h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik–

Erarbeitung einer Ontologie zur Darstellung von Systemanforderungen in einer Graphdatenbank

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von

Christian Meyer

Referent	:	Prof. Dr. Gunter Grieser
Korreferent	:	Björn Bär
Betreuer	:	Dr. Julius Geppert

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit wurden von mir selbst erstellt oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 08. März 2019

Christian Meyer

ABSTRACT

In the digital age, the amount of information available determines the competitiveness of a company. The ability to extract new relevant data from incomplete data is therefore a strategic advantage. In this thesis, system requirements are used to investigate the possibilities of determining new unknown data using dependencies. The work shows that a relational database system used so far is unsuitable for this case of application. Instead, the solution is a more flexible and powerful system, which automatically determines new information from existing data sets on the basis of rules. Ontologies are a proven, universally usable possibility to define such rules and to describe system requirements in a generally valid way. This ensures valuable knowledge about the internal structure of the system requirements. For the development of the necessary ontology, a specific modelling process is developed with the help of the literature. This is applied to the problem and an ontology is developed with the help of the analysis of an existing database of system requirements.

The result of this work is a functional ontology, which among other things guarantees the quality and completeness of the data. In addition, a control system is presented, which makes it possible to derive software-technical system requirements as well as to uncover possible logical inconsistencies. Since no automatic conclusions are possible for the hardware requirements, database queries are developed which enable conclusions to be shifted into the program logic. The implementation in the company shows that the use of a graph database instead of a relational database is more suitable for this application. Finally, a possible prototypical implementation is presented.

ZUSAMMENFASSUNG

Im digitalen Zeitalter bestimmt die Menge der zur Verfügung stehenden Informationen die Wettbewerbsfähigkeit eines Unternehmens. Die Fähigkeit, aus unvollständigen Daten neue relevante Daten zu erschließen, ist deshalb von strategischem Vorteil. In dieser Arbeit wird anhand von Systemanforderungen untersucht, welche Möglichkeiten existieren, mittels Abhängigkeiten neue unbekannte Daten zu bestimmen. Die Arbeit zeigt auf, dass ein bisher verwendetes relationales Datenbanksystem für diesen Anwendungsfall ungeeignet ist. Als Lösung wird stattdessen ein flexibleres und leistungsfähigeres System angestrebt, welches automatisch und auf Basis von Regeln aus vorhanden Datensätzen neue Informationen bestimmt. Ontologien sind eine erprobte universell einsetzbare Möglichkeit, solche Regeln zu definieren und damit auch Systemanforderungen allgemeingültig zu beschreiben. Dadurch wird wertvolles Wissen über die interne Struktur der Systemanforderungen gesichert. Für die Entwicklung der notwendigen Ontologie wird mit Hilfe der Literatur ein spezifischer Modellierungsprozess entwickelt. Dieser wird auf die Problemstellung angewendet und eine Ontologie mit Hilfe der Analyse einer vorhandenen Datenbasis von Systemanforderungen entwickelt.

Ergebnis dieser Arbeit ist eine funktionsfähige Ontologie, die unter anderem die Qualität und Vollständigkeit der Daten gewährleistet. Außerdem wird ein Regelsystem vorgestellt, das ermöglicht, softwaretechnische Systemanforderungen herzuleiten sowie etwaige logische Inkonsistenzen aufzudecken. Da für die Hardwareanforderungen keine automatischen Schlüsse möglich sind, werden Datenbankabfragen entwickelt, die es ermöglichen, Schlussfolgerungen in die Programmlogik zu verlagern. Bei der Umsetzung im Unternehmen zeigt sich, dass die Verwendung einer Graphdatenbank an Stelle einer relationalen Datenbank für diesen Anwendungsfall besser geeignet ist. Abschließend wird eine mögliche prototypische Implementierung vorgestellt.

ACKNOWLEDGMENTS

Namentlich gilt herzlichsten Dank meinem Referenten Herrn *Prof. Dr. Gunter Grieser* und meinem Unternehmensbetreuer Herrn *Dr. Julius Geppert*. Beide haben mich im Praxisprojekt und bei der Erstellung dieser Bachelorarbeit maßgeblich unterstützt. Ebenso bedanke ich mich bei meinem Korreferenten Herrn *Björn Bär*, der mich in den vergangen Praxisphasen aber auch bei der Bachelorarbeit konstruktiv begleitet hat.

Darüber hinaus möchte ich mich besonders bei *Ralf* und *Almut Meyer* bedanken für deren Hilfestellung und Motivation.

Außerdem möchte ich mich bei *Prof. Dr. Zander*, *Dr. Ulrich Walther* und *Dr. Peter Haase* bedanken, die mich bei fachlichen Fragen unterstützten. Für die Bereitstellung der Software und Server bedanke ich mich bei *Metaphacts*.

Zuletzt möchte ich mich bei allen bedanken, die mir sonst bei der Vollendung der Arbeit geholfen haben.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ABKÜRZUNGSVERZEICHNIS

BOM	"Bill-of-Material"
CRUD	"**C**reate-, **R**ead-, **U**pdate- und **D**elete-"
CWA	"Closed-World-Assumption"
DB	"Datenbank"
DBMS	"Datenbankmanagementsystem"
ER	"Entity-Relationship"
JDK	"Java-Development-Kit"
LPG	"Labeled-Property-Graph"
NoSQL	"nicht-relationalen Datenbanken"
OS	"Operating-System"
OWA	"Open-World-Assumption"
OWL	"Web-Ontology-Language"
PVP	"Product_Version_Platform"
PVSR	"Product_Version_Software_Requirement"
RDF	"Ressource-Description-Framework"
RDF-S	"Resource-Description-Framework-Schema"
SQL	"Structured-Query-Language"
SPARQL	"**S**PARQL-**P**rotocol-**a**nd-**R**DF-**Q**uery-**L**anguage"
W3C	"World-Wide-Web-Consortium"

1 EINLEITUNG

1.1 MOTIVATION

Die IT-Landschaften von Unternehmen oder Organisationen werden fortlaufend umfangreicher und komplexer. Die IT-Infrastruktur ist eine Kombination aus Abhängigkeiten und Kritikalitäten, wodurch Systemausfälle erhebliche Kosten verursachen können. Die Britische TSB Bank hat beispielsweise einen Verlust von 200 Millionen englischen Pfund und Kundenvertrauen hinnehmen müssen, durch eine gescheiterte Migration (vgl. [18]). Vor einer Aktualisierung, Migration, Änderung oder Erweiterung der Systeme ist es deshalb von entscheidendem Vorteil zu wissen, ob die Infrastruktur danach immer noch funktionsfähig ist. Eine garantiert reibungslose Migration gibt es nicht, aber der Datenabgleich von minimalen Systemanforderungen mit der zukünftigen Serverkonfiguration kann ein stichhaltiger Indikator für die Funktionsfähigkeit sein. Diese Daten werden im Moment empirisch ermittelt und in einer Datenbank händisch hinterlegt. Dies kann zu logischen Inkonsistenzen und hohen manuellen Aufwänden führen. Eine offenzugängliche Definition der Systemanforderungen in einem geeigneten Wissensmodell, zum Beispiel einer Ontologie, ermöglicht es, Unternehmen neue Datensätze und gefährliche Diskrepanzen automatisiert festzustellen. Dies steigert die Datenqualität und reduziert manuelle Aufwände.

Eine IT-Planungssoftware könnte mit den Daten über die IT-Landschaft und die Systemanforderungen eine Migration der bestehenden Systeme simulieren. Das bedeutet, für ein ganzes Netzwerk kann verglichen werden, ob die Systeme danach immer noch den Anforderungen entsprechen.

Ein weiterer Aspekt ist, dass Mitarbeiter die Systeme der Kunden derzeit manuell inspizieren, um sicherzustellen, dass diese den Anforderungen der Produkte entsprechen. Die im Unternehmen verwendete Installationssoftware könnte die Funktionsfähigkeit der Software automatisch vorab feststellen, wodurch der Aufwand sowie potentielle Fehler weiter reduziert werden.

Hierzu ist es erforderlich, dass die Migrationssimulation als auch die Systemanalyse auf eine entsprechende Datenqualität zugreifen können.

1.2 ZIELSETZUNG DER ARBEIT

Ziel dieser Arbeit ist die Modellierung des Produktportfolios und den zugehörigen Systemanforderungen in einem geeigneten wissensbasierten System. Hierfür wird eine möglichst allgemeingültige Ontologie vorgeschlagen, die die Besonderheiten von Systemanforderungen beschreibt. Ein weiteres Ergebnis ist deren Darstellung in einer Ontologie-Beschreibungssprache. Die Korrektheit des Modells wird durch Testdaten in einer geeigneten Softwareumgebung festgestellt. Die Implementierung als Produktivsystem innerhalb des Unternehmens erfolgt mittels einer Graphdatenbank. Dabei wird die Integrität der Datenbank auf Grundlage der Ontologie-Beschreibung sichergestellt. Das wesentliche Ziel hierbei ist es, Abfragen über Datensätze zu stellen, die nicht explizit existieren, sondern vom System geschlussfolgert wurden.

1.3 STRUKTURIERUNG DER ARBEIT

2 Grundlagen von Datenbanksystemen
Zunächst werden Konzepte eines Datenbankmanagementsystems sowie unterschiedliche Datenbanktypen erörtert.

3 Ausgangssituation und Problembeschreibung
Die Systemanforderungen sowie das aktuell eingesetzte Datenbanksystem zur Speicherung werden vorgestellt. Unzulänglichkeiten und Problembeschreibung werden dargestellt.

4 Methodik aus dem Wissensmanagement
Das Lösungssystem soll logische Zusammenhänge erkennen und möglichst fehlerfrei einordnen können. Dafür sind wissensbasierte Systeme erforderlich. Grundlagen über "Wissen" werden in diesem Abschnitt behandelt. Ontologien sind eine Möglichkeit zur Repräsentation von explizitem Wissen in wissensbasierten Systemen. Funktionsweise, Aufbau, Konstruktion und der Modellierungsprozess sind auch Gegenstand von diesem Abschnitt.

5 Ontologie zur Darstellung von Systemanforderungen
Die gesammelten Informationen über das Wissensmanagement und den Ontologien werden auf die konkrete Problemstellung angewendet. Das Resultat ist eine erarbeitete Ontologie, welche Systemanforderungen abbildet. Darüber hinaus werden Möglichkeiten zur automatischen Berechnung von weiteren Datensätzen vorgestellt.

