



TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur Datenverwaltungssysteme

Studienarbeit

**Anbietersuche für
Arbeitsplanungsaufgaben im
Informationstechnischen Modellkern**

Michael Rentzsch

Chemnitz, 15. Dezember 2004

Autor: Michael Rentzsch
Reichenhainer Straße 37/486
09126 Chemnitz
E-Mail: michael.rentzsch@repc.de
geboren am 07.04.1981 in Werdau

Hochschullehrer: Professor Dr. W. Benn

Betreuer: Dipl.-Inf. R. Neubert, Dipl.-Inf. O. Görlitz

Ausgabe: 01.07.2004

Eingereicht am: 17.12.2004

Aufgabenstellung

Im Rahmen des SFB 457 „Hierarchielose regionale Produktionsnetze“ wurde von der Professur Datenverwaltungssysteme ein System zur auftragsspezifischen Suche potentieller Kooperationspartner entwickelt. Das System basiert bisher auf einem mehrkriteriellen Abgleich zwischen den Anforderungen einzelner Prozessschritte und dem Angebotsprofil der potentiellen Partner für Fertigungsaufgaben. In dieses System soll nunmehr auch die Verwaltung und Suche von Partnern aus dem Bereich Arbeitsplanung integriert werden.

Zielstellung der Studienarbeit ist es, die Suche nach Partnern für eine Kooperation, von der Eingabe der im Bereich Arbeitsplanung auszuführenden Aktivitäten bis hin zur Auswahl möglicher Kandidaten, durchgehend zu spezifizieren und zu implementieren. Für die effiziente Durchführung des Abgleichs wurde an der Professur eine Komponente entwickelt, welche sich an die Signalverarbeitung im menschlichen Gehirn anlehnt und die Beschreibungen entsprechend indexiert. Kernstück der Arbeit ist die Abbildung von Kompetenzprofilen der Arbeitsplanung im objektorientierten Datenbanksystem „FastObjects“ und die Kopplung der Abgleichskomponente mit dem Datenbanksystem. Durch die Realisierung von zwei Schnittstellen muss sich die Studienarbeit in den Gesamtkontext des SFB eingliedern. Zum einen ist eine Schnittstelle als Grundlage der Kompetenz- und Anforderungsbeschreibung zu entwickeln. Zum anderen sind die ermittelten Kandidaten in geeigneter Weise an das Managementsystem des Produktionsnetzes zu übergeben.

Die Aufgabenstellung umfasst im einzelnen:

- Analysieren Sie die Struktur und die Inhalte des vorliegenden Analysemodells der Arbeitsplanung und transformieren Sie diese in ein geeignetes Designmodell für die Spezifikation von Angebots- und Anforderungsbeschreibungen!
- Erweitern Sie das bestehende Datenbankschema des Informationstechnischen Modellkerns für die Verwaltung der potentiellen Kooperationspartner im Bereich Arbeitsplanung und deren Angebotsprofile!
- Konzipieren und Implementieren Sie eine Verbindung zwischen dem Abgleichsmechanismus und dem Datenbanksystem, so dass die Angebotsprofile indexiert und für den Abgleich mit den Anforderungen herangezogen werden können! Die innerhalb des Index ermittelten Kandidaten sollen dann in der Datenbank zugegriffen und an das Managementsystem übergeben werden.
- Entwickeln Sie geeignete Schnittstellen zur Instantiierung des Designmodells!
- Entwickeln Sie geeignete Schnittstellen zur Übernahme der angeforderten Aktivitäten und zur Weitergabe der ermittelten Kandidaten!
- Validieren Sie das von Ihnen entwickelte System anhand von Beispielanfragen!

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Michael Rentzsch
Chemnitz, den 15. Dezember 2004

Inhaltsverzeichnis

1	Einleitung	1
1.1	Zielstellung des SFB457	1
1.2	Vorstellung des IMK	3
1.3	Aufgabe dieser Arbeit	4
2	Designmodell	6
2.1	Partialmodell Arbeitsplanungskompetenzen	6
2.2	Partialmodell Arbeitsplaner	9
2.3	Partialmodell Anforderung	9
2.4	Bewertungsmodell	9
2.4.1	Anpassung der Bewertung	11
2.4.2	Berechnung der Durchschnittsbewertung	12
2.5	Vollständiges Designmodell	13
3	Lösungsansätze zur Kandidatensuche	15
3.1	Naiver Algorithmus	15
3.2	Kandidatensuche mit GraphLearning	16
3.2.1	Verwaltung der Angebotsprofile	17
3.2.2	Anfragebearbeitung im GraphLearning	17
3.2.3	Reduktion der Ergebnismenge auf relevante Profile	18
4	Prototyp IMK-APL	20
4.1	Beschreibung der Entwicklungsumgebung	20
4.1.1	Objektorientiertes Datenbankmanagementsystem	20
4.1.2	Programmiersprache	21
4.1.3	Grafische Benutzeroberfläche	21
4.1.4	XML-Parsing	22
4.1.5	Zusammenfassung	23
4.2	Anbindung des GraphLearning	24
4.2.1	Kommunikation über Prozesse	24
4.2.2	Kommunikation über CORBA	24
4.2.3	Anbindung über JNI	25
4.2.4	Umsetzung	26
4.2.5	Zusammenfassung	28
4.3	Benutzung von IMK-APL	28
4.3.1	Laden der Metainformationen	30
4.3.2	Anzeigen des Angebotsbaums	31

4.3.3	Anlegen von Arbeitsplanern	31
4.3.4	Importieren von Arbeitsplanern	33
4.3.5	Anzeigen von Arbeitsplanern	33
4.3.6	Initialisieren der Kandidatensuche	34
4.3.7	Kandidatensuche	35
4.4	Definition der XML-Dokumente	35
4.4.1	Beschreibung der Metainformationen	36
4.4.2	Beschreibung der Arbeitsplaner	37
4.4.3	Anfragen	37
4.4.4	Rückgaben der Kandidatensuche	38
4.5	Spezifikation der Webservices	39
4.5.1	Webservices mit SOAP	39
4.5.2	Import eines Arbeitsplaners	40
4.5.3	Anfrage zur Kandidatensuche	41
4.6	Installation	42
4.6.1	Initialisieren der Datenbank	42
4.6.2	Installation und Konfiguration von IMK-APL	43
4.6.3	Installation der GraphLearning Komponente	43
4.6.4	Installation der Webservices	44
5	Kritik und Ausblick	46
5.1	Kritik am Designmodell	46
5.2	Kritik am GraphLearning	47
5.3	Kritik an der Prototypapplikation IMK-APL	47
5.3.1	Funktionsumfang	47
5.3.2	Zeitaufwand bei der Bearbeitung von XML-Daten	48
5.4	Weiterführende Arbeiten	49
5.4.1	Definition der Äquivalenz von Größen- und Gewichtsklassen	49
5.4.2	Bewertung von Kompetenzen durch Dritte	50
5.4.3	Neutraining der GraphLearning-Komponente	51
5.5	Zusammenfassung	52
A	Abkürzungsverzeichnis	53
B	Weitere Screenshots	55
C	Document Type Definitions	57
C.1	Metainformationen	57
C.2	Arbeitsplaner	59
C.3	Anfragen	60
C.4	Rückgabe der Kandidatensuche	61
C.5	Rückgabe beim Import von Arbeitsplanern	62
D	Inhalt der beigelegten CD	63

Abbildungsverzeichnis

1.1	Bedeutung der KMU, Quelle: [SFB457]	1
1.2	Struktur einer Kompetenzzelle, Quelle: [SFB457]	2
1.3	Modell des Hierarchielosen regionalen Produktionsnetzes, Quelle: [SFB457]	3
1.4	Struktur des IMK nach [Oes03]	4
2.1	Hierarchie der Arbeitsplanungskompetenzen	6
2.2	Darstellung von APLKPen als Baum	7
2.3	Beispiel für Angebotsbaum nach ([DM01])	7
2.4	Partialmodell Arbeitsplanungskompetenzen	8
2.5	Partialmodell Arbeitsplaner	10
2.6	Partialmodell Anfrage	11
2.7	Klasse Bewertung	11
2.8	Vollständiges Designmodell	14
3.1	Stufen der Prüfung, ob ein Angebot relevant ist	15
3.2	Angebotsprofile und Anfragen im GL, Quelle: [GNM04]	18
4.1	FastObjects: Struktur, Quelle: [FO]	21
4.2	Ein Frame mit Swing (links) und AWT (rechts)	22
4.3	Transformation eines XML-Dokuments in einen Objektbaum	23
4.4	CORBA Architektur, Quelle [Oes03]	25
4.5	JNI als Bindeglied zwischen C und Java, Quelle [JNI2]	26
4.6	Beispiel für die Verwendung des GraphLearning mit Java	29
4.7	Screenshot („Bildschirmfoto“) der IMK-APL GUI	30
4.8	Anzeige der APLKPen	31
4.9	Anlegen eines Arbeitsplaners	32
4.10	Anzeigen eines Angebots	34
4.11	Ergebnis der Kandidatensuche	35
4.12	Tomcat und SOAP im TCP/IP-Protokollstack	39
4.13	SOAP Beispielnachricht, Quelle: Wikipedia [Wiki2]	40
4.14	Anfrage an den Webservice aplanlegen mit Java	41
4.15	Anfrage an den Webservice anfrage mit Java	42
4.16	IMK-APL Setupprogramm	43
5.1	Vergleich des Laufzeitverhaltens von SAX und DOM, Quelle: [DEV]	48
5.2	Äquivalenzbereich für $m = 2, 5$ und $p = 0, 5$	50
B.1	Statistik über Import von Metainformationen	55

B.2	Details einer Arbeitsplanungskompetenz	55
B.3	Auswahl eines Geschäftsobjekts (mit Gewichts- und Größenklasse)	56
B.4	Auswahl der Teil- und Werkstoffklassen	56
B.5	Hinzufügen der Arbeitsplanungskompetenzen	56

1 Einleitung

Temporäre Netzwerke werden als die herausragende Organisationsform der nächsten Jahre für kleine und mittelständische Unternehmen (KMU) angesehen ([Wir01] und [DEL98]), um flexibel und kundenorientiert zu planen und produzieren. Sie ermöglichen es unter Verwendung moderner Informations- und Kommunikationstechnologien, dass sich die Unternehmen auf ihre Kernkompetenzen und -geschäfte konzentrieren können. Gleichzeitig unterstützen sie Geschäftsabläufe in hierarchiearmen und räumlich verteilten Organisationsstrukturen.

Mit einem Lösungsansatz für solche Netze, der auf der Verknüpfung von kleinsten Leistungseinheiten, den sogenannten Kompetenzzellen (KPZ) beruht, beschäftigt sich der Sonderforschungsbereich (SFB) 457¹ der Technischen Universität Chemnitz, „Hierarchielose regionale Produktionsnetze“, in dessen Kontext sich diese Arbeit eingliedert.

1.1 Zielstellung des SFB457

Abbildung 1.1 zeigt, dass kleine und mittlere Unternehmen einen großen Teil unserer Wirtschaft ausmachen. Sie zeichnen sich durch eine hohe Fachkompetenz auf bestimmten Teilgebieten aus. Dafür fehlt es ihnen aber oft an übergreifenden Kompetenzen und Ressourcen, um komplexe und stark kundenorientierte Projekte selbstständig ausführen zu können. Die fehlenden Ressourcen müssen dann durch Kooperationen ausgeglichen werden.

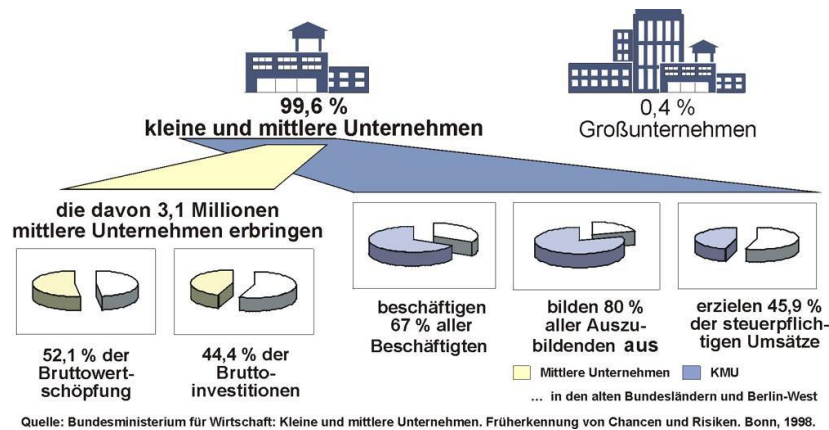


Abbildung 1.1: Bedeutung der KMU, Quelle: [SFB457]

¹<http://www.tu-chemnitz.de/sfb457>

Diese Kooperationen sind häufig hierarchisch strukturiert, das heißt die kleineren Unternehmen nehmen einseitige Abhängigkeiten² und fehlenden Kontakt zum Endkunden in Kauf. Gleichzeitig ist es für neu gegründete Unternehmen schwer, sich etablierten Kooperationsbeziehungen anzuschließen. Damit werden unter Umständen Unternehmensgründungen behindert und dadurch regionale Potenziale und Ressourcen nicht genutzt. Dieses Problem kann durch den direkten Zusammenschluss von KMU behoben werden. Hierbei bietet es sich an, regionale Bezüge zu beachten, da dann strukturelle und mentale Kopplungen geknüpft werden können.

Vision des SFB 457 ist es, dass Unternehmer in Form von eigenständigen Kompetenzzellen repräsentiert werden. Abbildung 1.2 zeigt die Struktur einer solchen KPZ. Sie besteht aus dem Menschen mit seinen (individuellen) Kompetenzen und den Ressourcen, die ihm zur Verfügung stehen.

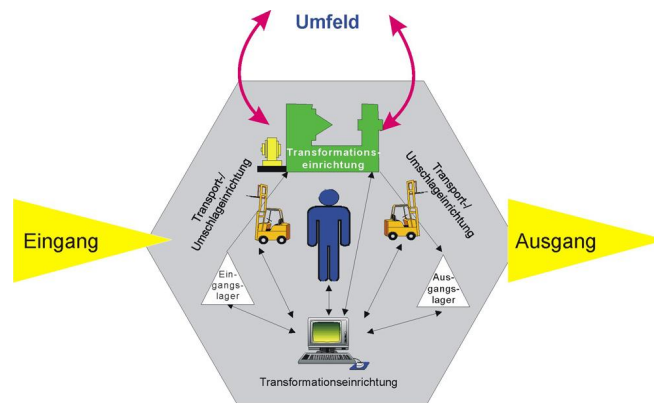


Abbildung 1.2: Struktur einer Kompetenzzelle, Quelle: [SFB457]

Diese Kompetenzzellen werden direkt miteinander vernetzt. Es entstehen also – im Gegensatz zu den eben beschriebenen, bisher existierenden Vernetzungsansätzen – keine Hierarchieebenen. Alle Zellen sind bei Entscheidungsfindungen gleichberechtigt. Die Vernetzung selbst findet regional und temporär entsprechend der zu erfüllenden Kundenwünsche statt. Fehlende Kompetenzen und Ressourcen werden durch Gründung weiterer KPZ oder durch Zukauf von außerhalb ausgeglichen. Abbildung 1.3 zeigt ein Modell des beschriebenen Ansatzes.

Das Ziel des SFB 457 ist nun „die wissenschaftliche Durchdringung des visionären Ansatzes kundenorientierter hierarchieloser regionaler Produktionsnetze auf Basis von Kompetenzzellen.“³ Dazu wurden die folgenden vier Teilaufgaben spezifiziert:

1. Definition der Kompetenzzelle,
2. Untersuchung der Genese und des Betriebens von hierarchielosen Produktionsnetzen,
3. Untersuchung der Rückkopplungen bei der Vernetzung von Kompetenzzellen und

²Z. B. vorgeschriebene zu verwendende Software

³[SFB457]

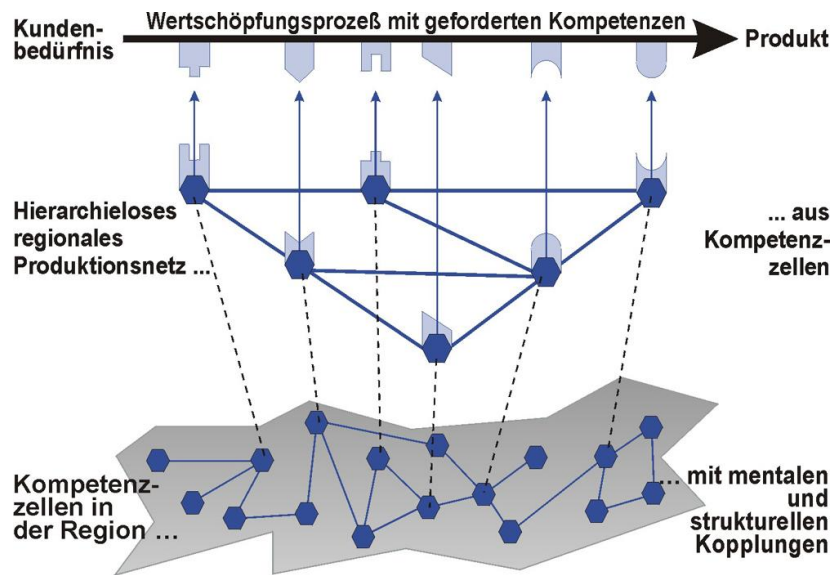


Abbildung 1.3: Modell des Hierarchielosen regionalen Produktionsnetzes, Quelle: [SFB457]

4. Untersuchung der Entwicklung von Produktionsnetzen zu regionalen Kompetenzzentren.

Der Sonderforschungsbereich „Hierarchielose regionale Produktionsnetze“ ist in vier Projektbereiche aufgeteilt. Die Professur Datenverwaltungssysteme, an der diese Arbeit entstanden ist, ist am Projektbereich A, „Grundlagen der Elementarisierung und Vernetzung“, beteiligt.

1.2 Vorstellung des IMK

Wie im letzten Abschnitt bereits erläutert, ist ein Produktionsnetzwerk im Kontext des SFB 457 eine temporäre Vernetzung von Kompetenzzellen. Zum Bilden eines solchen Netzwerks werden verschiedene Informationen über die einzelnen Zellen benötigt. Diese Informationen werden im *Informationstechnischen Modellkern (IMK)* gespeichert und von diesem dem Kompetenzzellen-netzwerk (KPZN) zur Verfügung gestellt.

Der schon existierende Teil des IMK wurde im Rahmen der Diplomarbeit [Oes03] entwickelt und implementiert. Bisher werden durch ihn Funktionen zum Beschreiben, Speichern und Suchen von Fertigungskompetenzzellen angeboten. Bei der Suche nach passenden Kandidaten für einen Auftrag wird der an der Professur Datenverwaltungssysteme entwickelte Intelligent Cluster Index (ICIX) verwendet. Abbildung 1.4 zeigt die Struktur des IMK.

Der Informationstechnische Modellkern wurde in der Programmiersprache Java implementiert. Als Datenbanksystem wird das objektorientierte Datenbankmanagementsystem FastObjects eingesetzt. Weitere Informationen zum Entwicklungsstand des IMK zu Beginn dieser Arbeit können [Oes03] entnommen werden.

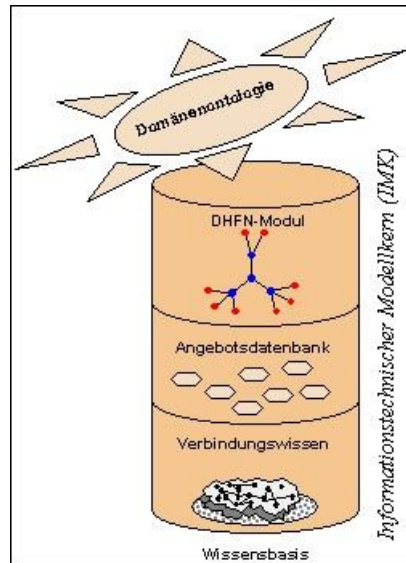


Abbildung 1.4: Struktur des IMK nach [Oes03]

1.3 Aufgabe dieser Arbeit

Aufgabe dieser Arbeit war die Erweiterung der Informationstechnischen Modellkerns, so dass in diesem planerisch tätige Anbieter⁴ beschrieben und gesucht werden können. Einen Ansatz dazu liefern [DM01] und [GNM04].

Diese Aufgabe wurde in drei Schritten erfüllt:

1. Aufstellen eines Modells zur Beschreibung von Arbeitsplanern,
2. Entwicklung einer Möglichkeit zum Suchen von Kandidaten und
3. Implementierung zweier Schnittstellen zum Instanzieren des Modells sowie zur Kandidatensuche.

Das entworfene Designmodell wird im Kapitel 2 vorgestellt und formal mit Hilfe der grafischen Modellierungssprache UML spezifiziert.

Im dritten Kapitel, „Lösungsansätze zur Kandidatensuche“, werden zwei Möglichkeiten vorgestellt, wie die Suche nach passenden Arbeitsplanern für einen Auftrag implementiert werden kann. Dabei wird der Einsatz einer für diese Zwecke entwickelten Indexierungskomponente, die auf neuronalen Netzen basiert, diskutiert.

