourcen unterschiedlich wichtig sein können, ist es sinnvoll, wenn der Benutzer die Möglichkeit hat, jeder Ressource R eine Ressource-Bewertungszahl RBZ_R zuzuordnen. Daher werden die Befehle **`Declare-Standard-Resources`** und **`Declare-Environment-Resources`** aus Abschnitt 5.2.1 folgendermaßen erweitert:

`(Declare-Standard-Resources` *{⟨Ressource-Angabe⟩}*$^+$**`)`**

`(Declare-Environment-Resources` *{⟨Ressource-Angabe⟩}*$^+$, mit

⟨Ressource-Angabe⟩ ::= *⟨Ressource-Name⟩* | **`(`***⟨Ressource-Name⟩ ⟨Bewertungszahl⟩*)

Defaultmäßig wird dabei jeder Ressource die Bewertungszahl 1 zugeordnet.

Die Komponenten-Bewertungsfunktion:

Es ist geplant, daß der Benutzer eine Komponenten-Bewertungsfunktion anhand eines generischen Schemas weitgehend frei definieren kann. Die Implementierung einer solchen allgemeinen Benutzerschnittstelle würde aber den Rahmen dieser Diplomarbeit sprengen. Deshalb wird hier nur (als Beispiel) eine feste Bewertungsfunktion für die *Realisierung* implementiert, die nach folgender Strategie arbeitet:

Prinzip:

Diese feste Bewertungsfunktion zur Unterstüzung von CONTROL bei der *Realisierung* arbeitet zweistufig: Jede in Frage kommende Komponentenausprägung (Terminalkonzept) wird primär nach den von ihr bereitgestellten bzw. geforderten Standard-Ressourcen (z. B. elektr. Leistung) im prozentualen Verhältnis[45] zum entsprechenden Ressource-Bedarf bewertet. In der zweiten Stufe werden alle Komponenten zusätzlich noch aufgrund ihres Verbrauches an Environment-Ressourcen (z. B. Geld) im Vergleich zueinander bewertet.

In beiden Bewertungsstufen werden die einzelnen Ressourcen gemäß den zugehörigen Ressource-Bewertungszahlen gewichtet. Dabei ist es meistens sinnvoll, die Ressource-Bewertungszahlen so zu wählen, daß die Bewertung der ersten Stufe den Hauptausschlag gibt. Denn

wenn man z. B. ein Netzteil aus dem Komponentenkatalog aussuchen soll, dann ist normalerweise das Hauptauswahlkriterium, daß das Netzteil ausreichend viel von der Standard-Ressource "elektr. Leistung" zur Verfügung stellt. Erst an zweiter Stelle kommt es darauf an, daß das Netzteil nicht mehr von der Environment-Ressource "Geld" verbraucht, als es unbedingt notwendig ist.

Die zugehörigen mathematischen Funktionen werden nachfolgend beschrieben:

Die zugehörigen mathematischen Funktionen:

Sei **MTK** := {TK1, TK_2, ... TK_n} die Menge aller für die Realisierung des Individuums **Ind** in Frage kommenden Terminalkonzepte $\mathbf{TK_i}$, mit $i \in \{1, 2, ... n\}$.
MR sei die Menge aller bei den Terminalkonzepten in MTK vorkommenden Ressourcen. Weiter sei **MSR** die Menge der in MR enthaltenen Standard-Ressourcen. Analog dazu sei **MER** die Teilmenge der Environment-Ressourcen in MR.
RBZ (R) sei die Ressource-Bewertungszahl der Ressource R, und $\mathbf{RM^-}$ **(TK, R)** sei die Ressource-Menge, die eine Instanz des Terminalkonzeptes TK von der Ressource R benötigt.

Innerhalb von zwei Konzept-Berwertungsstufen werden für jedes Terminalkonzept TK_i die zwei unterschiedlichen Konzept-Bewertungszahlen $\mathbf{KBZ_1}$ und $\mathbf{KBZ_2}$ durch die nachfolgenden Formeln berechnet und anschließend zu einer (Gesamt-)Konzept-Bewertungszahl **KBZ** addiert:

(1) Sei **F (Ind, TK, R)** eine Funktion, die angibt, wieviel Prozent (Zahl zwischen 0 und 100) des Bedarfs an der Ressource R gedeckt ist, wenn das Individuum Ind zum Terminalkonzept TK realisiert wird.
Dann wird jedem Terminalkonzept TK_i mit $i \in \{1, 2, ... n\}$ in der ersten Bewertungsstufe folgende Konzept-Bewertungszahl $\mathbf{KBZ_1}$ **(Ind, $\mathbf{TK_i}$)** zugeordnet:

$$KBZ_1 (Ind, TK_i) := \sum_{R \in MSR} \left(F (Ind, TK_i, R) * RBZ (R) \right)$$

(2) Sei **Pos (TK, R, MTK)** eine Funktion, die angibt, an welcher Position das Terminalkonzept $TK \in MTK$ stehen würde, wenn man TK_1, TK_2, ... TK_n nach dem Ressource-Verbrauch RM^- (TK, R) absteigend sortiert, wobei jeweils zwei Konzepte TK_i, TK_j mit $RM^- (TK_i, R) = RM^- (TK_j, R)$ parallel zueinander an der gleichen Position stehen sollen.
Die Konzept-Bewertungszahl der zweiten Stufe $\mathbf{KBZ_2}$ **($\mathbf{TK_i}$, MTK)** mit $i \in \{1, 2, ... n\}$ wird dann folgendermaßen berechnet:

---

45. Die reine Differenz zwischen Bedarf und Bereitstellung ist bei der Berücksichtigung mehrerer Ressourcen normalerweise weniger geeignet, da die Ressource-Mengen-Angaben verschiedener Ressourcen oft sehr unterschiedliche Größenordnungen haben.
Beispiel:
Eine SCSI-Festplatte stellt von der Ressource "Speicher" z. B. die Menge 500 zur Verfügung, fordert aber "nur" die Menge 1 von der Ressource "SCSI-Ports".

$$KBZ_2\,(TK_i, MTK) := \sum_{R \in MER} \left(Pos\,(TK_i, R, MTK) * RBZ\,(R) \right)$$

(3) Die (Gesamt-)Konzept-Bewertungszahl **KBZ (Ind, TK_i, MTK)** eines Terminalkonzeptes TK_i mit $i \in \{1, 2, \ldots n\}$ erechnet sich damit wie folgt:

$$KBZ\,(Ind, TK_i, MTK) = KBZ_1\,(Ind, TK_i) + KBZ_2\,(TK_i, MTK)$$

Hierbei ist zu beachten, daß ein Terminalkonzept TK_i bei der Realisierung um so bevorzugter ausgewählt wird, je höher dessen Konzept-Bewertungszahl KBZ ist.

6. Anwendungsbeispiel für die ressourcen-orientierte Konfigurierung mit TAXRES

In diesem Kapitel wird anhand eines etwas größeren Beispiels aus einer PC-Domäne demonstriert, wie man mit Hilfe von TAXRES im TOOCON-Werkzeugkasten ressourcen-orientiert konfigurieren kann. Gleichzeitig soll mit diesem Beispiel gezeigt werden, daß es durchaus sinnvoll sein kann, in ein und demselben Beispiel sowohl ressourcen- als auch struktur-orientierte Teile miteinander zu vermischen.

6.1 Umgangssprachliche Wissensbeschreibung

Ressourcen-orientiertes Wissen:

In diesem Beispiel werden folgende Ressourcen verwendet:

Standard-Ressourcen	Environment-Ressourcen
HD-Speicher, RAM-Speicher, RAM-Steckplätze, ISA-Slots, VLB-Slots, PCI-Slots, IDE-Ports, SCSI-Ports, FD-Ports, Gehäuse-Einschübe, Leistung	Geld

Nachfolgende Tabelle zeigt eine Übersicht, welche der in diesem Beispiel vorkommenden Komponenten von welcher Ressource eine bestimmte Menge verbrauchen bzw. zur Verfügung stellen:

Komponente	verbrauchte Ressourcen	bereitgestellte Ressourcen
PC-Gehäuse	Geld	Gehäuse-Einschübe
Netzteil	Geld	Leistung
Motherboard	Geld, Leistung	RAM-Steckplätze, ISA/VLB/PCI-Slots, evtl. IDE/SCSI-Ports, evtl. FD-Ports
Prozessor	Geld Leistung	
RAM-Modul	Geld Leistung, RAM-Steckplätze	RAM-Speicher

Tabelle 6.1: Übersicht über den Ressourcen-Verbrauch bzw. die Ressourcen-Bereitstellung der verwendeten Komponenten-Typen

Komponente	verbrauchte Ressourcen	bereitgestellte Ressourcen
Grafikkarte	Geld, Leistung, ISA/VLB/PCI-Slots	
Monitor	Geld	
Controller	Geld, Leistung, ISA/VLB/PCI-Slots	IDE/SCSI-Ports, FD-Ports
Festplatte	Geld, Leistung, Gehäuse-Einschübe	HD-Speicher
FD-Laufwerk	Geld, Leistung, FD-Ports, Gehäuse-Einschübe	

Tabelle 6.1: Übersicht über den Ressourcen-Verbrauch bzw. die Ressourcen-Bereitstellung der verwendeten Komponenten-Typen (Forts.)

Struktur-orientiertes Wissen:

- Jedes *PC-Gehäuse* enhält ein Netzteil
- Jedes *Motherboard* hat einen Prozessor[46], RAM-Speicher und eine Grafikkarte.
- Ein *Personal-Computer* besteht z. B. aus:
  einem PC-Gehäuse (mit einem Netzteil),
  einem Motherboard (mit Prozessor, RAM-Speicher und Grafikkarte)
  und einem Monitor.

6.2 Die Modellierung des Beispiels mit dem TOOCON-Werkzeugkasten unter Verwendung von TAXRES

In diesem Unterkapitel ist nur beschrieben, wie ein solches Beispiel prinzipiell mit dem TOOCON-Werkzeugkasten modelliert werden kann (vgl. [Wache94]). Nähere Informationen dazu und Auszüge aus dem Protokoll einer Beispielsitzung befinden sich in Anhang A.

Deklarierung der Ressourcen:

Die beiden nachfolgenden Befehle deklarieren alle in diesem Unterkapitel vorkommenden Ressourcen (vgl. Kapitel 5.2.1).

46. Wenn man diese 1:1-Beziehung zwischen Motherboard und Prozessor ressourcen-orientiert modellieren wollte, dann müßte man speziell für diese eine Beziehung eine eigene Ressource (z. B. Prozessor-Anzahl) einführen. Da von dieser Ressource dann aber immer genau die Menge 1 verbraucht bzw. zur Verfügung gestellt würde und man diese Ressource sonst nirgends verwenden könnte, wäre das eine inadäquate Modellierung.

```
(DECLARE-STANDARD-RESOURCES  HD-Speicher
                             IDE-Ports SCSI-Ports
                             Gehaeuse-Einschuebe
                             Leistung)

(DECLARE-ENVIRONMENT-RESOURCES Geld)
```

Nachdem alle Ressourcen deklariert wurden, wird das spezielle Konzept ENVIRONMENT definiert (siehe Unterkapitel 5.2.2):

```
(DEFINE-ENVIRONMENT-CONC)
```

Definition der Konkreten Prädikate für die nachfolgenden Konzept-Definitionen:

Alle Konkreten Prädikate, die in irgendeiner Konzept-Definition verwendet werden, müssen vorher (gemäß den Syntax-Regeln aus Unterkapitel 5.1.1) definiert werden[47]. Nachfolgend werden aber nur die Konkreten Prädikate definiert, die zum Verständnis des nächsten Abschnitts benötigt werden.

```
(CPRED Leistung-Minus (:TAXRES (?X) (Leistung :Consumer (?X >=! 0))))
(CPRED Leistung-Minus-10 (:TAXRES (?X) (Leistung :Consumer (?X =! 10))))

(CPRED HD-Speicher-Plus
       (:TAXRES (?X) (HD-Speicher :Supplier (?X >=! 100))))
(CPRED HD-Speicher-Plus-400
       (:TAXRES (?X) (HD-Speicher :Supplier (?X =! 400))))
(CPRED HD-Speicher-Plus-500
       (:TAXRES (?X) (HD-Speicher :Supplier (?X =! 500))))

(CPRED Gehaeuse-Einschuebe-Minus-1
       (:TAXRES (?X) (Gehaeuse-Einschuebe :Consumer (?X =! 1))))

(CPRED IDE-Ports-Minus-1 (:TAXRES (?X) (IDE-Ports :Consumer (?X =! 1))))
(CPRED SCSI-Ports-Minus-1 (:TAXRES (?X) (IDE-Ports :Consumer (?X =! 1))))

(CPRED Geld-Minus (:TAXRES (?X) (Geld :Consumer (?X >! 0))))
(CPRED Geld-Minus-300 (:TAXRES (?X) (Geld :Consumer (?X =! 300))))
(CPRED Geld-Minus-400 (:TAXRES (?X) (Geld :Consumer (?X =! 400))))
```

Definition einiger Konzepte:

Um das Prinzip einer ressourcen-orientierten Konzept-Definition zu veranschaulichen, werden in diesem Abschnitt als Beispiel eine IDE-Festplatte, eine SCSI-Festplatte und die zugehörigen Oberkonzepte definiert. Die anderen Konzept-Definitionen findet man in Anhang A.

Eine Festplatte ist nicht nur irgendein Objekt, das Leistung aufnimmt, HD-Speicher zur Verfügung stellt, einen Gehäuse-Einschub benötigt und Geld kostet. Um eine Festplatte eindeutig zu charakterisieren, müßte man noch wesentlich mehr Eigenschaften beschreiben.
Insbesondere ist auch die Umgebung eines technischen Systems theoretisch in der Lage, von

47. In der jetzigen Version von TOOCON ist es noch so, daß der Benutzer jedes verwendete Konkrete Prädikat vorher explizit definieren muß. Das ist aber kein grundsätzliches Problem, sondern nur eine Schwäche der derzeitigen Benutzerschnittstelle. Rein technisch wäre es z. B. auch möglich, daß der Benutzer innerhalb einer Konzept-Definition (mit CONC) Konkrete Prädikate implizit definieren könnte. Eine solche Erweiterung wird möglicherweise in einer zukünftigen Programm-Version implementiert.

jeder deklarierten Ressource eine gewisse Menge zur Verfügung zu stellen bzw. zu verbrauchen. Dennoch ist aber z. B. das Konzept *Festplatte* kein Unterkonzept des speziellen Konzeptes *ENVIRONMENT* !
Aus diesem Grund wird im folgenden ein primitives Konzept *Festplatten-Objekt* definiert. Dieses Konzept dient zur Beschreibung aller nicht modellierten Eigenschaften einer Festplatte.

```
(prim Festplatten-Objekt)

(CONC Festplatte
      (AND  Festplatten-Objekt
            (SOME (Leistung) Leistung-Minus)
            (SOME (HD-Speicher) HD-Speicher-Plus)
            (SOME (Gehaeuse-Einschuebe) Gehaeuse-Einschuebe-Minus-1)
            (SOME (Geld) Geld-Minus) ))

(CONC IDE-Festplatte
      (AND Festplatte
           (SOME (IDE-Ports) IDE-Ports-Minus-1) ))

(CONC SCSI-Festplatte
      (AND Festplatte
           (SOME (SCSI-Ports) SCSI-Ports-Minus-1) ))
```

Die Definition des Konzepte "Festplatte", "IDE-Festplatte" und "SCSI-Festplatte" sind hier nach dem gleichen Prinzip aufgebaut, Der Einfachheit halber wird an dieser Stelle aber nur die Definition der IDE-Festplatte erklärt[48]:

> Nach dieser Konzept-Definition (→ *CONC*-Befehl) ist eine "IDE-Festlpatte" ein Unterkonzept von "Festplatte" und (→ *AND*-Operator) gleichzeitig ein Konzept, bei dem ein Attribut "IDE-Ports" existiert (→ Exist-Quantor *SOME*), dessen Wertebereich durch das Konkrete Prädikat "IDE-Ports-Minus-1" festgelegt wird.