6 Produktivsetzung in einer Graphdatenbank
Für die Implementierung des Szenarios im Unternehmen ist es notwendig, das System im Netzwerk zur Verfügung zu stellen. Dies ist mit der Entwicklungsumgebung zum Erstellen von Ontologien nicht möglich, da diese ausschließlich lokal funktionieren. Die Lösung ist ein Datenbanksystem, welches auf einem Server installiert werden kann. Die Daten müssen in das neue System migriert werden und das erstellte Modell wird hinsichtlich der Korrektheit evaluiert.

7 Fazit
Im Fazit wird zusammengefasst inwieweit die Problemstellung und die damit verbundene Zielsetzung von der Arbeit erfüllt wurden. Darüber hinaus wird im Fazit ein Ausblick für weitere Forschungsmöglichkeiten vorgestellt.

2 GRUNDLAGEN VON DATENBANKSYSTEMEN

2.1 DEFINITION VON ZEICHEN UND DATEN

Daten sind eine Folge von Zeichen mit einer Ordnungsregelung. Das heißt, eine Aneinanderreihung von beispielsweise Bytes, Symbolen oder Buchstaben nach einem Regelsystem. So sind Sprachen der menschlichen Kommunikation ein solches Regelsystem und geben vor, in welcher Reihenfolge Zeichen angeordnet werden müssen, um Worte zu bilden (Daten). Die Ordnungsregelung wird in der Linguistik als "Syntax" bezeichnet. Dabei müssen Daten nicht zwangsläufig in textueller Form vorliegen. Denkbar sind auch komplexere Daten, zum Beispiel Bilddaten, Tonaufnahmen und sonstige Medien. In einem Computer können Daten erzeugt, manipuliert oder gelöscht werden. Die Bedeutung des Begriffs "Daten" ist nicht eindeutig und somit interpretierbar (vgl. [15]).

2.2 KOMPONENTEN EINES DATENBANKSYSTEMS

Ein Vorteil bei der Verwendung einer Datenbank ist das Trennen der Daten von der Anwendungsebene. Dieses Konzept ermöglicht es, dass mehrere unterschiedliche Anwendungen dieselbe Datenbasis haben. Datenbanksysteme bestehen aus der "Datenbank" (DB) und dem "Datenbankmanagementsystem" (DBMS). Die Datenbank ist eine Ansammlung von zusammengehörigen Daten. Das Datenbankmanagementsystem ist eine Softwarekomponente, die es dem Benutzer ermöglicht, die Datenbank zu erstellen und zu verwalten. Softwareanwendungen sind in der Lage mit "Abfragen" (engl. "Queries") sogenannte "**C**reate-, **R**ead-, **U**pdate- und **D**elete-" (CRUD) Befehle durchzuführen (vgl. [6]). In der schematischen Abbildung 2.1 eines Datenbanksystems verarbeitet das DBMS die Aufrufe der Softwareanwendung mit der Datenbank.

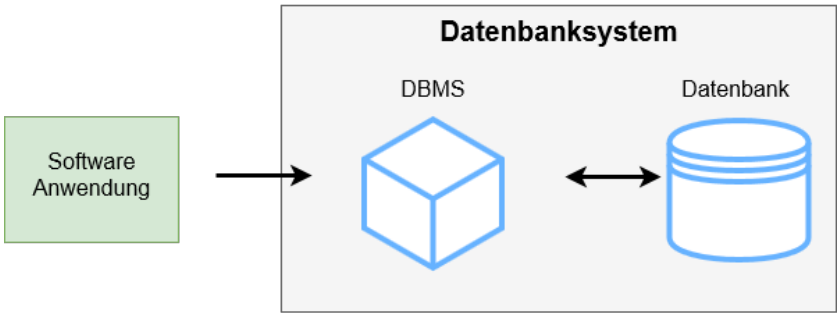


Abbildung 2.1: Datenbanksystem mit einzelnen Komponenten

2.3 AUFGABEN EINES DATENBANKMANAGEMENTSYSTEMS

Ein Datenbankmanagementsystem stellt ein Rechtesystem zur Verfügung, welches Benutzergruppen mit entsprechenden Rechten den Zugriff gewährt. Bei einem System- oder Anwenderfehler bietet es die Möglichkeit, Daten wiederherzustellen. Das DBMS verhindert die mehrfache Implementation von Datenzugriffsroutinen, indem es zum Beispiel "Stored Procedures" und "Trigger" zur Verfügung stellt. Dabei handelt es sich um gespeicherte Abfragen, die entweder auf Befehl oder automatisch mit einem CRUD-Aufruf ausgeführt werden (vgl. [6]).

2.4 DATENBANKTYPEN UND ANWENDUNGSGEBIETE

2.4.1 *Relationale Datenbanksysteme*

Der am häufigsten verwendete Datenbanktyp ist die relationale Datenbank (vgl. [12]). Diese setzt sich aus "Entities" (Tabellen), "Attributes" (Spalten), "Records" (Zeilen) und "Values" (Daten) zusammen. Den Spalten werden Datentypen zugeordnet, zum Beispiel "Integer" für ganze Zahlen oder "Varchar" für Zeichenketten. Die Datenbank überprüft die Korrektheit der Daten anhand der Datentypen. Die Modellierung einer solchen Datenbank geschieht in der Regel in drei Phasen. In der "Konzeptionellen Designphase" werden die zu speichernden Datensätze abstrahiert und in einem "Entity-Relationship" (ER) Diagramm dargestellt. Dieses ER-Diagramm wird als Vorlage für ein logisches "relationales Modell" verwendet. Das logische Modell unterscheidet sich durch eine umfangreichere Darstellung. In der physischen Umsetzung wird dann das Modell in die Datenbank übertragen, um die Daten hinsichtlich ihrer Integrität im Zusammenhang mit dem Modell zu validieren (vgl. [6]). Die eindeutige Zuordnung eines Datensatzes geschieht über "Primary-Keys" (PK). Diese können mit einer künstlichen Identifikationsnummer versehen oder durch mehrere Attribute zusammengesetzt werden. Mathematische Grundlage ist die relationale Algebra. Eine Tabelle über Softwareanwendungen könnte wie folgt implementiert werden (siehe Abb. 2.2).

Softwareanwendungen

ID	SoftwareName	Version	Betriebssysteme
1	Apama Server	10.2	Windows Server 2012
2	Adabas	10.1	IBM Mainframe
3	Apama Server	10.2	Windows Server 2016

Abbildung 2.2: Relationale Datenbank am Beispiel von Softwareanwendungen

Der doppelte Eintrag in der Tabelle zu "Apama Server" in der Softwareversion 10.2 bedeutet, dass sowohl "Windows Server 2012" als auch "Windows Server 2016" als Betriebssystem unterstützt werden. Das Speichern von mehreren Werten in einem Attribut, wie in einem Array, ist nicht möglich. Die Zuordnung von mehreren Betriebssystemen zu einer Softwareanwendung wird wie in dem Schaubild durch redundante Zeilen realisiert (siehe Apama Server). Die künstliche Identifikationsnummer "ID" garantiert in diesem Beispiel die Eindeutigkeit und ist entscheidend, da ansonsten ungewollt Datensätze durch CRUD-Befehle beschädigt werden könnten. Unterschiedliche "Normalformen" definieren Regeln zum Verhindern von vermeidbaren Redundanzen, eine davon empfiehlt die Verwendung von "Foreign-Keys".
Die Umsetzung der Normalform, geschieht in der Abbildung 2.3. Es wird eine weitere Entität "Betriebssysteme" eingeführt zum Trennen der Abhängigkeit zu den Softwareanwendungen. Diese neue Tabelle beinhaltet die zu speichernden Attribute "Name" (des Betriebssystems) sowie den Primary-Key "SoftwareId" als eigener Foreign-Key. In der Darstellung ist erkennbar, dass der doppelte Eintrag von Apama Server aufgelöst und die Verbindung zu den Betriebssystemen über die ID hergestellt wird. Jedes weitere unterstützte Betriebssystem von Apama Server wird als neuer Eintrag in der Betriebssystemtabelle gespeichert.

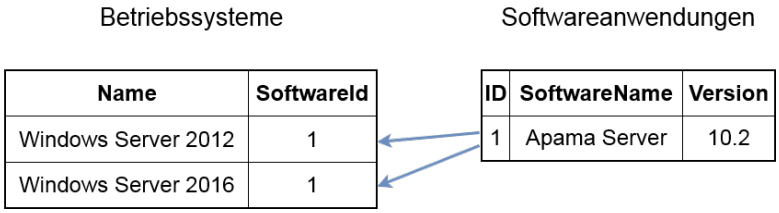


Abbildung 2.3: Softwareanwendungen mit Foreign-Key

Diese Kardinalität, also das Verhältnis zwischen Software und Betriebssystemen ist hier 1:n. Das heißt, zu einer Software können mehrere Betriebssysteme gespeichert werden. Konzeptionell wird zwischen drei verschiedenen Kardinalitäten oder Beziehungstypen unterschieden (1:n, 1:1 und n:m). Bei einer n:m Beziehung wird immer eine "Assoziationstabelle" oder umgangssprachlich "Zwischentabelle" angelegt, die die Datensätze von "n" mit "m" verknüpft (siehe Abb. 2.4).