Die beiden entwickelten Schnittstellen zur Instanzierung des Designmodells und zur Suche von Kandidaten werden im vierten Kapitel, „Prototyp IMK-APL“ detailliert vorgestellt. Dabei wird

⁴Z. B. Konstrukteure, Entwickler oder Technologen

sowohl auf die technischen Einzelheiten der Implementierung als auch auf die Anwendung durch die Nutzer eingegangen.

Im letzten Kapitel, „Kritik und Ausblick“ wird der entwickelte Ansatz noch einmal kritisch bewertet und es werden Vorschläge für weitere Arbeiten im Kontext der Beschreibung und Suche von Arbeitsplanern im Informationstechnischen Modellkern gegeben.

2 Designmodell

In diesem Kapitel wird das entworfene Designmodell vorgestellt, das in drei Teile gegliedert ist:

1. Partialmodell *Arbeitsplanungskompetenzen*,
2. Partialmodell *Arbeitsplaner* und
3. Partialmodell *Anfrage*.

Alle Teilmodelle werden erläutert und in UML¹-Notation grafisch dargestellt. Die Modellierung erfolgte, dem Paradigma der zu verwendenden Programmiersprache² und des Datenbankmanagementsystems³ (DBMS) folgend, objektorientiert.

Im letzten Abschnitt des Kapitels wird die Notwendigkeit eines Modells zur Bewertung der Arbeitsplaner begründet und das in dieser Arbeit definierte und verwendete Modell eingeführt.

2.1 Partialmodell Arbeitsplanungskompetenzen

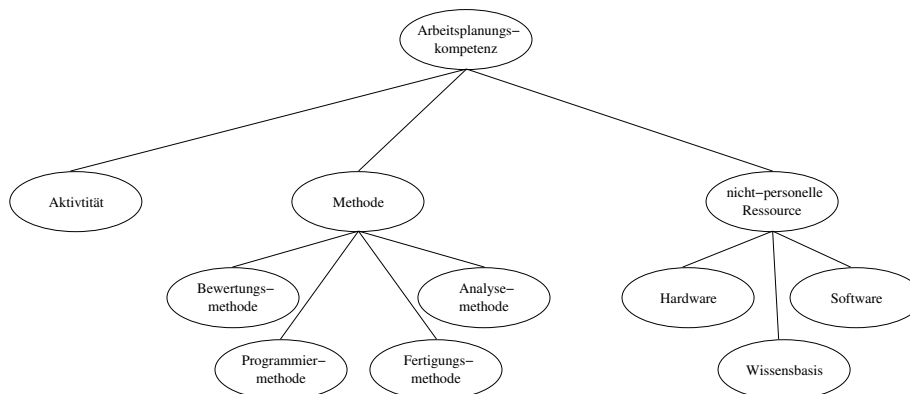


Abbildung 2.1: Hierarchie der Arbeitsplanungskompetenzen

Die Arbeitsplanungskompetenz (APLKP⁴) ist das zentrale Kriterium für die Suche nach Arbeitsplanern. Sie beschreibt eine Aktivität, Methode oder nichtpersonelle Ressource (nPR), die

¹Unified-Modelling-Language

²Java

³FastObjects

⁴Hinweis: Im Quellcode der in dieser Arbeit entwickelten Prototypapplikation (siehe Kapitel 4) wird aus Versionsgründen die Abkürzung APLKPZ als Klassenname verwendet.

ein Arbeitsplaner beherrscht oder über die er verfügt. Abbildung 2.1 zeigt die vollständige Hierarchie der APLKPen.

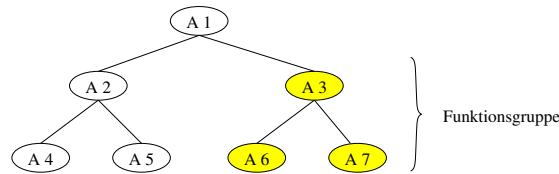


Abbildung 2.2: Darstellung von APLKPen als Baum

Jede Arbeitsplanungskompetenz kann in weitere Kompetenzen, im Folgenden *Teilkompetenzen* genannt, aufgeteilt werden [GNM04]. Die entstehende Anordnung kann man in einem n -ären Baum darstellen, wie Abbildung 2.2 zeigt. Im Kontext des in dieser Arbeit entwickelten Prototypen (siehe Kapitel 4) heißt dieser Baum *Angebotsbaum*, dieser darf im System nur genau einmal vorhanden sein. Abbildung 2.3 zeigt ein Beispiel für einen Angebotsbaum (nach [DM01]). Eine Zusammenfassung mehrerer APLKPen zu einem Aufgabenkomplex ist eine *Funktionsgruppe* (FG). Dies entspricht einem Teilbaum des Angebotsbaumes.

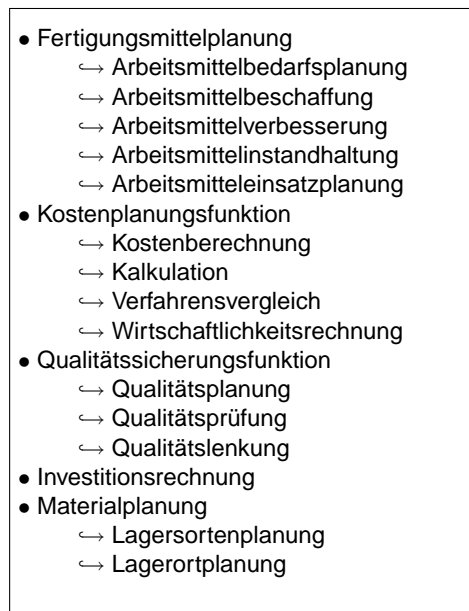


Abbildung 2.3: Beispiel für Angebotsbaum nach ([DM01])

Erfüllt ein Arbeitsplaner eine Arbeitsplanungskompetenz, die Wurzel einer solchen Funktionsgruppe ist, so erfüllt er automatisch auch alle APLKPen, die Knoten im entsprechenden Teilbaum sind. Angewendet auf das Beispiel aus Abbildung 2.2 heißt das: Beherrscht ein Arbeitsplaner die APLKP A3, so ist er ebenfalls für A6 und A7 qualifiziert. Der Umkehrschluss gilt *nicht*⁵.

⁵Der Umkehrschluss wäre: Ist man für alle Teilkompetenzen einer Arbeitsplanungskompetenz qualifiziert, dann automatisch auch für die übergeordnete APLKP.

Die Unterteilung der Arbeitsplanungskompetenzen in Aktivitäten, Methoden und nichtpersonelle Ressourcen, etc. ist im Datenmodell in einer *is-a* Beziehung umgesetzt. Das heißt, die Klasse APLKP ist eine *Oberklasse*⁶ der Klassen Aktivität, Methode und nPR. Diese können selbst wieder Oberklassen für andere Kompetenzen sein (siehe Abbildung 2.1).

Alle Arbeitsplanungskompetenzen haben die Attribute id (einen vom System vergebenen, eindeutigen Identifikator) und bezeichnung. Auch die Bezeichnung muss eindeutig sein, da sie als Auswahlattribut bei der Spezifikation von Angebotsvektoren und Anfragen gilt (siehe dazu Abschnitt 3.2). Aktivitäten haben keine weiteren Merkmale. Methoden verfügen über ein Attribut eigenschaften und nichtpersonelle Ressourcen über einen standort. Auch die Unterklassen von Methoden und nPRs haben spezifische Attribute. Diese sind in der UML-Notation des Partialmodells in Abbildung 2.4 detailliert dargestellt.

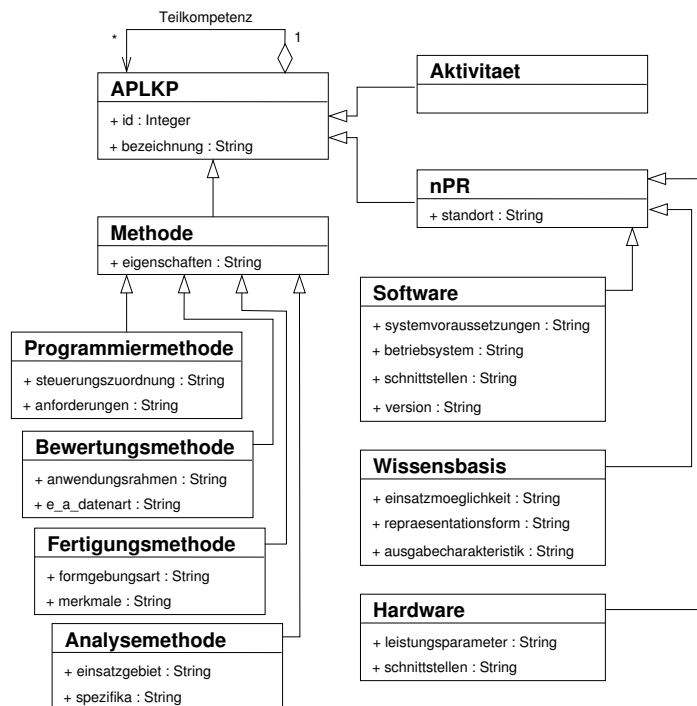


Abbildung 2.4: Partialmodell Arbeitsplanungskompetenzen

Ist eine APLKP Wurzel einer Funktionsgruppe, so sind ihr andere Kompetenzen als „Kinder“ zugeordnet. Diese Teilkompetenzen werden in einer Komponenten-Beziehung ausgedrückt. Abbildung 2.4 zeigt das gesamte Partialmodell *Arbeitsplanungskompetenzen* als UML-Diagramm.

⁶Zur Begriffsbildung vgl. [Heu92]

2.2 Partialmodell Arbeitsplaner

Arbeitsplaner (APL) sind Personen oder Unternehmen, die Nachfragen für eine Menge von Arbeitsplanungskompetenzen erfüllen können. Sie sind durch die folgenden Attribute beschrieben: *id*, einem vom System vergebenen, eindeutigen Identifikator, *name*, *ort*, *plz* (Postleitzahl), *strasse* und *webadresse*. Bei der Webadresse handelt es sich hierbei nicht um die URL⁷ der Homepage des APL sondern die Adresse, über die dieser später (auf elektronischem Weg⁸) Anfragen entgegen nehmen wird.

Zusätzlich zu den eben genannten, einfachen Attributen verfügt jeder Arbeitsplaner über einen *Angebotsvektor* (AGV), der aus einer Liste von Angeboten besteht. Dass es sich hierbei um eine Liste handelt ist wichtig, da bei der Kandidatensuche (vgl. Abschnitte 3.2.3 und 4.3.7) der Index eines Angebots im AGV zum identifizierenden Attribut wird.

Ein *Angebot* bezieht sich auf ein Geschäftsobjekt (GO) in einer Gewichts- und Größenklasse⁹. Weiterhin werden eine Menge von Teil- und Werkstoffklassen angegeben, in denen der Arbeitsplaner das GO projektieren kann. Diese beiden Mengen dürfen jeweils nicht leer sein. Für diese Kombination (aus Geschäftsobjekt mit Gewichts- und Größenklasse, Teil- und Werkstoffklassen) kann der APL dann eine (nicht-leere) Menge von Arbeitsplanungskompetenzen ausführen.

Zu jeder Arbeitsplanungskompetenz im Angebot gehört außerdem eine Menge von Bewertungen. Die Begründung, warum diese Bewertungen notwendig sind und das benutzte Berechnungsmodell werden im Abschnitt 2.4 erläutert. Die Kombination aus einer APLKP und den Bewertungen wird *Kompetenz* (KP) genannt.

Das vollständige Partialmodell *Arbeitsplaner* ist in Abbildung 2.5 dargestellt.

2.3 Partialmodell Anforderung

Eine Anforderung ist das Pendant zum Angebot eines Arbeitsplaners. Sie enthält eine Menge von Arbeitsplanungskompetenzen und die minimalen Anforderungen, die an diese gestellt werden. Diese APLKPen müssen für ein bestimmtes Geschäftsobjekt (aus einer Gewichts- und Größenklasse) in einer Menge von Werkstoff- und Teilklassen durchgeführt werden können. Die Auswertung einer solchen Anfrage wird im Kapitel 3, „Kandidatensuche“, beschrieben.

Abbildung 2.6 zeigt ein UML-Diagramm des Partialmodells Anfrage.

2.4 Bewertungsmodell

Der bisher vorgestellte Ausschnitt des in dieser Arbeit entworfenen Designmodells ermöglicht die Suche nach Arbeitsplanern anhand der bei ihnen vorhandenen Arbeitsplanungskompetenzen

⁷Uniform Resource Locator

⁸Zum Beispiel mit Webservices

⁹Bestehend aus den Attributen Höhe, Breite und Tiefe

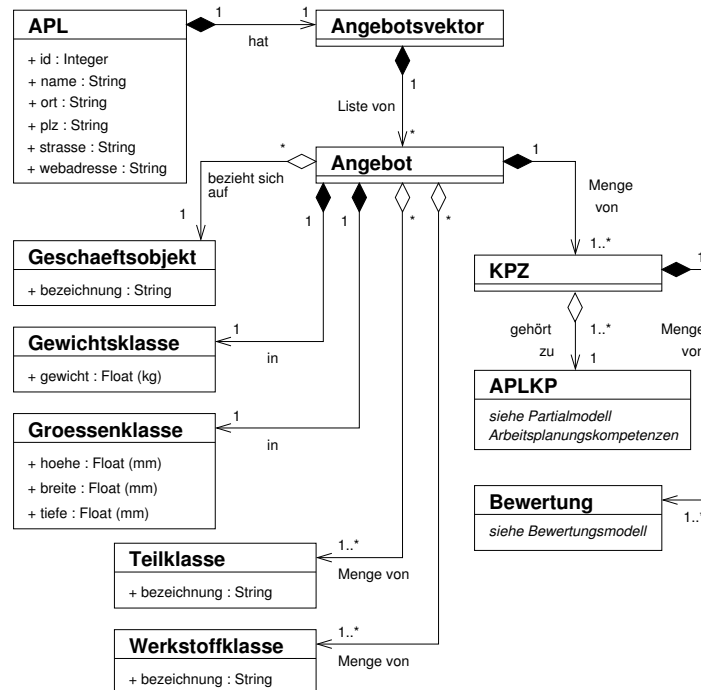


Abbildung 2.5: Partialmodell Arbeitsplaner

(bezogen auf Geschäftsobjekte in Teil- und Werkstoffklassen). Dieses Suchkriterium reicht im regulären Produktionsablauf möglicherweise nicht aus. Hier ist es nicht nur wichtig, welche Aktivitäten ein Planer ausführen kann, sondern auch, wie gut er diese beherrscht. Deswegen wird in diesem Abschnitt eine Möglichkeit vorgestellt, wie die Frage nach dem Grad der Qualifikation für eine APLKP beantwortet werden und in die Kandidatensuche einfließen kann.

Dazu wird zusätzlich zu den APLKPen in jedem Angebot eines Arbeitsplaners angegeben, wie qualifiziert der APL diese ausführen kann. Hierbei wird unterschieden zwischen Methoden bzw. nichtpersonellen Ressourcen – bei denen nur die Unterscheidung: „qualifiziert“ oder „nicht qualifiziert“ sinnvoll ist – und Aktivitäten, bei denen die Qualifikation differenzierter betrachtet werden kann. Dabei sind folgende umgangssprachlich ausgedrückten Bewertungsstufen vorgesehen, die dann in Zahlenwerte (in Klammern dahinter) transformiert werden¹⁰:

- „Expertenstatus“ (0,85)
- „Sehr erfahren“ (0,65)
- „Erfahren“ (0,40)
- „Durchführbar“ (0,15)

Wird eine Aktivität zum Angebotsvektor eines Arbeitsplaners hinzugefügt, so muss dieser angeben, mit welcher Qualifikation er sie ausführen kann.

¹⁰Bei Methoden und nichtpersonellen Ressourcen wird automatisch qualifiziert (1) angenommen.

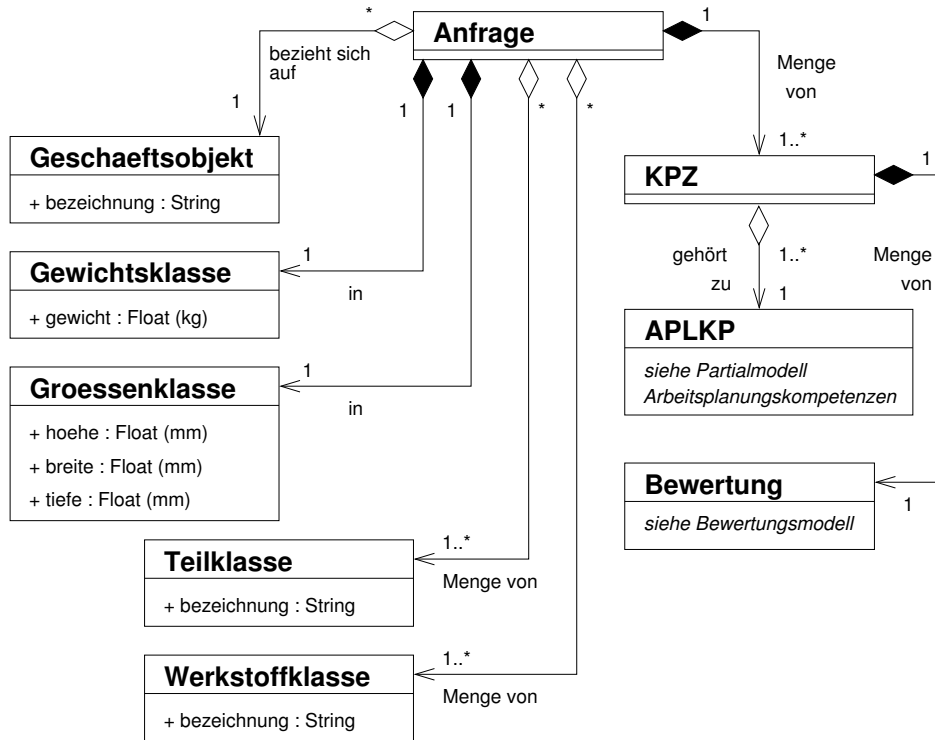


Abbildung 2.6: Partialmodell Anfrage

Bisher ist damit gesichert, dass eine konkrete Bewertung für eine APLKP im Angebotsvektor des Planers vorliegt. Allerdings wird diese Wertung vom APL selbst vergeben und ist deshalb subjektiv, muss also nicht den „realen Fähigkeiten“ des Anbieters entsprechen. Deshalb wird im Folgenden eine Möglichkeit beschrieben, diese subjektive Wertung an die tatsächlichen Leistungen des Arbeitsplaners anzupassen.

2.4.1 Anpassung der Bewertung

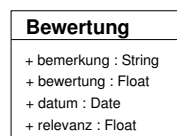


Abbildung 2.7: Klasse Bewertung

Abbildung 2.7 zeigt die UML-Darstellung der Klasse Bewertung. Jeder APLKP im Angebotsvektor eines APL werden mehrere solche Bewertungen zugeordnet (vgl. Abbildung 2.5). Eine davon (im Regelfall die erste) ist die Selbsteinschätzung des Arbeitsplaners. Alle weiteren Bewertungen kommen von anderen Instanzen. Je nachdem, von wem eine Bewertung stammt, wird

dieser eine bestimmte Relevanz zugeordnet. Die folgende Tabelle listet die möglichen Werte für Relevanz auf:

Quelle der Bewertung	Relevanz
Selbsteinschätzung des APL	0,3
Bewertung durch fachfremden Dritten	0,2
Bewertung durch fachkompetenten Dritten	0,5

Durch die Vergabe von verschiedenen Relevanz-Werten wird gesichert, dass subjektive oder nicht-kompetente Bewertungen weniger Gewicht haben, als die von fachkompetenten, (hoffentlich) objektiven dritten Parteien. Welchen Einfluss die Relevanz auf die Bestimmung der Durchschnitts- bzw. Ergebnisbewertung haben, erläutert der folgende Abschnitt¹¹.

2.4.2 Berechnung der Durchschnittsbewertung

Bei der Bestimmung einer adäquaten Formel zur Berechnung der Durchschnittsbewertung wird die Formel für das arithmetische Mittel als Ausgangspunkt gewählt. Diese lautet:

$$\bar{b} = \frac{1}{n} \cdot \sum_{i=1}^n b_i$$

wobei n die Anzahl der Bewertungen ist und b_i die einzelnen Bewertungen sind. In dieser Gleichung findet die Relevanz noch keine Beachtung. Um eine Gewichtung nach der Relevanz r_i durchzuführen wird jede Bewertung mit dem entsprechenden Wert für r multipliziert. Dieser muss allerdings vorher durch Division mit der Summe aller r_i normiert werden. Es wird also folgende Formel zur Berechnung der Durchschnittsbewertung angewendet:

$$\begin{aligned} \bar{b} &= \frac{1}{n} \cdot \sum_{i=1}^n b_i \cdot \frac{r_i}{\sum_{j=1}^n r_j} \\ &= \frac{1}{n \cdot \sum_{j=1}^n r_j} \cdot \sum_{i=1}^n b_i \cdot r_i \end{aligned}$$

Ein Beispiel soll den Sinn dieser Berechnung erläutern: Dazu wird angenommen, dass folgende Bewertungen vorliegen:

Bewertung	Relevanz
0,65	0,3
0,85	0,2
0,85	0,2
0,45	0,5
0,45	0,5

¹¹Vgl. dazu auch [JT02]

Das arithmetische Mittel beträgt 0,65. Der mit oben genannter Formel berechnete Durchschnittswert ist 0,58. Im zweiten Fall spiegeln sich die beiden schlechteren Bewertungen (je 0,45) mit hoher Relevanz (0,5) besser wider als bei der Verwendung des „einfachen“ Durchschnitts.