Die Festplatte-Typ-A (Terminal-Konzept) sei ein Beispiel für eine 400 MB IDE-Festplatte, die 10 W Leistung aufnimmt und 300 DM kostet:

```
(CONC Festplatte-Typ-A (AND IDE-Festplatte
                            (SOME (Leistung) Leistung-Minus-10)
                            (SOME (HD-Speicher) HD-Speicher-Plus-400)
                            (SOME (Geld) Geld-Minus-300) ))
```

Die Festplatte-Typ-B ist nach dem gleichen Prinzip definiert:

```
(CONC Festplatte-Typ-B (AND SCSI-Festplatte
                            (SOME (Leistung) Leistung-Minus-10)
                            (SOME (HD-Speicher) HD-Speicher-Plus-500)
                            (SOME (Geld) Geld-Minus-400) ))
```

Die T-Box am Ende dieser Konzept-Definitionen:

Nachdem diese Konzepte definiert wurden, sieht die textuelle Darstellung des (durch den Classifier der T-Box) berechneten Subsumptionsgraphen folgendermaßen aus:

48. Ausführlichere Informationen dazu findet man [RSW94] und in [Wache94].

```
                                TOP  <===  (HD-SPEICHER-SUPPLIER-OBJECT)
                                          (IDE-PORTS-SUPPLIER-OBJECT)
                                          (SCSI-PORTS-SUPPLIER-OBJECT)
                                          (GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT)
                                          (LEISTUNG-SUPPLIER-OBJECT)
                                          (ENVIRONMENT-CONCEPT)
                                          (FESTPLATTEN-OBJEKT)
          HD-SPEICHER-SUPPLIER-OBJECT  <===  (FESTPLATTE)
                              FESTPLATTE  <===  (IDE-FESTPLATTE)
                                          (SCSI-FESTPLATTE)
                           IDE-FESTPLATTE  <===  (FESTPLATTE-TYP-A)
                          SCSI-FESTPLATTE  <===  (FESTPLATTE-TYP-B)
                        FESTPLATTE-TYP-A  <===  (BOTTOM)
                        FESTPLATTE-TYP-B  <===  (BOTTOM)
                  IDE-PORTS-SUPPLIER-OBJECT  <===  (BOTTOM)
                 SCSI-PORTS-SUPPLIER-OBJECT  <===  (BOTTOM)
      GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT  <===  (BOTTOM)
                  LEISTUNG-SUPPLIER-OBJECT  <===  (BOTTOM)
                       ENVIRONMENT-CONCEPT  <===  (ENVIRONMENT)
                               ENVIRONMENT  <===  (BOTTOM)
                         FESTPLATTEN-OBJEKT  <===  (FESTPLATTE)
```

Die Schreibweise `A <=== (B) (C) (D),`
bedeutet, daß die Konzepte B, C und D direkte Unterkonzepte des (Ober-)Konzeptes A sind. Dabei ist das spezielle Konzept TOP Oberkonzept aller Konzepte und das spezielle Konzept BOTTOM Unterkonzept aller Konzepte. Diese speziellen Konzepte werden durch TAXON automatisch erzeugt. Bild 6.1 zeigt eine graphische Darstellung dieser Taxonomie:

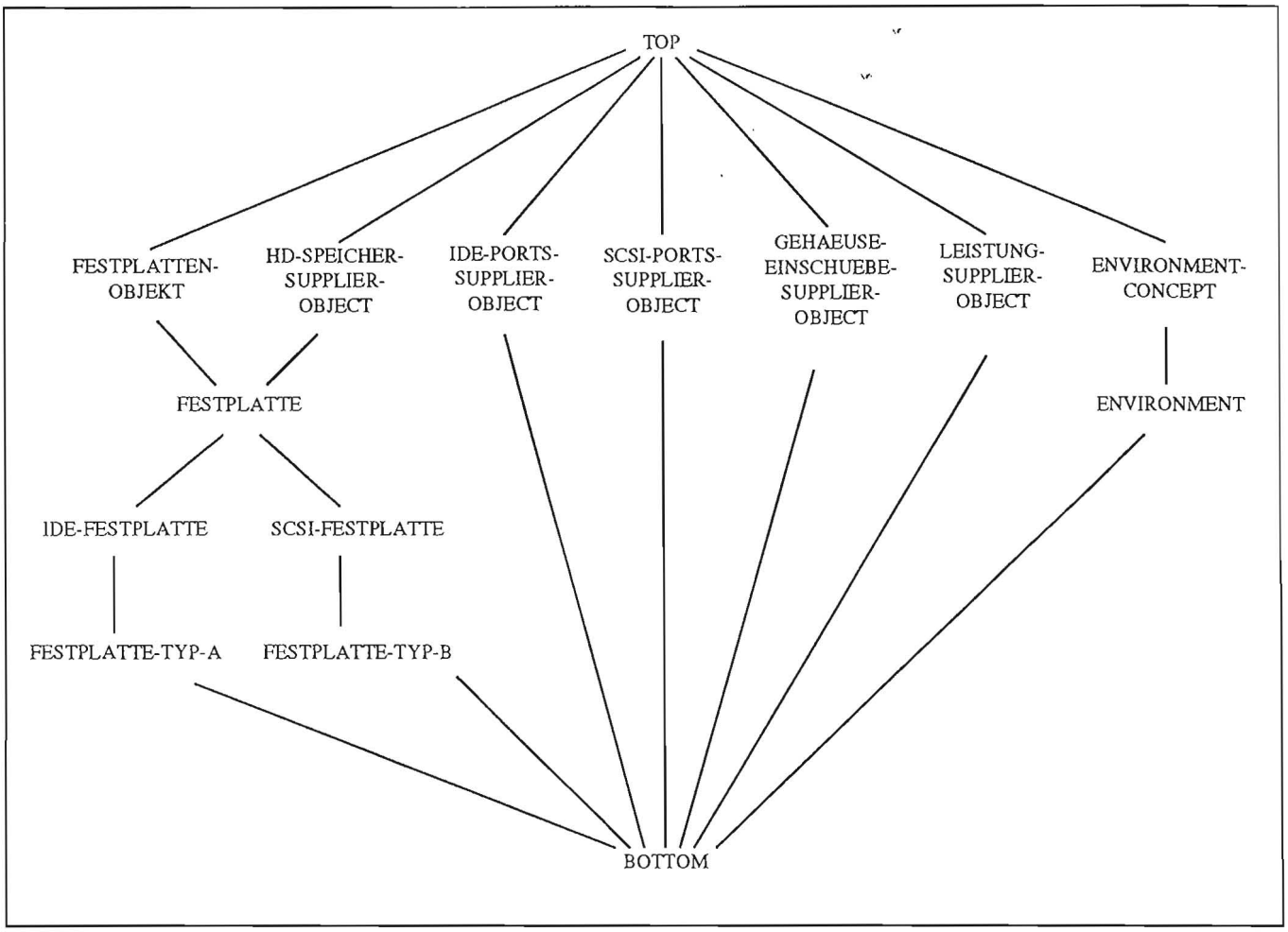


Bild 6.1: Graphische Darstellung der Taxonomie

Neben den mit dem CONC-Befehl explizit erzeugten Konzepten, wurde aufgrund der Ressource-Deklaration zu jeder Ressource R ein Konzept mit dem Namen ⟨R⟩-SUPPLIER-OBJECT generiert, das Oberkonzept aller Konzepte ist, die eine bestimmte Menge von R zur Verfügung stellen (siehe Kapitel 5.2.1 und 5.2.3).

Das spezielle Konzept ENVIRONMENT repräsentiert die Umgebung des technischen Systems und kann nur bei Formulierung der Aufgabenspezifikation verwendet werden (siehe Kapitel 5.2.2). Das (Hilfs-)Konzept ENVIRONMENT-CONCEPT ist ein primitives Konzept, das ausschließlich für die Definion des Konzeptes ENVIRONMENT benötigt wird, um dieses eindeutig von allen anderen Konzepten zu unterscheiden.

Spezifikation einer Konfigurierungsaufgabe:

Einfaches Beispiel[49] für eine vom Benutzer formulierte Aufgabenspezifikation:

```
(solve ( (Env :in Environment)
         (Env = Geld => ?G)
         (Geld :Supplier (?G =! 4000))

         (F :in Festplatte)
       )
       :weight 1000 )
```

Hiermit wird durch die Umgebung maximal 4000 DM zur Verfügung und lediglich gefordert, daß in der zu generierenden Konfiguration eine Festplatte F vorhanden ist.

Damit die Agenda-Operatoren, die sich direkt aus der Aufgabenspezifikation ergeben, zuerst abgearbeitet werden, wird ihnen hier jeweils zunächst ein Gewicht von 1000 zugeordnet.[50]

Die Agenda nach der Aufgabenspezifikation:

Wenn man sich nach dieser Aufgabenspezifikation die Agenda anzeigen läßt, dann erhält man folgendes:

```
*******************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (ENV :IN ENVIRONMENT)


*******************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (ENV = GELD => ?G)


*******************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (GELD :SUPPLIER (?G =! 4000))
```

49. Im Anhang A wird diese Aufgabenspezifikation noch etwas erweitert
50. Dieses Gewicht könnte sich theoretisch durch die Anwendung entsprechender Strategie-Regeln noch ändern.

```
**************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (F :IN FESTPLATTE)


**************************************************************************
```

Der prinzipielle Ablauf des Konfigurierungsprozesses:

Um eine Konfigurationslösung zu finden, müssen alle Operatoren auf der Agenda nacheinander abgearbeitet werden, wodurch in machen Fällen erneut Operatoren auf die Agenda eingetragen werden bzw. bei einer Inkonsistenz in der A-Box Backtracking eingeleitet werden muß (siehe Kapitel 2.2 und 2.3).

Nachfolgend sind ein paar markante Zwischenschritte dieses Konfigurierungsprozesses abgedruckt, an denen man erkennt, wie durch die Abarbeitung der Operatoren die A-Box verändert wird:

Durch die ersten drei Operatoren wird in die A-Box eine Instanz des Konzeptes ENVIRONMENT eingetragen und dieser über das Attribut GELD eine Ressource-Supplier-Variable mit dem Wert 4000 zugeordnet, wonach die A-Box folgendermaßen aussieht:

```
-------------------------------------------------
ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD    ---> ?G
-------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )
-------------------------------------------------
```

Danach wird die Festplatte F in die A-Box eingetragen und automatisch die entspechenden Attribut-Informationen hinzugefügt, wodurch folgende A-Box entsteht:

```
##########################################################################

ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD    ---> ?G
--------------------------------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )

##########################################################################

F is related to the concepts: (FESTPLATTE)
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-170
  |-A- HD-SPEICHER ---> ?_NEW-HD-SPEICHER-169
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-168
  |-A- GELD    ---> ?_NEW-GELD-167
--------------------------------------------------------------------------
?_NEW-LEISTUNG-170 is a Resource-CONSUMER-Variable of the type LEISTUNG
                   and in the range ( ((>= 0) NIL) )
--------------------------------------------------------------------------
```

```
?_NEW-HD-SPEICHER-169 is a Resource-SUPPLIER-Variable of the type HD-SPEICHER
                      and in the range ( ((>= 100) NIL) )
-------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-168 is a Resource-CONSUMER-Variable of the type
                              GEHAEUSE-EINSCHUEBE
                              and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------------
?_NEW-GELD-167 is a Resource-CONSUMER-Variable of the type GELD
               and in the range ( ((> 0) NIL) )
-------------------------------------------------------------------------------
```

An dieser Stelle wird beim Ressourcen-Bilanzieren festgestellt, daß bei der Ressource GEHAEUSE-EINSCHUEBE ein Defizit vorliegt und daher Operator auf die Agenda geschrieben, der eine Instanz des Ressource-Oberkonzeptes GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT in die A-Box eintragen soll:

```
Resource-Check:
There is too less of the resource GEHAEUSE-EINSCHUEBE !  [Resource-Deficit: (= 1) ]
--> The new GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT ?_IND-1 will be inserted in the A-Box.
```

Im weiteren Verlauf der Konfiguration werden diese Konzepte irgendwann in der A-Box instanziiert. Außerdem wird die Festplatte F zu einer Festplatte realisiert. Da in der Aufgabenspezifikation keine Anforderungen an die Festplatte F gestellt werden, wird hierfür die billigste Festplatte, die in der T-Box vorhanden ist (Festplatte-Typ-A) verwendet. Aufgrund dessen wird dann beim Bilanzieren erkannt, daß noch Komponenten fehlen, die mindestens 10 W elektr. Leistung bzw. mindestens einen IDE-Port zur Verfügung stellen, worauf die Oberkonzepte LEISTUNG-SUPPLIER-OBJECT und IDE-PORTS-SUPPLIER-OBJECT instanziiert werden. Desweiteren werden alle Komponenten-Variablen irgendwann vom Skolemizer zu Komponenten-Konstanten gemacht.

Die gefundene Konfigurationslösung:

Am Ende des Konfigurierungsprozesses enthält die A-Box folgende Lösungsmenge:

```
################################################################################

%_IND-1 is related to the concepts: (GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT)
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-171
-------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-171 is a Resource-SUPPLIER-Variable of the type
                              GEHAEUSE-EINSCHUEBE
                              and in the range ( (NIL NIL) )

################################################################################

%_IND-3 is related to the concepts: (LEISTUNG-SUPPLIER-OBJECT)
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-172
-------------------------------------------------------------------------------
?_NEW-LEISTUNG-172 is a Resource-SUPPLIER-Variable of the type LEISTUNG
                   and in the range ( (NIL NIL) )

################################################################################

%_IND-5 is related to the concepts: (IDE-PORTS-SUPPLIER-OBJECT)
  |-A- IDE-PORTS ---> ?_NEW-IDE-PORTS-177
-------------------------------------------------------------------------------
```

```
?_NEW-IDE-PORTS-177 is a Resource-SUPPLIER-Variable of the type IDE-PORTS
                    and in the range ( (NIL NIL) )

#########################################################################

ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD    ---> ?G
-------------------------------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )

#########################################################################

F is related to the concepts: (FESTPLATTE-TYP-A)
  |-A- IDE-PORTS ---> ?_NEW-IDE-PORTS-176
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-170
  |-A- HD-SPEICHER ---> ?_NEW-HD-SPEICHER-169
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-168
  |-A- GELD    ---> ?_NEW-GELD-167
-------------------------------------------------------------------------
?_NEW-IDE-PORTS-176 is a Resource-CONSUMER-Variable of the type IDE-PORTS
                    and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------
?_NEW-LEISTUNG-170 is a Resource-CONSUMER-Variable of the type LEISTUNG
                   and in the range ( ((>= 10) (<= 10)) )
-------------------------------------------------------------------------
?_NEW-HD-SPEICHER-169 is a Resource-SUPPLIER-Variable of the type HD-SPEICHER
                      and in the range ( ((>= 400) (<= 400)) )
-------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-168 is a Resource-CONSUMER-Variable of the type
                              GEHAEUSE-EINSCHUEBE
                              and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------
?_NEW-GELD-167 is a Resource-CONSUMER-Variable of the type GELD
               and in the range ( ((>= 300) (<= 300)) )
-------------------------------------------------------------------------
```

Da es in der derzeitigen T-Box (im Gegensatz zur der T-Box in Anhang A) keine Unterkonzepte von GEHAEUSE-EINSCHUEBE-OBJECT, LEISTUNG-SUPPLIER-OBJECT bzw. IDE-PORTS-SUPPLIER-OBJECT gibt, konnten hier die Instanzen von diesen Konzepten (%_IND-1, %_IND-3 und %_IND-5) nicht weiter realisiert werden.

Bezüglich der aktuellen T-Box ist die Konfigurationsaufgabe damit gelöst: Alle Operatoren auf der Agenda wurden abgearbeitet und damit unter anderem alle Individuen zu Komponenten-Konstanten skolemisiert und auch zu Terminal-Konzepten realisert. Außerdem sind alle Ressourcen-Bilanzen ausgeglichen[51].