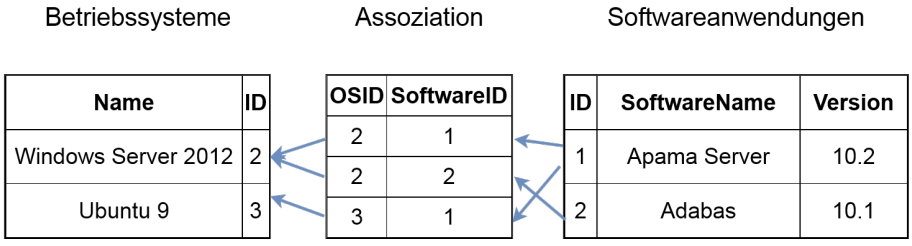


Abbildung 2.4: Softwareanwendungen mit Assoziationstabelle

Das heißt, in der Assoziationstabelle werden jeweils die Primary-Keys als Foreign-Keys gespeichert. Ein Beispiel dafür ist die Beziehung zwischen Soft-

wareanwendungen und Betriebssystemen. Eine Software kann auf mehreren unterschiedlichen Betriebssystemen ausgeführt werden und ein Betriebssystem kann mehrere unterschiedliche Softwareanwendungen unterstützen.
In der Assoziationstabelle könnten weitere Attribute zwischen Software und Betriebssystem gespeichert werden, zum Beispiel, ob bei explizit dieser Kombination Probleme auftreten könnten. Die Abfragesprache "Structured-Query-Language" (SQL) ist syntaktisch an die englische Sprache angelehnt und der Standard bei relationalen Datenbanken. Bei einer Datenbankabfrage wird eine erwartete Ergebnismenge definiert. Auf welche Weise diese vom Datenbanksystem performant ausgeführt wird, entscheidet der "Abfrageoptimierer". Daten über mehrere Tabellen werden mit "Joins" abgefragt. Dabei stellt die Datenbank, solange es nicht näher spezifiziert ist, automatisch die Zusammenhänge der Datensätze über die Primary- und Foreign-Keys her. Speziell bei diesen Abfragen ist die vom Optimierer festgelegte Reihenfolge "execution order" der auszuführenden Einzelschritte entscheidend für eine performante Ausführung. Joins sind assoziativ, das bedeutet, "(A Join B) Join C" ergibt das gleiche Ergebnis wie "A Join (B Join C)". Darüber hinaus sind Joins kommutativ, also "A Join B" ist gleich "B Join A". Für den Benutzer hat dieses Verhalten den Vorteil, dass die Reihenfolge des Ausdrucks in der Abfrage irrelevant ist. Der Optimierer generiert möglichst viele Ablaufroutinen und "wählt" anhand von Heuristiken die möglichst effizienteste Auflösung (vgl. [7]).

2.4.2 *Nicht-Relationale Datenbanksysteme*

Neben den relationalen Datenbanken hat sich ein weiterer Typ von "nicht-relationalen Datenbanken" (NoSQL) etabliert. Diese Gruppe setzt bewusst auf Konzepte, die nicht auf einer Tabellenstruktur und der Abfragesprache SQL basieren. Graphdatenbanken hingegen basieren beispielsweise auf der Graphentheorie aus der Mathematik. Dabei bilden "Knoten" Datenpunkte, die über "Kanten" miteinender verbunden werden. Das "Ressource-Description-Framework" (RDF) ist eine allgemeine Modellierungsart von Graphen. Dabei verbindet das Prädikat das Subjekt mit dem Objekt. Die drei Werte werden deshalb auch als "Triple" bezeichnet. Eine Graphdatenbank, welche Daten in dieser Form abspeichert wird deshalb auch "Triple-Store" genannt. Zur Selektion von Daten wird der Graph "traversiert", das heißt, beginnend von einem Startknoten wird über Kanten und Knoten der oder die entsprechenden Zielknoten gesucht (vgl. [12]). Dabei können sämtliche Wegoptimierungsalgorithmen der Graphentheorie angewandt werden. Das folgende Beispiel stellt die Knoten als blaue Sechsecke dar, die mit Pfeilen (den Kanten) verbunden werden (siehe Abb. 2.5). Linien zu unterstrichenen Werten bezeichnen in der Grafik konkrete Attribute und sind keine Kanten.

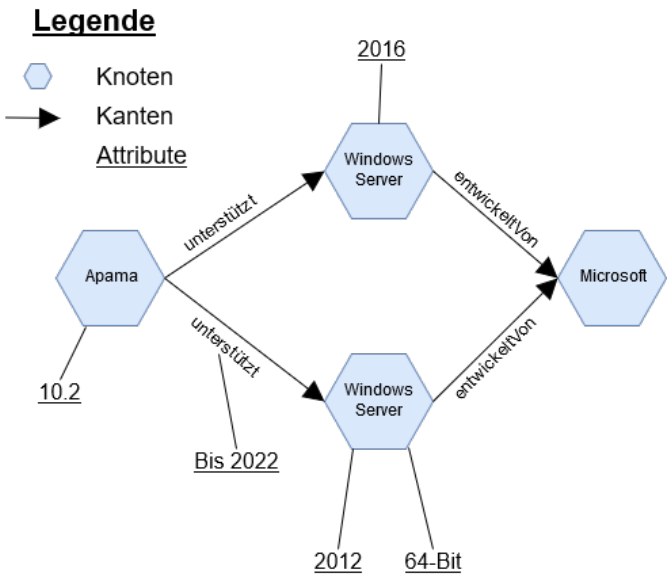


Abbildung 2.5: Schematische Darstellung einer Graphdatenbank

Die Traversierung könnte bei dem Knoten “Apama” starten und als Ergebnis sollen die Softwarehersteller der unterstützten Betriebssysteme ausgegeben werden. Die Graphdatenbank sucht ausgehend von Apama alle Knoten, die mit der Kante “unterstützt” verknüpft sind. Es existieren zwei Knoten “Windows Server” mit der entsprechenden Kante. Von diesen Knoten werden dann alle Kanten traversiert über die “entwickeltVon”-Bezeichnung. Das Resultat wäre der Hersteller Microsoft, da dieser beide Betriebssysteme entwickelt. Die Kante von “Apama” zu “Windows Server 2012” ist ein Beispiel für ein “Labeled-Property-Graph” (LPG), denn nur in diesem können Kante weitere Daten speichern (“unterstützt bis 2022”). Graphdatenbanken sind unter anderem durch das fehlende Datenbankmodell im höchsten Maß flexibel einsetzbar. Bei einem gerichteten Graph können die Kanten, im Gegensatz zu einem ungerichteten Graph, ausschließlich in der vorgegebenen Richtung traversiert werden (vgl. [1]).

3 AUSGANGSSITUATION UND PROBLEMBESCHREIBUNG

3.1 BESCHREIBUNG VON SYSTEMANFORDERUNGEN

Systemanforderungen sind spezielle Daten, die beschreiben, welche Anforderungen eine Computersoftware an ein bestehendes System hat. Dabei wird zwischen Hard- und Softwareanforderungen unterschieden. Die Hardware muss beispielsweise ausreichend Rechenleistung und Speicher aufweisen, damit die Software erwartungsgemäß funktioniert. Verschiedene Hersteller definieren in dem Zusammenhang mindestens zu erreichende und empfohlene Systemeigenschaften. In dieser Arbeit werden ausschließlich die mindestens zu erreichenden Anforderungen betrachtet, weil diese für den Betrieb der Anwendung erforderlich sind. Neben Hardwareeigenschaften können andere Softwareapplikationen als Softwareanforderung vorausgesetzt werden. Softwarehersteller versuchen ihre Anwendung auf möglichst vielen Systemen anzubieten. Dadurch kann eine Software beispielsweise nicht ausschließlich ein Betriebssystem voraussetzen, sondern unterstützt mehrere Alternativen, von der mindetens eine installiert sein muss. Ein Produkt unterstützt zum Beispiel neben Windows Server auch verschiedene Linux Varianten. Ein Eigenprodukt, kann andere Eigenprodukte inkludieren, die wiederum unterschiedliche Systemanforderungen besitzen. Ein Merkmal von “Software-Suites” ist die Integration von mehreren zusammengehörigen Produkten zu einem gebündelten Softwarepaket. Zur Vorbeugung von Missverständnissen wird das Wort “Produkt” und “Eigenprodukt” ausschließlich verwendet, um eigene Softwareentwicklungen zu beschreiben. Ein Produkt ist demzufolge immer “hauseigene” Software und mit “Software” wird situationsabhängig ein Eigenprodukt oder eine Drittanbieter-Software beschrieben.

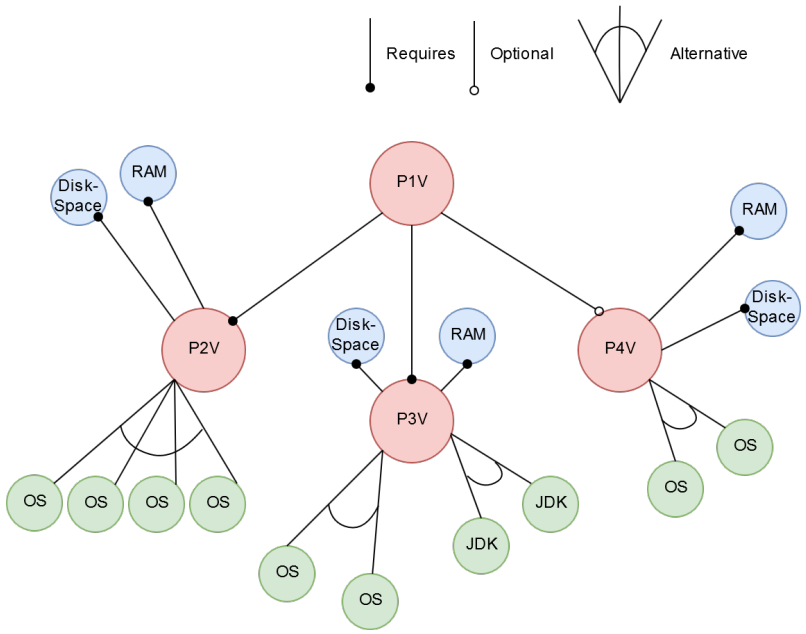


Abbildung 3.1: Darstellung einer typischen Produkthierarchie

Die Abbildung 3.1 zeigt in Rot dargestellt das Eigenprodukt "P1V". Dies ist aus den Produkten "P2V", "P3V" sowie optional "P4V" zusammensetzt. Ein Produkt, welches sich aus anderen zusammensetzt, wird in dieser Ausarbeitung als "Elternprodukt" und die inkludierten Produkte als "Kindprodukte" bezeichnet. Die Software P4V wird mitausgeliefert, aber für einen ordnungsgemäßen Betrieb nicht benötigt. Der Benutzer könnte die Installation möglicherweise im Installationsmenü verhindern. In der Abbildung sind Softwareanforderungen grün dargestellt. Das "Operating-System" (OS) (deutsch "Betriebssystem") und "Java-Development-Kit" (JDK) sind Softwareanforderungen, die in einer bestimmten Version installiert sein müssen. In Blau sind die Hardwareanforderungen gekennzeichnet. Diese sind Mindestanforderungen an das System für den ordnungsgemäßen Betrieb.