Das beschriebene Bewertungsmodell bietet eine Möglichkeit, die Bewertung eines Arbeitsplanners für eine bestimmte Arbeitsplanungskompetenz dynamisch an den real anzunehmenden Wert anzupassen. Die Frage, woher die zusätzlichen Bewertungen kommen, wird an dieser Stelle noch nicht beantwortet. Einen Lösungsansatz dazu wird Unterabschnitt 5.4.2 liefern.

2.5 Vollständiges Designmodell

Abbildung 2.8 zeigt das vollständige Designmodell.

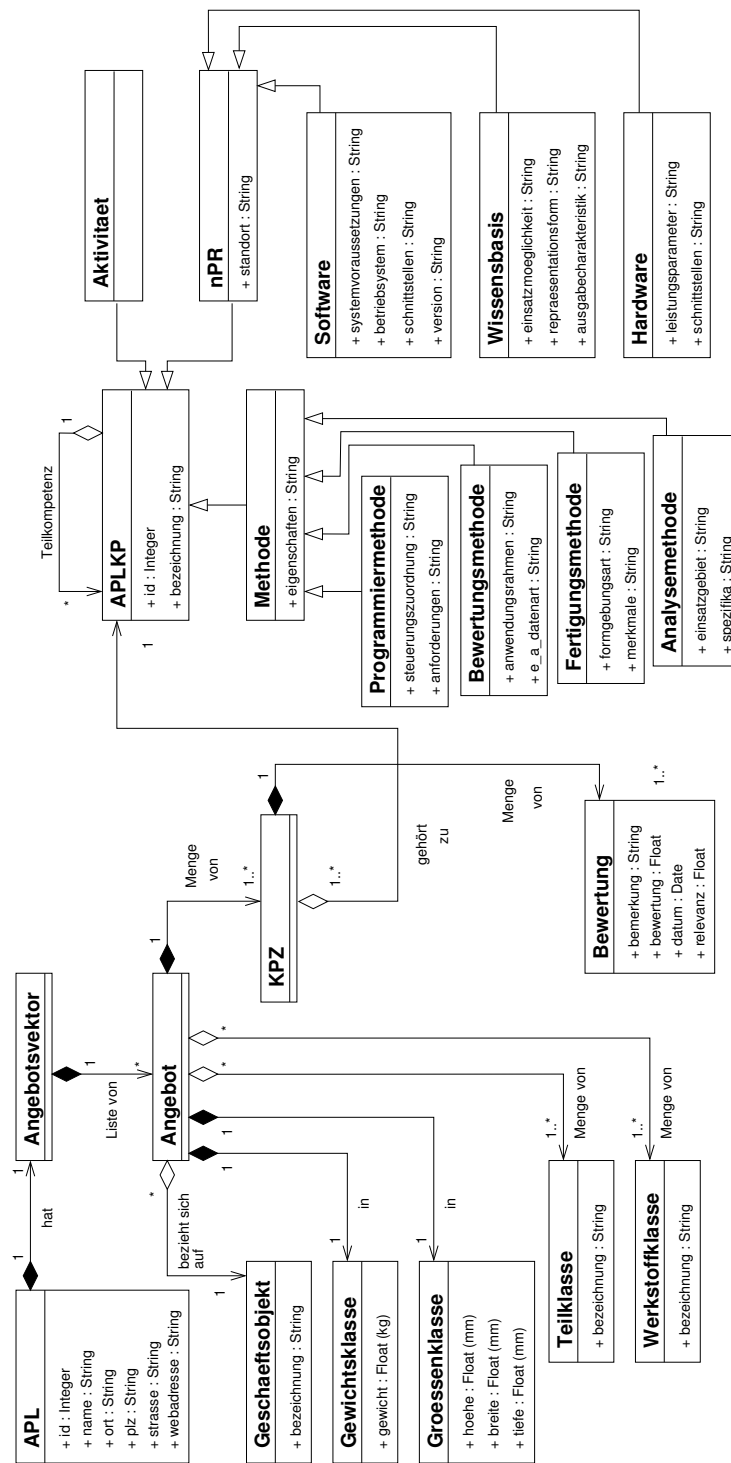


Abbildung 2.8: Vollständiges Designmodell

3 Lösungsansätze zur Kandidatensuche

Wenn dem System eine Menge von APLKPen (in einem Angebotsbaum), Geschäftsobjekten und Teil- und Werkstoffklassen bekannt gemacht wurden und in der Datenbank des IMK (IMKDB) abgelegt wurden, können Arbeitsplaner anhand dieser *Metainformationen* spezifiziert werden. Diese werden ebenfalls in der IMKDB gespeichert. Nun ist das System in der Lage, Anfragen (Kandidatensuchen) zu bearbeiten. In diesem Kapitel werden Möglichkeiten vorgestellt, wie diese Anfragebearbeitung vorgenommen werden kann.

3.1 Naiver Algorithmus

Der naive Algorithmus liest alle in der IMKDB vorhandenen Arbeitsplaner aus und betrachtet deren Angebotsvektoren. Für jedes Angebot aus dem AGV wird geprüft, ob es für die Anfrage relevant ist. Dazu müssen die Geschäftsobjekte übereinstimmen und in äquivalenten¹ Größen- und Gewichtsklassen vorliegen. Des Weiteren müssen alle Teil- und Werkstoffklassen der Anfrage in der Menge der Teil- und Werkstoffklassen des Angebots vorkommen. Die einzelnen Schritte der Prüfung, ob ein Angebot relevant ist, sind in Abbildung 3.1 noch einmal grafisch verdeutlicht.

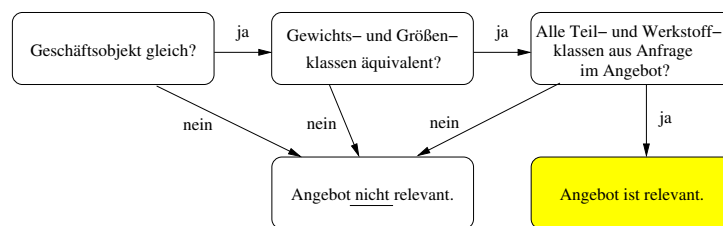


Abbildung 3.1: Stufen der Prüfung, ob ein Angebot relevant ist

Ist ein Angebot relevant, wird geprüft, ob dieses alle in der Anfrage geforderten Arbeitsplanungskompetenzen erfüllt. Dabei muss beachtet werden, dass eine APLKP auch dann als vorhanden gilt, wenn eine ihr übergeordnete Kompetenz im Angebot enthalten ist (vgl. Abschnitt 2.1). Es wird also für jede angebotene APLKP der Angebotsbaum durchlaufen, um zu bestimmen, welche gesuchten Kompetenzen ebenfalls angeboten werden. Diese Suche arbeitet rekursiv auf dem Angebotsbaum.

Sind alle gesuchten Arbeitsplanungskompetenzen im aktuell zu prüfenden Angebot vorhanden, so wird der Arbeitsplaner, zu dem es gehört, der Menge der *Kandidaten* hinzugefügt.

¹Vgl. Abschnitt 5.4.1

Im Folgenden eine formale Beschreibung des naiven Algorithmus:

Algorithmus: Naiver Algorithmus zur Kandidatensuche

Parameter: Anfrage A , Menge von Arbeitsplanern M aus IMKDB

Rückgabe: Menge von Arbeitsplanern

Return $\leftarrow \emptyset$

Für jeden APL aus M :

Erstelle Liste L mit Arbeitsplanungskompetenzen aus A

Markiere alle Elemente aus L als „unbefriedigt“

Für jedes $Angebot$ aus APL .Angebotsvektor:

Wenn $((Angebot.Geschaeftsobjekt == A.Geschaeftsobjekt) \wedge$

$(Angebot.Groesse \approx A.Groesse) \wedge$

$(Angebot.Gewicht \approx A.Gewicht))$:

Wenn $((A.Teilklassen \subseteq Angebot.Teilklassen) \wedge$

$(A.Werkstoffklassen \subseteq Angebot.Werkstoffklassen))$:

Für jede $APLKP$ aus $Angebot$:

Wenn $APLKP$ in L vorkommt:

Markiere $APLKP$ in L als „befriedigt“

Analog für jede Teilkp aus $APLKP$

Wenn alle Elemente in L als „befriedigt“ markiert:

Return $\leftarrow Return \cup \{APL\}$

Gib *Return* zurück

Der naive Algorithmus besteht aus zwei geschachtelten for-Schleifen über den Arbeitsplanern und deren Angeboten (im Angebotsvektor). Während der Abarbeitung der inneren Schleifen wird zusätzlich der Angebotsbaum rekursiv durchlaufen. Im schlechtesten Fall² wird dabei jede APLKP, die im Angebotsbaum definiert ist, einmal betrachtet.

Dieses Laufzeitverhalten ist sehr schlecht, insbesondere da in jedem Schritt die benötigten Objekte (vollständig) aus der Datenbank gelesen werden müssen. Aus diesem Grund ist es notwendig, eine Möglichkeit zu finden, die für die Kandidatensuche relevanten Daten zu extrahieren und die Anfrage dann über dieser Index-ähnlichen Struktur auszuführen.

3.2 Kandidatensuche mit GraphLearning

Für den schnelleren Abgleich einer Anfrage mit den zur Verfügung stehenden Angebotsvektoren der Arbeitsplaner wurde an der Professur Datenverwaltungssysteme der TU Chemnitz eine Softwarekomponente implementiert, die sich an die Signalverarbeitung im menschlichen Gehirn

²Dieser Fall tritt allerdings nur dann auf, wenn der Angebotsbaum eine Liste ist und der APL über die Kompetenz an der Wurzel verfügt.

anlehnt. Eine Beschreibung dieser Komponente wird in diesem Abschnitt vorgenommen, detailliertere Informationen sind in [GNM04] und [GN04] zu finden. Diese Softwarekomponente wird nachfolgend als *GraphLearning* (GL) bezeichnet.

3.2.1 Verwaltung der Angebotsprofile

Die Hierarchie der Arbeitsplanungskompetenzen, wie sie in Abbildung 2.2 auf Seite 7 dargestellt ist, wird im GraphLearning als Menge von Rezeptoren interpretiert, die miteinander verbunden sind, wenn eine Teilkompetenzen-Beziehung vorliegt³. Enthält ein Angebot eine APLKP, so wird der entsprechende Rezeptor erregt. Die Stärke der Erregung entspricht der Durchschnitts- bzw. Gesamtbewertung, die nach dem in Abschnitt 2.4 beschriebenen Modell bestimmt wird.

Um zu gewährleisten, dass das Vorhandensein einer Arbeitsplanungskompetenz, die Wurzel einer Funktionsgruppe ist, automatisch das Zurverfügungstehen aller Teilkompetenzen nach sich zieht, werden die Rezeptorerregungen zu den Kindknoten propagiert. Ein Angebot entspricht dann einem spezifischen Erregungsmuster des Rezeptorfeldes.

Die Erregungsmuster werden in zwei Schritten geclustert. Die erste, grobe Clusterung wird durch eine Komponente durchgeführt, die auf einem selbstorganisierenden künstlichen neuronalen Netz, dem *Growing Neutral Gas* (GNG) Netz, basiert. Dazu werden die Felder mehrfach an die Clusterkomponente angelegt, die dann typische Erregungsmuster lernt und die Angebotsprofile diesen zuordnet. Ähnliche Kombinationen und Stärken der markierten Arbeitsplanungskompetenzen werden dabei in eine Gruppe einsortiert.

Jeder der im ersten Schritt entstandenen Cluster wird danach einer weiteren, feineren Clusterung zugeführt. Dazu wird ein anderes neuronales Netz, das *Adaptive Resonance Theory* Netz [GC88], verwendet. Dieses Netz lernt neue Muster besser, ohne bereits bekannte wieder zu vergessen ([Zel94], [Roj93]).

3.2.2 Anfragebearbeitung im GraphLearning

Eine Anfrage stellt – genau wie ein Angebot – ein Erregungsmuster des Rezeptorfeldes dar. Im Gegensatz zum naiven Algorithmus, wie er in Abschnitt 3.1 vorgestellt wurde, der die „Propagierung von Kompetenzen“ durch eine rekursive Suche über dem Angebotsbaum simuliert, wird im GraphLearning die Propagierung der Rezeptorerregung auch in der Anfrage durchgeführt. Damit wird die Ähnlichkeit zwischen dem durch die Anfrage erstellten Erregungsmuster und dem Muster aus den Angebotsclustern sichergestellt.

Im nächsten Schritt wird das Suchmuster dem Grobcluster zugeordnet, zu dem es am besten passt (vgl. Abbildung 3.2). Dabei werden (wieder) ähnliche Kombinationen der Kompetenzen in eine Gruppe eingeordnet. Das verbleibende Muster wird der Feinclusterung zugeführt. Hier werden die entsprechenden Angebotsprofile geladen und mit der Anfrage verglichen. Enthält

³Vgl. UML-Diagramm in Abbildung 2.4

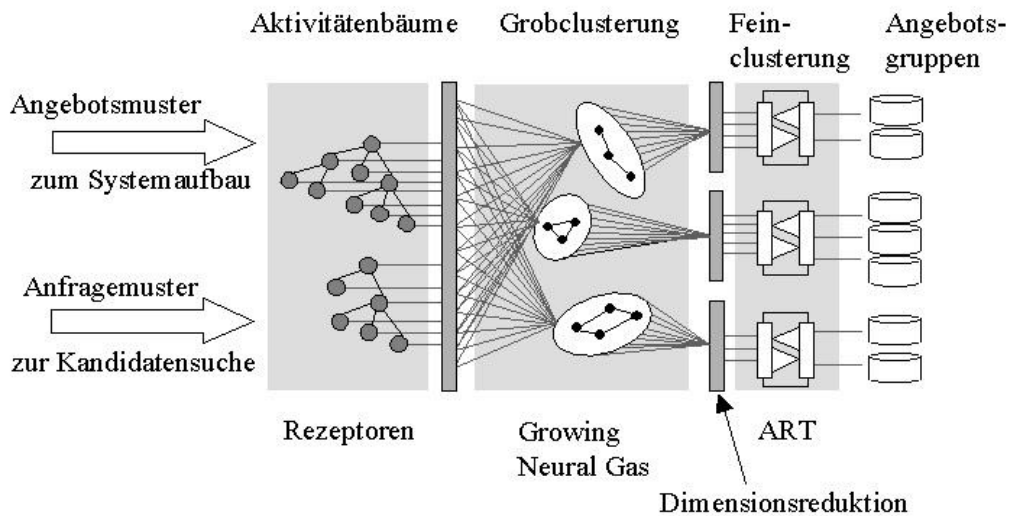


Abbildung 3.2: Angebotsprofile und Anfragen im GL, Quelle: [GNM04]

das Angebot die geforderten Kompetenzen (in der notwendigen Güte), so wird es in die Ergebnismenge eingeordnet.

3.2.3 Reduktion der Ergebnismenge auf relevante Profile

In den Rezeptorfeldern des GraphLearning werden (nur) die Arbeitsplanungskompetenzen in den einzelnen Angeboten registriert und für die Suche durch einen Musterabgleich verwendet. Dabei geht der Bezug der Angebote zu den Geschäftsobjekten, Gewichts-, Größen-, Teil- und Werkstoffklassen, wie er in Abbildung 2.5 gezeigt wird, verloren. Da diese Informationen nicht in den Datenstrukturen des GL enthalten sind, können sie auch bei der Suche nicht berücksichtigt werden. Das hat zur Folge, dass bei der Kandidatensuche mit GL Angebote in der Ergebnismenge sind, die zwar die richtigen Kompetenzen (mit einer ausreichend guten Bewertung) beinhalten, aber für ein anderes Geschäftsobjekt als das geforderte gelten.

Um die korrekte Ergebnismenge zu finden, muss das Ergebnis der GraphLearning Komponente nachträglich gefiltert werden. Dabei werden alle gefundenen Angebote auf ihre Relevanz für die Anfrage geprüft. Diese Prüfung erfolgt analog zum naiven Algorithmus für die Kandidatensuche (vgl. Abbildung 3.1) in drei Schritten: Vergleich des Geschäftsobjekts, Prüfung der Äquivalenz der Größen- und Gewichtsklassen und Prüfung, ob alle Teil- und Werkstoffklassen der Anfrage im Angebot enthalten sind.

Eingabe des Algorithmus ist eine Menge von Tupeln (Arbeitsplaner⁴, Angebotsnummer). Die Angebotsnummer ist der Index des Angebots im Angebotsvektor des APL. Daraufhin wird das

⁴Eindeutig identifiziert durch die id

bestimmte Angebot aus der Datenbank gelesen und geprüft. Ist es für die zu bearbeitende Anfrage relevant, so wird der Arbeitsplaner, zu dem das Angebot gehört, der korrekten Ergebnismenge hinzugefügt.

Nachfolgend wird der Algorithmus zu Korrektur des GraphLearning-Ergebnisses noch einmal spezifiziert:

Algorithmus: Korrektur der Ergebnismenge aus dem GL

Parameter: Anfrage A , Menge M von Tupeln $(APL.id, \#Angebot)$

Rückgabe: Menge von Arbeitsplanern

Return $\leftarrow \emptyset$

Für jedes $(APL.id, \#Angebot)$ aus M :

$APL \leftarrow$ Hole APL mit $APL.id$ aus IMKDB

$Angebot \leftarrow APL.getAngebotsvektor().getAngebot(\#Angebot)$

Wenn $((Angebot.Geschaeftsobjekt == A.Geschaeftsobjekt) \wedge$

$(Angebot.Groesse \approx A.Groesse) \wedge$

$(Angebot.Gewicht \approx A.Gewicht))$:

Wenn $((A.Teilklassen \subseteq Angebot.Teilklassen) \wedge$

$(A.Werkstoffklassen \subseteq Angebot.Werkstoffklassen))$:

$Return \leftarrow Return \cup \{APL\}$

Gib $Return$ zurück

Ist die Ergebnismenge nach der Korrektur leer, erfüllt keiner der APL die Anfrage. In diesem Fall werden die Arbeitsplaner als Ergebnis zurückgegeben, die vom GL ermittelt wurden⁵. Die aufrufende Instanz muss dann entscheiden, wie sie mit dem Ergebnis verfährt. Der in dieser Arbeit entwickelte Prototyp (wie er in Kapitel 4 beschrieben wird) zeigt dann einen entsprechenden Hinweis an.

In diesem Kapitel wurden zwei verschiedene Lösungsansätze für die Kandidatensuche diskutiert. Der naive Algorithmus ist sehr aufwendig und findet nur Arbeitsplaner, die die Anforderungen in einer Anfrage exakt erfüllen. Der zweite Ansatz verwendet die GraphLearning-Komponente, die eine Art Index über den Angeboten aller Arbeitsplaner erstellt. Die Suche wird dann über diesem Index ausgeführt. Deshalb ist dieses Verfahren effizienter. Außerdem wird es durch die Gruppierung der Angebotsprofile durch die Verwendung neuronaler Netze ermöglicht, Angebote, die der Anfrage ähnlich sind, zu finden. In dieser Arbeit wird deshalb die zweite Variante implementiert.

⁵Diese verfügen über alle geforderten APLKPen, bieten sie aber nicht für die gesuchte Kombination aus Geschäftsobjekt, Gewichts- und Größenklassen, und Teil- und Werkstoffklassen an.

4 Prototyp IMK-APL

In den beiden vorangegangenen Kapiteln wurden die theoretischen Grundlagen für die Beschreibung und Suche von Arbeitsplanern im Informationstechnischen Modellkern beschrieben. Aufgabe dieser Arbeit war es weiterhin, geeignete Schnittstellen zur Instanziierung des Designmodells und zur Kandidatensuche zu implementieren. Diese Implementierung erfolgte in zwei Schritten:

1. Entwicklung einer GUI¹ zum Importieren und Anzeigen der Metadaten und Arbeitsplaner sowie zur Kandidatensuche und
2. Entwicklung von *Webservices* zum Importieren von Arbeitsplanern und zur Kandidatensuche.

Die Details der Implementation und die dazu verwendeten Werkzeuge und Systeme werden in diesem Kapitel vorgestellt.

4.1 Beschreibung der Entwicklungsumgebung

Die Entwicklungsumgebung für die in dieser Arbeit entstandene Prototyp-Applikation, die den Namen *IMK-APL* trägt, war durch die Professur Datenverwaltung und die bisherigen Implementierungen am IMK (u. a. [Oes03]) größtenteils vorgegeben. In diesem Abschnitt werden die einzelnen Werkzeuge und Bibliotheken genauer vorgestellt.