51. Man beachte hierbei, daß jedes Ressource-Oberkonzept (wie z. B. LEISTUNG-SUPPLIER-OBJECT) per Definition beliebig viel von der betreffenden Ressource zur Verfügung stellt.

7. Erweiterung des Grundkonzeptes um lokale Ressourcen

In einigen Fällen ist es erwünscht, die Ressourcen-Bilanzierung nicht über alle Objekte in der A-Box durchzuführen, sondern die Bilanzierung auf eine Teilmenge davon einzuschränken. Diese Ausdrucksmöglichkeit braucht man, um zusammengehörende *Komponenten-Gruppen* modellieren zu können.

Beispiel 7.1:

Ein Computernetzwerk besteht aus mehreren einzelnen Rechnern. Jeder einzelne dieser Rechner ist jeweils ein eigenes Gerät. Alle Komponenten, die sich in einem eigenständigen Rechnergehäuse befinden, bilden dabei jeweils eine eigene *Komponenten-Gruppe*.
Das bedeutet z. B., daß sie alle jeweils ein eigenes Netzteil brauchen. Man kann also nicht einfach ein großes Netzteil für alle elektrischen Komponenten im gesamten Computernetzwerk verwenden. Aus diesem Grund sollte man hier die Ressource "elektrische Leistung" für jeden einzelnen Rechner des Netzwerks separat (d. h. *lokal*) ausbilanzieren.

Damit bestimmte Komponenten-Gruppen bezüglich einer Ressource lokal ausbilanziert werden können, muß angegeben werden, welche Komponenten zu welcher Gruppe gehören. Um diese Gruppen-Zugehörigkeit zu modellieren, wird zwischen jeder Komponente einer Gruppe und einem Konzept, das die Gruppe als Ganzes repräsentiert (→ Name der Gruppe), eine Rollenbeziehung aufgebaut.

Beispiel 7.2:

Um in einem Computernetz den Rechner_1 lokal ausbilanzieren zu können, würde man zwischen jeder einzelnen Komponente von Rechner_1 und dem Konzept Rechner_1 jeweils eine bestimmte Rollenbeziehung aufbauen, z. B.:

$$(Komponente\_1 = gehört\text{-}zu => Rechner\_1)$$

Alle Komponenten K in der A-Box mit

$$(K = gehört\text{-}zu => Rechner\_1)$$

bilden dann mit zusammen mit den zugehörigen Ressource-Variablen einer bestimmten Ressource eine *lokale (Ressourcen-)Bilanzierungs-Gruppe* von Rechner_1.

Bild 7.1 zeigt die entsprechende Aufteilung der A-Box in verschiedene Teilmengen, wenn ein Computernetz, das aus drei Rechnern besteht, mit lokalen Ressourcen konfiguriert wird.

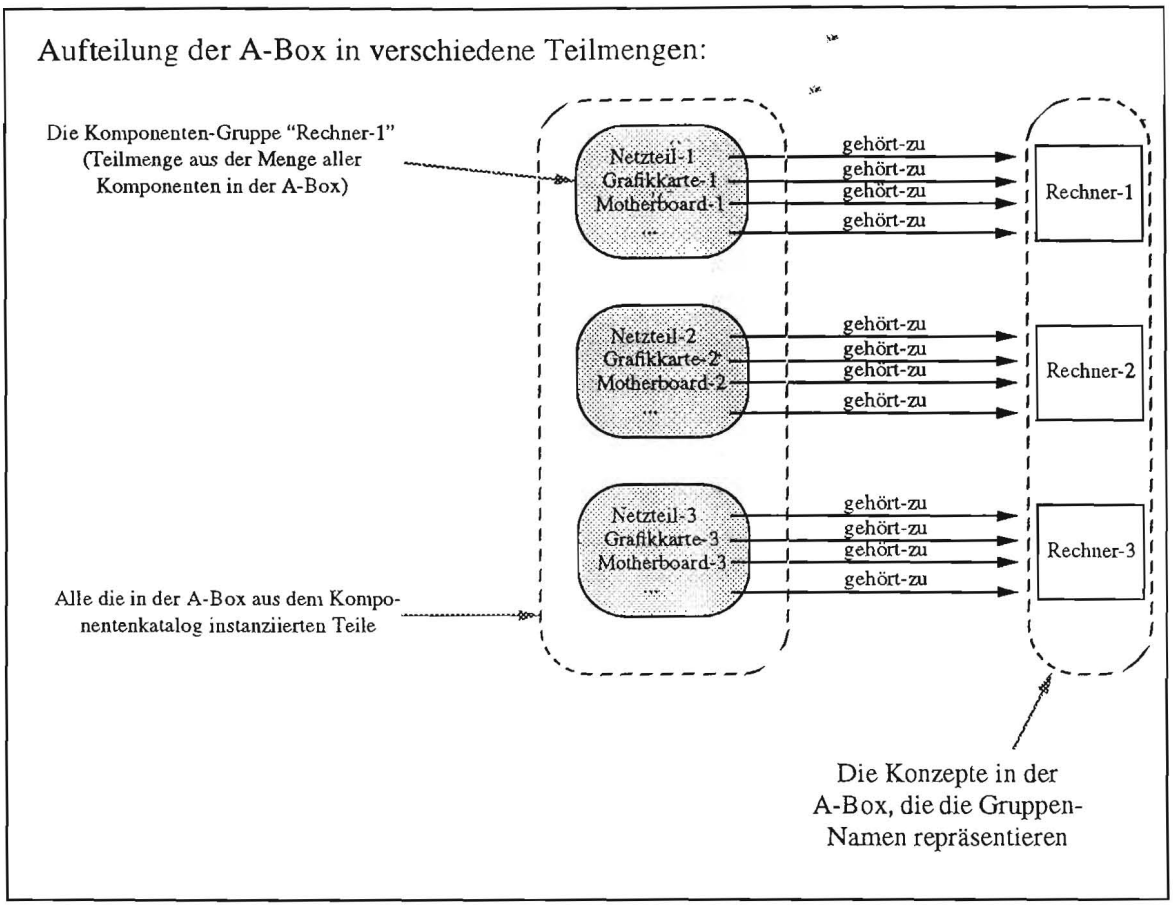


Bild 7.1: Graphische Veranschaulichung lokaler Ressourcen

7.1 Wie kann man die Komponenten-Zugehörigkeits-Rollenbeziehungen in die A-Box eintragen?

Es stellt sich die Frage, wie man diese Komponenten-Zugehörigkeits-Rollenbeziehungen in die A-Box bekommt. Dazu gibt es mehrere Möglichkeiten:

a) Man macht die Zugehörigkeitseinträge manuell. Das ist aber nur in Einzelfällen möglich bzw. sinnvoll, wie z. B. bei der Spezifikation der gewünschten Konfiguration. Danach kann man beim laufenden Konfigurierungsprozeß, bei dem immer wieder Einträge in die A-Box erfolgen, nicht vom Benutzer verlangen, daß er manuell z. B. jedes Computernetzteil einem bestimmten Rechner zuordnet.

b) Man verwendet dazu Vorwärtsregeln, die diese Eintragungen nach bestimmten Kriterien automatisch vornehmen.

c) Man trägt in der T-Box bei bestimmten Komponenten ein, daß jeweils jede Instanz davon irgendeinem Rechner zugeordnet werden muß, z. B.:

(conc computernetzteil (and netzteil (some (gehört-zu) Rechner)))

Solche "gehört-zu"-Rollen[52] können also sowohl in der T-Box als auch direkt in der A-Box eingetragen werden.[53]

7.2 Die Definition lokaler Bilanzierungs-Gruppen

Damit TAXRES lokale Bilanzierungs-Gruppen überwachen und evtl. auch ausbilanzieren kann, muß (bei der T-Box-Definition) explizit angegeben werden, auf welcher Teilmenge der A-Box welche Ressource lokal bilanziert werden soll. Dazu wird der Befehl

(Def-Local-Resource-Group
⟨Group-Name-Variable-List⟩
(*⟨Komponenten-Variable⟩* **=** *⟨Ressource-Name⟩* **=>** *⟨Resource-Variable⟩***)**
⟨Kontext-Bedingungen⟩ **)**

verwendet. Eine solche Definition enthält die folgenden Bestandteile:

Bestandteil der Definition	Beschreibung	Beispiel
⟨Komponenten-Variable⟩	Variable, die die Komponenten symbolisiert, die zur lokalen Gruppe gehören	?K
⟨Ressource-Name⟩	Name der Ressource, die in dieser Gruppe lokal ausbilanziert werden soll (Das muß auch gleichzeitig der Name des Ressource-Attributes sein.)	elektr-Leistung
⟨Ressource-Variable⟩	Variable, die die von einer Komponente bereitgestellte bzw. geforderte Ressource-Menge beschreibt	?RV
⟨Kontext-Bedingungen⟩	Liste mit Observer-Bedingungen zur Beschreibung der lokalen Gruppe	((?K = gehört-zu => ?R) (?R :in ?Rechner))
⟨Group-Name-Variable-List⟩	Die Kontext-Bedingungen können mehrere Variablen enthalten. Welcher dieser Variablen als Zugriffsschlüssel und auch für die interne Namensgebung der Gruppe benutzt werden soll, wird hier in Form einer Variablen-Liste festgelegt.	(?R)

Tabelle 7.1: Bestandteile bei der Definition lokaler Bilanzierungs-Gruppen

Beispiel 7.3:

```
(Def-Local-Resource-Group (?R)
                          (?K = Speicher => ?RV)
                          ((?K = gehört-zu => ?R)(?R :in ?Rechner)) )
```

52. Statt "gehört-zu" könnte auch jeder andere Rollen-Name verwendet werden.
53. Falls bei einer Ressourcen-Bilanzierungs-Gruppe ein behebbares Ressource-Defizit erkannt wird erfolgt automatisch ebenfalls ein solcher Rollen-Eintrag (siehe Abschnitt 7.3.3).

Diese Definition könnte man als eine Art Bilanzierungs-Bedingung ansehen, die genau dann erfüllt ist, wenn alle Komponenten, die jeweils zu einem Rechner gehören, bezüglich der Ressource Speicher lokal ausbilanziert sind.

Beispiel 7.4:

Angenommen, man möchte, daß man die Bilanzierung der Ressource Speicher bei einem speziellen Teil-Rechner wie z. B. Rechner_1, unbedingt lokal ausbilanziert wird, um zu gewährleisten, daß alle Programme auf Rechner_1 (Server) genügend lokalen Speicher besitzen, damit sie auch bei großer Netzbelastung hinreichend schnell ablaufen können. Bei allen anderen Rechnern im Netzwerk würde es aber ausreichen, wenn global im gesamten Netz genügend Speicherplatz vorhanden ist. Bei allen anderen Rechner würde man deshalb die Ressource Speicher nur global ausbilanzieren. Für diesen Spezialfall könnte man dann folgende Definition verwenden[54]:

```
(Def-Local-Resource-Group (?R)
                          (?K = Speicher => ?RV)
                          ((?K = gehört-zu => ?R) (?R :in Rechner_1)))
```

Mit den Komponenten-Zugehörigkeits-Rollenbeziehungen kann man auch Komponenten-Hierarchien und damit strukturierte Ressourcen-Bilanzierungs-Gruppen aufbauen:

Beispiel 7.5:

Wenn man mehrere Computernetze zu konfigurieren hat, dann kann man die Definition aus Beispiel 7.4 folgendermaßen noch weiter schachteln:

```
(Def-Local-Resource-Group (?R)
(?K = Speicher => ?RV)
((?K = gehört_zu => ?R)
(?R :in ?Rechner)
(?R = ist_Bestandteil_von => ?CN)
(?CN :in ?Computernetz) ))
```

Diese Notation bedeutet, daß jeweils alle Komponenten ?K, die zu einem Rechner ?R gehören, wobei ?R Bestandteil von einem Computernetz ?CN ist, bezüglich der Ressource Speicher lokal ausbilanziert werden sollen.

7.3 Das Implementierungskonzept der lokalen Ressourcen

In den nachfolgenden Unterkapiteln werden verschiedene Implementierungsaspekte bezüglich lokaler Ressourcen behandelt. Dazu wird zunächst auf die Repräsentation und die Definition lokaler Ressourcen-Bilanzierungs-Gruppen eingegangen (Abschnitt 7.3.1 und 7.3.2). Anschließend wird beschrieben, wie lokale Ressourcen bilanziert werden (Abschnitt 7.3.3).

54. Der einzige Unterschied zur Definition in Beispiel 7.3 besteht darin, daß *?Rechner* eine Variable und *Rechner_1* eine Konstante ist.

7.3.1 Die Repräsentation lokaler Ressourcen-Bilanzierungs-Gruppen

Jede lokale Ressourcen-Bilanzierungs-Gruppe besteht auf der Seite von TAXON aus einer Menge von Komponenten (*Komponenten-Gruppe*) und auf der Seite von TAXRES aus der Menge der zugehörigen Ressource-Variablen (*Ressource-Variablen-Gruppe*) von einer bestimmten Ressource (siehe Bild 7.2). Dabei wird intern in TAXRES jeder Ressource-Variablen-Gruppen ein eindeutiger Zugriffsschlüssel (→ interner Name der Gruppe) zugeordnet.

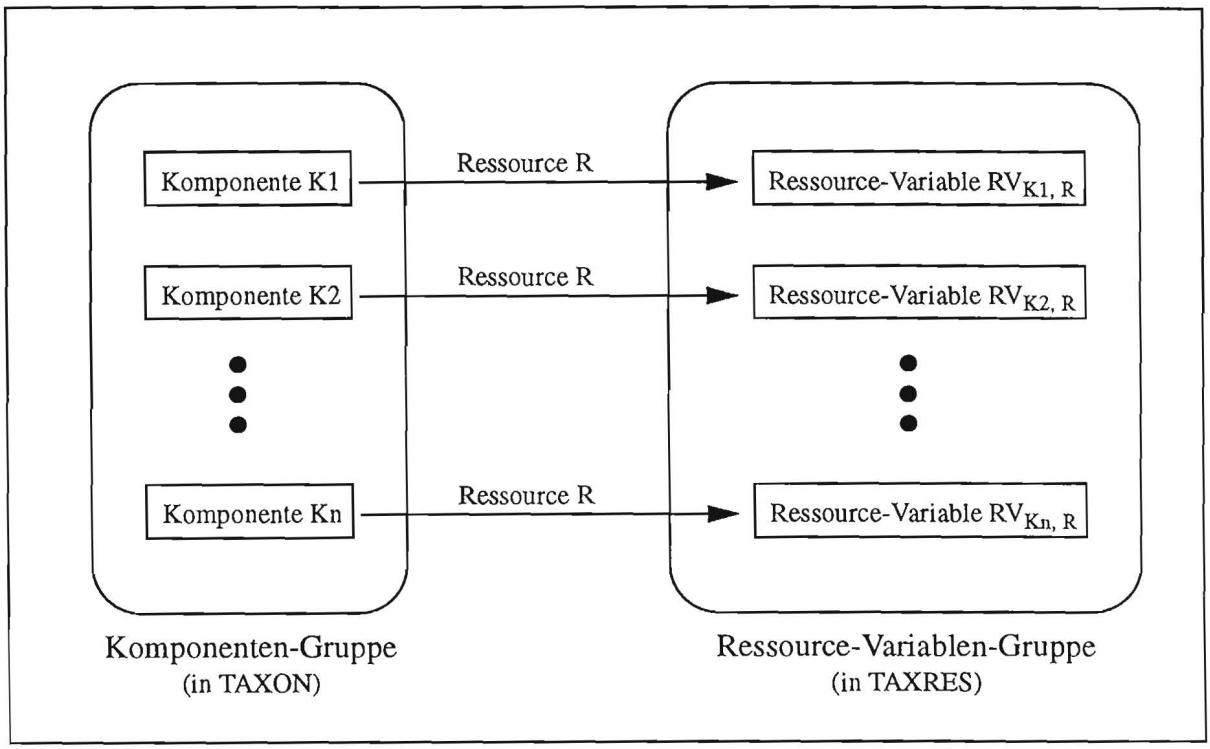


Bild 7.2: Die Repräsentation einer lokalen Ressourcen-Bilanzierungs-Gruppe

7.3.2 Was passiert durch den Befehl "Def-Local-Resource-Group"?

Wenn mit *Def-Local-Resource-Group* eine lokale Bilanzierungs-Gruppe definiert wird, dann wird daraufhin eine Observer-Regel[55] generiert. Diese Observer-Regel nennt TAXRES den Namen jeder Ressource-Variablen von der angegebenen Ressource, die zu einer Komponente aus der lokalen Bilanzierungs-Gruppe gehört. Eine solche Ressource-Variable wird dann von TAXRES in die Ressource-Variablen-Gruppe mit dem Zugriffsschlüssel (⟨Ressource-Name⟩, ⟨Group-Name-Variable-List⟩) eingetragen.