Zusammenfassung der Eigenschaften der Systemanforderungen:

1. Produkt kann Produkte inkludieren
2. Produkt kann Software voraussetzen
3. Produkt kann Hardware voraussetzen

3.2 AUSGANGSSITUATION - AKTUELLES SYSTEM

Das aktuell eingesetzte System, welches unter anderem die Systemanforderungen abspeichert, ist eine relationale Datenbank mit einem integrierten Datenbankmanagementsystem. Durch das verwendete SQL-Schema können die Daten repräsentiert werden. Eine vereinfachte Version des ER-Diagramms zur Speicherung der Systemanforderungen zeigt Abbild 3.2.

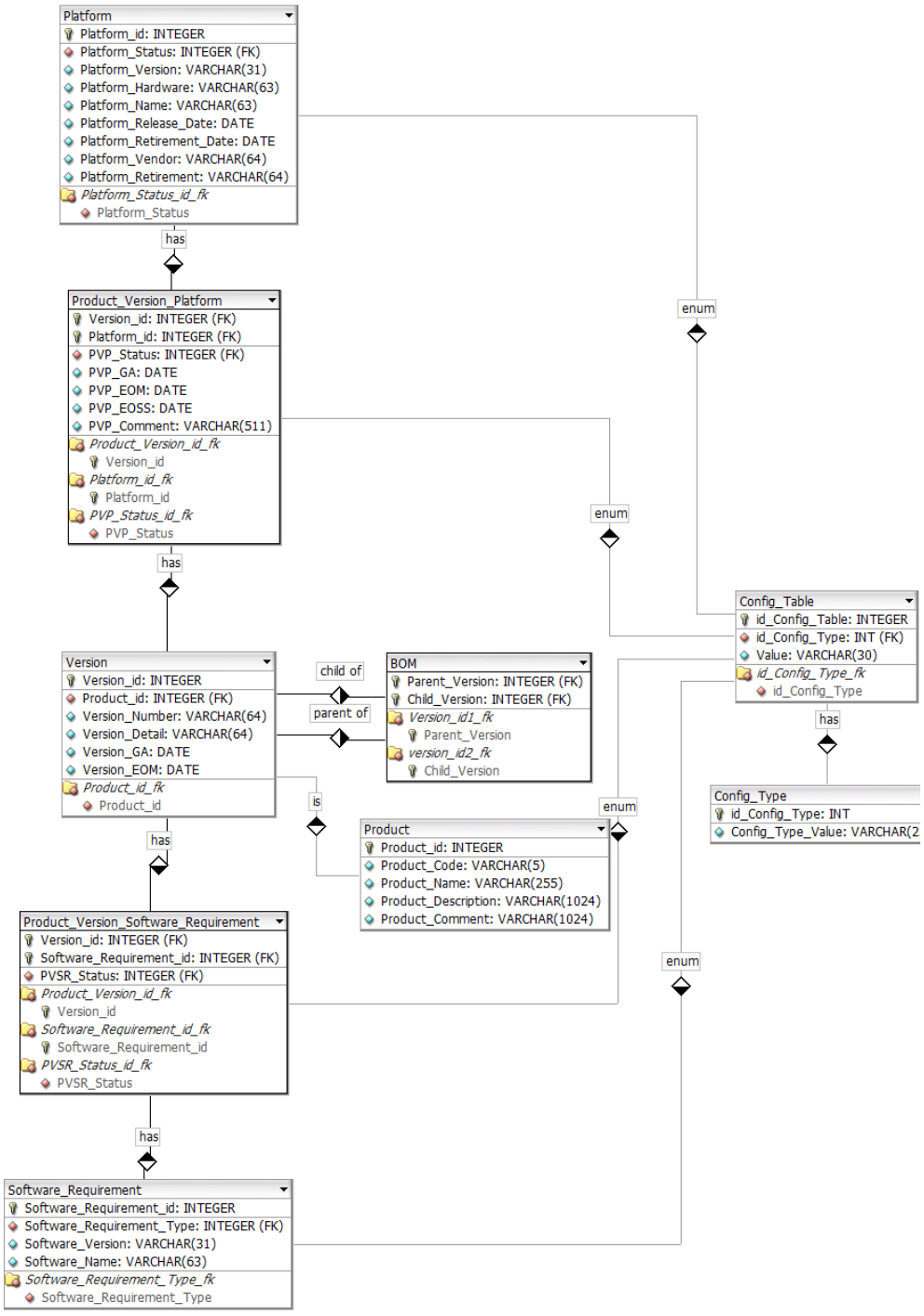


Abbildung 3.2: ER-Diagramm des aktuellen Datenbanksystems

Besonderes Augenmerk liegt auf der Stückliste, "Bill-of-Material" (BOM) und der "Version"-Tabelle (Mittig im Schaubild). Letztere speichert Daten wie den Produktnamen und Softwareversionen von sämtlichen Eigenprodukten ab, die in verschieden Versionen vorliegen können. Die BOM besteht aus den Spalten "Parent" und "Child". Diese Spalten beinhalten die IDs aus der Versionstabelle der verschiedenen Elternprodukte und der direkten

Kindprodukte. Die BOM simuliert damit die Zusammensetzung eines Produkts aus anderen Produkten.

Die Tabelle "Software_Requirement" speichert in diesem Zusammenhang eine Software von Drittanbietern in einer bestimmten Version ab, wie zum Beispiel (JDK 1.8, Firefox 53, ...). Die dazugehörige Assoziationstabelle "Product_Version_Software_Requirement" (PVSR) verknüpft diese externe Softwareversion mit der Eigenproduktversion aus der Versionstabelle.

Die Tabelle "Platform" beinhaltet alle unterstützten Betriebssysteme in unterschiedlichen Versionen. Die Assoziationstabelle "Product_Version_Platform" (PVP) verknüpft ein Betriebssystem mit einer Produktversion aus der Versionstabelle (Beispiel: Produkt A in der Version 2.0 unterstützt Windows Server 2012).

Die Tabellen "Config_Table" und "Config_Type" sind eine Möglichkeit sogenannte "Enums", also eine vordefinierte Auswahlmöglichkeit zu simulieren, da diese in der bisherigen Datenbankanwendung nicht unterstützt werden (Rechts im Schaubild). Bisher werden keine hardwarespezifischen Daten in der Datenbank abgelegt wie zum Beispiel RAM, CPU und Festplattenspeicher.

3.3 PROBLEMBESCHREIBUNG - BISHERIGE UNZULÄNGLICHKEITEN

Die historisch bedingte Aufteilung der Tabellen in "Betriebssystem" (Platform) und "Drittanbietersoftware" (Software_Requirement) ist semantisch nicht korrekt, da beides Computersoftware ist, die von Drittanbietern entwickelt wird. Die Notwendigkeit, Systemanforderungen in der bisherigen Datenbank abzubilden, wurde erst Jahre nach der Entwicklung des Systems erkannt. Das Aufteilen der vorhandenen Daten in eine detailliertere Datenbankstruktur würde bedeuten, dass existierende Datensätze in neue Tabellen und Zwischentabellen migriert werden müssten. Bei dieser Migration könnten Daten unwissentlich und ungewollt beschädigt werden. Alle formulierten Abfragen, Trigger und Stored Procedures wären bei einer Aktualisierung unbrauchbar und müssten angepasst werden. Darüber hinaus müssten alle Systeme hinsichtlich der Zugriffsmethode angepasst werden. Da jedoch andere Systeme mit einer hohen Kritikalität die Datenbank direkt aufrufen, wurde es als zu riskant bewertet, die bestehende Datenbankstruktur zu verändern. Stattdessen wurde die Datenbank um die Tabellen Software_Requirement und PVSR erweitert. Das Hinzufügen der Tabellen ermöglicht das Speichern der Systemanforderungen, ohne Modifikationen an der existierenden Struktur vornehmen zu müssen. Dieses Vorgehen verringert Qualität, Realitätsnähe und Anpassbarkeit der Datenbank. Die Ursachen des Problems sind ungelöst und dadurch wird langfristig der Wartungsaufwand und die Komplexität des System erhöht.

Neben der Anpassbarkeit des Datenbankschemas existieren Probleme hinsichtlich der Ausführung von Abfragen. Die Datenstruktur der Systemanforderungen ist sowohl hierarchisch als auch zyklisch. Hierarchisch, weil das Produkt A, das Produkt B inkludieren kann, welches wiederum Produkt C inkludiert. Zyklisch aufgrund der starken Abhängigkeit zwischen den Produkten, denn das Produkt A kann das Produkt B voraussetzen und umgekehrt (siehe Abb. 3.3).

BOM - Bill of Material

Parent	Child
Product A	Product B
Product B	Product C
Product B	Product A

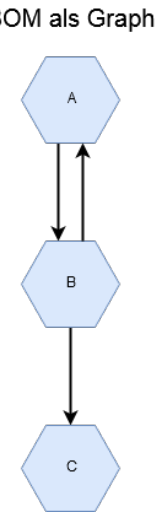


Abbildung 3.3: Beispieldatensätze in der Bill-of-Material (BOM)

Die Anzahl der Joins für eine Abfrage, welche alle inkludierten Produkte von einem Produkt zurückliefert, ist unter der Annahme, dass die Tiefe der Hierarchie bekannt ist gleich "n-1". Die Tiefe entspricht der maximalen Anzahl der Kanten bis zum Startknoten. Das Beispiel 3.3 entspricht demzufolge einer Tiefe von zwei, somit wird ein einziger Join benötigt für die Ergebnismenge. Das Ergebnis enthält lediglich IDs der verschiedenen Produktversionen. Eine Auflösung der IDs mit der Versiontabelle um zusätzlich die Informationen wie Produktnamen und Produktversion zu erhalten, erfordert weitere Joins. Die Anzahl der benötigten Joins entspricht der Menge an aufzulösenden Tabellen "x", multipliziert mit der Tiefe des Baums "n". In diesem Fall wird von der Tiefe "n" nicht "1" subtrahiert, da auch der Startknoten aufgelöst werden muss. Daraus folgt, dass grundsätzlich die Anzahl der Joins nach dieser Abfrage "(x*n)+(n-1)" ist. Diese kann je nach Implementierung und Durchführung abweichen. Für einen Join muss die komplette Tabelle durchsucht werden, folglich wächst die benötigte Zeit proportional zum Wachstum der Datenmengen. Zur erfolgreichen Abfrage der hierarchischen Daten müssen komplexe, rekursiv geschachtelte Joins verwendet werden, welche in der Regel kompliziert und auch performanceaufwändig sind (vgl. [12]). Aus diesem Grund implementieren Anwendungen in der Regel die Logik für die Zuordnung der Produktbeziehungen selbst.