4.1.1 Objektorientiertes Datenbankmanagementsystem

Als objektorientiertes Datenbanksystem kommt *FastObjects* zum Einsatz. *FastObjects* ist das Nachfolgeprodukt von *Poet*² und bietet unter anderem folgende Funktionen:

- Multi-User bzw. Multi-Prozess Zugriff
- Schemaevolution
- (optionale) Daten- bzw. Kommunikationsverschlüsselung
- ACID Transaktionen

¹Graphical User Interface (Grafische Benutzeroberfläche)

²Persistent Objects and Extended Database Technology

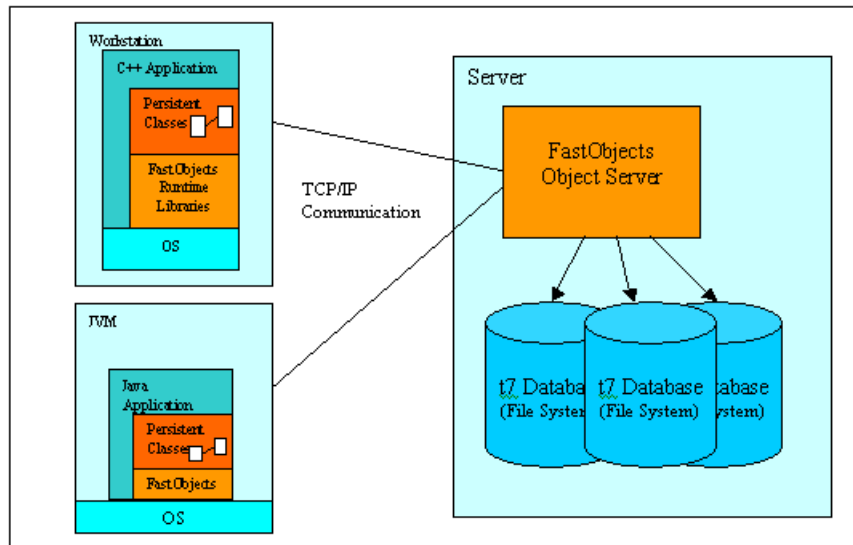


Abbildung 4.1: FastObjects: Struktur, Quelle: [FO]

Der Zugriff auf eine FastObjects-Datenbank kann mit C++, Java oder den mitgelieferten Entwicklungswerkzeugen erfolgen. Die Struktur des Systems ist in Abbildung 4.1 dargestellt. Weitere Informationen zu FastObjects sind in [FO] zu finden.

4.1.2 Programmiersprache

Als Programmiersprache wurde *Java* verwendet. Java ist eine von Sun Microsystems entwickelte, objektorientierte Programmiersprache. Ihre wichtigsten Merkmale sind:

- Anlehnung (in Aufbau und Syntax) an C++
- Plattformunabhängigkeit
- Sichere Speicherverwaltung (inkl. Garbage-Collection)

Weitere Einführungen zu Java findet man unter [Ull04] und [Kru03].

4.1.3 Grafische Benutzeroberfläche

Für die Entwicklung von grafischen Oberflächen stehen in Java zwei verschiedene integrierte Bibliotheken zur Verfügung: das *Abstract Windowing Toolkit* (AWT) und *Swing*. Swing gilt als Verbesserung des bereits in Version 1.0 des Java Development Kits (JDK) eingeführten AWT. Seine Vorteile sind (nach [Kru03]):

- Einheitliches Aussehen und Verhalten („Look and Feel“)

- Keine plattformspezifischen Besonderheiten
- Größere Menge an Dialogelementen als in AWT

Gleichzeitig stellt Swing aber höhere Anforderungen an die Rechenleistung als AWT. Die von IBM entwickelte Alternative zum Abstract Windowing Toolkit, das *Standard Widget Toolkit* (SWT), wird in dieser Arbeit nicht näher betrachtet.

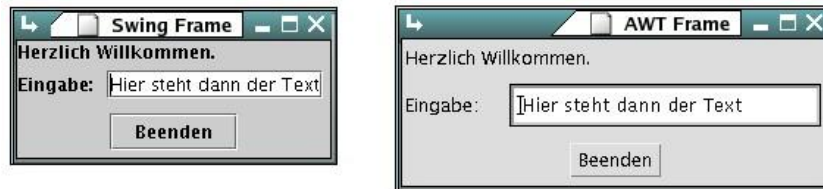


Abbildung 4.2: Ein Frame mit Swing (links) und AWT (rechts)

Aufgrund der genannten Vorteile verwendet IMK-APL die Swing-Bibliotheken zur Generierung grafischer Oberflächen.

4.1.4 XML-Parsing

Die *Extensible Markup Language* (XML) ist eine Sprache zur Beschreibung semistrukturierter Daten. Sie wurde vom World Wide Web Consortium³ (W3C) spezifiziert (vgl. [Min02]).

In dieser Arbeit wird XML dazu verwendet, die in Kapitel 2 beschriebenen Metainformationen, zum Beispiel über die vorhandenen Arbeitsplanungskompetenzen und Geschäftsobjekte, und die Informationen über Arbeitsplaner zu importieren. Dies kann über die grafische Oberfläche (siehe Abschnitt 4.3) oder über Webservices (vgl. Abschnitt 4.5) vorgenommen werden. Die dabei zu beachtenden Strukturen sind – mit Hilfe der *Document Type Definition* (DTD) Sprache – in Abschnitt 4.4 beschrieben.

Zum Parsen⁴ der XML-Dokumente wird die *Xerces* Bibliothek, die vom Apache XML Project⁵ entwickelt wird, verwendet. Xerces erlaubt den Zugriff auf XML-basierte Inhalte mit SAX⁶ oder dem vom W3C standardisierten DOM⁷. SAX ist eine ereignis-basierte Schnittstelle. Die Dokumente werden dabei sequentiell durchlaufen und analysiert. Tritt ein bestimmtes Ereignis⁸ auf, wird eine entsprechende Callback-Funktion aufgerufen. Weitere Informationen zu SAX sind unter [SAX] zu finden. Beim Zugriff über DOM wird aus einem XML-Dokument ein Objektbaum erzeugt. Dieser kann dann über entsprechende Methoden durchsucht und abgearbeitet werden.

³<http://www.w3.org>

⁴Grammatische Analyse

⁵<http://xml.apache.org>

⁶Simple API for XML

⁷Document Object Model

⁸Zum Beispiel „Auftreten eines öffnenden Tags“

Abbildung 4.3 zeigt ein Beispiel für eine Transformation im DOM. Die offizielle Homepage zum Document Object Model ist unter [DOM] zu finden.

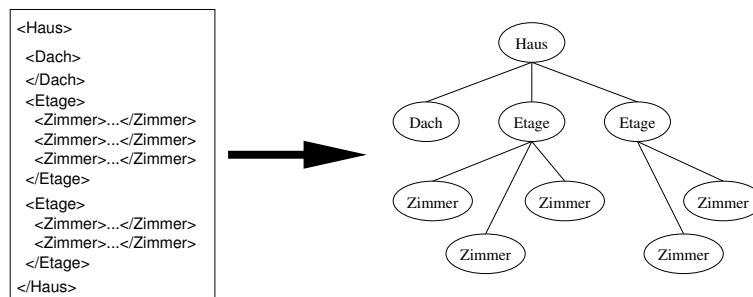


Abbildung 4.3: Transformation eines XML-Dokuments in einen Objektbaum

Die Applikation IMK-APL verwendet DOM zum Auslesen der XML- Dokumente, da die damit zur Verfügung stehenden Schnittstellen einfacher zu verwenden sind als ein SAX-Parser. Dadurch wird auch garantiert, dass der entstehende Programmcode besser wartbar ist. Gleichzeitig ist die Transformation eines XML-Dokuments in einen Objektbaum sehr rechenaufwändig. Darauf wird in Unterabschnitt 5.3.2 noch einmal eingegangen.

Die Verwendung von SAX und DOM mit Xerces und Java wird in [McL02] sehr detailliert beschrieben.

4.1.5 Zusammenfassung

Im ersten Abschnitt dieses Kapitels wurden die einzelnen verwendeten Werkzeuge und Bibliotheken vorgestellt und ihre Verwendung begründet. Die Entwicklung des Prototypen erfolgte parallel unter Windows und Linux. Damit wird sichergestellt, dass die Applikation später auf beiden Plattformen lauffähig ist. Nachfolgende Tabelle stellt beide Entwicklungsumgebungen, jeweils mit der benutzten Version der einzelnen Komponenten, gegenüber:

Windows XP	(Linux) Fedora Core 2, Kernel 2.6.8
Java SDK 1.4.2_04	Java SDK 1.4.2_04
FastObjects T7	FastObjects Trial Web 9.5.17.194-1
Xerces 2.5.0	Xerces 2.5.0
Tomcat 5.0.19	Tomcat 5.0.12
SOAP 2.3.1	SOAP 2.3.1

Die Werkzeuge Tomcat und SOAP und ihre Verwendung wurden bisher noch nicht erläutert. Sie werden in Abschnitt 4.5 vorgestellt.

4.2 Anbindung des GraphLearning

Die GraphLearning Komponente lag zu Beginn dieser Arbeit bereits als Implementierung in C++ vor. Angeboten wurden u. a. Funktionen zum Initialisieren, Trainieren und Anfragen des Systems. In diesem Abschnitt werden mehrere Möglichkeiten für die Anbindung des GraphLearning an die in Java implementierte Applikation IMK-APL vorgestellt.

4.2.1 Kommunikation über Prozesse

Die erste betrachtete Möglichkeit ist die Verwendung des GraphLearning als eigenständiger Prozess. Dazu werden zwei unabhängige Applikationen zum Initialisieren und Trainieren und zum Anfragen des GL implementiert. Diese können von der IMK-APL Applikation gestartet werden⁹. Zum Austausch von Daten (Angebotsprofile, Angebotsbaum, Anfragen) bieten sich dann Dateien oder die Standardein-/ausgabe an.

Die Kommunikation über eigenständige Prozesse ist einfach umzusetzen, insbesondere da das Verwenden des GraphLearning als Applikation schon vorgesehen (und implementiert) war. Allerdings bietet diese Art der Kommunikation auch einige Nachteile: Zum einen benötigt das Starten eines neuen Prozesses eine viel längere Zeit als das Aufrufen einer (Bibliotheks-) Funktion. Zum anderen könnte es vorkommen, dass die GL Applikation in einem undefinierten Zustand beendet wird¹⁰. Damit die aufrufende Instanz (IMK-APL) mit dieser Situation umgehen kann, müssen aufwändigere Prüffunktionen entworfen und umgesetzt werden.

4.2.2 Kommunikation über CORBA

Die *Common Request Broker Architecture* (CORBA) wurde 1991 von der Object Management Group (OMG) spezifiziert. Es handelt sich hierbei nicht um ein (Software-) Produkt, sondern um eine Definition von Protokollen und Diensten für verteilte Anwendungen in heterogenen Umgebungen.

CORBA ist nicht an eine bestimmte Programmiersprache gebunden. Stattdessen werden alle gemeinsam genutzten Anwendungsobjekte (Client- und Serverseitig) mittels der *Interface Definition Language* (IDL) beschrieben. Aus diesen IDL-Beschreibungen werden dann¹¹ Skelettklassen (zum Beispiel in C++ oder Java) erzeugt, die um die benötigte Programmlogik ergänzt werden. Alternativ kann das *Dynamic Invocation Interface* (DII) bzw. *Dynamic Skeleton Interface* (DSI) verwendet werden, um unbekannte Objekte ansprechen zu können.

Abbildung 4.4 zeigt die Architektur von CORBA. Zentrale Komponente ist dabei der *Object Request Broker* (ORB). Dieser nimmt die Daten des lokal laufenden Klienten entgegen und überträgt sie zum ORB des entfernt erstellten Objekts. Dabei kommt das *General Inter-ORB*

⁹Zum Starten von externen Programmen unter Java vgl. [Ull04].

¹⁰Zum Beispiel bei einem Abbruch des Prozesses durch Speicherzugriffsfehler

¹¹Mit Hilfe eines sogenannten IDL-Compilers

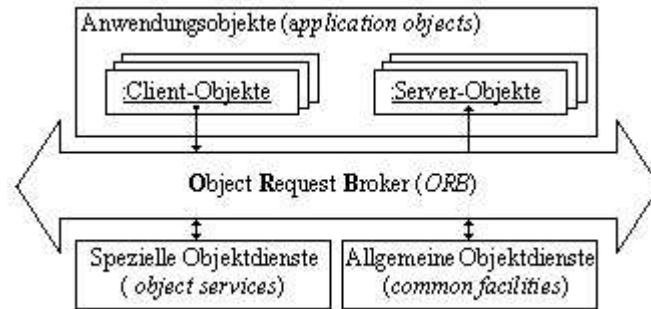


Abbildung 4.4: CORBA Architektur, Quelle [Oes03]

Protocol (GIOP) zum Einsatz. CORBA wird in [Pop98] genauer beschrieben. Die Verwendung von CORBA mit Java wird unter anderem in [Bog99] erklärt.

Beispiele für ORBs sind:

- MICO (<http://www.mico.org>),
- ORBit (<http://www.gnome.org/projects/ORBit2>),
- omniORB (<http://www.uk.research.att.com/omniORB>) und
- JacORB (<http://www.jacorb.org>).

Die Verwendung von CORBA zur Anbindung des GraphLearning an IMK-APL ist prinzipiell möglich, hat allerdings drei Nachteile: Alle Objekte, die in der Kommunikation benutzt werden müssten in der IDL beschrieben werden. Weiterhin müsste ein Object Request Broker ausgewählt, für das konkrete System konfiguriert und im laufenden Betrieb gewartet¹² werden. Schließlich entsteht durch die Benutzung von CORBA, das neben den genannten Merkmalen auch Netzwerkfähigkeit, Naming Service u. a. anbietet, im System Mehraufwand, der sich negativ auf das Laufzeitverhalten der Anwendung auswirkt.

4.2.3 Anbindung über JNI

Java-Programme laufen in einer virtuellen Maschine, der Java-VM, ab. Diese Vorgehensweise sichert die Plattformunabhängigkeit. In einigen Fällen ist allerdings der Zugriff auf native Bibliotheken nötig, zum Beispiel wenn ein bereits implementiertes Werkzeug (weiter) benutzt werden soll, die Benutzung plattformabhängiger Funktionen benötigt wird oder aufwändige Berechnungen abgearbeitet werden müssen. Um dies zu ermöglichen, wurde das *Java Native Interface* (JNI) entwickelt. Damit kann von Java-Programmen aus auf native Methoden zugegriffen werden aber auch die umgekehrte Verwendung (Zugriff auf Java-Funktionen von nativen Programmen) ist möglich. In den meisten Fällen beschränkt sich die Nutzung von JNI jedoch auf den erstgenannten Fall.

¹²Z. B. durch Einspielen sicherheitsrelevanter Updates

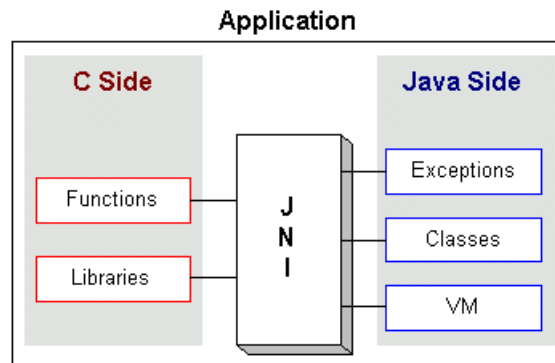


Abbildung 4.5: JNI als Bindeglied zwischen C und Java, Quelle [JNI2]

JNI ist seit Version 1.1 des JDK verfügbar. Der Compiler, mit dem die native Bibliothek implementiert werden soll, muss das Erstellen von *dynamischen Bibliotheken*¹³ unterstützen. Die einzelnen Schritte im Entwicklungsprozess mit JNI sind ([JNI1]):

1. Schreiben der Java Klasse
2. Compilieren der Java Klasse
3. Generieren einer C-kompatiblen Headerdatei aus der Java Klasse
4. Schreiben der C Methode(n)
5. Compilieren der C Bibliothek
6. Ansprechen der C Methode aus Java heraus

JNI bietet eine schlanke Schnittstelle zwischen Java-Applikationen und nativen Bibliotheken. Dies führt zu einem besseren Laufzeitverhalten als bei der Verwendung von CORBA. Außerdem ist der Programmcode zur Anbindung von externen Modulen mit JNI weniger aufwändig und damit leichter wartbar. Nachteilig ist die Tatsache, dass durch die Verwendung von nativem Code die Plattformunabhängigkeit verloren gehen kann. Da das GraphLearning unter Windows und Linux¹⁴ kompiliert werden kann, ist dieser Nachteil allerdings weniger schwerwiegend.

Deshalb wird in dieser Arbeit das Java Native Interface zur Kopplung des GraphLearning an die IMK-APL-Applikation verwendet.

4.2.4 Umsetzung

Wie im vorigen Unterabschnitt beschrieben, gliedert sich die Entwicklung mit JNI in sechs Schritte auf. Die Umsetzung dieser Schritte verlief in dieser Arbeit wie folgt:

¹³ .dll-Dateien unter Windows bzw. .so-Dateien im Unix/Linux

¹⁴ Windows und Linux werden in dieser Arbeit als relevante Betriebssysteme betrachtet.

Schritt 1. Schreiben der Java Klasse: Als erstes wurde die Java Klasse `GraphLearningWrapper` im Paket `de.dvs.apl.jni` erstellt. Diese enthält folgende Methodendeklarationen¹⁵:

- `initialize` (Initialisierung der GL Komponente)
- `loadTrainSet` (Laden einer Menge von Angebotsprofilen)
- `train` (Training der Netze)
- `createCluster` (Clusterbildung)
- `prepareTrainPatterns` (Trainingspattern vorbereiten)
- `initializeART` (ART-Netze initialisieren)
- `trainART` (ART-Netze trainieren)
- `print` (Ausgabe von Statusmeldungen)
- `setCurDir` (Arbeitsverzeichnis¹⁶ festlegen)
- `query` (Anfrage stellen)

Alle aufgeführten Methoden wurden durch das Schlüsselwort `native` als native Funktionen gekennzeichnet. Eine Implementierung erfolgte also in der Java Klasse nicht.

Schritt 2. Compilieren der Java Klasse: Zum Compilieren der im ersten Schritt erstellten Klasse wurde der Java Compiler (`javac`) aus dem JDK benutzt. Der exakte Aufruf lautet:

```
javac de/dvs/apl/jni/GraphLearningWrapper.java
```

Schritt 3. Generieren einer C-kompatiblen Headerdatei aus der Java Klasse Zum Generieren der C bzw. C++ kompatiblen Headerdatei wurde das Tool `javah`¹⁷ verwendet. Diesem wird der qualifizierte Name der Java Klasse als Parameter mitgegeben. Der Aufruf lautet also:

```
javah -jni de.dvs.apl.jni.GraphLearningWrapper
```

Der Parameter `-jni` ist optional, da JNI der Standardmodus für die Ausgabe ist. Als Ergebnis entsteht die Headerdatei `de_dvs_apl_jni_GraphLearningWrapper.h`.

Schritt 4. Schreiben der C Methode(n): Die im dritten Schritt entstandene Headerdatei enthält die Methodendeklarationen (analog zur Java Klasse). Diese Methoden mussten nun implementiert werden. Dazu wurde die Datei `graphlearning.cc` angelegt. In dieser Datei wurden alle Methodenköpfe übernommen und die Methoden ausprogrammiert.

Die Implementation erfolgte analog zu einem Beispiel, das in der `GraphLearning` Komponente enthalten war. Dabei musste beachtet werden, dass in jeder Methode als erster Schritt das aktuelle (Arbeits-) Verzeichnis neu gesetzt werden musste, damit das GL seine eigenen erzeugten Dateien wiederfinden konnte. Dazu wurde die Funktion `gotoCurDir` verwendet.

¹⁵Genaue Beschreibung und Parameterlisten siehe Klassendokumentation

¹⁶Hier werden durch das `GraphLearning` einige Dateien erstellt und gelesen.

¹⁷C header and stub file generator, ebenfalls aus dem JDK

Schritt 5. Compilieren der C Bibliothek: Zum Compilieren der C Bibliothek wurde sowohl unter Linux als auch unter Windows der *GNU C Compiler* (GCC) verwendet. Dabei unterscheidet sich der Aufruf in den beiden Betriebssystemen. Wichtigster Unterschied ist der Dateiname des entstehenden Moduls. Unter Linux lautet dieser `libgraphlearning.so` und unter Windows `graphlearning.dll`. In beiden Fällen mussten noch einige Bibliotheken aus dem GraphLearning eingebunden werden, nämlich `nno`¹⁸ und `artgal`¹⁹.

Den korrekten Aufruf zum Compilieren der Bibliothek unter Linux bzw. Windows findet man in dem dafür erstellten und verwendeten Makefile²⁰ auf der CD, die dieser Arbeit beigelegt ist (siehe Abschnitt D im Anhang).

Schritt 6. Ansprechen der C Methode aus Java heraus Um die GraphLearning Komponenten nun aus der Java-Applikation heraus zu nutzen, muss eine Instanz der Klasse `GraphLearningWrapper` erstellt werden. Dabei wird die native Bibliothek mit dem von Java zur Verfügung gestellten Befehl `System.loadLibrary` geladen. Sie muss sich dazu im Suchpfad des Betriebssystems befinden²¹. Wird die Moduldatei nicht gefunden oder kann sie aus irgendeinem Grund nicht geladen werden, so wird eine Ausnahme (*Exception*) geworfen. Diese Ausnahme sollte von der aufrufenden Instanz abgefangen und bearbeitet werden²².