55. siehe Kapitel 2.1.1

Beispiel 7.6:

Um beispielsweise auszudrücken, daß für alle in der A-Box vorhandenen Rechner jeweils die Ressource Leistung lokal ausbilanziert werden soll (vgl. Beispiel 7.1), würde man z. B. folgendes in TAXRES eintragen:

(Def-Local-Resource-Group (?R)
(?K = Leistung => ?RV)
((?K = gehört_zu => ?R) (?R :in Rechner))).

Aufgrund dieser Definition wird automatisch eine Observer-Regel mit folgender *Observer-Condition*[56] generiert:

((?K = Leistung => ?RV) (?K = gehört_zu => ?R) (?R :in Rechner))

Wenn irgendeine Komponente ?K in der A-Box diese Bedingungen erfüllt, dann wird die Ressource-Variable ?RV in TAXRES in die Ressource-Variablen-Gruppe mit dem Zugriffsschlüssel *(Leistung, ?R)* eingetragen[57].

7.3.3 Die Bilanzierung lokaler Ressourcen

Die Ausbilanzierung einer lokalen Ressourcen-Bilanzierungs-Gruppe funktioniert im Prinzip genauso wie bei globalen Ressourcen (siehe Kapitel 5.2.3). Der Hauptunterschied besteht darin, daß hier nur auf einer Teilmenge der A-Box gearbeitet wird.

Zur Ausbilanzierung einer lokalen Ressourcen-Bilanzierungs-Gruppe kann es in günstigen Fällen schon genügen, wenn man einfach eine bereits in der A-Box vorhandene Komponente dieser Gruppe zuordnet. In diesen Fällen braucht man zur lokalen Ausbilanzierung weder ein in der A-Box vorhandenes Oberkonzept einzuschränken noch eine neue Komponente in die A-Box einzuführen.

Wenn bei einer lokalen Bilanzierungs-Gruppe mit den Ressourcen-Erzeuger-Variablen ($?RV_1$, $?RV_2$, ... , $?RV_n$) noch eine gewisse Ressourcen-Menge fehlt, dann wird mit Hilfe eines Agenda-Operators eine neue Komponenten-Variable angefordert, die einerseits von der fehlenden Ressource etwas zur Verfügung stellt und andererseits Element der betreffenden lokalen Bilanzierungs-Gruppe ist. In einem solchen Fall würde TAXRES deshalb einen Operator auf die Agenda schreiben, der prinzipiell folgende Struktur aufweist:

(AND *(?K* **:in** *R-Supplier-Object)*
(?K = R => ?RV)
(?RV **:unequal** *$?RV_1$)* *(?RV* **:unequal** *$?RV_2$)* ... *(?RV* **:unequal** *$?RV_n$)*
⟨Alle bei Def-Local-Resource-Group angegebenen Kontext-Bedingungen, wobei jedes Vorkommen der Komponenten-Variable durch ?K und alle Vorkommen einer Group-Name-Variable durch die entsprechende Konstante ersetzt wird.⟩)

56. Die *Observer-Condition* ist eine Bedingung, die erfüllt sein muß, damit die in der betreffenden Observer-Regel angegebene Anweisung ausgeführt wird (siehe [Wache94]).
57. Dabei ist zu beachten, daß eine solche Observer-Regel erst dann feuert, wenn die Komponenten-Variablen in der Observer-Condition mit Komponenten-Konstanten unifiziert wurden (siehe Kapitel 2.1.3).

Beispiel 7.7:

Es sei eine lokale Ressourcen-Bilanzierungs-Gruppe für die Ressource Leistung wie in Beispiel 7.6 definiert. In der A-Box sei eine Komponenten-Gruppe Rechner-1 mit den Komponenten K-1, K-2 und K-3 vorhanden, die jeweils eine bestimmte Menge von der Ressource Leistung zur Verfügung stellen. Die zugehörigen Ressource-Variablen seien ?RV-1, ?RV-2, und ?RV-3. Um in dieser Konstellation ein Ressource-Defizit bei der Ressource Leistung in der Komponenten-Gruppe Rechner-1 lokal zu beseitigen, würde TAXRES folgenden Operator auf die Agenda schreiben:

(AND (?K-4 :in Leistung-Supplier-Object)
(?K-4 = Leistung => ?RV-4)
(?RV-4 :unequal ?RV-1) (?RV-4 :unequal ?RV-2) (?RV-4 :unequal ?RV-3)
(?K-4 = gehört_zu => Rechner-1) (Rechner-1 :in Rechner)) ,

wobei ?K-4 und ?RV-4 automatisch generierte neue Symbol-Namen sind.

8. Vorschläge für Verbesserungen und Erweiterungen

Abgesehen davon, daß für die Anwendung des TOOCON-Werkzeugkastens in der Praxis eine effizientere Implementierung (z. B. in C oder C++)[1] sinnvoll wäre, gibt es aber auch noch grundsätzliche Möglichkeiten dieses wissensbasierte Konfigurierungssystem zu verbessern und zu erweitern, die aber den Rahmen dieser Diplomarbeit sprengen würden. Einige dieser Möglichkeiten werden in den folgenden Unterkapiteln aber zumindest ansatzweise vorgestellt.

8.1 Verarbeitung von Ressourcen-Abhängigkeiten

Gerade bei abhängigen Ressourcen treten immer wieder eine Vielzahl von Problemen auf. Man betrachte z. B. die Formel P=U*I . Wenn man sowohl P als auch I als Ressourcen modellieren will, hat man folgendes Problem: Immer, wenn etwas von der Ressource I verbraucht wird, dann wird auch automatisch etwas von der Ressource P verbraucht. Nun könnte man deshalb fordern, daß grundsätzlich bei jedem elektrischen Verbraucher sowohl I als auch P angegeben werden muß. Diese Forderung wäre dann aber nicht immer erfüllbar, wie z. B. bei einem Lastwiderstand R_L. Die Leistungsaufnahme eines Lastwiderstandes kann man nämlich erst dann bestimmen, wenn man weiß, an welche Spannung oder Strom er angeschlossen wird.

Für solche Fälle sollte es möglich sein, anstelle einer konkreten Ressource-Menge ein Constraint anzugeben, durch das beschrieben wird, wie der Ressourcen-Verbrauch berechnet wird. Sobald die entsprechende Ressourcen-Informationen bekannt sind, wird der Ressourcen-Verbrauch berechnet und in die A-Box eingetragen, wodurch indirekt eine Ressourcen-Bilanzierung angestoßen wird.

Für einen ohmschen Widerstand könnte man z. B. folgenden Constraint eintragen: $P=U^2/R$. Wenn dieser Wiederstand dann irgendwann an eine bestimmte Spannung angeschlossen wird, kann der Leistungsverbrauch nach diesem Constraint ausgerechnet und anschließend in die A-Box eingetragen werden.

In TOOCON gibt es zwar Constraintsolver (z. B. CONTAX oder EPILYTIS), aber derzeit kann eine Variable immer nur aus höchstens einem Konkreten Bereich sein, aber z. B. nicht gleichzeitig in TAXRES und EPILYTIS. Hierfür wäre eine Integration verschiedener Constraint-Mechanismen notwendig.

8.2 Harte und weiche Constraints

Angenommen, ein menschlicher Experte soll nach bestimmten Spezifikationen ein technisches System (z. B. ein Computer-Netzwerk) konfigurieren, das z. B. maximal 5.000 DM kosten darf. Wenn dieser Experte eine Konfigurationslösung finden würde, die zwar alle technischen Spezifikationen erfüllt, aber z. B. 5.010 DM kosten würde, dann würde er dem Kunden wahrscheinlich diese Lösung trotzdem vorschlagen, obwohl sie etwas teurer ist.

---

1. Derzeit ist der TOOCON-Werkzeugkasten komplett in LISP [Steele90] bzw. CLOS [Keene89] implementiert.

Anhand dieses Beispiels erkennt man, daß es sinnvoll wäre, wenn man bei der Aufgabenspezifikation auch *weiche Constraints* verwenden könnte. Weiche Constraints sollten zwar erfüllt sein, müssen es aber nicht unbedingt. Dennoch sollte man versuchen, diese Constraints möglichst gut zu erfüllen. Dazu könnte man den weichen Constraints durch Gewichte unterschiedliche Prioritäten zuordnen[2].

8.3 Übergang von der Konfigurierung zur Konstruktion durch das Verbindungskonzept

Angenommen, in der Lösungsmenge (bzw. in einer betrachteten Teilmenge davon)[3] seien einige Komponenten, die von der Ressource R eine gewisse Menge zur Verfügung stellen und andere, die davon etwas verbrauchen. Nach dem bisherigen Konzept gilt diese Ressource R genau dann als ausbilanziert, wenn durch die Komponenten in der Lösungs(teil)menge insgesamt mindestens soviel von der Ressource R zur Verfügung steht, wie in der Lösungs(teil)menge benötigt wird. Hierbei ist es nur wichtig, daß genügend viel von der Ressource R vorhanden ist.

Bei dieser Betrachtung wird allerdings völlig vernachlässigt, daß sich in der Realität Ressourcen nur dann ausgleichen können, wenn zwischen ihnen eine *geeignete Verbindung* besteht. So können sich z. B. die Ladungen zwischen einem elektrischen Verbraucher und einer Batterie nur dann ausgleichen, wenn zwischen beiden eine elekrische Verbindung (z. B. in Form von zwei Drähten) existiert.

Um so etwas modellieren zu können, ist eine Konzept-Erweiterung notwendig. Bei diesem erweiterten Konzept (*Verbindungskonzept*) geht man davon aus, daß es nicht selbstverständlich ist, daß man gleichartige Ressourcen immer miteinander (z. B. über irgendwelche Adapter) verbinden kann[4], sondern daß die Verbindung zwischen Ressourcen eine wesentliche Information darstellt (siehe Beispiel 8.1).

2. vgl. Dissertation [Meyer94]
3. Bei lokalen Ressourcen betrachtet man Teilmengen der gesamten Lösungsmenge (siehe Kapitel 7).
4. Es ist z. B. so, daß man nicht jeden Speicherbaustein in jedes beliebige Motherboard stecken kann. In dem Fall gibt meistens keinen Adapter, da dieser große technische Schwierigkeiten verursachen würde (z. B. Signal-Laufzeit-Probleme) und/oder teurer wäre als ein passender Speicherbaustein.

Beispiel 8.1:

Gegeben sei eine A-Box mit einem Motherboard, einer Grafikkarte und einem Speicherbaustein mit folgenden bereitgestellten bzw. geforderten Ressourcen:

Komponente	Ressourcen-Bereitstellung	Ressourcen-Forderung
Motherboard M	1 Steckplatz vom Typ A	Speicherkapazität der Menge X
Grafikkarte G -	1 Steckplatz vom Typ B (für optionale Aufrüstung mit zusätzlichem Speicher)	Keine Ressourcen-Forderung, da auf der Grafikkarte onboard bereits genügend Speicher vorhanden ist.
Speicherbaustein S1	Speicherkapazität der Menge X	1 Steckplatz vom Typ B

Wenn man hier die Steckplatz-Ressourcen und die Speicher-Ressourcen getrennt betrachtet, sind sie beide ausbilanziert, d. h. es werden mindestens soviel Ressourcen bereitgestellt, wie gefordert.

Allerdings nützt es in der Praxis wenig, wenn sich der vom Motherboard geforderte Speicherplatz auf der Grafikkarte befindet. Daher kann man mit dieser Art von Ressourcen-Bilanzierung hier nichts anfangen. Um dieses Problem zu lösen, muß man in der Modellierung berücksichtigen, daß hier Steckplatz und Speicherkapazität auf eine bestimmte Art und Weise zusammengehören.

Beim *Verbindungskonzept* kann sich z. B. die geforderte und bereitgestellte Speicherkapazität nur dann ausgleichen, wenn zwischen beiden eine *geeignete Verbindung* besteht. Aber damit eine solche Verbindung überhaupt aufgebaut werden kann, ist es notwendig, daß sowohl Ressourcen-Verbraucher als auch Ressourcen-Erzeuger jeweils über einen *freien und zueinander kompatiblen Anschluß* verfügen. Durch dieses Konzept wird berücksichtigt, ob der Austausch einer oder mehrerer Ressourcen überhaupt möglich ist.

Damit sollte man in Beispiel 8.1 einen Steckplatz nicht als Ressource, sondern besser als Anschluß(typ) modellieren. Zum Beispiel stellt der Speicherbaustein S1 über einen Anschluß vom Typ B von der Ressource Speicherkapazität die Menge X zur Verfügung. Da das Motherboard M und der Speicherbaustein S1 nur über inkompatible Anschlüsse verfügen, kann jetzt hier die Ressource Speicherkapazität nicht mehr so einfach ausbilanziert werden, wie oben. Zur Ausbilanzierung braucht man nun erst einen anderen Speicherbaustein S2, der über den entsprechenden Anschluß verfügt. Obiges Fehlverhalten wird also damit vermieden.

Zusatzbemerkung:

Als Ergebnis einer erfolgreichen rein ressourcen-orientierten Konfigurierung erhält man normalerweise nur eine Stückliste der benötigten Komponenten, die unter Umständen unübersichtlich viele Bauteile enthält.

Im Gegensatz dazu könnte bei der Verwendung des Verbindungskonzeptes noch zusätzlich ein Vorschlag generiert werden, wie die einzelnen Komponenten miteinander verbunden werden können. Das kann in manchen Fällen die nachfolgende Konstruktion sehr erleichtern. Daher geht diese Konzept-Erweiterung etwas über das ursprüngliche Ziel hinaus und bewegt sich schon in Richtung Konstruktion. Man könnte diese Erweiterung aber auch einfach nur als weiteren Test (zusätzliche Integritätsbedingung) oder als Plausibilitätskontrolle für den Benutzter verwenden, um zu überprüfen, ob eine generierte Konfiguration überhaupt realisierbar ist.

Auf jeden Fall sollte der Benutzer aber für jede einzelne Ressource selbst entscheiden können, ob er dieses Verbindungskonzept verwenden möchte oder nicht, d. h. ob eine Ressource beliebig oder nur über Anschlüsse und Verbindungen ausbilanziert werden kann.

8.3.1 Vergleich zwischen lokaler Ressourcen-Bilanzierung und dem Verbindungskonzept

Bei der lokalen Ressourcen-Bilanzierung betrachtet man für die Bilanzierung einiger bestimmter Ressourcen jeweils nur eine Teilmenge aller in der A-Box vorhandenen Komponenten. Beim Vergleich zwischen Ressourcen-Forderung und Ressourcen-Bereitstellung werden nur die Komponenten aus dieser Teilmenge (*Lokalmenge*) betrachtet.

Im Gegensatz dazu betrachtet man beim Verbindungskonzept jeweils Mengen von Anschlüssen, die miteinander verbunden sind. Die Ressourcen von Komponenten können sich nur dann ausgleichen, wenn es möglich ist, eine Verbindung zwischen ihnen herzustellen.