Die Systemanforderungen zu einem Produkt werden mit einem speziellen Programm ermittelt und in die Datenbank eingepflegt. Bestimmte Datensätze lassen sich durch die hierarchische Struktur mit ausreichend hoher Sicherheit aus den vorhandenen Daten schlussfolgern. Infolgedessen können neue Datensätze erstellt und existierende auf logische Integrität überprüft werden. Das jetzige Datenbanksystem unterstützt keine automatisierte Schlussfolgerung. In der existierenden Architektur können logische Schlussfolgerungen ausschließlich mit der Abfragesprache SQL durchgeführt werden. Jede Anwendung, die diese Informationen benötigt, muss komplizierte Abfragen formulieren, was den Wartungsaufwand und die Fehleranfälligkeit vergrößert. Alternativ könnten die Abfragen in dem System als Trigger implementiert werden, der bei einem Update der Datensätze aktiviert wird und eine Berechnung durchführt. Bei einem Massenupdate durch einen Trigger wären die Folgen unter anderem wegen den zuvor beschriebenen Abfrageproblematiken unvorhersehbar. Da Datenbanksysteme Daten, aber kein Wissen über die Daten speichern, können weder existierende Datensätze überprüft, noch neue Daten automatisiert berechnet werden.
Im folgenden Abschnitt wird in das Wissensmanagement eingeführt. Das Wissensmanagement verspricht unter anderem die Problematik der Schlussfolgerungen zu lösen. Darüber hinaus werden die Begriffe Wissensrepräsentation, Wissensbasis und wissensbasierte Systeme erörtert. Dabei handelt es sich um Werkzeuge, die mit einer "intelligenten" Datenbank vergleichbar sind. Der Abschnitt baut thematisch auf die bisher vorgestellten Themen "Daten" und "Datenbanksysteme" auf.

Zusammenfassung der Probleme der jetzigen Datenbanklösung:

1. Eingeschränkte und aufwändige Anpassbarkeit des Systems
2. Möglichkeit Unvorhersehbarer Abfragekonstrukte
3. Keine Überprüfung von logischer Integrität
4. Keine Schlussfolgerung von neuen logischen Datensätzen

3.4 VERWANDTE ARBEITEN UND KONZEPTE

Die Entwicklung eines Lösungsansatzes, ist aus zwei unterschiedlichen Perspektiven erfolgt. Zunächst wurde nach Verwandten Arbeiten innerhalb des Umfelds der Systemanforderungen gesucht. Da die Ergebnisse aus der Literatur zu diesem Thema nicht erfolgsversprechend für die Lösung des Problems waren, wurde ein Lösungsansatz aus dem Wissensmanagement verfolgt. Der Abschnitt über Wissensmanagement wurde maßgeblich von Davenport, Thomas H. and Prusak, Laurence beeinflusst.
Für die Themengebiete "Daten", "Informationen" und "Wissen" gibt es jeweils unterschiedliche Definitionen. Für diese Arbeit wird der Erklärungsansatz von Klaus North aus seinem Buch "Wissensmanagement für Qualitätsmanager" angewendet.

Neben der theoretischen Analyse und praktischen Evaluierung von Lösungen wurden auch Befragungen mit erfahrenen Experten von semantischen Lösungen und Entwicklern der "Enterprise Cloud und Data-Center Ontology" durchgeführt. Hierdurch konnten weitere Lösungsansätze identifiziert und konzipiert werden.

4

METHODIK AUS DEM WISSENSMANAGEMENT

4.1 ABGRENZUNG VON DATEN, INFORMATIONEN UND WISSEN

Im Grundlagenkapitel wurde erläutert, dass Zeichen mithilfe von Syntax (einem Regelsystem) zu Daten werden. "Informationen" sind darauf aufbauend Daten mit einer Bedeutung. Dies wird auch "Kontext" , bzw. "Semantik" genannt. "Wissen" ist ein fortlaufender Prozess, um unterschiedliche Informationen miteinander zu verflechten. Mit Wissen kann aus Daten neue Information gewonnen werden. Wissen und Informationen ergänzen sich gegenseitig, da mit Wissen aus Informationen neues Wissen entstehen kann und umgekehrt (vgl. [15]). Eine schematische Übersicht bietet die "Wissenstreppe" nach North (siehe Abb. 4.1).

Abbildung 4.1: Wissenstreppe nach North (Kopie aus: [15])

Die unterste Stufe bilden "Zeichen" und die oberste die "Wettbewerbsfähigkeit" des Unternehmens. Das operative Wissensmanagement ist unterhalb der Treppenstufe und ist die zu erfüllende Bedingung zum Erklimmen einer Stufe. Oberhalb ist das strategische Wissensmanagement und analysiert von oben nach unten Lücken in der Umsetzung.

Wissen ist eine Stufe zum Ziel der Wettbewerbsfähigkeit. Es gibt zwei unterschiedliche Arten von Wissen.
Das sogenannte **"Implizite Wissen"** differenziert von Mensch zu Mensch. Beispielsweise werden mit Wörtern unterschiedliche Emotionen und Interpretationen verknüpft, obwohl die Semantik identisch ist. Dieses Wissen ist personengebunden und schwer zu kommunizieren. Demgegenüber ist **"Explizites Wissen"** solches, das identifiziert, gespeichert und geteilt werden kann. Es lässt sich artikulieren und ist systematisch (vgl. [15]).

Zur Veranschaulichung ein Beispiel mit der Zeichenkette **"RAM"** :
Die Zeichen entsprechen dem Regelsystem, also der Syntax der englischen Sprache. Es handelt sich hier um ein Datum. Ein englischsprachiger Mensch, könnte diese Zeichenkombination als die Wörter "Ramme", oder "Widder" verstehen. Dabei kommt es auf die Semantik bzw. den Kontext an, in der sich das Datum befindet. Würden wir die Zeichenkette zu "RAM is defect in the computer" erweitern, so wird das Datum zur Information, denn es befindet sich in einem Kontext. Mit hoher Wahrscheinlichkeit ändert sich daher die Interpretation zu einer Komponente im Computer.
Der Empfänger kann diese Information mit seinem Wissen verarbeiten, wenn er weitere Informationen zur Bedeutung und den Umständen der Äußerung hat. Zum Beispiel mit der Information, dass diese Komponente entscheidend für die Funktionsfähigkeit des Gerätes ist. Der Empfänger kann schlussfolgern, dass der Computer des Senders nicht mehr funktioniert. Auf dieser Grundlage könnte ein Handeln folgen (Pragmatik), dass der Empfänger dem Sender einen Computer leiht. Bei einer erfolgreichen Kommunikation würde der Empfänger, die vom Sender gewünschte Reaktion ausführen. Eine Garantie zur korrekten Interpretation gibt es aber nicht. Der Empfänger kann durch den transportierten Informationsgehalt lediglich die Intension des Senders vermuten. Die Erkenntnis daraus ist, dass der Empfänger ausschließlich auf Grundlage von vorhandenen Informationen Entscheidungen treffen kann (vgl. [2]). Im späteren Verlauf der Arbeit hilft diese Abgrenzung, das angestrebte System besser nachzuvollziehen.

4.2 PRAKTISCHE ANWENDUNG VON WISSENSMANAGEMENT

Die Aussage "Wissen ist Macht", die auf den englischen Philosophen "Francis Bacon" zurückzuführen ist, bekommt im digitalen Zeitalter einen neuen Stellenwert. Wissen wird als Produktionsfaktor gehandelt und Investitionen in diesem Bereich sind ähnlich profitabel wie in anderen Sektoren (vgl. [10]). Als Produktionsfaktoren werden die aufzuwendenden Leistungen zusammengefasst, die zur Herstellung von Gütern benötigt werden. Wissenmanagement beschäftigt sich mit der Frage, wie Wissen innerhalb des Unternehmens strukturiert und nutzbar gemacht werden kann. Davenport und Prusak definieren Wissensmanagement als "die Arbeit der Erfassung, Verteilung und Nutzung von Wissen" ([5]). Nutzeffekte beim Einsatz der Techniken sind unter anderem das Verfügbarmachen von Wissen, die Möglichkeit

Wissen zu vervielfältigen und letztlich auch zu sichern (vgl. [11]). Gerade wenn Experten eine Organisation verlassen ist es wichtig, dass ihr explizites Wissen konserviert wird und nicht verloren geht. Wissensbasierte Systeme sind eine Möglichkeit dieses Prinzip softwaregestützt umzusetzen.

4.2.1 *Wissensbasierte Systeme*

In der Softwareentwicklung werden meistens die Datenhaltung und die Algorithmen zur Berechnung getrennt, was zum Verlust von Flexibilität führt. Datenbanksysteme bilden den Datenspeicher und Anwendungsprogramme ermitteln mit Algorithmen unter Anderem neue Daten. Das Wissen über die Daten wird in einer solchen Architektur in Form von Algorithmen abgelegt. Anders ist die Architektur von wissensbasierten Systemen. Sie bestehen aus einer Inferenzkomponente, einer Wissensbasis und einer Benutzerschnittstelle. Das ermöglicht solchen Systemen, Wissen und Daten gemeinsam abzuspeichern (vgl. [11]).

4.2.2 *Wissensbasis und Wissensrepräsentation*

Die Wissensbasis ist eine Art Datenbank, die Zusammenhänge, Fakten und Regeln als Wissen abspeichert. Die Wissensrepräsentation ist die formale Abbildung des Wissens und unterscheidet sich unter anderem hinsichtlich der Mächtigkeit. Mögliche Varianten der Wissensrepräsentation aufsteigend der semantischen Mächtigkeit und Komplexität sind: Taxonomien, Thesauri und Ontologien. Taxanomien unterstützen ausschließlich hierarchische Strukturen. Zyklische Abhängigkeiten wie in den Daten der Systemanforderungen sind nicht zu modellieren, beispielsweise “A benötigt B” und “B benötigt A”. Thesauri erweitern Taxanomien um vordefinierte, nicht hierarchische Beziehungen. Die Ontologie ist die einzige der vorgestellten Repräsentationen, die wegen der semantischen Aussagekraft und mathematische Genaugikeit das Schlussfolgern von Daten überhaupt ermöglicht (vgl. [11]). Neben der fehlenden Ausdrucksstärke von Taxanomien und Thesauri existieren bereits Ontologien, um die komplette IT-Infrastruktur darzustellen. Mit Hilfe dieser vorhandenen Ontologien in Kombination mit einer Darstellung der Systemanforderungen, könnte die in der Motivation vorgestellte Vision verwirklicht werden.