Ist das Objekt vom Typ `GraphLearningWrapper` erfolgreich erstellt worden, kann über dieses auf die Funktionen des GL zugegriffen werden. Abbildung 4.6 zeigt ein Beispiel, dass die eben beschriebenen Schritte demonstriert.

4.2.5 Zusammenfassung

Im zweiten Abschnitt dieses Kapitels wurden drei Möglichkeiten zur Anbindung des GraphLearning, das in C++ implementiert ist, an die in Java geschriebene Prototypapplikation IMK-APL erläutert. In dieser Arbeit wurde dann das Java Native Interface, JNI, benutzt. Die Implementierungsschritte sind im letzten Unterabschnitt nachvollzogen und ein Beispiel für die Benutzung der entstandenen Schnittstelle angegeben worden. JNI hat sich als einfache und gut dokumentierte Methode erwiesen, um mit Java auf native Bibliotheken zuzugreifen.

4.3 Benutzung von IMK-APL

In diesem Abschnitt wird die Grafische Oberfläche (GUI) der Prototypapplikation IMK-APL vorgestellt. Sie stellt eine einfache und interaktive Schnittstelle zur Instanziierung des Designmodells und zur Kandidatensuche dar. Nachfolgend eine Auflistung aller über die GUI zu erreichenden Funktionen:

¹⁸<http://www.ep1.ruhr-uni-bochum.de/marcel/RhoNNO.html>

¹⁹<http://cns-web.bu.edu/pub/laliden/WWW/nnet.html>

²⁰Zur Verwendung mit dem Tool `make`

²¹Unter Linux: `$LD_LIBRARY_PATH` und unter Windows: `%PATH%`

²²Zum Konzept der Exceptions in Java vgl. [Ull04] bzw. [Kru03]

```
1 import de.dvs.apl.jni.GraphLearningWrapper;
2 import java.lang.UnsatisfiedLinkError;
3
4 public class GraphLearningDemo {
5     public static void main(String[] args) {
6         GraphLearningWrapper wrapper;
7
8         try {
9             wrapper = new GraphLearningWrapper();
10        }
11        catch (UnsatisfiedLinkError e) {
12            System.err.println("Bibliothek nicht gefunden:\n" +
13                e.getMessage());
14            System.exit(-1);
15        }
16        catch (Exception e) {
17            System.err.println("Konnte Bibliothek nicht laden:\n" +
18                e.getMessage());
19            System.exit(-1);
20        }
21
22        wrapper.setCurDir("/tmp/graphlearning");
23        wrapper.initialize();
24        ...
25    }
26 }
```

Abbildung 4.6: Beispiel für die Verwendung des GraphLearning mit Java

- Laden und Anzeigen der Metainformationen (Arbeitsplanungskompetenzen, Geschäftsobjekte, Teil- und Werkstoffklassen)
- Anlegen und Anzeigen von Arbeitsplanern
- Importieren von Arbeitsplanern
- Initialisieren der Kandidatensuche
- Kandidatensuche durchführen
- Alle Informationen der Datenbank löschen

Zum Importieren der Metainformationen und Arbeitsplaner sowie bei der Kandidatensuche werden XML-Dokumente als Datenquelle verwendet. Diese werden im nächsten Abschnitt definiert.

Abbildung 4.7 zeigt die Applikation IMK-APL, so wie sie dem Nutzer nach dem Start präsentiert wird²³. Alle Funktionen werden über die entsprechenden Menüeinträge erreicht. Die Konfiguration der Applikation wird während der Installation, die in Abschnitt 4.6 beschrieben wird, vorgenommen.

In den folgenden Unterabschnitten werden die wichtigsten Funktionen und ihre Umsetzung in der grafischen Oberfläche erläutert.

²³Alle in dieser Arbeit verwendeten Screenshots wurden unter Linux und dem Desktopsystem GNOME (GNU Network Object Model Environment) erstellt.



Abbildung 4.7: Screenshot („Bildschirmfoto“) der IMK-APL GUI

4.3.1 Laden der Metainformationen

Als Metainformationen (vgl. Kapitel 3) werden die Arbeitsplanungskompetenzen (angeordnet in einem Angebotsbaum), Geschäftsobjekte, Teilklassen und Werkstoffklassen bezeichnet. Sie müssen dem System bekannt gemacht werden, damit sich Arbeitsplaner spezifizieren und Anfragen bearbeitet werden können.

Die Funktion zum Laden/Importieren der Metainformationen kann über die Menüeinträge APL-Kompetenzen → Metainformationen laden erreicht werden. Nun kann die XML-Datei ausgewählt werden, die als Datenquelle dient. Diese wird dann ausgelesen und Objekte mit den angegebenen Daten werden erstellt. Alle Arbeitsplanungskompetenzen werden dabei in der Reihenfolge ihres Auftretens im XML-Dokument nummeriert.

Im nächsten Schritt wird geprüft, ob sich schon ein Angebotsbaum in der Datenbank des IMK befindet. Ist dies der Fall, so schlägt der Import fehl, es darf immer nur ein Angebotsbaum existieren (vgl. Partialmodell Arbeitsplanungskompetenzen im Abschnitt 2.1). Danach werden die Objekte in der IMKDB gespeichert. Dabei wird für jedes Objekt geprüft, dass die Bezeichnung in der Extension²⁴ der Klasse, zu der es gehört, eindeutig ist. Ansonsten kann es nicht gespeichert werden. Existiert zum Beispiel in der Datenbank ein Geschäftsobjekt mit der Bezeichnung „NC-Programm“, so kann eine nichtpersonelle Ressource mit der gleichen Bezeichnung importiert werden, aber kein weiteres Geschäftsobjekt mit diesem Namen.

Ist das Laden der Metainformationen erfolgreich beendet worden, wird dem Nutzer eine Statistik angezeigt, die die Anzahl der importierten Arbeitsplanungskompetenzen, Geschäftsobjekte, Teil- und Werkstoffklassen enthält. Ein Screenshot dieser Statistik ist im Anhang B „Weitere Screenshots“ (ab Seite 55) abgebildet.

²⁴Vgl. [Heu92]

4.3.2 Anzeigen des Angebotsbaums

Über die Menüpunkte APL-Kompetenzen → Angebotsbaum anzeigen kann sich der Nutzer die Hierarchie der Arbeitsplanungskompetenzen anzeigen lassen. Dabei wird das Swing-Widget²⁵ JTree verwendet, dass sich an die Anzeige von Verzeichnisbäumen anlehnt (siehe Abbildung 4.8).

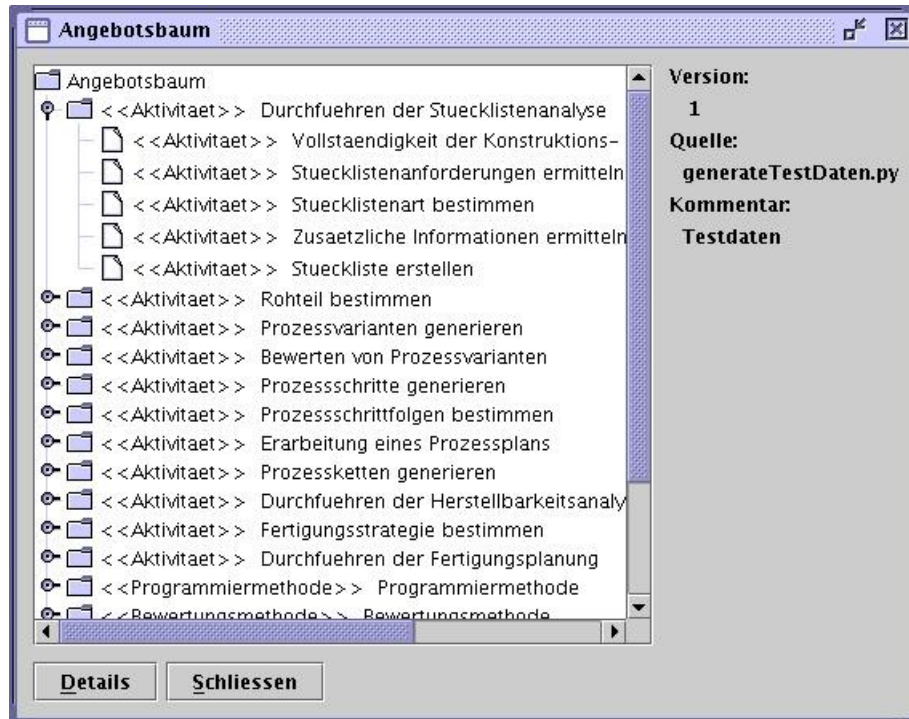


Abbildung 4.8: Anzeige der APLKPen

Zu jeder APLKP wird angezeigt, um was für eine Art Kompetenz (Aktivität, Bewertungsmethode, Fertigungsmethode usw.) es sich handelt. Durch Anklicken des Schalters²⁶ Details kann sich der Nutzer die genauen Eigenschaften der Kompetenz anschauen. Welche Eigenschaften zu einer Kompetenz gehören, ist abhängig davon, aus welcher Unterklasse von APLKP sie stammt, man beachte hierzu das UML-Diagramm in Abbildung 2.1. Ein Screenshot eines Detail-Fensters ist ebenfalls im Anhang B zu finden.

Zusätzlich zur Hierarchie der Arbeitsplanungskompetenzen werden noch Version, Quelle und Kommentar des Angebotsbaums angezeigt.

4.3.3 Anlegen von Arbeitsplanern

Die Beschreibung eines Arbeitsplaners erfolgt in IMK-APL in zwei Schritten:

²⁵Bezeichnung für vordefinierte Komponenten in der GUI-Programmierung

²⁶Button

1. Anlegen des APL und
2. Hinzufügen der einzelnen Angebote²⁷ des APL,

wobei der letzte Schritt für jedes Angebot aus dem AGV des Arbeitsplaners wiederholt werden muss²⁸.

Abbildung 4.9: Anlegen eines Arbeitsplaners

Das Dialogfenster zum Anlegen eines APL-Objekts kann unter Arbeitsplaner → Anlegen aufgerufen werden. Es ist in Abbildung 4.9 dargestellt. In diesem Fenster müssen Name, Ort, PLZ, Straße und Webadresse²⁹ des Arbeitsplaners angegeben werden. Sind alle Angaben erfolgt, kann der APL gespeichert werden. Ihm wird dann vom System eine eindeutige id zugewiesen.

Ist der Arbeitsplaner angelegt worden, können Angebote zu seinem Angebotsvektor hinzugefügt werden. Die entsprechende Funktion kann der Nutzer mit Arbeitsplaner → Angebote hinzufügen erreichen. Im nächsten Schritt muss er den APL auswählen, zu dem das neue Angebot gehört. Dazu werden ihm alle in der Datenbank vorhandenen Arbeitsplaner (geordnet nach der ID, die ihnen vom System gegeben wurde) aufgelistet.

Danach muss das Geschäftsobjekt ausgewählt werden, auf das sich das Angebot bezieht. Im gleichen Dialogfeld werden die Werte für die Gewichts- und Größenklasse³⁰ angegeben. Als nächstes sind die Teil- und Werkstoffklassen anzugeben, in denen das Geschäftsobjekt geplant werden kann. Dabei ist es möglich, mehrere Klassen zu bestimmen. Gleichzeitig muss aus jeder der beiden Mengen mindestens ein Element selektiert werden.

Sind das Geschäftsobjekt (mit Gewichts- und Größenklasse) und die Teil- und Werkstoffklassen bestimmt, so können die Arbeitsplanungskompetenzen festgelegt werden, über die der Arbeitsplaner für diese Kombination verfügt. Dazu wird dem Nutzer der Angebotsbaum (wie er in Abbildung 4.8 dargestellt ist) präsentiert. Aus diesem kann die Kompetenz ausgewählt werden. Dabei wird geprüft, dass diese nicht schon im Angebot vorhanden ist. Handelt es sich bei der selektierten APLKP um eine Aktivität, muss im nächsten Dialogfenster die Eigenbewertung (vgl. Abschnitt 2.4) des Arbeitsplaners angegeben werden. Der Nutzer kann aus den Werten

²⁷Im Angebotsvektor

²⁸Vgl. Unterabschnitt 5.1

²⁹Nicht URL der Homepage, vgl. Abschnitt 2.2

³⁰Besteht aus Höhe, Breite und Tiefe

- „Expertenstatus“ (0,85)
- „Sehr erfahren“ (0,65)
- „Erfahren“ (0,40)
- „Durchführbar“ (0,15)

wählen.

Die Dialoge der drei zuletzt beschriebenen Funktionen sind im Anhang B in den Abbildung B.3 bis B.5 zu sehen. Sind alle Komponenten des Angebots angegeben, kann es in der IMKDB gespeichert werden.

4.3.4 Importieren von Arbeitsplanern

Im vorigen Unterabschnitt wurde das Anlegen von Arbeitsplanern unter Benutzung der grafischen Oberfläche beschrieben. Dieser Vorgang ist sehr aufwändig, besonders wenn der APL viele verschiedene Angebote (in Bezug auf die Kombination Geschäftsobjekt (Gewicht, Größe), Teil- und Werkstoffklassen und Arbeitsplanungskompetenzen) anlegen möchte (vgl. Unterabschnitt 5.1). Deshalb gibt es die Möglichkeit, einen APL aus einer XML-Datei zu lesen und in die Datenbank zu importieren. Diese Funktion ist im Menü unter Arbeitsplaner → Laden zu finden.

Der Nutzer muss dann die Datei auswählen, die die Daten des Arbeitsplaners enthält. Diese wird geparkt und ein entsprechendes Objekt vom Typ APL erstellt. Auch hier wird dem APL eine eindeutige ID vom System gegeben. Beim Anlegen der einzelnen Angebote wird geprüft, ob die angegebenen Metainformationen auch in der IMKDB existieren, ist dies nicht der Fall, schlägt der Import fehl. Bezieht sich zum Beispiel ein Angebot auf das Geschäftsobjekt „Stückliste“ und dieses wurde vorher nicht in der Datenbank angelegt, so enthält der Nutzer eine entsprechende Meldung und der Arbeitsplaner kann nicht gespeichert werden.

Läuft der Importvorgang erfolgreich ab, wird der APL in der Datenbank des IMK abgelegt.

4.3.5 Anzeigen von Arbeitsplanern

Der Nutzer kann sich die angelegten bzw. importierten Arbeitsplaner unter Arbeitsplaner → Anzeigen anzeigen lassen. Dabei muss im ersten Schritt der gewünschte APL ausgewählt werden. Danach werden seine „persönlichen Daten“³¹ ausgegeben. Durch Anklicken des Buttons Angebote, können die einzelnen Angebote im Angebotsvektor des Arbeitsplaners angeschaut werden. Alle Informationen zu einem Angebot werden hier in einem Fenster dargestellt (siehe Abbildung 4.10).

³¹Name, Ort, etc.

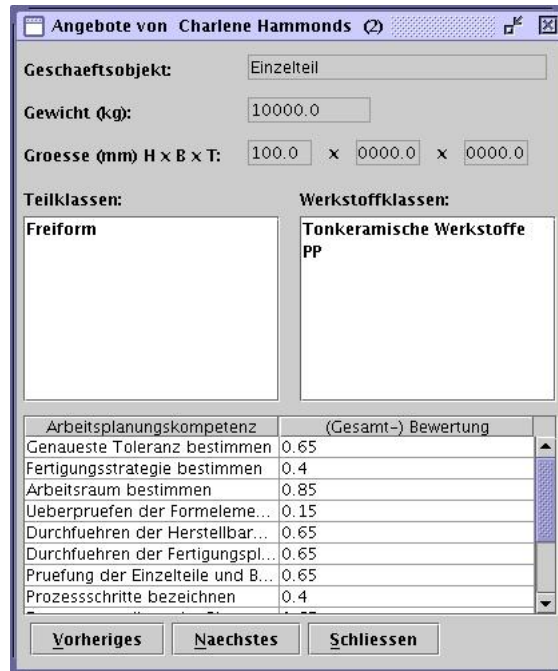


Abbildung 4.10: Anzeigen eines Angebots

4.3.6 Initialisieren der Kandidatensuche

Im Menü Kandidatensuche können die GraphLearning-Komponente initialisiert und die Netze³² trainiert werden. Dazu werden zwei Dateien mit den vom GraphLearning benötigten Informationen erstellt: Die erste Datei enthält den Angebotsbaum in Form einer Adjazenzmatrix³³. Die zweite Datei enthält alle Angebote, die in der IMKDB vorhandenen Arbeitsplaner bereitstellen. Dabei entspricht jede Zeile einem Angebot. In einem dritten File wird die Zuordnung: Nummer des Angebots im GraphLearning ↔ ID des Arbeitsplaners und Nummer des Angebots im AGV gespeichert.

Sind die Dateien geschrieben, werden die folgenden Funktionen aus dem GraphLearningWrapper aufgerufen:

- initialize,
- loadTrainSet,
- train,
- createCluster,
- prepareTrainPatterns,

³²Vgl. Abschnitt 3.2

³³Eine Adjazenzmatrix ist die Darstellung eines Graphen mit n Knoten in Form einer $n \times n$ -Matrix A . Dabei ist der Wert von $A[i, j] = 1$, wenn eine Kante vom Knoten i zum Knoten j verläuft.

- initializeART und
- trainART.

Der Methode initialize wird der Name der Datei, die den Angebotsbaum als Adjazenzmatrix enthält, als Parameter übergeben. Der Name des Files mit den Angeboten wird an die Funktion loadTrainSet weitergereicht. Alle anderen Funktionen benötigen keine Parameter. Wenn bei der Bearbeitung der Methoden keine Fehler bzw. Ausnahmen auftreten, ist die GraphLearning-Komponente initialisiert und es können Anfragen beantwortet werden.

4.3.7 Kandidatensuche

Über die Menüeinträge Kandidatensuche → Anfrage kann eine Anfrage an das System gestellt werden. Dies setzt voraus, dass die im letzten Unterabschnitt beschriebene Initialisierung der GL-Komponente bereits erfolgt ist. Eine Anfrage wird ebenfalls als XML-Dokument eingelesen. Dabei wird – wie beim Importieren eines Arbeitsplaners – geprüft, ob alle genannten Metainformationen in der Datenbasis des IMK vorhanden sind. Ist dies nicht der Fall, wird die Anfrage zurückgewiesen. Andernfalls wird das erstellte Anfrageobjekt an die Kandidatensuche, die im Kapitel 3, „Kandidatensuche“, detailliert beschrieben wird, übergeben. Das Ergebnis wird dem Nutzer dann in einem entsprechenden Dialog (siehe Abbildung 4.11) angezeigt.



Abbildung 4.11: Ergebnis der Kandidatensuche

4.4 Definition der XML-Dokumente

XML-Dokumente werden in dieser Arbeit an vier Stellen als Datenquellen verwendet: Für

1. den Import von Metainformationen,
2. den Import von Arbeitsplanern,
3. Anfragen und

4. Rückgaben bei der Kandidatensuche als Webservice³⁴

In diesem Abschnitt werden die Strukturen der XML-Dokumente definiert.

Folgt eine XML-Datei einer vorgegebenen Struktur, so wird sie *gültig* (*valid*) genannt ([Min02]). Die drei wichtigsten Sprachen zur Definition von XML-Strukturen³⁵ sind

- Document Type Definition,
- XML Schema und
- Relax NG.

Diese drei Sprachen werden unter anderem in [Kun04] ausführlich erläutert und gegenübergestellt.

In dieser Arbeit wird die *Document Type Definition* (DTD) Sprache verwendet. Sie ist die älteste der drei genannte Schemasprachen und wurde 1999 vom W3C eingeführt. Dabei wird die Struktur der Instanzdokumente durch grammatikbasierte Muster vorgegeben. DTD gilt als schnell erlernbar, ist dafür aber nicht so ausdrucksstark wie zum Beispiel XML Schema. Da es sich bei den hier beschriebenen Dokumentstrukturen jedoch um relativ einfache handelt, reicht die Mächtigkeit von DTD aus.

4.4.1 Beschreibung der Metainformationen

Die in diesem Unterabschnitt beschriebene XML-Struktur gilt für die Datenquellen beim Import der Metainformationen.

Das Wurzel-Tag (*Root*) eines XML-Dokuments, das die Metainformationen für IMK-APL enthält, muss den Namen *meta* haben. Unterelemente von *meta* sind dann die Arbeitsplanungskompetenzen, Geschäftsobjekte (*geschaeftsobjekt*), Teilklassen (*teilklass*) und Werkstoffklassen (*werkstoffklasse*). Jedes Element, das eine APLKP beschreibt hat einen der Namen:

- *aktivitaet*,
- *programmiermethode*,
- *bewertungsmethode*,
- *fertigungsmethode*,
- *analysemethode*,
- *software*,
- *wissensbasis* oder
- *hardware*.

³⁴Vgl. Unterabschnitt 4.5.3

³⁵sogenannte *Metasprachen* oder auch *Schemasprachen*

Methoden und nichtpersonelle Ressourcen, die nicht Element einer der genannten Unterklassen sind, dürfen nicht vorkommen.