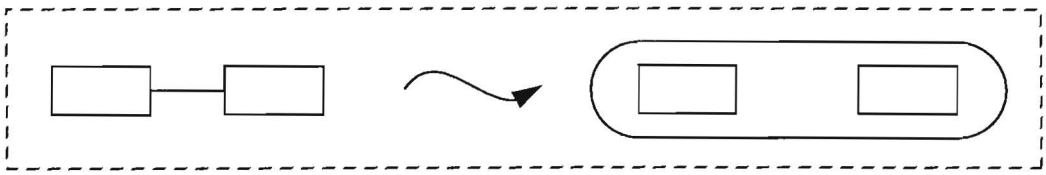
Die lokale Ressourcen-Bilanzierung und das Verbindungskonzept haben unterschiedliche Ansätze und Zielsetzungen. Mit Hilfe der lokalen Ressourcen-Bilanzierung soll man in der Lage sein, zusammengehörende Komponenten-Gruppen (wie z. B. eigenständige Geräte oder Module) getrennt auszubilanzieren. Das Verbindungskonzept hat einen anderen Ursprung: In der Praxis gibt es manchmal für gleiche Ressourcen unterschiedliche Anschlüsse. Ressourcen können sich aber nur dann ausgleichen, wenn ihre Anschlüsse zueinander kompatibel sind. Mit dem Verbindungskonzept hat man also die Möglichkeit, in natürlicher Weise den Zusammenhang zwischen Ressource und zugehörigem Anschluß zu modellieren.

Dennoch stellt sich hier die Frage, ob das Verbindungskonzept wirklich eine echte Erweiterung des bisherigen Konzeptes darstellt, oder ob man dieses Konzept irgendwie durch lokale Ressourcen simulieren kann. Umgekehrt ist dann natürlich auch zu klären, ob das Verbindungskonzept die lokale Ressourcen-Bilanzierung mit einschließt bzw. überflüssig macht. Beide Fragestellungen werden im nächsten Abschnitt diskutiert.

8.3.2 Kann man mit der lokalen Ressourcen-Bilanzierung das Verbindungskonzept simulieren bzw. macht das Verbindungskonzept lokale Ressourcen überflüssig?

Kann man das Verbindungskonzept mit lokalen Ressourcen vollständig simulieren?

Idee: Ersetze eine Verbindung durch eine Lokalmenge und bilanziere dann die Ressource und den Anschluß über diese Lokalmenge.



Dieses Verfahren mag zwar in vielen Fällen funktionieren, aber nicht immer. Es funktioniert z. B. nicht, wenn mehrere Komponenten benötigt werden, um eine bestimmte Ressource auszubilanzieren und diese auch noch unterschiedliche Anschlüsse haben. Ein weiterer Problemfall entsteht dann, wenn eine Komponente dieselbe Ressource über unterschiedliche Anschlüsse zur Verfügung stellt (siehe Beispiel 8.2).

Beispiel 8.2:

Annahme, man hat ein Netzteil N und einen elektrischen Verbraucher V mit folgenden Eigenschaften:

Table 1:

Komponente	Ressourcen-Bereitstellung	Ressourcen-Forderung
Netzteil N	1 Anschluß für 5V mit einer max. Leistungsabgabe von 5W 1 Anschluß für 12V mit einer max. Leistungsabgabe von 12W gesamte max. Leistungsabgabe 15W	
Verbraucher V		5V-Anschluß mit einer Leistungsabgabe von mindestens 12W

In diesem Beispiel gibt es zwar einen 5V-Anschluß und eine Leistungsressource von 12W, aber trotzdem ist hier das Netzteil N zusammen mit dem Verbraucher V nicht ausbilanziert. Außerdem gibt es hier noch das Problem, daß das Netzteil zweimal die gleiche Ressource hat. Deshalb wäre man in dem Fall sogar gezwungen, für die gleiche Ressource Leistung zwei verschiedene Namen zu vergeben, wie z. B. Leistung-bei-5V und Leistung-bei-12V. Man müßte also den Anschlußtyp in den Namen hineincodieren. Aber wie könnte man dann noch die Ressource Leistung bilanzieren? Außerdem müßte man bei dieser Art der Namensgebung Constraints der Form P=U*I für jede Spannung getrennt behandeln.

Im Gegensatz dazu könnte man beim Verbindungskonzept ohne Probleme bei ein und derselben Komponente den gleichen Ressourcetyp mehrfach verwenden, da sich eine Ressourcemengen-Angabe immer auf einen bestimmten Anschluß bezieht.

Bei der lokalen Ressourcen-Bilanzierung kann man nur Mengen von Komponenten (nicht Anschlüsse!) bezüglich bestimmter Ressourcen vollkommen isoliert von anderen Komponenten ausbilanzieren. Wenn mehrere Komponenten über unterschiedliche Anschlüsse dieselbe Ressource R austauschen, kann man diese Komponenten bezüglich der Ressource R nicht in getrennten Lokalmengen ausbilanzieren. In solchen Fällen kann man das Verbindungskonzept nicht durch lokale Ressourcen simulieren.

Macht das Verbindungskonzept lokale Ressourcen überflüssig?

Beim Verbindungskonzept kann man jede einzelne Verbindung als Menge miteinander verbundener Anschlüsse (nicht nur Komponenten!) betrachten. Das Verbindungskonzept operiert also auf einer kleineren Granulat-Ebene und ist damit viel genauer als die lokale Ressourcen-Bilanzierung.

Bei der Modellierung mit lokalen Ressourcen werden alle verwendeten Komponenten in verschiedene separat auszubilanzierende Teilmengen (Komponenten-Gruppen) eingeordnet, wodurch *grobe Strukturen* repräsentiert werden. Im Gegensatz dazu entstehen bei einer Konfigurierung mit dem Verbindungskonzept durch die explizit generierten Verbindungen *feine Strukturen* zwischen den einzelen Komponenten. Bild 8.1 versucht die Unterschiede zwischen beiden Modellierungsarten graphisch zu verdeutlichen.

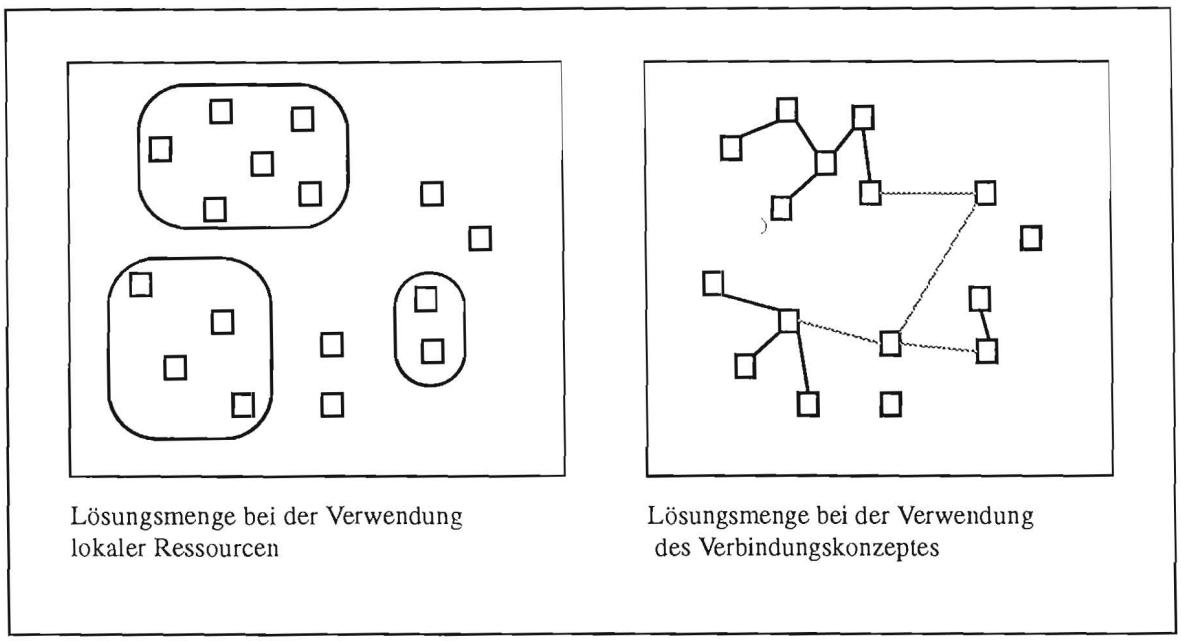


Bild 8.1: Graphischer Vergleich zwischen der Modellierung mit lokalen Ressourcen und der Modellierung mit dem Verbindungskonzept

Mit dem Verbindungskonzept können zwar feine Strukturen repräsentiert werden, es ist aber im allgemeinen für die Charakterisierung und eindeutige Abgrenzung größerer zusammengehörender Komponenten-Gruppen (z. B. einzelne Computer in einem Netzwerk) nicht geeignet. Denn wie sollte man mit diesem Konzept adäquat ausdrücken, daß jeder einzelne Computer (im zu konfigurierenden Netzwerk) ein eigenes Netzteil haben muß bzw. daß nicht alle elektrischen Verbraucher mit einem zentralen Netzteil verbunden werden dürfen?

Daher ist das Verbindungskonzept kein Ersatz für die Modellierung mit lokalen Ressourcen, sondern eine Ergänzung!

9. Zusammenfassung

In dieser Diplomarbeit wurden der struktur-orientierte und der ressourcen-orientierte Konfigurierungsansatz vorgestellt und miteinander kritisch verglichen. Es wurde gezeigt, daß beide Ansätze jeweils Vor- und Nachteile haben und nicht für alle Teile einer Modellierungsaufgabe gleich gut geeignet sind. Aus dieser Feststellung leitete sich die Motivation ab, beide Ansätze in einem Konfigurierungssystem miteinander zu integrieren, damit ein Wissensingenieur die Vorteile beider Modellierungsarten gleichzeitig nutzen kann.

Dazu wurde untersucht, ob in dem bereits vorhandenen struktur-orientierten Konfigurierungssystem TOOCON ressourcen-orientierte Techniken irgendwie integriert werden können. Dabei wurde zunächst festgestellt, daß die Ausdrucksmächtigkeit der Tools, die in diesem System schon enthalten waren, nicht ausreichte, um ressourcen-orientierte Techniken zu simulieren.

Allerdings konnte dann konzeptionell und durch die zugehörige praktische Implementierung gezeigt werden, wie sich ressourcen-orientierte Techniken in Form eines neuen Tools (das mit dem Namen TAXRES bezeichnet wurde) als Konkreter Bereich in den TOOCON-Werkzeugkasten integrieren lassen. Dabei wurde festgestellt, daß die reine Konkrete-Bereiche-Schnittstelle zwischen TAXRES und dem terminologischen Systen TAXON für ressourcen-orientiertes Konfigurieren nicht ausreicht. Daher wurde die Konkrete-Bereiche-Schnittstelle zu TAXON entsprechend erweitert und eine Schnittstelle zur Steuerungseinheit CONTROL hinzugefügt. Aus diesem Grund ist TAXRES nicht einfach nur ein normaler Konkreter Bereich, sondern eher ein *"Meta" - Konkreter Bereich.*

Durch diese Integration ressourcen-orientierter Techniken mit einem terminologischen System konnte hier eine Ausdrucksmöglichkeit und eine Verzahnung unterschiedlicher Inferenz-Prozesse erreicht werden, die weder bei einem rein ressourcen-orientierten noch bei einem rein struktur-orientierten Konfigurierungssystem vorhanden sind.

Anhand eines etwas ausführlicheren Beispiels aus einer PC-Domäne wurde demonstriert, wie die neue Ausdrucksmächtigkeit des erweiterten TOOCON-Systems in der Praxis angewendet werden kann. Bei diesem Beispiel wurden struktur- und ressourcen-orientierte Techniken gleichzeitig eingesetzt, um eine Konfigurierungsaufgabe zu modellieren und zu lösen.

Anschließend wurde gezeigt, daß es Anwendungsbeispiele gibt, bei denen die vorhandenen Ressourcen nicht nur global über alle Komponenten in der Lösungsmenge, sondern teilweise auch lokal innerhalb bestimmter Komponenten-Gruppen, ausbilanziert werden sollen. Damit wurde die Nützlichkeit *lokaler Ressourcen* motiviert und deshalb TAXRES entsprechend erweitert.

Nachfolgende Tabelle 9.1 zeigt zum Abschluß noch einen zusammenfassenden Vergleich zwischen einem rein struktur- bzw. ressourcen-orientierten Konfigurierungssystem und dem um TAXRES erweiterten TOOCON-System:

	ein rein **struktur-orientiertes Konfigurierungs-system**	ein rein **ressourcen-orientiertes Konfigurierungs-system**	**bisheriges TOOCON-System + TAXRES**
Komponenten-wissen:	Strukturierung des Komponentenwissens in einer Begriffshierarchie: - Beschreibung der Aufbaustruktur von Komponenten(-Gruppen) durch andere Komponenten mit Hilfe von "HAS-Part"- Beziehungen - Spezialisierung von allgemeineren Begriffen durch Verwendung von "IS-A"-Beziehungen	Modellierung der Funktionalität jeder vorhandenen Komponente durch die Angabe von bereitgestellten bzw. geforderten Ressourcen	um Konkrete Bereiche erweitertes Terminologisches System: - Repräsentation der Komponenten durch Konzepte - Strukturierung des Komponentenwissens durch allgemeine Attribut- und Rollen-Beziehungen zwischen den Konzepten und Attribut-Beziehungen von Konzepten zu Konkreten Prädikaten - Einordnung der Konzepte in eine "IS-A"- Taxonomie + - Ressourcen-Informationen bei den einzelnen Konzepten
Integritäts-bedingungen:	Integritätsbedingungen bezüglich der Aufbaustruktur der Komponenten(-Gruppen)	Ressourcen-Bilanzierungs-Bedingungen	taxonomische Integritätsbedingungen, Constraints über endlichen Wertebereichen, algebraische Constraints, räumliche Constraints + globale und lokale Ressourcen-Bilanzierungs-Bedingungen
die wichtigsten Inferenzen:	Zerlegung, Parametrierung, Spezialisierung	Ressourcen-Bilanzierung	Zerlegung, Parametrierung, Spezialisierung + Ressourcen-Bilanzierung

Tabelle 9.1: Vergleich zwischen einem rein struktur- bzw. ressourcen-orientierten Konfigurierungssystem und dem um TAXRES erweiterten TOOCON-System

	ein rein **struktur-orientiertes Konfigurierungs-system**	ein rein **ressourcen-orientiertes Konfigurierungs-system**	**bisheriges TOOCON-System + TAXRES**
Vorteile:	- strukturierte Lösung durch semantische Beziehungen zwischen den Komponenten - Erklärbarkeit der Lösungsfindung - Effizienz durch strukturierte Suche	- Bei den einzelnen Komponenten braucht nur angegeben zu werden, wieviel sie von welcher Ressource bereitstellen bzw. fordern. - Die Struktur der zu konfigurierenden Lösung muß nicht vorgegeben werden.	Der struktur- und der ressourcen-orientierte Konfigurierungsansatz können miteinander kombiniert werden: - struktur- und ressourcen-orientierte Integritätsbedingungen - Zur Beseitigung von Ressourcen-Defiziten können Oberkonzepte verwendet und dadurch in vielen Fällen Backtracking-Operationen vermieden werden.
			Bei einer ressourcen-orientierten Konfigurierung können durch lokale Ressourcen grobe Strukturen in Form von zusammengehörenden Komponenten-Gruppen modelliert werden.
Nachteile:	Der Komponentenkatalog muß explizit strukturiert werden, wodurch weitgehend auch die Struktur einer potentiellen Lösung vorgegeben wird.	- Vorhandenes Struktur-Wissen läßt sich zum Teil nur sehr inadäquat modellieren. - Es können nur unstrukturierte Konfigurationslösungen generiert werden.	- kompliziertere Modellierungsmöglichkeiten - aufwendigere Inferenzmechanismen

Tabelle 9.1: Vergleich zwischen einem rein struktur- bzw. ressourcen-orientierten Konfigurierungssystem und dem um TAXRES erweiterten TOOCON-System (Forts.)