4.2.3 *Inferenzmaschine*

Mittels der formalen Spezifikation von Wissen durch die Wissensrepräsentation in der Wissensbasis ist eine Inferenzmaschine in der Lage, nicht vorhandene Datensätze zu schlussfolgern (engl. Reasoning). Eine Inferenzmaschine ermöglicht unter anderem, nicht kategorisierte Daten zu klassifizieren, die Integrität von Daten sicherzustellen, oder neues Wissen aus vorhandenen Daten abzuleiten (vgl. [11]).

4.3 ONTOLOGIE ALS WISSENSREPRÄSENTATION

Der Begriff "Ontologie" (engl. Ontology) hat seine Ursprünge in der Philosophie und prägt hier die "Lehre des Sein" (vgl. [4]). Es soll die Welt durch Strukturen und Beziehungen zwischen den einzelnen Elementen als Ganzes beschrieben werden. Eine einheitliche Definition gibt es sowohl in der Philosophie, als auch in der Informatik nicht (vgl. [4]). Teilweise sind die Unterschiede in der Definition erheblich. Eine häufig zitierte Definition von Tom Gruber lautet: "An ontology is an explicit specification of a conceptualization" ([8]). In dieser Arbeit genügt die ergänzende Definition von Studer als "[...] formal explicit [...]" ([17]). Eine Wissensrepräsentationsmöglichkeit wie eine Ontologie kann durch die Mächtigkeit einen beliebigen Sachverhalt formal und präzise spezifizieren. Zu unterscheiden sind "Foundational"- oder auch "Upper"- von einer "Domain"-Ontologie. Eine "Upper"-Ontologie ist generisch und hat den Anspruch ähnlich wie in der Philosophie, die Welt abzubilden. Dabei wird versucht, Wissen interdisziplinär über alle Bereiche zu sammeln und in einer Ontologie zu strukturieren. Demgegenüber ist die "Domain"-Ontologie ein spezifischer Ausschnitt eines Sachverhalts. Mit "Verbindungsontologien" können unterschiedliche Ontologien miteinander verknüpft werden (vgl. [16]).

4.3.1 *Aufbau und Konstruktion*

Konzeptionell wird zwischen sogenannten "lightweight" und "heavyweight" Ontologien unterschieden. Lightweight-Ontologien beschränken sich hauptsächlich auf Klassen, Relationen und deren Eigenschaften. In der Regel sind Computer nicht in der Lage, anhand dieser Modelle schlusszufolgern. Heavyweight-Ontologien ergänzen diese um formale Definitionen über die Konzeptionalisierung. Ontologiebeschreibungssprachen bieten die Möglichkeit, beide Ontologie-Typen maschinenlesbar aufzubereiten. Für die Wahl der Beschreibungssprache sollte sowohl die Ausdrucksstärke, Semantik als auch die mathematische Genauigkeit berücksichtigt werden. Im Abschnitt 2.4.2 wurde RDF als Möglichkeit zur Speicherung von Daten in einer Graphdatenbank vorgestellt. Darauf aufbauend wurde im Umfeld des semantischen Webs, das "Resource-Description-Framework-Schema" (RDF-S) und die darauf aufbauende "Web-Ontology-Language" (OWL) des "World-Wide-Web-Consortium" (W3C) entwickelt. Beide eignen sich zum Beschreiben von Ontologien und unterscheiden sich hinsichtlich der Beschreibungsmöglichkeiten. Lightweight-Ontologien können in unterschiedlichen Formaten dargestellt werden, wie zum Beispiel IDEF5, Mind-Maps, UML-Diagrammen oder auch durch selbstentwickelte Modelle. Voraussetzung für eine eigene Darstellung sind klare Regeln und die Nachvollziehbarkeit. Die Lightweight-Ontologie eignet sich gut für die Mensch-zu-Mensch-Kommunikation, um grundlegende Ideen vorzustellen (vgl. [19]).

4.3.2 *Funktionsprinzipien*

Die wesentlichen Bausteine einer Ontologie sind "Klassen", "Relationen" und "Instanzen". Eine Klasse beschreibt eine Kategorie von Elementen. Wie zum Beispiel "Softwarehersteller", also Firmen die Software entwickeln. Konkrete Softwarehersteller, wie zum Beispiel "SoftwareAG", "SAP" oder "Microsoft" sind Instanzen der Klasse "Softwarehersteller". Instanzen beschreiben immer ein spezifisches Objekt einer Klasse. Relationen verbinden einzelne Instanzen. Dies wird über die Relationen der Klassen definiert. Häufige Relationstypen sind unter anderem die "Subsumption" (umgangssprachlich "Vererbung") und die "Assoziation". Bei der Subsumption werden die Eigenschaften von dem Elternteil (Startknoten) dem Kind (Endknoten) vererbt (vgl. [19]). Eine nicht standardisierte Darstellung könnte wie in Abbildung 4.2 aussehen.

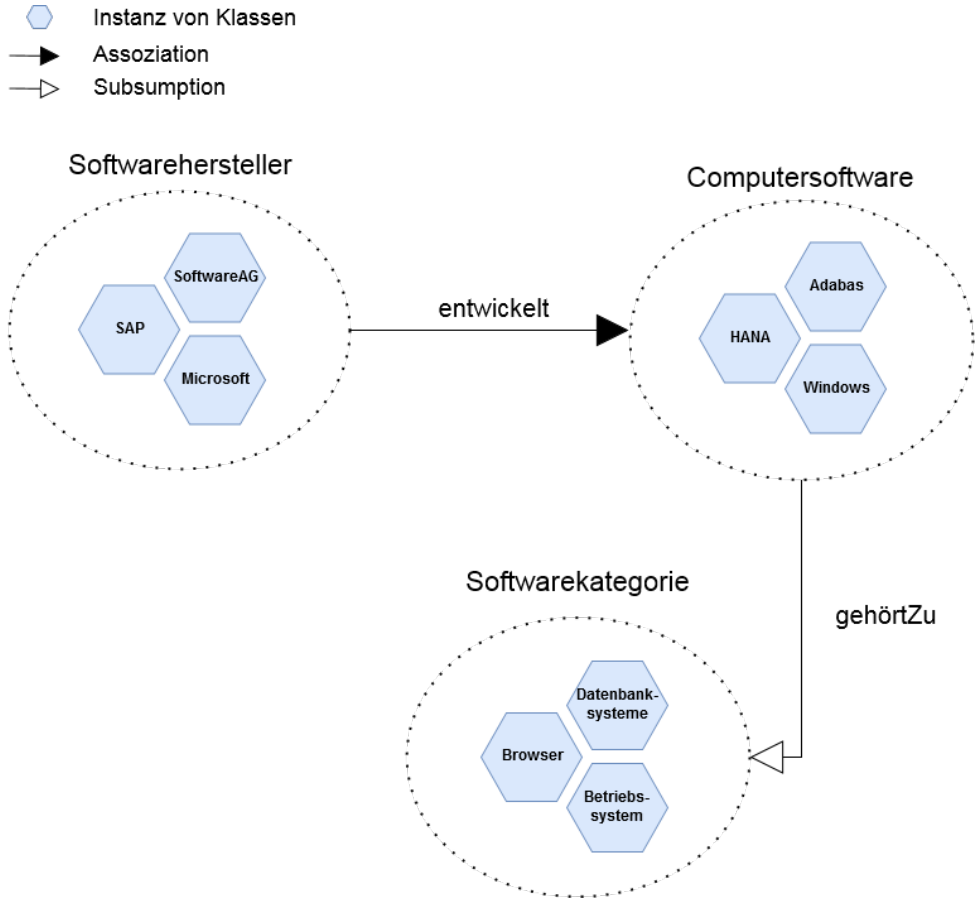


Abbildung 4.2: Grundkomponenten einer Ontologie

Durch die Relationen zwischen den Klassen werden die Beziehungen zwischen den Instanzen "erlaubt".

4.3.3 *Abgeschlossenheit*

Datenbanksysteme arbeiten meistens mit der "Closed-World-Assumption" (CWA), was bedeutet, dass ausschließlich Aussagen über vorhandene Datensätze beantwortet werden können. Bei der "Open-World-Assumption" (OWA) wird der Datenbestand nicht als vollständig betrachtet, sondern als ein Ausschnitt der Welt. Das heißt, Aussagen über einen Sachverhalt werden als "unbekannt" beantwortet, so lange sie nicht explizit anderweitig definiert wurden (vgl. [16]).
Die Erkenntnis des Beispiels von "4.1 Abgrenzung von Daten, Informationen und Wissen" ist, dass ein Empfänger Entscheidungen ausschließlich auf der Grundlage seiner vorhanden Informationen treffen kann. Eine Interpretation von fehlenden Informationen wird erst möglich unter einer Open-World-Assumption, also der Annahme, dass Aspekte unmodelliert sind und / oder fehlen. Philosophisch betrachtet (siehe Abschnitt 4.3) ist eine Ontologie abgeschlossen, durch das Abbilden der Welt als Ganzes. Da dies nicht möglich ist, gilt eine Ontologie als korrekt und abgeschlossen durch Abbildung des gewünschten Sachverhalts. Daraus folgt keine Notwendigkeit, dass mit der Ontologie zwangsläufig geschlussfolgert werden muss. Darüber hinaus wird bei der Entwicklung das Sparsamkeitsprinzip angewandt. Dieses verlangt eine möglichst geringe Verwendung von Komponenten für die Durchführung, ohne dass die logische Integrität darunter leidet (vgl. [16]).

4.3.4 *Modellierungsstrategie*

Für die Konstruktion einer Ontologie, gibt es kein allgemeingültiges Erfolgsrezept, welches zwangsläufig zu einer funktionierenden Ontologie führt. Es gibt jedoch Methodiken und Werkzeuge, die sich beim Erstellungsprozess bewährt haben. Die Modellierungsstrategien sind aber nicht als strikte Abarbeitung von Aufgaben zu verstehen, sondern als ein Prozess der iterativ und inkrementell durchlaufen wird (vgl. [14]).