Während die Geschäftsobjekte, Teil- und Werkstoffklassen alle nur über ein Attribut bezeichnung verfügen, haben die Arbeitsplanungskompetenzen spezifische Eigenschaften. Diese sind aus dem UML-Diagramm in Abbildung 2.4 ersichtlich. Jedes Element, das eine APLKP beschreibt, kann außerdem Unterelemente haben, die dann allerdings vom gleichen Typ (z. B. aktiviert) sein müssen. Dies sind die Teilkompetenzen (vgl. Abschnitt 2.2).

Die DTD für die Metainformationen ist im Anhang „Document Type Definitions“, Abschnitt C.1, zu finden.

4.4.2 Beschreibung der Arbeitsplaner

Ein Arbeitsplaner (apl) hat fünf obligatorische Attribute: name, ort, plz, strasse und webadresse. Diese Attribute beschreiben die „persönlichen Daten“ des Arbeitsplaners. Außerdem kann in der XML-Beschreibung eines Arbeitsplaners bereits ein Angebotsvektor angegeben werden³⁶, dieser muss dann mindestens ein Angebot enthalten.

Jedes Angebot im AGV (vergleiche Abschnitt 2.2) bezieht sich auf ein bestimmtes Geschäftsobjekt, das durch seine Bezeichnung identifiziert wird, in einer Größen- und Gewichtsklasse. Da diese Eigenschaften pro Angebot genau einmal vorkommen, werden sie als Attribute spezifiziert. Außerdem wird eine Menge von Teil- und Werkstoffklassen angegeben (jeweils mindestens eine). Zuletzt werden die Arbeitsplanungskompetenzen³⁷ genannt, auf die sich das Angebot bezieht. Eine aplkpz besteht aus zwei Attributen: bezeichnung, dem Namen, über die die Kompetenz identifiziert wird, und bewertung, in dem die (subjektive) Eigenbewertung des Arbeitsplaners für diese APLKP angegeben wird. Die Instanz, die das XML-Dokument erzeugt, sollte dafür sorgen, dass bei Methoden und nichtpersonellen Ressourcen der Wert für bewertung automatisch 1 ist³⁸.

Die formale Beschreibung der XML-Struktur für den Import von Arbeitsplanern ist ebenfalls im Anhang 4.4 zu finden.

4.4.3 Anfragen

Eine Anfrage stellt (vergleiche Abschnitt 2.3) das Gegenstück zu einem Angebot dar. Daraus folgt, dass für ein anfrage-Element ebenfalls die Attribute

- geschaeftsobjekt,
- gewicht,
- hoehe,

³⁶Ist dies nicht der Fall, so wird ein leerer Angebotsvektor für ihn angelegt.

³⁷Der Name des Tags ist aus Versionsgründen aplkpz.

³⁸Siehe Abschnitt 2.4

- breite und
- tiefe

angegeben werden müssen.

Außerdem werden wieder eine Menge von Teil- und Werkstoffklassen genannt. In diesen soll der Arbeitsplaner das Geschäftsobjekt planen können. Beide Mengen dürfen nicht leer sein. Schließlich werden noch die Arbeitsplanungskompetenzen angegeben, die gesucht werden. Zu jeder Kompetenz (Element aplkpz) wird ein Attribut bewertung hinzugefügt, das die *minimale Bewertung* angibt, mit der der APL die Kompetenz zur Verfügung stellen soll. Auch hier sollte die Instanz, die das XML-Dokument erstellt, darauf achten, dass diese Attribute bei Methoden und nichtpersonellen Ressourcen automatisch den Wert 1 hat.

Die DTD für Anfragen ist im Anhang, Abschnitt C.3, abgebildet.

4.4.4 Rückgaben der Kandidatensuche

Wenn eine Suchanfrage an die Applikation IMK-APL mittels der grafischen Oberfläche gestellt wurde, wird das Ergebnis in einem Dialogfenster (dargestellt in Abbildung 4.11) angezeigt. Erfolgt die Anfrage jedoch über die Webservice-Schnittstelle, die im nächsten Abschnitt beschrieben wird, so wird das Ergebnis ebenfalls in einem XML-Dokument zurückgegeben. Dabei findet die in Anhang C.4 beschriebene Dokumentstruktur Anwendung.

Das Wurzelement rueckgabe hat zwei Attribute: errorcode und errormsg. Das erste hat den Wert 0, wenn die Kandidatensuche fehlerfrei durchgeführt werden konnte. Sind Fehler bei der Suche aufgetreten, so kann errorcode einen der folgenden Werte aufnehmen:

Wert v. errorcode	Bedeutung
1	Fehler beim Laden der Konfigurationsdatei
2	Fehler beim Verbinden zur Datenbank
3	GraphLearning-Bibliothek konnte nicht geladen werden
4	Fehler beim Schreiben des temporären Files
5	XML-Format (Anfrage) ungültig
6	Fehler bei der Bearbeitung der Anfrage

Zusätzlich zu den oben genannten Codes wird im Fall eines Fehlers eine Beschreibung des Problems³⁹ angegeben. Diese ist dann im Attribut errormsg zu finden.

Tritt bei der Anfrage kein Fehler auf, so folgt die Liste der gefundenen Kandidaten (kandidaten). Das Attribut anzahl gibt die Anzahl der Arbeitsplaner in der Kandidatenliste an. Der Wert von exakt ist 1, wenn die gefundenen Kandidaten über genau die geforderten Kompetenzen verfügen, wie es im Unterabschnitt 3.2.3 beschrieben ist. Ansonsten ist der Wert 0.

³⁹Zum Beispiel „Erstes Element muss vom Typ 'anfrage' sein“

Jeder Kandidat wird dann als apl-Element genannt. Die Attribute name, ort, plz, strasse und web-adresse entsprechen denen aus der DTD zum Import von Arbeitsplanern. Außerdem tritt pro Kandidat ein Element *sfmatrix* auf. In diesem wird die *SoftFacts-Matrix*⁴⁰, die zu dem APL gehört, angegeben. Sie wird in Zeilen und Spalten unterteilt, analog der Darstellung von Tabellen in HTML⁴¹.

Die komplette Document Type Definition für die Rückgabe der Kandidatensuche befindet sich im Anhang C.4.

4.5 Spezifikation der Webservices

Aufgabe dieser Arbeit war es nicht nur, ein Designmodell für die Beschreibung von Arbeitsplanungskompetenzen und Arbeitsplanern und eine Applikation zur Instanziierung dieses Modells zu entwickeln. Es war auch eine Schnittstelle gefordert, durch die sich der neue Teil des Informationstechnischen Modellkerns (IMK-APL) in den Gesamtkontext des SFB „Hierarchielose regionale Produktionsnetze“ eingliedert. Diese Schnittstelle und die dafür verwendeten Werkzeuge werden in diesem Abschnitt beschrieben.

4.5.1 Webservices mit SOAP

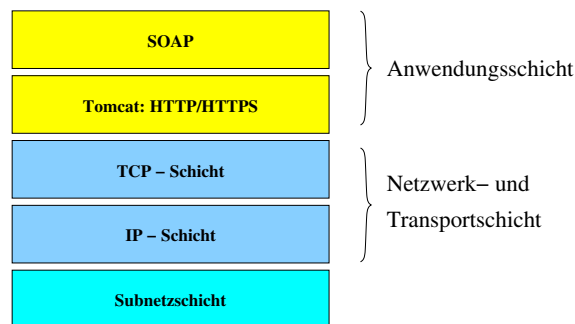


Abbildung 4.12: Tomcat und SOAP im TCP/IP-Protokollstack

Die bisher existierende Kommunikation mit dem IMK basiert auf *Webservices*. Ein Webservice gilt als „Dienst, der mit Hilfe von XML auf der Basis von Internet-Netzwerkprotokollen erbracht wird“ (Quelle: Wikipedia [Wiki1]). Dafür wurde Apache SOAP⁴² verwendet, das als *Servlet* im Tomcat-Server (entwickelt vom Apache Jakarta Project) läuft⁴³. Abbildung 4.12 zeigt die Einordnung von Tomcat und SOAP im TCP/IP-Protokollstack.

⁴⁰Vgl. [JT02]

⁴¹Hypertext Markup Language

⁴²Ursprünglich: Simple Object Access Protocol, <http://ws.apache.org/soap>

⁴³SOAP kann auch auf anderen Protokollen (FTP, u. a.) aufgesetzt werden.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5   <soapenv:Body>
6     <ns1:add soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
7       xmlns:ns1="urn:CardValidator">
8       <number xsi:type="xsd:string">1234 5678 9876 5432</number>
9       <valid xsi:type="xsd:string">12/08</valid>
10      </ns1:add>
11    </soapenv:Body>
12  </soapenv:Envelope>
13

```

Abbildung 4.13: SOAP Beispielnachricht, Quelle: Wikipedia [Wiki2]

Eine SOAP-Nachricht (*Envelope*) ist in *Header* und *Body* aufgeteilt. Im Header, der nicht durch ein spezielles Tag eingeleitet wird, stehen Metainformation zur Nachricht, zum Beispiel über das Routing oder eventuelle Verschlüsselung. Im Body befinden sich die eigentlichen „Nutzdaten“. Dies beinhaltet den Namen der Methode, die aufgerufen werden soll, und die Parameter, die an sie übergeben werden. In Abbildung 4.13 ist eine Beispielnachricht ohne HTTP⁴⁴-Kopfdaten dargestellt. Genauere Beschreibungen der Funktionsweise von SOAP sind in [Wiki2] und (im Zusammenhang mit Java) in [McL02] zu finden.

Um eine einheitliche Umgebung im Informationstechnischen Modellkern zu gewährleisten, wurden in dieser Arbeit ebenfalls Webservices (unter Verwendung von Tomcat und SOAP) entwickelt. Zur Zeit werden dabei zwei Methoden angeboten:

1. Import eines Arbeitsplaners und
2. Anfrage zur Kandidatensuche.

Diese beiden Methoden werden in den folgenden Unterabschnitten kurz beschrieben.

4.5.2 Import eines Arbeitsplaners

Die Funktion zum Importieren eines Arbeitsplaners heißt `aplanlegen`, ihr wird als Parameter ein String⁴⁵ übergeben. Dieser String muss die gültige XML-Beschreibung eines APL nach der DTD aus Abschnitt C.2 enthalten. Das übergebene XML-Dokument wird dann – wie in Unterabschnitt 4.3.4, „Importieren von Arbeitsplanern“, beschrieben – geparkt, ein Objekt der Klasse APL wird erstellt und anschließend in der Datenbank des IMK gespeichert.

Der Rückgabewert der Methode ist ebenfalls ein String. Dieser enthält ein XML-Dokument nach der DTD aus Anhang C.5. Hier werden nur zwei Attribute, `errorcode` und `errormsg`⁴⁶, angegeben.

⁴⁴Hypertext Transfer Protocol

⁴⁵Zeichenkette

⁴⁶Vgl. Unterabschnitt 4.4.4

Die genaue Deklaration der Methode (in Java) lautet:

```
1 public String aplanlegen(String xml)
```

Die folgende Abbildung enthält Beispielcode in Java zum Anfragen des in diesem Unterabschnitt beschriebenen Webservices. Der Code ist ein Ausschnitt der Klasse APLANlegenClient, die im Paket de.dvs.apl.application.clients zu finden ist.

```
1 /** Datei einlesen */
2 String xml = readFile("apl.xml");
3
4 /** Erzeugen des Aufrufobjekts und der Parameter */
5 Call call = new Call();
6 call.setTargetObjectURI("urn:imkapl");
7 call.setMethodName("aplanlegen");
8 call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
9 Vector params = new Vector();
10 params.addElement(new Parameter("xml", String.class, xml, null));
11
12 /** Aufruf */
13 call.setParams(params);
14 Response response = call.invoke(new URL(server), "localhost");
```

Abbildung 4.14: Anfrage an den Webservice aplanlegen mit Java

4.5.3 Anfrage zur Kandidatensuche

Die Methode anfrage nimmt ebenfalls einen String als Parameter. Dieser enthält die XML-Beschreibung einer Anfrage, deren Struktur in der DTD in Anhang C.3 festgelegt ist. Die Anfrage wird dann, wie in Unterabschnitt 4.3.7 beschrieben, bearbeitet. Ergebnis der Kandidatensuche ist ein XML-String, der eine Liste der gefundenen Arbeitsplaner inklusive der SoftFacts-Matrix beinhaltet.

Die genaue Deklaration der Funktion ist:

```
1 public String anfrage(String xml)
```

Abbildung 4.15 enthält Beispielcode für eine Anfrage an den beschriebenen Webservice in Java. Dieser Quellcode ist Teil der Klasse AnfrageClient, die sich ebenfalls im Paket de.dvs.apl.application.clients befindet.

```
1  /** Datei einlesen */
2  String xml = readFile("anfrage.xml");
3
4  /** Erzeugen des Aufrufobjekts und der Parameter */
5  Call call = new Call();
6  call.setTargetObjectURI("urn:imkapl");
7  call.setMethodName("anfrage");
8  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
9  Vector params = new Vector();
10 params.addElement(new Parameter("xml", String.class, xml, null));
11
12 /** Aufruf */
13 call.setParams(params);
14 Response response = call.invoke(new URL("localhost"), "");
```

Abbildung 4.15: Anfrage an den Webservice anfrage mit Java

4.6 Installation

In diesem Abschnitt werden die nötigen Schritte beschrieben, um die Applikation IMK-APL, inklusive der im letzten Abschnitt beschriebenen Webservices, zu installieren. Voraussetzungen für die Installation sind:

- Installiertes und konfiguriertes FastObjects-Datenbankmanagementsystem,
- Installation des Java SDK (inklusive Ant⁴⁷),
- Installation der Xerces-Bibliotheken und
- Installation und Konfiguration von Tomcat und SOAP.

Alle in dieser Arbeit entstandenen Java-Klassen sind auf der beigelegten CD, im Verzeichnis `IMKAPL-1.0`, enthalten. Der Nutzer sollte sich dieses Verzeichnis kopieren und in der Kopie⁴⁸ die nachfolgend genannten Befehle ausführen.

4.6.1 Initialisieren der Datenbank

Eine Datenbank (DB) wird im FastObjects-DBMS mit Hilfe des Tools `ptj` erstellt. Dazu werden in einer Datei, die dem Programm als Parameter übergeben wird, alle Java-Klassen aufgeführt, die im Schema der Datenbank auftreten sollen. Es wird empfohlen, diese vorher (auf dem Zielsystem) neu übersetzen zu lassen. Dazu kann das Java-Werkzeug `ant`⁴⁹ verwendet werden. Zuerst müssen allerdings alle alten `.class`-Files mit `ant clean` gelöscht werden. Das Neuübersetzen der Klassen ist nicht unbedingt erforderlich, es kann aber zu Laufzeitfehlern bei der Benutzung der FastObjects-Bibliotheken kommen, wenn die Bytecode-Dateien der Klassen auf einer anderen Plattform erstellt wurden.

⁴⁷<http://ant.apache.org>

⁴⁸Muss beschreibbar sein

⁴⁹Java-basierte Alternative des Standardtools `make`

Im nächsten Schritt muss das Programm `ptj` aufgerufen werden. Ihm wird der Dateiname `ptj.opt` als Parameter mitgegeben. Der exakte Aufruf lautet:

```
ptj -enhance -inplace -create -conf ptj.opt
```

Alternativ kann das Batch-File `updatePtj.bat` verwendet werden. Wenn `ptj` fehlerfrei ausgeführt wurde, ist ein Verzeichnis `IMKAPLbase` entstanden, in dem sich alle zur Datenbank gehörenden Dateien befinden. Dieses Verzeichnis kann nun an eine vom Nutzer gewünschte Stelle kopiert bzw. verschoben werden.

4.6.2 Installation und Konfiguration von IMK-APL

Wenn die Datenbank initialisiert und an den „richtigen Platz“ kopiert oder verschoben wurde, kann die Installation und Konfiguration der grafischen Oberfläche vorgenommen werden. Dazu wurde ein Setup-Programm implementiert. Es kann mit

```
java de.dvs.apl.application.APLManagerSetup
```

bzw. dem Batch-File `APLSetup.bat` aufgerufen werden. In dem daraufhin angezeigten Dialogfenster muss der Nutzer das Zielverzeichnis der Installation, den Zugriffspfad zur Datenbank⁵⁰ und das temporäre Verzeichnis, das von der GraphLearning Komponente verwendet werden soll, angeben.

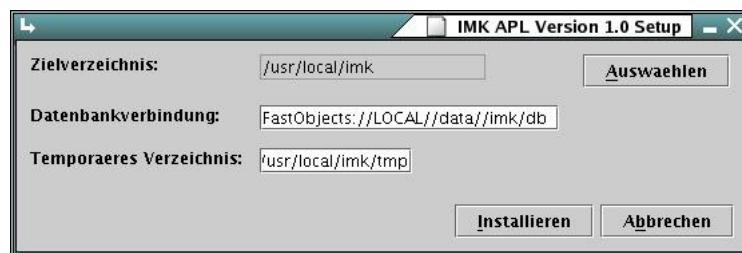


Abbildung 4.16: IMK-APL Setupprogramm

Durch Betätigen des Buttons `Installieren` wird dann die Installation gestartet. Dabei werden einige Daten in das Zielverzeichnis kopiert und eine Konfigurationsdatei wird im Heimatverzeichnis des Nutzers angelegt. Es wird zusätzlich empfohlen, das Verzeichnis `IMKAPL-1.0` auf die Festplatte des zu verwendenden Systems zu kopieren, damit die Applikation nicht jedesmal von der CD gestartet werden muss.

4.6.3 Installation der GraphLearning Komponente

Die GraphLearning Komponente ist als dynamische Bibliothek auf der beliegenden CD vorhanden. Der exakte Dateiname lautet unter Windows:

⁵⁰Z. B.: `FastObjects://LOCAL//data/IMKAPLbase`

```
IMKAPL-1.0\GraphLearning.windows\graphlearning.dll
```

für Linux Systeme muss die Datei

```
IMKAPL-1.0/GraphLearning.linux/libgraphlearning.so
```

verwendet werden. Die Linux-Bibliothek wurde auf einem *i686*-System mit dem GNU C Compiler Version 3.3.3 für glibc-2.3.3 erzeugt. Wird auf der Zielplattform eine andere Konfiguration eingesetzt, so kann es nötig sein, das Modul neu zu kompilieren. Dazu sind entsprechende Makefiles vorhanden. Weitere Hinweise zum Erstellen von dynamischen Bibliotheken sind unter [JNI1] und [JNI2] zu finden.

Das GraphLearning Modul muss von der Java Virtual Machine (JVM) zur Laufzeit geladen werden. Dazu muss es sich im Suchpfad des Betriebssystems für dynamische Bibliotheken befinden. Der Nutzer kann also die passende Datei in ein Verzeichnis aus diesem Pfad kopieren. Alternativ kann er die Umgebungsvariable (`$LD_LIBRARY_PATH` unter Linux und `%PATH%` im Windows) um das Verzeichnis erweitern, in dem sich die Datei befindet.

Wurden die ersten drei in diesem Abschnitt beschriebenen Schritte erfolgreich durchgeführt, kann die Prototypapplikation gestartet werden. Dazu kann der Befehl

```
java de.dvs.apl.application.APLManager
```

oder das Batch-File `APLManager.bat` verwendet werden. Voraussetzung ist, dass der Nutzer sich im Verzeichnis `IMKAPL-1.0` (bzw. in der Kopie auf der Festplatte) befindet oder der Pfad zu diesem Verzeichnis in den Java Suchpfad für Klassen⁵¹ eingetragen wurde.

4.6.4 Installation der Webservices

Im Abschnitt 4.5 werden die beiden in dieser Arbeit entstandenen Webservices beschrieben: Import eines Arbeitsplaners und Anfrage zur Kandidatensuche. Beide wurden unter Verwendung von Tomcat und SOAP entwickelt. Es wird im Folgenden davon ausgegangen, dass der Zielrechner bereits über eine funktionierende Installation und Konfiguration von Tomcat und SOAP verfügt.

Im ersten Schritt muss die Java Klasse, die die beiden Methoden enthält, in das Verzeichnis der SOAP-Services⁵²

```
/usr/local/tomcat/webapps/soap/WEB-INF/classes
```

kopiert werden. Diese Klasse ist in der Datei

```
de/dvs/apl/application/services/IMKAPL.class
```

enthalten.

⁵¹ `$CLASSPATH` (Linux) bzw. `%CLASSPATH%` (Windows)

⁵² `/usr/local/tomcat` sei das Verzeichnis, in dem der Tomcat Server installiert wurde.

Danach müssen die von der Applikation verwendeten Klassen, die in den Verzeichnissen `IMKAPL-1.0/de` und `IMKAPL-1.0/mren` vorliegen, in den Suchpfad für Java-Klassen von Tomcat kopiert werden. Der Name dieses Verzeichnisses lautet

```
/usr/local/tomcat/common/classes
```

Die Bibliothek für den Datenbankzugriff ist in einem Java Archiv (`.jar`) enthalten und muss nach