Anhang A: Ergänzung zum Anwendungsbeispiel aus Kapitel 6

Dieser Anhang ist die Ergänzung zu dem Anwendungsbeispiel aus Kapitel 6. In Form einer relativ ausführlichen Modellierungsbeschreibung und durch kommentierte Auszüge aus dem Protokoll einer Beispielsitzung wird gezeigt, wie mit dem erweiterten TOOCON-Werkzeugkasten eine Konfigurierung durchgeführt werden kann:

Die Initialisierung des TOOCON-Werkzeugkastens:

```
(DESTROY-TOOCON)
```

Mit diesem Befehl werden alle geladene Tools von TOOCON (einschließlich TAXRES) in einen definierten Grundzustand versetzt.

Deklarierung der Ressourcen:

```
(DECLARE-STANDARD-RESOURCE-TYPES HD-Speicher
                                 RAM-Speicher RAM-Steckplaetze
                                 ISA-Slots VLB-Slots PCI-Slots
                                 IDE-Ports SCSI-Ports
                                 FD-Ports
                                 Gehaeuse-Einschuebe
                                 Leistung )

(DECLARE-ENVIRONMENT-RESOURCE-TYPES Geld)
(DEFINE-ENVIRONMENT-CONC)
```

Definition der Konkreten Prädikate für die nachfolgenden Konzept-Definitionen:

An dieser Stelle werden alle die in den nachfolgenden Konzept-Definitionen vorkommenen Konkreten Prädikate definiert. Da aber diese Definitionen zum einen insgesamt recht umfangreich und zum anderen völlig analog zu denen in Kapitel 6.2 sind, werden sie hier nicht abgedruckt.

Definition der verwendteten Konzepte:

Zu jedem der in Tabelle 6.1 aufgeführten Komponenten(typen) werden nachfolgend nur jeweils ein paar wenige einfache Beispiel-Konzepte und die zugehörigen Oberkonzepte definiert:

```
;---------------------------------------------------------------
;Definition einiger Netzteile:

(CONC Netzteil (AND (SOME (Leistung) Leistung-Plus)
                    (SOME (Geld) Geld-Minus) ))

(CONC Netzteil-100 (AND Netzteil
                        (SOME (Leistung) Leistung-Plus-100)
                        (SOME (Geld) Geld-Minus-130) ))

(CONC Netzteil-150 (AND Netzteil
                        (SOME (Leistung) Leistung-Plus-150)
                        (SOME (Geld) Geld-Minus-150) ))

(CONC Netzteil-200 (AND Netzteil
                        (SOME (Leistung) Leistung-Plus-200)
```

```
                        (SOME (Geld) Geld-Minus-180) ))

;-------------------------------------------------------------
;Definition einiger Gehaeuse:

(attr hat-Netzteil)

(CONC Gehaeuse (AND (SOME (hat-Netzteil) Netzteil)
                    (SOME (Gehaeuse-Einschuebe) Gehaeuse-Einschuebe-Plus)
                    (SOME (Geld) Geld-Minus) ))

(CONC Gehaeuse-2 (AND Gehaeuse
                      (SOME (Gehaeuse-Einschuebe) Gehaeuse-Einschuebe-Plus-2)
                      (SOME (Geld) Geld-Minus-100) ))

(CONC Gehaeuse-3 (AND Gehaeuse
                      (SOME (Gehaeuse-Einschuebe) Gehaeuse-Einschuebe-Plus-3)
                      (SOME (Geld) Geld-Minus-130) ))

;-------------------------------------------------------------
;Definition einiger Prozessoren:

(prim Prozessor-Objekt) ;repraesentiert alle Prozessor-Eigenschaften

(CONC Prozessor (AND Prozessor-Objekt (SOME (Geld) Geld-Minus)))

(CONC i486DX-33 (AND Prozessor (SOME (Geld) Geld-Minus-200)))
(CONC i486DX/2-66 (AND Prozessor (SOME (Geld) Geld-Minus-300)))
(CONC i486DX/4-100 (AND Prozessor (SOME (Geld) Geld-Minus-400)))
(CONC Pentium-90 (AND Prozessor (SOME (Geld) Geld-Minus-1000)))

;-------------------------------------------------------------
;Definition einiger RAM-Module:

(CONC RAM-Modul (AND (SOME (RAM-Speicher) RAM-Speicher-Plus)
                     (SOME (RAM-Steckplaetze) RAM-Steckplaetze-Minus-1)
                     (SOME (Geld) Geld-Minus) ))

(CONC 4MB-RAM (AND RAM-Modul
                   (SOME (RAM-Speicher) RAM-Speicher-Plus-4)
                   (SOME (Geld) Geld-Minus-300) ))

(CONC 8MB-RAM (AND RAM-Modul
                   (SOME (RAM-Speicher) RAM-Speicher-Plus-8)
                   (SOME (Geld) Geld-Minus-600) ))

(CONC 16MB-RAM (AND RAM-Modul
                    (SOME (RAM-Speicher) RAM-Speicher-Plus-16)
                    (SOME (Geld) Geld-Minus-1000) ))

;-------------------------------------------------------------
;Definition einiger Grafikkarten:

(prim Grafikkarten-Objekt) ;repraesentiert alle Grafikkarten-Eigenschaften

(CONC Grafikkarte (AND Grafikkarten-Objekt
                       (SOME (Geld) Geld-Minus)
                       (SOME (Leistung) Leistung-Minus) ))

(CONC ISA-Grafikkarte (AND Grafikkarte (SOME (ISA-Slots) ISA-Slots-Minus-1)))
(CONC VLB-Grafikkarte (AND Grafikkarte (SOME (VLB-Slots) VLB-Slots-Minus-1)))
```

```
(CONC PCI-Grafikkarte (AND Grafikkarte (SOME (PCI-Slots) PCI-Slots-Minus-1)))

(CONC Grafikkarte-Typ-A (AND ISA-Grafikkarte
                             (SOME (Leistung) Leistung-Minus-10)

(CONC Grafikkarte-Typ-B (AND VLB-Grafikkarte
                             (SOME (Leistung) Leistung-Minus-15)
                             (SOME (Geld) Geld-Minus-400) ))

(CONC Grafikkarte-Typ-C (AND PCI-Grafikkarte
                             (SOME (Leistung) Leistung-Minus-20)
                             (SOME (Geld) Geld-Minus-500) ))

;-------------------------------------------------------------
;Definition einiger Motherboards:

(prim Motherboard-Objekt)
(attr hat-CPU hat-RAM hat-Grafikkarte)

(CONC Motherboard (AND Motherboard-Objekt
                       (SOME (hat-CPU) Prozessor)
                       (SOME (hat-RAM) RAM-Modul)
                       (SOME (hat-Grafikkarte) Grafikkarte)
                       (SOME (Geld) Geld-Minus)
                       (SOME (Leistung) Leistung-Minus) ))

(CONC Motherboard-Typ-A (AND Motherboard
                             (SOME (Geld) Geld-Minus-300)
                             (SOME (RAM-Steckplaetze) RAM-Steckplaetze-Plus-4)
                             (SOME (Leistung) Leistung-Minus-50)
                             (SOME (ISA-Slots) ISA-Slots-Plus-5)
                             (SOME (IDE-Ports) IDE-Ports-Plus-2) ))

(CONC Motherboard-Typ-B (AND Motherboard
                             (SOME (Geld) Geld-Minus-500)
                             (SOME (RAM-Steckplaetze) RAM-Steckplaetze-Plus-4)
                             (SOME (Leistung) Leistung-Minus-50)
                             (SOME (ISA-Slots) ISA-Slots-Plus-3)
                             (SOME (PCI-Slots) PCI-Slots-Plus-2)
                             (SOME (IDE-Ports) IDE-Ports-Plus-2)
                             (SOME (SCSI-Ports) SCSI-Ports-Plus-7) ))

;-------------------------------------------------------------
;Definition eines ISA-SCSI-Controllers:

(CONC ISA-SCSI-Controller (AND (SOME (Geld) Geld-Minus)
                               (SOME (Leistung) Leistung-Minus)
                               (SOME (ISA-Slots) ISA-Slots-Minus-1)
                               (SOME (SCSI-Ports) SCSI-Ports-Plus-7) ))

(CONC ISA-SCSI-Controller-Typ-08/15 (and ISA-SCSI-Controller
                                         (SOME (Geld) Geld-Minus-400)
                                         (SOME (Leistung) Leistung-Minus-10) ))

;-------------------------------------------------------------
```

```
;Definition einiger Festplatten:

(prim Festplatten-Objekt)

(CONC Festplatte (AND Festplatten-Objekt
                      (SOME (Leistung) Leistung-Minus)
                      (SOME (HD-Speicher) HD-Speicher-Plus)
                      (SOME (Gehaeuse-Einschuebe) Gehaeuse-Einschuebe-Minus-1)
                      (SOME (Geld) Geld-Minus) ))

(CONC IDE-Festplatte (AND Festplatte (SOME (IDE-Ports) IDE-Ports-Minus-1)))
(CONC SCSI-Festplatte (AND Festplatte (SOME (SCSI-Ports) SCSI-Ports-Minus-1)))


(CONC Festplatte-Typ-A (AND IDE-Festplatte
                             (SOME (Leistung) Leistung-Minus-10)
                             (SOME (HD-Speicher) HD-Speicher-Plus-400)
                             (SOME (Geld) Geld-Minus-300) ))

(CONC Festplatte-Typ-B (AND SCSI-Festplatte
                             (SOME (Leistung) Leistung-Minus-10)
                             (SOME (HD-Speicher) HD-Speicher-Plus-500)
                             (SOME (Geld) Geld-Minus-400) ))

(CONC Festplatte-Typ-C (AND SCSI-Festplatte
                             (SOME (Leistung) Leistung-Minus-15)
                             (SOME (HD-Speicher) HD-Speicher-Plus-1000)
                             (SOME (Geld) Geld-Minus-1100) ))

;-------------------------------------------------------------
;Definition eines Monitors:

(prim Monitor-Objekt)

(conc Monitor (AND Monitor-Objekt
                   (SOME (Geld) Geld-Minus) ))

(conc Monitor-Typ-XY (AND Monitor-Objekt
                          (SOME (Geld) Geld-Minus-1000)))

;-------------------------------------------------------------
;Einfache strukturelle Definion eines Personal-Computers:

(attr hat-Gehaeuse hat-Motherboard hat-Festplatte hat-Monitor)

(conc Personal-Computer
      (AND (SOME (hat-Gehaeuse) Gehaeuse)
           (SOME (hat-Motherboard) Motherboard)
           (SOME (hat-Festplatte) Festplatte)
           (SOME (hat-Monitor) Monitor) ))
```

Pre-Compilation der T-Box:

Um CONTROL bei der Realisierung von Oberkonzepten unterstützen zu können, wird mit dem Befehl

```
(init-TAXRES)
```

für jedes terminale Konzept der T-Box berechnet, wieviel es von welcher Ressource fordert bzw. bereitstellt (siehe Unterkapitel 5.3.4).

Die T-Box am Ende dieser Konzept-Definitionen:

```
> (pphi)

                                  TOP <===  (HD-SPEICHER-SUPPLIER-OBJECT)
                                            (RAM-SPEICHER-SUPPLIER-OBJECT)
                                            (RAM-STECKPLAETZE-SUPPLIER-OBJECT)
                                            (ISA-SLOTS-SUPPLIER-OBJECT)
                                            (VLB-SLOTS-SUPPLIER-OBJECT)
                                            (PCI-SLOTS-SUPPLIER-OBJECT)
                                            (IDE-PORTS-SUPPLIER-OBJECT)
                                            (SCSI-PORTS-SUPPLIER-OBJECT)
                                            (GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT)
                                            (LEISTUNG-SUPPLIER-OBJECT)
                                            (ENVIRONMENT-CONCEPT)
                                            (PROZESSOR-OBJEKT)
                                            (GRAFIKKARTEN-OBJEKT)
                                            (MOTHERBOARD-OBJEKT)
                                            (FESTPLATTEN-OBJEKT)
                                            (MONITOR-OBJEKT)
                                            (PERSONAL-COMPUTER)
          HD-SPEICHER-SUPPLIER-OBJECT <===  (FESTPLATTE)
                             FESTPLATTE <===  (IDE-FESTPLATTE)
                                            (SCSI-FESTPLATTE)
                       IDE-FESTPLATTE <===  (FESTPLATTE-TYP-A)
                    FESTPLATTE-TYP-A <===  (BOTTOM)
                     SCSI-FESTPLATTE <===  (FESTPLATTE-TYP-B)
                                            (FESTPLATTE-TYP-C)
                    FESTPLATTE-TYP-B <===  (BOTTOM)
                    FESTPLATTE-TYP-C <===  (BOTTOM)
        RAM-SPEICHER-SUPPLIER-OBJECT <===  (RAM-MODUL)
                           RAM-MODUL <===  (4MB-RAM)
                                            (8MB-RAM)
                                            (16MB-RAM)
                             4MB-RAM <===  (BOTTOM)
                             8MB-RAM <===  (BOTTOM)
                            16MB-RAM <===  (BOTTOM)
    RAM-STECKPLAETZE-SUPPLIER-OBJECT <===  (MOTHERBOARD-TYP-A)
                                            (MOTHERBOARD-TYP-B)
                  MOTHERBOARD-TYP-A <===  (BOTTOM)
                  MOTHERBOARD-TYP-B <===  (BOTTOM)
           ISA-SLOTS-SUPPLIER-OBJECT <===  (MOTHERBOARD-TYP-A)
                                            (MOTHERBOARD-TYP-B)
           VLB-SLOTS-SUPPLIER-OBJECT <===  (BOTTOM)
           PCI-SLOTS-SUPPLIER-OBJECT <===  (MOTHERBOARD-TYP-B)
           IDE-PORTS-SUPPLIER-OBJECT <===  (MOTHERBOARD-TYP-A)
                                            (MOTHERBOARD-TYP-B)
         SCSI-PORTS-SUPPLIER-OBJECT <===  (MOTHERBOARD-TYP-B)
                                            (ISA-SCSI-CONTROLLER)
                ISA-SCSI-CONTROLLER <===  (ISA-SCSI-CONTROLLER-TYP-08/15)
      ISA-SCSI-CONTROLLER-TYP-08/15 <===  (BOTTOM)
GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT <===  (GEHAEUSE)
                          GEHAEUSE <===  (GEHAEUSE-2)
                                            (GEHAEUSE-3)
                        GEHAEUSE-2 <===  (BOTTOM)
                        GEHAEUSE-3 <===  (BOTTOM)
           LEISTUNG-SUPPLIER-OBJECT <===  (NETZTEIL)
                          NETZTEIL <===  (NETZTEIL-100)
                                            (NETZTEIL-150)
                                            (NETZTEIL-200)
                      NETZTEIL-100 <===  (BOTTOM)
                      NETZTEIL-150 <===  (BOTTOM)
                      NETZTEIL-200 <===  (BOTTOM)
                ENVIRONMENT-CONCEPT <===  (ENVIRONMENT)
                        ENVIRONMENT <===  (BOTTOM)
                  PROZESSOR-OBJEKT <===  (PROZESSOR)
                         PROZESSOR <===  (I486DX-33)
                                            (I486DX/2-66)
```

```
                                    (I486DX/4-100)
                                    (PENTIUM-90)
                      I486DX-33 <===  (BOTTOM)
                    I486DX/2-66 <===  (BOTTOM)
                   I486DX/4-100 <===  (BOTTOM)
                     PENTIUM-90 <===  (BOTTOM)
               GRAFIKKARTEN-OBJEKT <===  (GRAFIKKARTE)
                      GRAFIKKARTE <===  (ISA-GRAFIKKARTE)
                                    (VLB-GRAFIKKARTE)
                                    (PCI-GRAFIKKARTE)
                  ISA-GRAFIKKARTE <===  (GRAFIKKARTE-TYP-A)
                GRAFIKKARTE-TYP-A <===  (BOTTOM)
                  VLB-GRAFIKKARTE <===  (GRAFIKKARTE-TYP-B)
                GRAFIKKARTE-TYP-B <===  (BOTTOM)
                  PCI-GRAFIKKARTE <===  (GRAFIKKARTE-TYP-C)
                GRAFIKKARTE-TYP-C <===  (BOTTOM)
              MOTHERBOARD-OBJEKT <===  (MOTHERBOARD)
                     MOTHERBOARD <===  (MOTHERBOARD-TYP-A)
                                    (MOTHERBOARD-TYP-B)
               FESTPLATTEN-OBJEKT <===  (FESTPLATTE)
                   MONITOR-OBJEKT <===  (MONITOR)
                          MONITOR <===  (MONITOR-TYP-XY)
                   MONITOR-TYP-XY <===  (BOTTOM)
                PERSONAL-COMPUTER <===  (BOTTOM)
```