In dieser Arbeit wird die IDEF5-Strategie verwendet, ein offener Standard des US-Militärs. Das Vorgehensmodell definiert Rahmenbedingungen, die in einzelne Phasen unterteilt werden, bestimmen aber nicht deren Umsetzung. Dadurch ist es möglich, das Konzept nach Belieben anzupassen.
IDEF5 unterteilt den Prozess in fünf Schritte, von der Zielsetzung bis zur Validierung und Verfeinerung der Ontologie. Im Folgenden wird die Durchführung der einzelnen Schritte innerhalb dieser Arbeit im Detail aufgezeigt.

IDEF5 Überblick:

1. Organizing and Scoping - Organisation und Zielsetzung
2. Data Collection - Informationsbeschaffung
3. Data Analysis - Datenanalyse
4. Initial Ontology Development - Initialer Entwurf
5. Ontology Refinement and Validation - Verfeinerung und Validierung

Organisation und Zielsetzung
Zunächst werden Verwendungszweck und Kriterien aufgestellt, die von der Ontologie erfüllt werden müssen. Der Sachverhalt sollte abgeschlossen sein, aber nicht zu eingegrenzt betrachtet werden. Zur Definition der Ziele haben sich unter anderem "Competency-Questions" bewährt, einem Fragekatalog, der später durch die Ontologie überprüft werden kann. Sollten "Questions" nicht durch die Ontologie beantwortet werden, müssen entweder die Ziele oder die Ontologie angepasst werden (vgl. [16]).

Informationsbeschaffung
In dieser Phase werden Rohdaten über die Domäne gesammelt, wobei es vorteilhaft ist, eine möglichst breite Informationsbasis aufzubauen. Dies soll in Kombination mit der Zielsetzung eine möglichst vollständige Abdeckung sicherstellen. Dabei wird zwischen ontologischen und nicht-ontologischen Informationen unterschieden. Empfehlenswerterweise wird überprüft, ob eine bestehende Ontologie komplett oder teilweise verwendet werden kann. Die Wiederverwendung von vorhandenen anerkannten Ontologien steigert deren Glaubwürdigkeit. Vorteile sind außerdem eine relative Sicherheit hinsichtlich der Korrektheit und die Einsparungen von Aufwänden einer Eigenentwicklung. Darüber hinaus eröffnet dies die Möglichkeit zum quasi barrierefreien Austausch von Infomationen durch die Verwendung eines Standards. Gleichwohl sollte abgewogen werden, ob die Einarbeitung in eine komplexe Ontologie lohnend ist. Denn in der Regel muss diese für den eigenen Anwendungsfall erweitert werden (vgl. [16]). Unter nicht-ontologischem Wissen werden Diagramme, Modelle und Datenbestände verstanden, die bisher nicht formal beschrieben wurden. Diese können bereits in digitaler Form vorliegen, oder müssen erzeugt werden. Zur Erzeugung können Werkzeuge der Wissensschöpfung, wie zum Beispiel "Brainstorming", "Umfragen", "Interviews", "Content Analysis" und "Focus Group" genutzt werden. Es empfiehlt sich mehrere dieser Techniken zu verwenden, um eine möglichst große Informationsbasis zu schaffen. Die Bewertung dieser Ansätze ist jedoch nicht Teil dieser Arbeit. Effizienz, Aufwände und die Informationsdichte der Techniken können variieren und sollten bei der Auswahl berücksichtigt werden (vgl. [19]).

Datenanalyse
Um die Extraktion von relevanten Daten zu erleichtern, müssen diese zunächst analysiert werden. Auf Grundlage der gesammelten Informationen können einfache Sätze gebildet werden, die die Realität widerspiegeln. Zum Beispiel "Softwarehersteller entwickeln Software", "Software benötigt Hardware". Die akquirierten Informationen werden zu einer Liste von kurzen prägnanten Sätzen überarbeitet, die in Kategorien zusammengefasst werden. Wörter ohne Aussagekraft werden eliminiert, so auch Verbindungswörter, Phrasen und umgangssprachliche Ausdrücke. Das Resultat sind logisch aufgebaute Sätze, bestehend aus Subjekt, Prädikat und Objekt. Wie in dem Beispiel mit der Zeichenkette "RAM" 4.1 gezeigt, ist die menschliche Sprache ein Regelsystem, in dem Daten zu Informationen umgewandelt werden können. Dieses Regelsystem definiert das Subjekt (Knoten), das mittels eines Prädikats (Kanten) ein Objekt (Knoten oder Attribute) verbindet.
Prädikate wie "X ist ein Y", "X ist eine Art von Y" oder "X ist vom Typ Y" indizieren, dass es sich um eine Subsumption handelt. Das Triple bestehend aus Subjekt, Prädikat und Objekt bilden zusammen einen einfachen Graphen (vgl. [19]).

Initialer Entwurf
Aus den gesammelten Informationen wird zunächst ein vorläufiges Lightweight-Modell mit Hilfe einer Visualisierung erstellt. Im nächsten Schritt wird dieses formalisiert und maschinenlesbar aufbereitet. Hierfür wird in dieser Arbeit Protégé (vgl. [13]), ein von der Stanford-Universität entwickeltes und häufig verwendetes Werkzeug im Bereich der Ontologiemodellierung eingesetzt. Die Software unterstützt die Abbildung von Klassen, Relationen und Instanzen, aber auch komplexe "Regelsysteme" und Abfragen. Der große Vorteil einer Software wie Protégé ist die Unabhängigkeit von der eingesetzten Modellierungssprache. Das garantiert die Flexibilität, Modelle als OWL, XML, JSON, RDF oder nach LaTeX zu exportieren (vgl. [19]).

Verfeinerung und Validierung
Nach der Übertragung des Lightweight-Modells in Protégé können weitere explizitere Beschreibungen hinzugefügt werden, wie zum Beispiel die Ergänzung fehlender mathematischer Regeln und logischer Zusammenhänge. Die in Protégé integrierte Inferenzmaschine, Debugging Umgebung und Abfragesprache ermöglichen die Validierung des Modells innerhalb von Protégé. Dabei werden die im ersten Schritt definierten Ziele (u.a. Competency Questions) mit den Ergebnissen verglichen. Können die Questions beantwortet werden, ist die Konzeption erfolgreich abgeschlossen. Ist dies nicht der Fall, müssen einzelne Schritte wiederholt werden (vgl. [16]).

5

ONTOLOGIE ZUR DARSTELLUNG VON SYSTEMANFORDERUNGEN

Im folgenden Abschnitt wird die entwickelte Ontologie vorgestellt. Zur Erstellung wurde die vorgestellte Modellierungsstrategie auf die Problemstellung der Systemanforderungen angewendet. Mit der Erstellung der Ontologie sollen die Probleme hinsichtlich der logischen Integrität gelöst werden. Der Entwicklungsprozess ist iterativ und durch Wiederholen der einzelnen Schritte wird die Ontologie korrekt. Im Folgenden werden ausschließlich die Ergebnisse des mehrfach durchlaufenen Prozesses vorgestellt, nicht der Prozess selbst.

5.1 ORGANISATION UND ZIELSETZUNG

Systemanforderungen sind Domänenwissen. Dieses ist spezifisch auf einen Anwendungsfall begrenzt. Deshalb ist das Ziel eine Domain-Ontologie, welche die Systemanforderungen abbilden kann. Im Zusammenhang der Systemanforderungen wurden folgende Competency-Questions identifiziert:

Competency Questions:

1. Aus welchen anderen Eigenprodukten setzt sich ein Eigenprodukt zusammen?
2. Welche andere Software muss darüber hinaus installiert sein und in welcher Version muss sie vorliegen? (Beispiele sind: Browser, Java, Betriebssystem)
3. Welchen Hardwareanforderungen muss das System mindestens genügen?
   3.1. RAM
   3.2. Festplattenspeicher
   3.3. CPU-Taktfrequenz

5.2 INFORMATIONSBESCHAFFUNG

Die Informationsbeschaffung wurde in drei Phasen durchgeführt. Zunächst wurde nach existierenden Ontologien gesucht, die komplett oder teilweise verwendet werden könnten. Die Recherche ergab, dass die "Enterprise Cloud and Datacenter Ontology" entwickelt wurde, um eine beliebige IT-Architektur innerhalb eines Unternehmens semantisch darzustellen (vgl. [9]). Diese Ontologie skizziert die Zusammenhänge von Servern, Netzwerken, virtuellen Maschinen und die auf den einzelnen Systemen installierte Software. Dies hat zwar keine Relevanz für die definierten Ziele der Competency-Questions oder der Zielsetzung der Arbeit. Gleichwohl könnte die Enterprise Cloud Ontologie mit der in der Arbeit entwickelten Ontologie kombiniert werden. Dies könnte die in der Motivation vorgestellte Idee, des automatischen Abgleichs der Systemanforderungen mit einer IT-Infrastruktur verwirklichen. Dadurch ließe sich eine Migration des Systems simulieren und eventuelle Komplikationen frühzeitig entdecken. Vor der Speicherung der Systemanforderungsdaten in dem jetzigen Datenbanksystem wurden Exceltabellen verwendet. Die Kernaussagen der Tabellen in Kombination mit dem vorgestellten ER-Diagramm könnten direkt als Grundlage für ein Lightweight-Modell verwendet werden. Beide sind an die Bedürfnisse des Unternehmens angepasst und realitätsfern. Aus diesem Grund werden bei der Erstellung der Ontologie die beiden Informationsquellen in einer abstrahierten Version als Orientierung verwendet. Dies soll sicherstellen, dass die spätere Ontologie mindestens die gleiche Menge an Informationen speichern kann. Fehlende Informationen im Prozess wurden durch Expertenbefragungen beantwortet. Alle gesammelten Informationen mit den verschiedenen Quellen wurden in einem Textdokument zusammengetragen und die verschiedenen linguistischen Besonderheiten markiert (Subjekt, Prädikat und Objekt). Die Ergebnisse der Akquise befinden sich unter anderem zusammengefasst im Anhang (siehe Anhang 7).