```
/usr/local/tomcat/common/lib
```

kopiert werden. Die Xerces Bibliotheken werden von Tomcat selbst benutzt und stehen deshalb jedem Service automatisch zur Verfügung.

Im letzten Schritt müssen der Tomcat Server gestartet und die Webservices eingebunden werden. Dazu kann die Java Klasse `ServiceManagerClient` aus dem Paket `org.apache.soap.server` verwendet werden. Ihr werden als Parameter die URL des SOAP Servlets, das Schlüsselwort `deploy`⁵³ und der Name einer XML-Datei, die den Service beschreibt, übergeben. Diese Datei ist auf der beigelegten CD vorhanden und heißt `IMKAPLservices.xml`.

Der vollständige Befehl, um den Service auf einem lokal laufenden Tomcat/SOAP Server anzumelden lautet:

```
java org.apache.soap.server.ServiceManagerClient
  http://localhost:8080/soap/servlet/rpcrouter
  deploy IMKAPLservices.xml
```

Um zu testen, ob die Webservices erfolgreich installiert sind, wurden zwei Testapplikationen implementiert. Diese sind im Paket `de.dvs.apl.application.clients` zu finden. Beide Klienten fordern als Parameter den Namen einer Datei, die eine XML-Beschreibung des Arbeitsplaners bzw. der Anfrage enthält, und die URL des SOAP Servlets. Ein Beispielaufruf könnte also lauten:

```
java de.dvs.apl.application.clients.AnfrageClient
  anfrage.xml http://localhost:8080/soap/servlet/rpcrouter
```

Beide Testklienten geben den XML-String, den sie vom Service erhalten, unbearbeitet auf der Standardausgabe aus.

⁵³Anwenden, Aufstellen

5 Kritik und Ausblick

In dieser Studienarbeit wurden ein Designmodell zur Beschreibung von Arbeitsplanern entworfen und zwei Schnittstellen zum Instanzieren des Modells sowie zur Kandidatensuche implementiert. Im letzten Kapitel dieser Arbeit soll der verwendete Ansatz noch einmal kritisch bewertet werden. Dabei werden die folgenden Abschnitte separat betrachtet:

1. das Designmodell,
2. die GraphLearning Komponente und
3. die Prototypapplikation IMK-APL.

Im letzten Abschnitt werden weiterführende Arbeiten vorgeschlagen und kurz beschrieben.

5.1 Kritik am Designmodell

Das in dieser Arbeit entworfene Designmodell, das im Kapitel 2 ausführlich beschrieben wird, führt eine Menge von Metainformationen ein, anhand derer sich ein Arbeitsplaner beschreiben kann. Die wichtigste Komponente sind dabei die Arbeitsplanungskompetenzen. Sie repräsentieren Aktivitäten, Methoden und nichtpersonelle Ressourcen, die ein Arbeitsplaner beherrscht bzw. über die er verfügt.

Jeder Arbeitsplaner hat einen Angebotsvektor, der aus einer Liste von Angeboten besteht. Diese Angebote können sofort beim Einpflegen des APL in das System angegeben oder nachträglich hinzugefügt werden. Dabei bezieht sich jedes Angebot auf genau ein Geschäftsobjekt in einer Größen- und Gewichtsklasse, eine Menge von Teil- und Werkstoffklassen sowie Arbeitsplanungskompetenzen.

Dieser Ansatz ist nicht besonders gut geeignet, wenn die Arbeitsplaner dieselben Kompetenzen für eine Vielzahl von Geschäftsobjekten und in verschiedenen Größen- und Gewichtsklassen (vgl. hierzu auch Unterabschnitt 5.4.1) anbieten. Dann muss nämlich für jede dieser Kombinationen ein eigenes Angebot erstellt werden. Deshalb ist das Beschreiben eines Arbeitsplaners eine sehr aufwändige Aufgabe. Außerdem entsteht eine sehr große Menge an (Angebots-) Daten.

Das Problem der Datenmenge kann nur durch Änderungen am Designmodell beseitigt werden. Das erste Problem kann man beheben, indem man dem Arbeitsplaner spezielle Werkzeuge zur Verfügung stellt, die das Anlegen von Angeboten vereinfachen. Bei diesen sollte es dann zum Beispiel möglich sein, ein Angebot so zu beschreiben, dass es mehrere Geschäftsobjekte beinhaltet. Soll das Angebot dann im IMK gespeichert werden, muss es von der Werkzeugapplikation in mehrere – dem Designmodell dieser Arbeit entsprechende – Angebote aufgespaltet werden. Eine weitere Unterstützung beim Anlegen des Angebotsvektors des Arbeitsplaners wäre

die Möglichkeit, bestimmte Arbeitsplanungskompetenzen als „Kernkompetenzen“ zu markieren, die dann in alle Angebote automatisch einfließen.

5.2 Kritik am GraphLearning

Das im Kapitel „Kandidatensuche“ beschriebene GraphLearning ist eine auf neuronalen Netzen basierende Komponente zur Indexierung und Suche der Angebotsprofile. Das GL erhält als Eingabe eine Beschreibung des Angebotsbaums (vgl. Abschnitt 2.1) und eine Menge von Angeboten. Diese Angebote bestehen aus mehreren Paaren (Arbeitsplanungskompetenz, Durchschnittliche Bewertung). Der Bezug zu den Geschäftsobjekten und Teil- bzw. Werkstoffklassen wird im GraphLearning nicht berücksichtigt und geht verloren.

Da dieser Bezug jedoch für die Suche nach geeigneten Kandidaten notwendig ist, muss die Ergebnismenge des GraphLearning nachträglich, wie in Unterabschnitt 3.2.3 beschrieben, korrigiert werden. Bei dieser Korrektur muss jedes Angebot, das vom GL ermittelt wurde, aus der Datenbank des IMK geladen und seine Relevanz bezüglich des Geschäftsobjekts geprüft werden. Diese Prüfung ist sehr rechenaufwändig.

Es wäre also besser, wenn bei der Indexierung der Angebotsprofile das Geschäftsobjekt mit berücksichtigt würde. Das kann erreicht werden, indem man für jedes GO einen eigenen Index anlegt, in dem dann nur die Angebote betrachtet werden, die sich auf dieses Geschäftsobjekt beziehen. Diese Methode ist allerdings bei einer großen Anzahl an Geschäftsobjekten nicht praktikabel. Besonders schwerwiegend ist der Aufwand, wenn auch verschiedene Indexe für unterschiedliche Kombinationen aus GO und Größen-/Gewichtsklasse oder GO und Teil-/Werkstoffklassen angelegt werden sollen.

Die Suche nach einer geeigneten Möglichkeit, die Geschäftsobjekte im GraphLearning zu berücksichtigen, sollte also Gegenstand weiterer Arbeiten sein.

5.3 Kritik an der Prototypapplikation IMK-APL

In diesem Abschnitt sollen zwei Kritikpunkte an der Applikation IMK-APL aufgezeigt werden.

5.3.1 Funktionsumfang

Die Forderung nach geeigneten Schnittstellen zur Instanziierung des Designmodells wurden durch die in Kapitel 4, „Prototyp IMK-APL“, beschriebene Prototypapplikation und zwei als Webservice angebotene Funktionen erfüllt.

Mit Hilfe der beiden genannten Implementierungen ist es möglich

- die Metainformationen¹ zu importieren,

¹Geschäftsobjekte, Teil- und Werkstoffklassen und Arbeitsplanungskompetenzen

- Arbeitsplaner (und Angebote) anzulegen,
- Arbeitsplaner zu löschen,
- die GraphLearning-Komponente zu trainieren und
- eine Kandidatensuche durchzuführen.

Es fehlen zur Zeit Funktionen zum Ändern von Arbeitsplanern und zum Ändern bzw. Löschen von einzelnen Angeboten. Diese Funktionen zu entwerfen und zu implementieren ist ebenfalls Aufgabe zukünftiger Betrachtungen.

5.3.2 Zeitaufwand bei der Bearbeitung von XML-Daten

Wie im Unterabschnitt 4.1.4 bereits erläutert, wird zur Bearbeitung von XML-Daten in der Prototypapplikation DOM verwendet. Dabei wird das Dokument in einen Objektbaum transformiert. Diese Vorgehensweise bietet dem Entwickler einen komfortablen Zugriff auf eine abstrakte Datenstruktur, was sich auch in leichter wartbarem Programmcode (gegenüber einem ereignis-basierten Ansatz) widerspiegelt.

Dafür benötigt ein DOM-Parser wesentlich mehr Zeit zum Laden eines XML-Dokuments. Abbildung 5.1 zeigt das Ergebnis eines Benchmarks der Firma Devsphere². Dabei wurden das Java Development Toolkit in der Version 1.3 und Xerces 1.4 eingesetzt, um das Laufzeitverhalten von SAX, DOM und Mischformen aus beiden miteinander zu vergleichen.

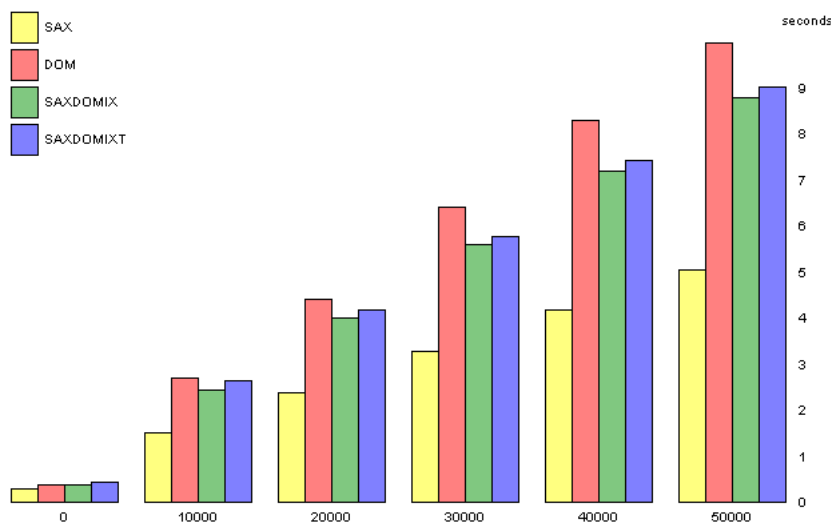


Abbildung 5.1: Vergleich des Laufzeitverhaltens von SAX und DOM, Quelle: [DEV]

Das Ergebnis des Benchmarks zeigt, dass DOM besonders bei großen XML-Dokumenten (mit

²<http://www.devsphere.com>

bis zu 50.000 Datensätzen) erheblich länger arbeitet. Deshalb empfiehlt es sich, die entsprechenden Klassen zum Parsen von XML-Dateien, auf die Verwendung von SAX umzustellen, sobald die Datenmenge im IMK größere Dimensionen erreicht.

5.4 Weiterführende Arbeiten

In den drei vorangegangenen Abschnitten wurden weiterführende Arbeiten genannt, die direkt aus Problemen mit der aktuellen Implementierung resultieren. In diesem Abschnitt sollen noch einige weitere Vorschläge für zukünftige Projekte gemacht werden.

5.4.1 Definition der Äquivalenz von Größen- und Gewichtsklassen

Im Unterabschnitt 3.2.3 wird beschrieben, wie die Ergebnismenge der GraphLearning-Komponente bei der Kandidatensuche auf relevante Angebote gefiltert wird. Dabei ist die Bedingung³

$$(Angebot.Groesse \approx A.Groesse) \wedge (Angebot.Gewicht \approx A.Gewicht)$$

von zentraler Bedeutung. Sie prüft, ob die Größen- und Gewichtsklasse, die im Angebot angegeben sind, die angefragten Größen- und Gewichtsklasse abdecken.

In der in dieser Arbeit entwickelten Prototypapplikation wird diese Prüfung nur dann positiv beantwortet, wenn die Werte für Gewicht und Größe⁴ genau gleich sind. Diese Interpretation ist für reale Geschäftsabläufe nicht adequat. Im Folgenden werden drei Lösungsansätze genannt, wie die Frage nach der Äquivalenz der Größen- und Gewichtsklassen besser beantwortet werden kann.

Die erste Möglichkeit ist eine Änderung des Modells dahingehend, dass für jedes Angebot ein Bereich, also ein Minimal- und ein Maximalwert, für die Gewichts- und Größenwerte angegeben wird, in dem das Geschäftsobjekt projiziert werden kann. Die Prüfung der Äquivalenz wäre dann einfach beantwortet durch die Bedingung (hier am Beispiel des Gewichts):

$$Angebot.GewichtMinimum \leq A.Gewicht \leq Angebot.GewichtMaximum$$

Eine weitere Möglichkeit ist es, anzunehmen, dass die im Angebot angegebenen Werte für Gewicht und Größe eine Obergrenze für die bearbeitbaren Gewichts- und Größenklassen darstellen. Dann ändert sich die Bedingung für die Äquivalenz zu:

$$A.Gewicht \leq Angebot.Gewicht$$

Die erste genannte Möglichkeit setzt Änderungen am Designmodell und damit auch umfangreiche Änderungen an den in dieser Arbeit entwickelten Implementierungen voraus. Die zweite Möglichkeit benötigt diese Änderungen nicht, lässt aber keine ausreichende Differenzierung zu. Deshalb soll an dieser Stelle noch eine weitere Alternative genannt werden.

³Variable *A* repräsentiert die Anfrage.

⁴Höhe, Breite und Tiefe

Man kann die Werte, die der Arbeitsplaner beim Anlegen eines Angebots für Gewicht und Größe angibt, als Mittelwert m eines Intervalls von gültigen Maßen betrachten. Dazu muss festgelegt werden, wie groß der Bereich um den angegebenen Wert sein soll. Es empfiehlt sich, hierfür keinen festen Betrag zu nehmen, sondern einen Wert, der relativ zum genannten Mittelwert ermittelt wird, zum Beispiel $p = 0,5$. Abbildung 5.2 zeigt ein Beispiel für einen solchen Bereich.

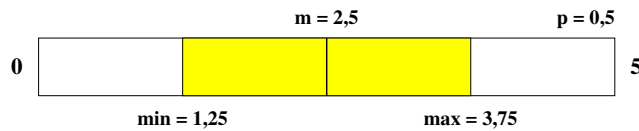


Abbildung 5.2: Äquivalenzbereich für $m = 2,5$ und $p = 0,5$

Ob ein Angebot bezüglich der Werte für Gewicht und Größe relevant ist, wird dann durch folgende Bedingung geprüft (wieder am Beispiel des Gewichts):

$$\text{Angebot.Gewicht} \cdot (1 - p) \leq A.\text{Gewicht} \leq \text{Angebot.Gewicht} \cdot (1 + p)$$

Ob die zuletzt genannte Möglichkeit am besten geeignet und wenn ja, welcher Wert für p optimal ist, muss ebenfalls in zukünftigen Arbeiten geklärt werden.

5.4.2 Bewertung von Kompetenzen durch Dritte

Im Abschnitt 2.4 wird ein Modell eingeführt, dass die stufenweise Anpassung der (Gesamt-) Bewertung einer Arbeitsplanungskompetenz an einen objektiv gerechtfertigten Wert ermöglichen soll. Dies wird erreicht, indem zusätzlich zur Eigenbewertung des Arbeitsplaners weitere Bewertungen (von fachfremden oder -kompetenten Instanzen) an das Angebot gebunden werden.

Während das Klassenmodell der Prototypapplikation IMK-APL die Speicherung und Beachtung weiterer Bewertungen bereits unterstützt, sind noch keine geeigneten Schnittstellen zum Anlegen dieser Bewertungen vorhanden. Außerdem ist noch nicht festgelegt, wer (an welchem Zeitpunkt) eine Bewertung für einen Arbeitsplaner abgeben darf.

Als Referenzmodell zur Klärung dieser Fragen soll an dieser Stelle das Bewertungssystem des Online-Auktionshauses eBay⁵ dienen. Es ist in [EBAY] beschrieben. Bei diesem System ist jeder Nutzer angehalten, nach einer Transaktion den Handelspartner⁶ zu bewerten. Dabei wird nur zwischen einer positiven, negativen oder neutralen Beurteilung unterschieden.

Analog kann man den Geschäftsablauf im Hierarchielosen regionalen Produktionsnetz so ändern, dass jede Instanz, die einen Arbeitsplaner für eine Transaktion gesucht und gefunden hat, diesen nach Ablauf des Auftrags bewerten kann. Für diese Bewertung muss ein entsprechender Service, der nach außen sichtbar ist, implementiert werden. Es entfällt – im Gegensatz zu dem Bewertungssystem von eBay – die Möglichkeit des Arbeitsplaners, zur abgegebenen Beurteilung Stellung zu nehmen.

⁵<http://www.ebay.de>

⁶Verkäufer, Käufer

5.4.3 Neutraining der GraphLearning-Komponente

Im Unterabschnitt 3.2.1 wurde das Training der GraphLearning-Komponente und im Abschnitt 4.2 die Anbindung an die in dieser Arbeit entwickelte Implementierung beschrieben. Die Suche im GraphLearning basiert auf neuronalen Netzen. Diese müssen, wenn sich die Angebotsmenge ändert, neu trainiert werden. Im letzten Abschnitt dieser Arbeit sollen drei verschiedene Möglichkeiten genannt werden, wann dieses Neutraining erfolgen kann.

Die erste Variante ist es, das Neutraining der neuronalen Netze nach jeder Änderung am Datenbestand des IMK, also nach jedem Einfügen, Ändern oder Löschen eines Arbeitsplaners bzw. Angebots⁷, vorzunehmen. Diese Vorgehensweise stellt sicher, dass das GraphLearning immer alle aktuellen Daten über die Arbeitsplaner und ihre Angebote berücksichtigt und keine Angebote als Ergebnis liefert, die inzwischen gelöscht wurden. Dafür ist dieses Verfahren sehr rechenaufwändig und sollte nur angewendet werden, wenn wenige Änderungen am Datenbestand erfolgen.

Alternativ kann festgelegt werden, dass ein Neutraining der Netze nicht nach jeder Änderung, sondern nach einer bestimmten Serie von Änderungen durchgeführt wird. Diese Variante reduziert die Anzahl der Trainingsdurchläufe, allerdings arbeiten dann die Netze auf einem älteren Datenbestand. Es kann also vorkommen, dass ein Angebot im Ergebnis des GraphLearning referenziert wird, dass in der IMKDB nicht mehr existiert. In diesem Fall muss dann die Kandidatensuche entsprechend reagieren. Außerdem kann bei dieser Alternative der Fall eintreten, dass eine Änderung für einen sehr langen Zeitraum nicht im GL berücksichtigt wird, da kein Neutraining erfolgt.

Eine dritte Möglichkeit ist es, das Neutraining des GraphLearning regelmäßig zu einem bestimmten Zeitpunkt, zum Beispiel jede Nacht 0:00 Uhr, durchzuführen. Da es sich beim IMK um ein System für regionale Produktionsnetze handelt, sollte es möglich sein, einen Zeitpunkt zu finden, an dem kein regulärer Geschäftsverkehr stattfindet. Mit dieser Variante wird gesichert, dass alle Änderungen nach einer bestimmten Maximaldauer in den neuronalen Netzen berücksichtigt werden. Es besteht allerdings auch hier die Schwierigkeit, dass in der Ergebnismenge der GraphLearning-Komponente bereits gelöschte Angebote auftauchen können.

Dieses Problem kann gelöst werden, indem nicht nur das Neutraining an dem bestimmten Zeitpunkt t vorgenommen wird, sondern auch die Änderungen am Datenbestand. Dazu müssen alle Änderungsaufträge, zum Beispiel „Lösche das zweite Angebot aus dem Angebotsvektor vom Arbeitsplaner mit der ID 19999“, in eine Warteschlange geschrieben werden. Wird dann t erreicht, so werden zuerst alle Änderungen aus der Warteschlange durchgeführt und danach erfolgt das Neutraining der Netze.

Die letztgenannte Möglichkeit bietet sicherlich den interessantesten Lösungsansatz, bedarf aber einiger Änderungen an der bisher existierenden Implementierungen.

⁷Es sind noch nicht alle genannten Funktionen implementiert.

5.5 Zusammenfassung

Das in dieser Arbeit entworfene Modell und die beiden implementierten Schnittstellen erfüllen die Aufgabe, Arbeitsplaner zu beschreiben und Kandidaten für eine Anfrage zu suchen. Mit Hilfe der GraphLearning-Komponente, die an der Professur Datenverwaltungssysteme der TU Chemnitz entwickelt wurde, ist die Kandidatensuche auch effizient durchführbar.

In diesem letzten Kapitel wurden einige Kritikpunkte an der entwickelten Lösung und Möglichkeiten, diese zu beheben, genannt. Außerdem wurden Vorschläge gemacht, welche weiterführenden Entwicklungen an der Applikation IMK-APL notwendig oder lohnenswert sind. Diese umzusetzen bleibt Aufgabe zukünftiger Arbeiten.

A Abkürzungsverzeichnis

AGV	Angebotsvektor
APL	Arbeitsplaner
APLKP	Arbeitsplanungskompetenz
ART	Adaptive Resonance Theory
AWT	Abstract Windowing Toolkit
CORBA	Common Request Broker Architecture
DB	Datenbank
DBMS	Datenbankmanagementsystem
DII	Dynamic Invocation Interface
DOM	Document Object Model
DSI	Dynamic Skeleton Interface
DTD	Document Type Definition
FG	Funktionsgruppe
GCC	GNU C Compiler
GIOP	General Inter-ORB Protocol
GNG	Growing Neutral Gas
GNOME	GNU Network Object Model Environment
GL	GraphLearning
GO	Geschäftsobjekt
GUI	Graphical User Interface (Grafische Benutzeroberfläche)
HTML	Hypertext Markup Language
ICIX	Intelligent Cluster Index
IDL	Interface Definition Language
IMK	Informationstechnischer Modellkern
IMKDB	Datenbank des IMK
JDK	Java Development Kit

JNI	Java Native Interface
JVM	Java Virtual Machine
KMU	Kleine und mittelständische Unternehmen
KP	Kompetenz
KPZ	Kompetenzzelle
KPZN	Kompetenzzellennetzwerk
NNO	Neural Network Objects
nPR	nichtpersonelle Ressource
OMG	Object Management Group
ORB	Object Request Broker
POET	Persistent Objects and Extended Database Technology
SAX	Simple API for XML
SFB	Sonderforschungsbereich
SWT	Standard Widget Toolkit
UML	Unified Modelling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

B Weitere Screenshots



Abbildung B.1: Statistik über Import von Metainformationen



Abbildung B.2: Details einer Arbeitsplanungskompetenz



Abbildung B.3: Auswahl eines Geschäftsobjekts (mit Gewichts- und Größenklasse)



Abbildung B.4: Auswahl der Teil- und Werkstoffklassen

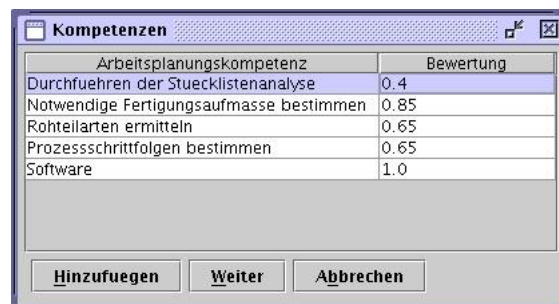


Abbildung B.5: Hinzufügen der Arbeitsplanungskompetenzen

C Document Type Definitions

C.1 Metainformationen

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Version 1.1, 02.08.2004
5 Version 1.2, 25.11.2004
6
7 Root-Element: meta,
8 NEU (02.08.2004): + beliebig viele
9 Geschäftsobjekte, Werkstoffklassen, Teilklassen
10 (diese werden nur fuer Beziehungen benoetigt)
11
12 Neu (25.11.2004): Root-Tag heisst jetzt META
13 (siehe Studienarbeit: Metainformationen)
14 -->
15
16 <!ELEMENT meta (
17 aktivitaet*,
18 programmiermethode*,
19 bewertungsmethode*,
20 fertigungsmethode*,
21 analysemethode*,
22 software*,
23 wissensbasis*,
24 hardware*,
25 geschaeftsobjekt*,
26 werkstoffklasse*,
27 teilklasse*) >
28
29 <!-- Angebotsbaum MUSS Version haben -->
30 <!ATTLIST angebotsbaum version CDATA #REQUIRED >
31 <!-- Optionale Daten -->
32 <!ATTLIST angebotsbaum quelle CDATA "" >
33 <!ATTLIST angebotsbaum kommentar CDATA "" >
34
35 <!-- Aktivitaet,
36 kann weitere Unteraktivitaeten haben,
37 MUSS Bezeichnung haben -->
38 <!ELEMENT aktivitaet (aktivitaet*) >
39 <!ATTLIST aktivitaet bezeichnung CDATA #REQUIRED >
40
41 <!-- Programmiermethode,
42 kann weitere Untermethoden haben,
43 MUSS Bezeichnung, Eigenschaften, Steuerungszuordnung
44 und Anforderungen haben (evtl. leerer String) -->
45 <!ELEMENT programmiermethode (
46 eigenschaften,
47 steuerungszuordnung,
48 anforderungen,
49 programmiermethode*) >
```

```

50 <!ATTLIST programmiermethode bezeichnung CDATA #REQUIRED >
51
52 <!-- Bewertungsmethode,
53 kann weitere Untermethoden haben,
54 MUSS Bezeichnung, Eigenschaften, Anwendungsrahmen
55 und e_a_datentyp haben (evtl. leerer String) -->
56 <!ELEMENT bewertungsmethode (
57 eigenschaften,
58 anwendungsrahmen,
59 e_a_datentyp,
60 bewertungsmethode*) >
61 <!ATTLIST bewertungsmethode bezeichnung CDATA #REQUIRED >
62
63 <!-- Fertigungsmethode,
64 kann weitere Untermethoden haben,
65 MUSS Bezeichnung, Eigenschaften, Formgebungsart
66 und Merkmale haben (evtl. leerer String) -->
67 <!ELEMENT fertigungsmethode (
68 eigenschaften,
69 formgebungsart,
70 merkmale,
71 fertigungsmethode*) >
72 <!ATTLIST fertigungsmethode bezeichnung CDATA #REQUIRED >
73
74 <!-- Analysemethode,
75 kann weitere Untermethoden haben,
76 MUSS Bezeichnung, Eigenschaften, Einsatzgebiet
77 und Spezifika haben (evtl. leerer String) -->
78 <!ELEMENT analysemethode (
79 eigenschaften,
80 einsatzgebiet,
81 spezifika,
82 analysemethode*) >
83 <!ATTLIST analysemethode bezeichnung CDATA #REQUIRED >
84
85 <!-- Software,
86 kann weitere Untersoftware haben,
87 MUSS Bezeichnung, Standort, Systemvoraussetzungen,
88 Betriebssystem, Schnittstellen und Version
89 haben (evtl. leerer String) -->
90 <!ELEMENT software (
91 standort,
92 systemvoraussetzungen,
93 betriebssystem,
94 schnittstellen,
95 version,
96 software*) >
97 <!ATTLIST software bezeichnung CDATA #REQUIRED >
98
99 <!-- Wissensbasis,
100 kann weitere Unterwb. haben,
101 MUSS Bezeichnung, Standort, Einsatzmoeglichkeit,
102 Repraesentationsform und Ausgabecharakteristik
103 haben (evtl. leerer String) -->
104 <!ELEMENT wissensbasis (
105 standort,
106 einsatzmoeglichkeit,
107 repraesentationsform,
108 ausgabecharakteristik,
109 wissensbasis*) >
110 <!ATTLIST wissensbasis bezeichnung CDATA #REQUIRED >
111