Spezifikation der Konfigurierungsaufgabe:

```
> (solve ((Env :in Environment)
          (Env = Geld => ?G)
          (Geld :Supplier (?G =! 4000))
          (F :in Festplatte)
          (Env = HD-Speicher => ?HD)
          (HD-Speicher :Consumer (?HD >=! 1000))
         )
        :weight 1000)
```

Die Agenda nach der Aufgabenspezifikation:

```
> (show-agenda)

**************************************************************************
NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (ENV :IN ENVIRONMENT)

**************************************************************************
NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (ENV = GELD => ?G)

**************************************************************************
NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (GELD :SUPPLIER (?G =! 4000))

**************************************************************************
NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
```

```
EXTERN                  : (F :IN FESTPLATTE)

*****************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (ENV = HD-SPEICHER => ?HD)

*****************************************************************************

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (HD-SPEICHER :CONSUMER (?HD >=! 1000))
```

Der Ablauf des Konfigurierungsprozesses (bzw. die Abarbeitung der Agenda):

Für die Reihenfolge, in der die Operatoren auf der Agenda abgearbeitet werden sollen, wurde hier der Einfachheit halber keine Strategie vorgegeben. Daher wählt CONTROL zuerst die Operatoren die sich aus der Aufgabenspezifikation ergeben, da diese hier das größte Gewicht haben (1000). Alle anderen Operatoren haben das Default-Gewicht 0. Desweiteren wird bei Operatoren mit gleichem Gewicht zuerst derjenige Operator ausgeführt, der als erstes auf die Agenda eingetragen wurde.

Die Abarbeitung der Agenda[5] erfolgte in einem Trace-Modus. Im folgenden werden aber nur die interessantesten Bildschirmausgaben davon abgedruckt und zum Teil am rechten Rand kommentiert:

```
> (next 2)
T

> (show-model)

-------------------------------------------------
ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD     ---> ?G
-------------------------------------------------
?G is a Resource-Variable of the type GELD
   and in the range ( (NIL NIL) )
-------------------------------------------------
T
> (next)
```

Die A-Box nach der Abarbeitung der ersten beiden Operatoren

5. Bermerkungen zu den verwendeten Befehlen:
   Mit **(next)** wird der nächste Operator und mit **(next <n>)** die nächsten <n> Operatoren von der Agenda abgearbeitet.
   Mit **(show-model)** und **(show-sorted-model)** kann man sich die jeweils aktuelle A-Box ansehen. Diese beiden Befehle unterscheiden sich nur in der Bildschirmdarstellung.

```
Control: The actual operation is:

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (GELD :SUPPLIER (?G =! 4000))
T
> (show-model)

-------------------------------------------------
ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD    ---> ?G
-------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )
-------------------------------------------------
T
> (next)
```

Der Wertebereich der Ressource-Variablen ?G wird auf den Wert 4000 eingeschränkt.

```
Control: The actual operation is:

NAME                    : „input"
WEIGHT                  : 1000
TYPE                    : :SINGLE
AGENT                   : :TAXON
EXTERN                  : (F :IN FESTPLATTE)
```

Das Individuum F wird als Instanz des Konzeptes Festplatte in die A-Box eingetragen.

```
Control: The following operation is being inserted in the agenda:

NAME                    : „REALIZE-OBJECT"
WEIGHT                  : 0
TYPE                    : :SINGLE
AGENT                   : :REALIZER
OBJECT                  : F
STATE                   : :REALIZE
EXTERN                  : (OPERATORS:REALIZE F)

TAXRES is checking the following A-Box:

##############################################################################

ENV is related to the concepts: (ENVIRONMENT)
  |-A- GELD    ---> ?G
------------------------------------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )

##############################################################################

F is related to the concepts: (FESTPLATTE)
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4185
  |-A- HD-SPEICHER ---> ?_NEW-HD-SPEICHER-4184
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-4183
  |-A- GELD    ---> ?_NEW-GELD-4182
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4185 is a Resource-CONSUMER-Variable of the type LEISTUNG
                    and in the range ( ((>= 0) NIL) )
```

Mit diesem automatisch generierten Operator soll das Individuum F zu einem Terminalkonzept realisiert werden

```
------------------------------------------------------------------------------
?_NEW-HD-SPEICHER-4184 is a Resource-SUPPLIER-Variable of the type HD-SPEICHER
                       and in the range ( ((>= 100) NIL) )
------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-4183 is a Resource-CONSUMER-Variable of the type
                               GEHAEUSE-EINSCHUEBE
                               and in the range ( ((>= 1) (<= 1)) )
------------------------------------------------------------------------------
?_NEW-GELD-4182 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 0) NIL) )
------------------------------------------------------------------------------

Resource-Check:
There is too less of the resource GEHAEUSE-EINSCHUEBE !  [Resource-Deficit: (= 1) ]
--> The new GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT ?_IND-25 will be inserted in the A-
Box.


Control: The following operation is being inserted in the agenda:

NAME                    : „TAXRES-AND-Operation"
WEIGHT                  : 0
TYPE                    : :AND
-----------------------------------------------------
 :AND

     NAME                    : „TAXRES-Single-Operation"
     WEIGHT                  : 0
     TYPE                    : :SINGLE
     AGENT                   : :TAXON
     EXTERN                  : (?_IND-25 :IN GEHAEUSE-EINSCHUEBE-SUPPLIER-OBJECT)

     NAME                    : „TAXRES-Single-Operation"
     WEIGHT                  : 0
     TYPE                    : :SINGLE
     AGENT                   : :TAXON
     EXTERN                  : (?_IND-25 = GEHAEUSE-EINSCHUEBE TAXON::=> ?_RV-26)
T
```

Durch den letzten abgearbeiteten Operator wurde ein ganze Kette von Aktionen ausgelöst:

- Zunächst wurde das Individuum F als Instanz des Konzeptes Festplatte in die A-Box eingetragen.
- Der Konsistenztester der A-Box hat dann das Individuum F anhand der Attribut-Informationen aus der T-Box vervollständigt.
- Es wurde einen Operator auf die Agenda eingetragen, der F zu einem Terminalkonzept realisieren soll.
- Außerdem hat TAXRES beim Resource-Check festgestellt, daß bei der Ressource "Gehäuse-Einschübe" ein Defizit vorliegt und einen entsprechenden AND-Operator auf die Agenda eingetragen, um dieses Defizit zu beseitigen.

Ab hier wird nur noch die Abarbeitung dieses letzten AND-Operators und der daraus resultierenden neuen Operatoren weiter verfolgt:

```
Control: The actual operation is:

NAME                    : „SKOLEM-OBJECT"
WEIGHT                  : 0
TYPE                    : :SINGLE
AGENT                   : :SKOLEMIZER
OBJECT                  : ?_IND-25
STATE                   : :SKOLEMIZE
EXTERN                  : (OPERATORS:SKOLEMIZE ?_IND-25)
```

Der Skolemizer soll das Individuum *?_IND-25* skolemisieren.

```
Control: The actual operation is:

NAME                    : „SKOLEM-OBJECT"
WEIGHT                  : 0
TYPE                    : :SINGLE
AGENT                   : :TAXON
STATE                   : :SKOLEMIZE
OBJECT                  : ?_IND-25
EXTERN                  : (%_IND-25 EQUAL ?_IND-25)
```

Hier wird *?_IND-25* zu der neuen Komponenten-Konstante *%_IND-25* skolemisiert.

```
Control: The actual operation is:

NAME                    : „REALIZE-OBJECT"
WEIGHT                  : 0
TYPE                    : :SINGLE
AGENT                   : :REALIZER
OBJECT                  : %_IND-25
STATE                   : :REALIZE
EXTERN                  : (OPERATORS:REALIZE %_IND-25)
```

Der Realizer soll das Individuum *%_IND-25* zu einem Terminalkonzept realisieren.

```
Control: The actual operation is:

NAME                    : „realize-to-object"
WEIGHT                  : 0
TYPE                    : :OR
OBJECT                  : %_IND-25
STATE                   : :REALIZE
-----------------------------------------------------
 :OR

     NAME                    : „realize-to-object"
     WEIGHT                  : 0
     TYPE                    : :SINGLE
     AGENT                   : :TAXON
     STATE                   : :REALIZE
     OBJECT                  : %_IND-25
     EXTERN                  : (%_IND-25 :IN GEHAEUSE-2)

     NAME                    : „realize-to-object"
     WEIGHT                  : 0
     TYPE                    : :SINGLE
     AGENT                   : :TAXON
     STATE                   : :REALIZE
     OBJECT                  : %_IND-25
     EXTERN                  : (%_IND-25 :IN GEHAEUSE-3)
```

Der Realizer hatte ermittelt, daß das Individuum *%_IND-25* noch zu den Terminalkonzepten *GEHAUESE-2* und *GEHAUESE-3* realisiert werden kann.
CONTROL muß nun entscheiden, zu welchem dieser Terminalkonzepte *%_IND-25* realisert werden soll, worauf der Realisierer dann die entsprechende Operation ausführt.

Nachdem diese Operationen ausgeführt wurden, sieht die die A-Box folgendermaßen aus:

```
> (show-sorted-model)

##############################################################################

%_IND-25 is related to the concepts: (GEHAEUSE-2)
  |-A- GELD     ---> ?_NEW-GELD-4198
  |-A- HAT-NETZTEIL ---> %_NEW-NETZTEIL-4195
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-4186
------------------------------------------------------------------------------
?_NEW-GELD-4198 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 100) (<= 100)) )
------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-4186 is a Resource-SUPPLIER-Variable of the type
                               GEHAEUSE-EINSCHUEBE
                               and in the range ( ((>= 2) (<= 2)) )

##############################################################################

?_NEW-NETZTEIL-4195 is related to the concepts: (NETZTEIL)
  |-A- GELD     ---> ?_NEW-GELD-4196
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4197
------------------------------------------------------------------------------
?_NEW-GELD-4196 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 0) NIL) )
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4197 is a Resource-SUPPLIER-Variable of the type LEISTUNG
                    and in the range ( ((>= 0) NIL) )

##############################################################################

ENV is related to the concepts: (ENVIRONMENT)
  |-A- HD-SPEICHER ---> ?HD
  |-A- GELD     ---> ?G
------------------------------------------------------------------------------
?HD is a Resource-CONSUMER-Variable of the type HD-SPEICHER
   and in the range ( ((>= 1000) NIL) )
------------------------------------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
  and in the range ( ((>= 4000) (<= 4000)) )

##############################################################################

F is related to the concepts: (FESTPLATTE)
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4185
  |-A- HD-SPEICHER ---> ?_NEW-HD-SPEICHER-4184
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-4183
  |-A- GELD     ---> ?_NEW-GELD-4182
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4185 is a Resource-CONSUMER-Variable of the type LEISTUNG
                    and in the range ( ((>= 0) NIL) )
------------------------------------------------------------------------------
?_NEW-HD-SPEICHER-4184 is a Resource-SUPPLIER-Variable of the type HD-SPEICHER
                       and in the range ( ((>= 100) NIL) )
------------------------------------------------------------------------------
```

```
?_NEW-GEHAEUSE-EINSCHUEBE-4183 is a Resource-CONSUMER-Variable of the type
                               GEHAEUSE-EINSCHUEBE
                               and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------
?_NEW-GELD-4182 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 0) NIL) )
-------------------------------------------------------------------------

T
```

Ab hier wird die Agenda im Automatik-Modus vollständig agbearbeitet, wodurch die nachfolgende Konfigurationslösung generiert wird:

Die gefundene Konfigurationslösung:

```
> (show-sorted-model)


##############################################################################

%_IND-25 is related to the concepts: (GEHAEUSE-2)
  |-A- GELD     ---> ?_NEW-GELD-4198
  |-A- HAT-NETZTEIL ---> %_NEW-NETZTEIL-4195
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-4186
------------------------------------------------------------------------------
?_NEW-GELD-4198 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 100) (<= 100)) )
------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-4186 is a Resource-SUPPLIER-Variable of the type
                               GEHAEUSE-EINSCHUEBE
                               and in the range ( ((>= 2) (<= 2)) )

##############################################################################

%_IND-27 is related to the concepts: (MOTHERBOARD-TYP-B)
  |-A- GELD     ---> ?_NEW-GELD-4236
  |-A- RAM-STECKPLAETZE ---> ?_NEW-RAM-STECKPLAETZE-4235
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4234
  |-A- ISA-SLOTS ---> ?_NEW-ISA-SLOTS-4233
  |-A- PCI-SLOTS ---> ?_NEW-PCI-SLOTS-4232
  |-A- IDE-PORTS ---> ?_NEW-IDE-PORTS-4231
  |-A- HAT-CPU ---> %_NEW-PROZESSOR-4229
  |-A- HAT-RAM ---> %_NEW-RAM-MODUL-4225
  |-A- HAT-GRAFIKKARTE ---> %_NEW-GRAFIKKARTE-4222
  |-A- SCSI-PORTS ---> ?_NEW-SCSI-PORTS-4203
------------------------------------------------------------------------------
?_NEW-GELD-4236 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 500) (<= 500)) )
------------------------------------------------------------------------------
?_NEW-RAM-STECKPLAETZE-4235 is a Resource-SUPPLIER-Variable of the type
                            RAM-STECKPLAETZE
                            and in the range ( ((>= 4) (<= 4)) )
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4234 is a Resource-CONSUMER-Variable of the type LEISTUNG
                    and in the range ( ((>= 50) (<= 50)) )
------------------------------------------------------------------------------
?_NEW-ISA-SLOTS-4233 is a Resource-SUPPLIER-Variable of the type ISA-SLOTS
                     and in the range ( ((>= 3) (<= 3)) )
------------------------------------------------------------------------------
?_NEW-PCI-SLOTS-4232 is a Resource-SUPPLIER-Variable of the type PCI-SLOTS
                     and in the range ( ((>= 2) (<= 2)) )
```

```
------------------------------------------------------------------------------
?_NEW-IDE-PORTS-4231 is a Resource-SUPPLIER-Variable of the type IDE-PORTS
                     and in the range ( ((>= 2) (<= 2)) )
------------------------------------------------------------------------------
?_NEW-SCSI-PORTS-4203 is a Resource-SUPPLIER-Variable of the type SCSI-PORTS
                      and in the range ( ((>= 7) (<= 7)) )

##############################################################################

%_NEW-NETZTEIL-4195 is related to the concepts: (NETZTEIL-100)
  |-A- GELD    ---> ?_NEW-GELD-4196
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4197
------------------------------------------------------------------------------
?_NEW-GELD-4196 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 130) (<= 130)) )
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4197 is a Resource-SUPPLIER-Variable of the type LEISTUNG
                    and in the range ( ((>= 100) (<= 100)) )

##############################################################################

%_NEW-GRAFIKKARTE-4222 is related to the concepts: (GRAFIKKARTE-TYP-A)
  |-A- ISA-SLOTS ---> ?_NEW-ISA-SLOTS-4240
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4223
  |-A- GELD    ---> ?_NEW-GELD-4224
------------------------------------------------------------------------------
?_NEW-ISA-SLOTS-4240 is a Resource-CONSUMER-Variable of the type ISA-SLOTS
                     and in the range ( ((>= 1) (<= 1)) )
------------------------------------------------------------------------------
?_NEW-LEISTUNG-4223 is a Resource-CONSUMER-Variable of the type LEISTUNG
                    and in the range ( ((>= 10) (<= 10)) )
------------------------------------------------------------------------------
?_NEW-GELD-4224 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 300) (<= 300)) )

##############################################################################

%_NEW-RAM-MODUL-4225 is related to the concepts: (4MB-RAM)
  |-A- GELD    ---> ?_NEW-GELD-4226
  |-A- RAM-STECKPLAETZE ---> ?_NEW-RAM-STECKPLAETZE-4227
  |-A- RAM-SPEICHER ---> ?_NEW-RAM-SPEICHER-4228
------------------------------------------------------------------------------
?_NEW-GELD-4226 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 300) (<= 300)) )
------------------------------------------------------------------------------
?_NEW-RAM-STECKPLAETZE-4227 is a Resource-CONSUMER-Variable of the type
                            RAM-STECKPLAETZE
                            and in the range ( ((>= 1) (<= 1)) )
------------------------------------------------------------------------------
?_NEW-RAM-SPEICHER-4228 is a Resource-SUPPLIER-Variable of the type RAM-SPEICHER
                        and in the range ( ((>= 4) (<= 4)) )

##############################################################################

%_NEW-PROZESSOR-4229 is related to the concepts: (I486DX-33)
  |-A- GELD    ---> ?_NEW-GELD-4230
------------------------------------------------------------------------------
?_NEW-GELD-4230 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 200) (<= 200)) )

##############################################################################
```

```
ENV is related to the concepts: (ENVIRONMENT)
  |-A- HD-SPEICHER ---> ?HD
  |-A- GELD    ---> ?G
-------------------------------------------------------------------------------
?HD is a Resource-CONSUMER-Variable of the type HD-SPEICHER
    and in the range ( ((>= 1000) NIL) )
-------------------------------------------------------------------------------
?G is a Resource-SUPPLIER-Variable of the type GELD
   and in the range ( ((>= 4000) (<= 4000)) )

###############################################################################

F is related to the concepts: (FESTPLATTE-TYP-C)
  |-A- SCSI-PORTS ---> ?_NEW-SCSI-PORTS-4202
  |-A- LEISTUNG ---> ?_NEW-LEISTUNG-4185
  |-A- HD-SPEICHER ---> ?_NEW-HD-SPEICHER-4184
  |-A- GEHAEUSE-EINSCHUEBE ---> ?_NEW-GEHAEUSE-EINSCHUEBE-4183
  |-A- GELD    ---> ?_NEW-GELD-4182
-------------------------------------------------------------------------------
?_NEW-SCSI-PORTS-4202 is a Resource-CONSUMER-Variable of the type SCSI-PORTS
                      and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------------
?_NEW-LEISTUNG-4185 is a Resource-CONSUMER-Variable of the type LEISTUNG
                    and in the range ( ((>= 20) (<= 20)) )
-------------------------------------------------------------------------------
?_NEW-HD-SPEICHER-4184 is a Resource-SUPPLIER-Variable of the type HD-SPEICHER
                       and in the range ( ((>= 1000) (<= 1000)) )
-------------------------------------------------------------------------------
?_NEW-GEHAEUSE-EINSCHUEBE-4183 is a Resource-CONSUMER-Variable of the type
                               GEHAEUSE-EINSCHUEBE
                               and in the range ( ((>= 1) (<= 1)) )
-------------------------------------------------------------------------------
?_NEW-GELD-4182 is a Resource-CONSUMER-Variable of the type GELD
                and in the range ( ((>= 1100) (<= 1100)) )
-------------------------------------------------------------------------------
T
>
```

Anhang B: Literaturverzeichnis

[Abecker94] Andreas Abecker
TaxLog: Taxonomische Wissensrepräsentation und Logisches Programmieren
Diplomarbeit, Universität Kaiserslautern, Juni 1994

[BH91] F. Baader, P. Hanschke
A Scheme for Integrating Concrete Domains into Concept Languages
Research Report RR-91-10, DFKI, Kaiserslautern, April 1991

[BBHLN92] F. Baader, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt
Terminologische Logiken
Fachbeitrag in KI 3/92, Fachbereich 1 der Gesellschaft für Informatik e.V., September 1992

[BBHNS90] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, J. H. Siekmann
Concept Logics
Research Report RR-90-10, DFKI, Kaiserslautern, September 1990

[BHHM93] H. Boley, P. Hanschke, K. Hinkelmann, M. Meyer
CoLab: A Hybrid Knowledge Representation and Compilation Laboratory
Research Report RR-93-08, DFKI, Kaiserslautern, Januar 1993

[BKN] M. Buchheit, R. Klein, W. Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration

[BS85] R. J. Brachman, J. Schmolze
An overview of the KL-ONE knowledge representation system
Cognitive Science, 9(2): Seite 171-216, April 1985

[CGS91] R. Cunis, A. Günter, H. Strecker
Das PLAKON-Buch
Informatik-Fachberichte, Springer-Verlag Berlin Heidelberg, 1991

[DDJSSW94] F. Dridi, D. Drollinger, M. Junker, A. Schoeller, F. Steinle, H. Wache
Systembeschreibung des TOOCON-Werkzeugkastens
Implementierungsbeschreibung, DFKI, Kaiserslautern, Juli 1994

[Dengel94] Andreas Dengel
Künstliche Intelligenz: Allgemeine Prinzipien und Modelle
Meyers Forum 13, BI-Taschenbuchverlag, Mannheim, 1994

[DP94] B. Dellen, J. Paulokat
Verwaltung von impliziten Abhängigkeiten bei der strukturorientierten Konfiguration in IDAX
In: R. Bergmann, J. Paulokat, A.-M. Schoeller, H. Wache (Hrsg.): PuK-94 8. Workshop "Planen und Konfigurieren", SEKI Working Paper SWP-94-01, Universität und DFKI Kaiserslautern

[Dilger88] Werner Dilger
Objektorientierte Wissensrepäsentation
Vorlesung an der Universität Kaiserslautern WS 88/89

[Günter94] Andreas Günter
Strukturierung der Vorgehensweisen beim Konfigurieren in Kontrolltypen
In: R. Bergmann, J. Paulokat, A.-M. Schoeller, H. Wache (Hrsg.): PuK-94 8. Workshop "Planen und Konfigurieren", SEKI Working Paper SWP-94-01, Universität und DFKI Kaiserslautern

[Günter95] Andreas Günter
KONWERK - ein modulares Konfigurierungswerkzeug
In: M. Richter, F. Maurer (Hrsg.): Expertensysteme 95, Proceedings in Artificial Intelligence, Infix-Verlag, Sankt Augustin, 1995

[Heinrich93] Michael Heinrich
Ressourcenorientiertes Konfigurieren
Fachbeitrag in KI 1/93, Fachbereich 1 der Gesellschaft für Informatik e.V., 1993

[HJ91] M. Heinrich, E.-W. Jüngst
A resource-based paradigm for the configuring of technical systems from modular components
In: Proceedings of the 7th IEEE Conference on AI Appl. CAIA-91, 1991

[HJ93] M. Heinrich, E.-W. Jüngst
Konfigurieren technischer Einrichtungen ausgehend von den Komponenten des technischen Prozesses: Prinzip und erste Erfahrungen
In: F. Puppe, A. Günter (Hrsg.): Expertensysteme 93, Springer-Verlag, 1993

[HN90] B. Hollunder, W. Nutt
Subsumption Algorithms for Concept Languages
Research Report RR-90-04, DFKI, Kaiserslautern, April 1990

[JMSY87] J. Jaffar, S. Michaylov, P. J. Stuckey, R. H. C. Yap
The CLP(R) Language and System, 1987

[JW93] M. Junker, H. Wache
Realisierung der Schnittstelle zwischen TAXON und konkreten Bereichen
DFKI, Kaiserslautern, November 1993

[Keene89] Sonya E. Keene
Object-Oriented Programming in COMMON LISP
A Programmer's Guide to CLOS
Addison-Wesley, 1989

[Klein] Rüdiger Klein
Constuctive Problem Solving: A Model Generation Approach to Configuration
Technical Report, Daimler-Benz Research Dept., Berlin

[Klein92] Rüdiger Klein
Überlegungen zu einem allgemeinen Konfigurierungs-Toolsystem
Technical Report, Daimler-Benz AG, 1992

[Klein93] Rüdiger Klein
Some Suggestions to Resource-based Modelling in Configuration Problem Solving
Technical Report, Daimler-Benz Research Dept., Berlin, 1993

[Luck91] Kai von Luck
Hybride logikbasierte Systeme
Fachbeitrag in KI 2/91, Fachbereich 1 der Gesellschaft für Informatik e.V., 1991

[Meyer94] Manfred Meyer
Finite Domain Constraints: Declarativity meets Efficiency, Theory meets Application
DISKI 79 - Dissertation zur Künstlichen Intelligenz, Infix-Verlag, Sankt Augustin, 1994

[MP94] H. Meyer auf'm Hofe, J. Paulokat
Verarbeitung von Constraints in der Expertensystementwicklungsumgebung IDAX
In: R. Bergmann, J. Paulokat, A.-M. Schoeller, H. Wache (Hrsg.): PuK-94 8. Workshop "Planen und Konfigurieren", SEKI Working Paper SWP-94-01, Universität und DFKI Kaiserslautern

[Paulokat95] Jürgen Paulokat
Entscheidungsorientierte Rechtfertigungsverwaltung zu Unterstützung des Konfigurierungsprozesses in IDAX
In: M. Richter, F. Maurer (Hrsg.): Expertensysteme 95, Proceedings in Artificial Intelligence, Infix-Verlag, Sankt Augustin, 1995

[PR93] J. Paulokat, H. Ritzer
Unterstützung der Kontrolle bei der Konfigurierung durch ein problemklassenspezifisches TMS
In: F. Puppe, A. Günter (Hrsg.): Expertensysteme 93, Springer-Verlag, 1993

[Peter90] Erwin T. Peter
Artificial Intelligence und Expertensysteme
IWT Verlag, Vaterstetten bei München, 1990

[Rich88] Elaine Rich
KI - Einführung und Anwendung
McGraw-Hill, Hamburg, 1988

[Richter92] Michael M. Richter
Prinzipien der Künstlichen Intelligenz
2. überarb. und erw. Auflage, B. G. Teubner, Stuttgart, 1992

[RSW94] M. M. Richter, A. Schoeller, H.Wache
Der TOOCON-Werkzeugkasten
Abschlußbericht, DFKI, Kaiserslautern, Juli 1994

[Schoeller94] Anna-Maria Schoeller
Räumliches Schließen bei der Konfigurierung
Abschlußbericht, DFKI, Kaiserslautern, 1994

[Steele90] Guy L. Steele JR.
COMMON LISP, THE LANGUAGE
2. Auflage, Digital Press, U. S. A., 1990

[SV94] U. Schüller, H. G. Veddeler
PC aufrüsten und reparieren
5. Auflage, DATA BECKER, Düsseldorf, 1994

[VDI91] Verein Deutscher Ingenieure
Erfolgreiche Anwendung wissensbasierter Systeme in Entwicklung und Konstruktion
Tagung in Heidelberg, 7.-8.10.91, VDI Berichte 903, VDI Verlag, Düsseldorf, 1991

[Wache93] Holger Wache
Integration von ressourcen-basierten Ansätzen im Werkzeugkasten von TOOCON
Diskussionspapier (DRAFT), DFKI, Kaiserslautern, 1993

[Wache94] Holger Wache
TOOCON USER-MANUAL und REFERENCE-MANUAL
DFKI, Kaiserslautern, Juli 1994

[Winston87] Patrick Henry Winston
Künstliche Intelligenz
Addison-Wesley, Bonn, 1987

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210

e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-
sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1995

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

1993

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder
Cardinality Restrictions on Concepts
20 pages

RR-93-46
Philipp Hanschke
A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-45
Rainer Hoch
On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt
Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-43
M. Bauer, G. Paul
Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-42
Hubert Comon, Ralf Treinen
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-41
Winfried H. Graf
LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-38
Stephan Baumann
Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt
Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.)
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

RR-93-34
Wolfgang Wahlster
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-33
Bernhard Nebel, Jana Koehler
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman
Corporate Agents
14 pages

RR-93-29
Armin Laux
Representing Belief in Multi-Agent Worlds via Terminological Logics
35 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker
Feature-Based Allomorphy
8 pages

RR-93-27
Hans-Ulrich Krieger
Derivation Without Lexical Rules
33 pages

RR-93-26
Jörg P. Müller, Markus Pischel
The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-25
Klaus Fischer, Norbert Kuhn
A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-24
Rainer Hoch, Andreas Dengel
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy
Comparative Study of Connectionist Simulators
20 pages

RR-93-22
Manfred Meyer, Jörg Müller
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-18
Klaus Schild
Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-17
Rolf Backofen
Regular Path Expressions in Feature Logic
37 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz
Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles,
Markus A. Thies, Wolfgang Wahlster
PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-13
Franz Baader, Karl Schlechta
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-12
Pierre Sablayrolles
A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-11
Bernhard Nebel, Hans-Jürgen Bürckert
Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf
Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-09
Philipp Hanschke, Jörg Würtz
Satisfiability of the Smallest Binary Program
8 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer
CoLab: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
Concept Logics with Function Symbols
36 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
On Skolemization in Constrained Logics
40 pages

RR-93-05
Franz Baader, Klaus Schulz
Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-03
Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-01
Bernhard Hollunder
An Alternative Proof Method for Possibilistic Logic and its Application to Terminological Logics
25 pages

DFKI Technical Memos

1995

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

1993

TM-93-05
Michael Sintek
Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-93-04
Hans-Günther Hein
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta
Conflict Resolving Negotiation for COoperative Schedule Management Agents (COSMA)
21 pages

TM-93-01
Otto Kühn, Andreas Birk
Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

DFKI Documents

1995

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache *ACCP*
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
Beiträge zum Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

1993

D-93-27
Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman,
Hans Uszkoreit (Eds.)
Report of the EAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken
110 pages

D-93-26
Frank Peters
Unterstützung des Experten bei der Formalisierung von Textwissen INFOCOM - Eine interaktive Formalisierungskomponente
58 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.)
Modeling Epistemic Propositions
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-24
Brigitte Krenn, Martin Volk
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-22
Andreas Abecker
Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

Note: This document is no longer available in printed form.

D-93-21
Dennis Drollinger
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-20
Bernhard Herbig
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt
Design & KI
74 Seiten

D-93-15
Robert Laux
Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-14
Manfred Meyer (Ed.)
Constraint Processing – Proceedings of the International Workshop at CSAM'93, St.Petersburg, July 20-21, 1993
264 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek,
Werner Stein
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.)
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

Note: This document is no longer available in printed form.

D-93-10
Elizabeth Hinkelman, Markus Vonerden, Christoph Jung
Natural Language Software Registry (Second Edition)
174 pages

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer
TDLExtraLight User's Guide
35 pages

D-93-08
Thomas Kieninger, Rainer Hoch
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-07
Klaus-Peter Gores, Rainer Bleisinger
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-06
Jürgen Müller (Hrsg.)
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz, Saarbrücken, 29. - 30. April 1993
235 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel,
Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster
PPP: Personalized Plan-Based Presenter
70 pages

D-93-04
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht
1992
194 Seiten

D-93-03
Stephan Busemann, Karin Harbusch (Eds.)
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter
User Manual of COKAM+
23 pages

D-93-01
Philipp Hanschke, Thom Frühwirth
Terminological Reasoning with Constraint Handling Rules
12 pages

Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON

Volker Ehresmann

D-95-10
Document