```

```

112 <!-- Hardware,
113 kann weitere Unterhardware haben,
114 MUSS Bezeichnung, Standort, Leistungsparameter
115 und Schnittstellen haben (evtl. leerer String) -->
116 <!ELEMENT hardware (
117   standort,
118   leistungsparameter,
119   schnittstellen,
120   hardware*) >
121 <!ATTLIST hardware bezeichnung CDATA #REQUIRED >
122
123 <!-- "Attribut-Elemente", CDATA -->
124 <!ELEMENT eigenschaften (#PCDATA) >
125 <!ELEMENT steuerungszuordnung (#PCDATA) >
126 <!ELEMENT anforderungen (#PCDATA) >
127 <!ELEMENT anwendungsrahmen (#PCDATA) >
128 <!ELEMENT e_a-datenart (#PCDATA) >
129 <!ELEMENT formgebungsart (#PCDATA) >
130 <!ELEMENT merkmale (#PCDATA) >
131 <!ELEMENT einsatzgebiet (#PCDATA) >
132 <!ELEMENT spezifika (#PCDATA) >
133 <!ELEMENT standort (#PCDATA) >
134 <!ELEMENT systemvoraussetzungen (#PCDATA) >
135 <!ELEMENT betriebssystem (#PCDATA) >
136 <!ELEMENT schnittstellen (#PCDATA) >
137 <!ELEMENT version (#PCDATA) >
138 <!ELEMENT einsatzmoeglichkeit (#PCDATA) >
139 <!ELEMENT repraesentationsform (#PCDATA) >
140 <!ELEMENT ausgabecharakteristik (#PCDATA) >
141 <!ELEMENT leistungsparameter (#PCDATA) >
142
143 <!-- Geschaeftsobjekt, MUSS Bezeichnung haben -->
144 <!ELEMENT geschaeftsobjekt () >
145 <!ATTLIST geschaeftsobjekt bezeichnung CDATA #REQUIRED >
146
147 <!-- Werkstoffklasse, MUSS Bezeichnung haben -->
148 <!ELEMENT werkstoffklasse () >
149 <!ATTLIST werkstoffklasse bezeichnung CDATA #REQUIRED >
150
151 <!-- Teilklasse, MUSS Bezeichnung haben -->
152 <!ELEMENT teilklasse () >
153 <!ATTLIST teilklasse bezeichnung CDATA #REQUIRED >

```

C.2 Arbeitsplaner

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4   Version 1.2, 04.10.2004
5
6   Root-Element: APL
7   NEU (28.09.2004): Bezeichnung des APL weg
8   NEU (04.10.2004): Angebotsvektor besteht aus >= 1 Angeboten
9   -->
10
11 <!-- Arbeitsplaner, kann Angebotsvektor haben -->
12 <!ELEMENT apl (angebotsvektor*) >

```

```

13
14 <!-- APL MUSS Name, Ort, PLZ, Strasse
15 und Webadresse haben -->
16 <!ATTLIST apl name CDATA #REQUIRED >
17 <!ATTLIST apl ort CDATA #REQUIRED >
18 <!ATTLIST apl plz CDATA #REQUIRED >
19 <!ATTLIST apl strasse CDATA #REQUIRED >
20 <!ATTLIST apl webadresse CDATA #REQUIRED >
21
22
23 <!-- Angebotsvektor, besteht aus >= 1 Angeboten -->
24 <!ELEMENT angebotsvektor (angebot+) >
25
26
27 <!-- Angebot, kann aus verschiedenen (je mind. 1)
28 Werkstoffklassen, Teilklassen und
29 Arbeitsplanungskompetenzen bestehen -->
30 <!ELEMENT anbot (teilklasse+, werkstoffklasse+, aplkpz+) >
31
32 <!-- Angebot muss (genau ein) Geschaeftsobjekt,
33 Gewicht und Groesse (Hoehe x Breite x Tiefe) haben -->
34 <!ATTLIST anbot geschaeftsobjekt CDATA #REQUIRED >
35 <!ATTLIST anbot gewicht CDATA #REQUIRED >
36 <!ATTLIST anbot hoehe CDATA #REQUIRED >
37 <!ATTLIST anbot breite CDATA #REQUIRED >
38 <!ATTLIST anbot tiefe CDATA #REQUIRED >
39
40
41 <!-- Teilklassen und Werkstoffklassen, muessen
42 jeweils Attribut Bezeichnung haben -->
43 <!ELEMENT teilklasse () >
44 <!ELEMENT werkstoffklasse () >
45 <!ATTLIST teilklasse bezeichnung CDATA #REQUIRED >
46 <!ATTLIST werkstoffklasse bezeichnung CDATA #REQUIRED >
47
48
49 <!-- APLKPZ, muss Bezeichnung und Bewertung haben -->
50 <!ELEMENT aplkpz () >
51 <!ATTLIST aplkpz bezeichnung CDATA #REQUIRED >
52 <!ATTLIST aplkpz bewertung CDATA #REQUIRED >

```

C.3 Anfragen

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Version 1.0, 05.10.2004
5
6 Root-Element: Anfrage
7 -->
8
9 <!-- Anfrage, bezieht sich auf (genau ein)
10 Geschaeftsobjekt (Attribute),
11 auf mehrere Teil- und Werkstoffklassen
12 und mehrere APLKPZ und deren minimale Bewertung
13 -->
14 <!ELEMENT anfrage (teilklasse+, werkstoffklasse+, aplkpz+)>

```

```

15
16 <!-- Anfrage muss sich auf (genau ein) Geschaeftsobjekt mit
17 Gewicht und Groesse (Hoehe x Breite x Tiefe) beziehen -->
18 <!ATTLIST anfrage geschaeftsobjekt CDATA #REQUIRED>
19 <!ATTLIST anfrage gewicht CDATA #REQUIRED>
20 <!ATTLIST anfrage hoehe CDATA #REQUIRED>
21 <!ATTLIST anfrage breite CDATA #REQUIRED>
22 <!ATTLIST anfrage tiefe CDATA #REQUIRED>
23
24
25 <!-- Teilklassen und Werkstoffklassen, muessen
26 jeweils Attribut Bezeichnung haben -->
27 <!ELEMENT teilklasse ()>
28 <!ELEMENT werkstoffklasse ()>
29 <!ATTLIST teilklasse bezeichnung CDATA #REQUIRED>
30 <!ATTLIST werkstoffklasse bezeichnung CDATA #REQUIRED>
31
32
33 <!-- APLKPZ, muss Bezeichnung und (Mindest-) Bewertung haben -->
34 <!ELEMENT aplkpz ()>
35 <!ATTLIST aplkpz bezeichnung CDATA #REQUIRED>
36 <!ATTLIST aplkpz bewertung CDATA #REQUIRED>

```

C.4 Rückgabe der Kandidatensuche

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Version 1.0, 14.10.2004
5
6 Root-Element: Rueckgabe
7 -->
8
9 <!-- Rueckgabe, Attribut errorcode <> 0 wenn Fehler aufgetreten
10 wenn errorcode <> 0, steht Fehlernachricht in Attribut errormsg
11 -->
12 <!ELEMENT rueckgabe (kandidaten) >
13 <!ATTLIST rueckgabe errorcode CDATA #REQUIRED >
14 <!ATTLIST rueckgabe errormsg CDATA >
15
16
17 <!-- Liste der in Frage kommenden Kandidaten, das Attribut
18 exakt ist 1, wenn die APL die geforderten Kompetenzen vollstaendig
19 erfuellen, sonst wird das Ergebnis aus GraphLearning genommen -> also
20 die aehnlichsten Kandidaten
21 Attribut anzahl gibt Anzahl der gefundenen Kandidaten an
22 -->
23 <!ELEMENT kandidaten (apl+) >
24 <!ATTLIST kandidaten exakt CDATA #REQUIRED >
25 <!ATTLIST kandidaten anzahl CDATA #REQUIRED >
26
27
28 <!-- Ein Kandidat (APL) -->
29 <!ELEMENT apl (sfmatrix) >
30 <!ATTLIST apl name CDATA #REQUIRED >
31 <!ATTLIST apl ort CDATA #REQUIRED >
32 <!ATTLIST apl plz CDATA #REQUIRED >

```

```
33 <!ATTLIST apl strasse CDATA #REQUIRED >
34 <!ATTLIST apl webadresse CDATA #REQUIRED >
35
36
37 <!-- SoftFacts-Matrix, 1 pro APL -->
38 <!ELEMENT sfmatrix (sfzeile+) >
39 <!ATTLIST sfmatrix anzahlZeilen CDATA #REQUIRED >
40 <!ATTLIST sfmatrix anzahlSpalten CDATA #REQUIRED >
41
42
43 <!-- Eine Zeile der SoftFacts-Matrix -->
44 <!ELEMENT sfzeile (sfspalte+) >
45
46
47 <!-- Eine Zelle der SoftFacts-Matrix -->
48 <!ELEMENT sfspalte () >
```

C.5 Rückgabe beim Import von Arbeitsplanern

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4   Version 1.0, 09.11.2004
5
6   Root-Element: Rueckgabe-APL
7   -->
8
9 <!-- Rueckgabe, Attribut errorcode <> 0 wenn Fehler aufgetreten
10   wenn errorcode <> 0, steht Fehlernachricht in Attribut errormsg
11 -->
12 <!ELEMENT rueckgabe-apl () >
13 <!ATTLIST rueckgabe-apl errorcode CDATA #REQUIRED >
14 <!ATTLIST rueckgabe-apl errormsg CDATA >
```

D Inhalt der beigelegten CD

Die CD, die dieser Arbeit beigelegt wurde, enthält folgende Verzeichnisse:

Verzeichnis	Inhalt
Aufgabenstellung	Aufgabenstellung als Microsoft Word-Dokument
Designmodell	UML-Diagramm des Designmodells
Documentation	JavaDOC API-Dokumentation
DTDs	Document Type Definitions und Beispiel XML-Dateien
GraphLearn	Quellen der GraphLearning-Komponente
Paper	Studienarbeit als PDF-Datei und die \LaTeX -Quellen
IMKAPL-1.0	Quell- und Bytecode-Dateien der Implementierungen
Testdaten	Skripte zum Generieren von Testdaten (inkl. einer Reihe von Testdaten)

Literaturverzeichnis

- [Bog99] Boger, M.: *Java in verteilten Systemen*, 1. Auflage. dpunkt.verlag (Heidelberg), 1999
- [DEL98] DELPHI '98 Umfrage: *Studie zur globalen Entwicklung von Wissenschaft Development, Design and Planning*, Proceedings of the IASTED Int'l. Conference on Knowledge Sharing and Collaborative Engineering, KSCE 2004 (St. Thomas USA), 2004.
- [DEV] WWW: *Devsphere: XML Parsing Benchmark*, Abruf am 14.12.2004
URL: <http://www.devsphere.com/xml/benchmark>
- [DM01] Dürr, H., Mehnert, J.: *Arbeitsplanung im hierarchielosen Produktionsnetz* in Teich, T. (Hrsg): *Hierarchielose Regionale Produktionsnetze*. Verlag der GUC (Chemnitz), 2001
- [DOM] WWW: *DOM Homepage*, Abruf am 22.11.2004
URL: <http://w3c.org/DOM>
- [EBAY] WWW: *Das eBay Bewertungssystem*, Abruf am 14.12.2004
URL: http://presse.ebay.de/news.exe?typ=HI&news_id=100223
- [FO] WWW: *Produktübersicht FastObjects T7*, Abruf am 22.11.2004
URL: <http://www.versant.com/products/fastobjects/t7>
- [GC88] Grossberg, S., Carpenter, G.: *The art of adaptive pattern recognition by self-organizing neural networks*, ACM Computer, 21 (3), März, 1988.
- [GN04] Görlitz, O., Neubert, R.: *Supporting the Search for Co-operation Partners in Product Development, Design and Planning*, Proceedings of the IASTED Int'l. Conference on Knowledge Sharing and Collaborative Engineering, KSCE 2004 (St. Thomas USA), 2004.
- [GNM04] Görlitz, O., Neubert, R., Mehnert, J.: *Die Suche nach Planungskompetenzen beim Aufbau von Produktionsnetzen* in ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 7/8 (2004). Hanser Verlag (München), 2004.
- [Heu92] Heuer, A.: *Objektorientierte Datenbanken: Konzepte, Modelle, Systeme*, 1. Auflage. Addison-Wesley (Bonn, München, Paris, u. a.), 1992
- [JNI1] WWW: *JNI (Java talks with C++) Einführung (haertfelder.com)*, Abruf am 23.11.2004
URL: <http://www.haertfelder.com/jni.html>

- [JNI2] WWW: *Overview of the JNI*, Abruf am 23.11.2004
URL: <http://java.sun.com/docs/books/tutorial/native1.1/concepts>
- [JT02] Jäckel, T., Tinter, A.: *Modellierung von SoftFacts in hierarchielosen regionalen Produktionsnetzen*, Studienarbeit (Chemnitz), 2002
- [Kun04] Kunis, R.: *WebFoG HTML Formulargenerator und -prüfer*, Studienarbeit (Chemnitz),
abrufbar unter URL: <http://www.rafaelkunis.de/studies/studienarbeit> 2004
- [Kru03] Krüger, G.: *Handbuch der Java-Programmierung*, 3. Auflage. Addison-Wesley (München), 2003
- [McL02] McLaughlin, B.: *Java & XML*, 2. Auflage. Übersetzung: Kersken, S., Key, J. O'Reilly (Köln), 2002
- [Min02] Mintert, S. (Hrsg): *XML & Co. Die W3C-Spezifikationen für Dokumenten- und Datenarchitektur*, 1. Auflage. Addison-Wesley (München), 2002
- [Oes03] Oestreich, E.: *Anbietersuche für Produktionsaufgaben auf der Basis des Intelligent Cluster Indexes*, Diplomarbeit (Chemnitz), 2003
- [Pop98] Pope, A.: *The CORBA Reference Guide*, 1. Auflage. Addison-Wesley (Reading, Mass. u. a.), 1998
- [Roj93] Rojas, R.: *Theorie der neuronalen Netze: Eine systematische Einführung*, 1. Auflage. Springer-Verlag (Berlin, Heidelberg), 1993
- [SAX] WWW: *SAX Homepage*, Abruf am 22.11.2004
URL: <http://sax.sourceforge.net>
- [SFB457] WWW: *Homepage des SFB457*, Abruf am 15.12.2004
URL: <http://www.tu-chemnitz.de/sfb457>
- [Ull04] Ullenboom, C.: *Java ist auch eine Insel*, 4. Auflage. Galileo Press (Bonn), 2004
- [Wiki1] WWW: *Wikipedia - Webservice*, Abruf am 30.11.2004
URL: <http://de.wikipedia.org/wiki/Webservice>
- [Wiki2] WWW: *Wikipedia - Simple Object Access Protocol*, Abruf am 30.11.2004
URL: <http://de.wikipedia.org/wiki/SOAP>
- [Wir01] Wirth, S.: *Der Kompetenzzellenbasierte Vernetzungsansatz für Produktion und Dienstleistung* in Teich, T. (Hrsg): *Hierarchielose Regionale Produktionsnetze*. Verlag der GUC (Chemnitz), 2001
- [Zel94] Zell, A.: *Simulation Neuronaler Netze*, 1. Auflage. Addison-Wesley (Bonn, Reading, Mass. u. a.), 1994