5.3 DATENANALYSE

Zur weiteren Analyse wurden alle markierten Wörter in ein Tabellenprogramm überführt. Dabei ist eine einheitliche Namenskonvention angewandt, und es werden, falls erforderlich, die Wörter aus dem Plural in den Singular umgewandelt. Dies vereinheitlicht das Modell und verringert die Gefahr von Missverständnissen. Jeder Satzbaustein ist mit den jeweiligen Entitäten vermerkt. Teile der gesammelten Informationen sind in diesem Schritt bereits herausgefiltert. Beispielsweise wurden die Beschreibungen zu Server, Cloud und Computer nicht in der entwickelten Ontologie modelliert, denn die vorhandenen Informationen dazu sind nicht vollständig und dieses Paradigma wird bereits ausführlich durch die entdeckte Enterprise-Ontologie beschrieben. Die Tabelle enthält Markierungen, wenn Informationen aus dem Textdokument nicht übertragen wurden mit entsprechender Begründung (siehe Anhang 7).

5.4 INITIALER ENTWURF

Aus den gefundenen Informationen wurde ein Lightweight-Modell in Form einer Mind-Map konstruiert (siehe Abb. 5.1).

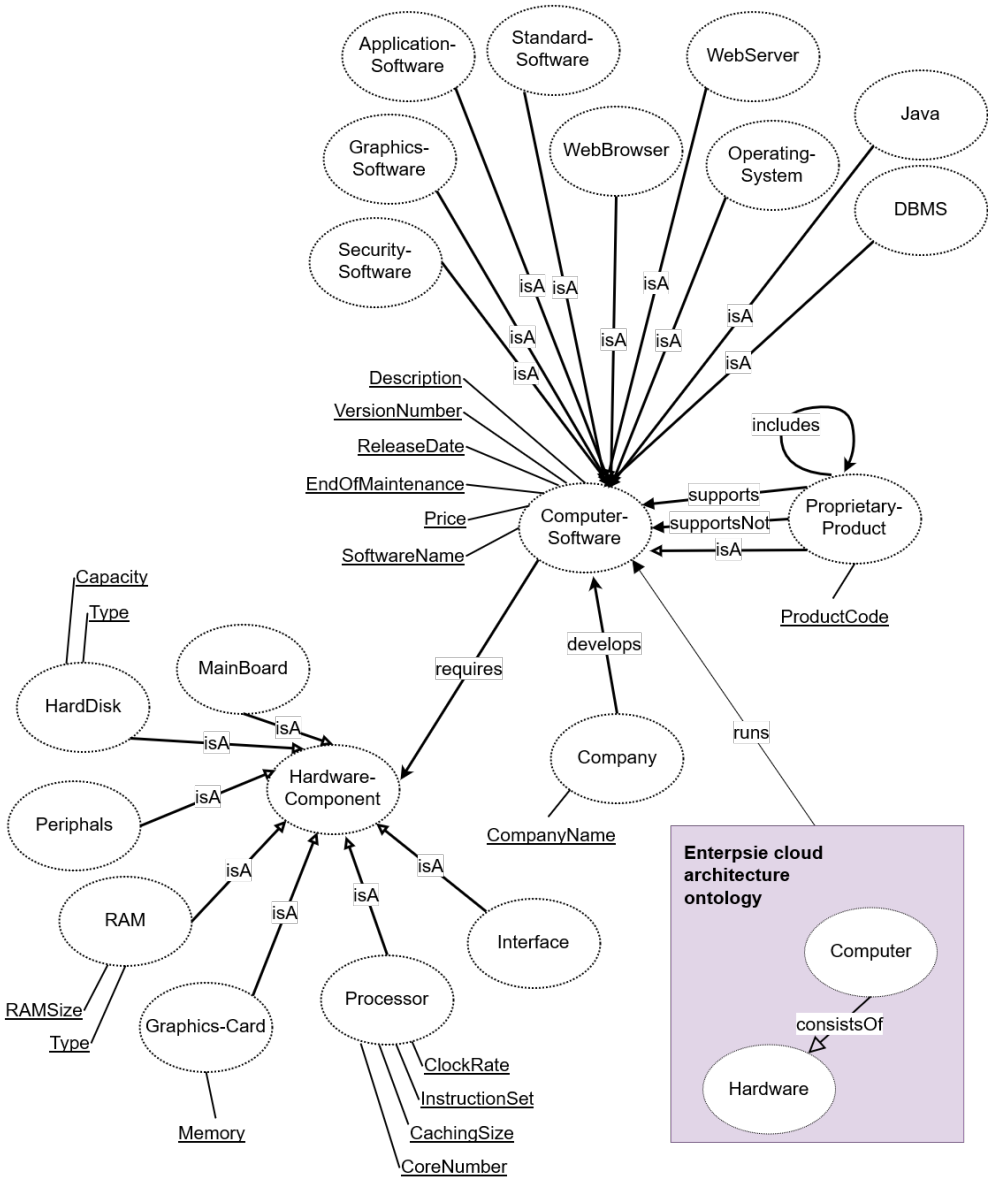


Abbildung 5.1: Lightweight-Ontologie über Systemanforderungen

Die Klasse “ComputerSoftware” beinhaltet sämtliche Computersoftware. Das bedeutet sowohl extern entwickelte Software, als auch die Produkte in der Eigenentwicklung. Die Beziehung “isA” beschreibt eine Subsumption, also Vererbung der Attribute von der Klasse “ComputerSoftware” an die jeweilige Kategorie. Die Klasse “ProprietaryProduct” (Eigenprodukt) ist eine spezielle Kategorie, die Produkte aus der Eigenentwicklung speichert. Diese enthält neben den vererbten Attribute der Oberklasse Computersoftware auch einen Identifier “ProductCode” zur Unterscheidung der Eigenprodukte. Die

Beziehung “includes” verweist von der Klasse “ProprietaryProduct” auf sich selbst. Dies ermöglicht in dem Modell das Inkludieren eines Produktes in einem Elternprodukt. Die Relationen “supports” und “supportsNot” beschreiben, ob ein Eigenprodukt eine Instanz der Klasse ComputerSoftware oder deren Unterkategorien unterstützt oder nicht unterstützt. Eine ComputerSoftware und demzufolge auch ein Eigenprodukt kann Hardware voraussetzen. Die Klasse “ComputerSoftware” ist über die Beziehung “requires” mit der Klasse “HardwareComponent” verbunden. Die “HardwareComponent” beinhaltet die wichtigsten Komponenten eines Computers. Die Klassen der Komponenten sind über eine “isA”-Beziehung mit der “HardwareComponent” verbunden. Das Paradigma, dass ein Unternehmen Software entwickelt, wurde mit der Klasse “Company” über die Beziehung “develops” mit der Klasse “ComputerSoftware” realisiert. Diese Mind-Map ist die Grundlage für die Modellierung in Protégé und damit auch für das darauf basierende heavyweight Modell.

5.5 VERFEINERUNG UND VALIDIERUNG

Das oben dargestellte Lightweight-Modell hat in dieser Darstellung jedoch noch nicht genügend “Wissen” für korrekte Schlussfolgerungen. In diesem Abschnitt wird das Modell deshalb um zusätzliche Aspekte erweitert, die für eine Mensch-zu-Mensch-Kommunikation weniger relevant sind. Aus diesem Grund wurden diese nicht im Lightweight-Modell visualisiert.

Da Ontologien auf Grundlage der Open-World-Annahme Entscheidungen treffen, ist es unabdingbar, falsche Interpretationsmöglichkeiten als inkonsistent auszuschließen. Das bedeutet, eine Instanz der Klasse Betriebssystem könnte in der OWA zeitgleich auch von der Klasse WebBrowser stammen. Dies ist möglich, da das bisherige Modell kein explizites Verbot vorsieht. Deshalb werden “unmögliche” Interpretationen mit einem “disjoint” (deutsch: disjunkt) verboten. Disjoints verhindern auch, dass die Inferenzmaschine solche verbotenen Zuordnungen schlussfolgert. An einer anderen Stelle in der Ontologie könnte dieses Verhalten jedoch erwünscht sein. Ein Beispiel hierfür wäre, dass ein Eigenprodukt gleichzeitig ein WebBrowser ist. Eine Liste bezüglich der disjointen Klassen befindet sich im Anhang (siehe Anhang 7).

Das Hinzufügen von inversen Beziehungen, ermöglicht den bis dato gerichteten Graphen direkt invers zu traversieren. Beispielsweise beschreibt die Relation “includes”, dass ein Eigenprodukt ein anderes Produkt inkludiert. Die Inverserelation, dieser Beziehung könnte zum Beispiel “includedBy” heißen und beschreibt in welchen anderen Produkten das Eigenprodukt inkludiert wurde. Wenn ein Produkt ein anderes enthält, muss es zwangsläufig auch dessen Kinder enthalten. Dieses Verhalten wird als transitiv bezeichnet und sowohl “includes” als auch “includedBy” müssen in Protégé als transitiv definiert werden. Dadurch ist die Inferenzmaschine in der Lage schlusszu-

folgern, welche weiteren Produkte von dem Produkt inkludiert werden müssen, ohne den Graphen zu traversieren. Durch die inverse Beziehung werden darüber hinaus alle Produkte festgestellt, in denen das Eigenprodukt inkludiert ist.

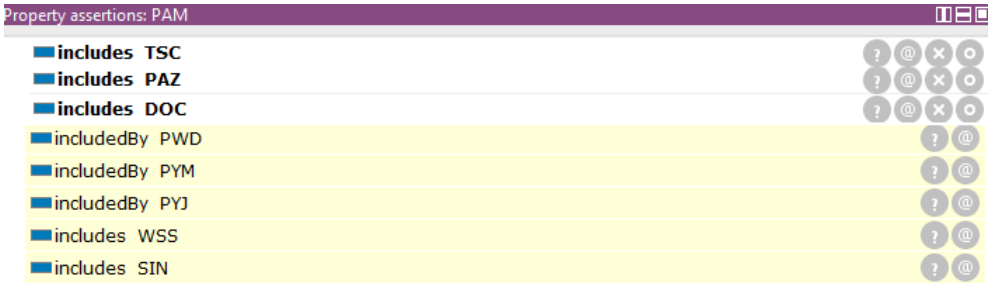


Abbildung 5.2: Screenshot aus Protégé (transitive Schlussfolgerung)

Die Abbildung 5.2 zeigt diese Inferenz am Beispiel von dem Eigenprodukt mit den Produktcode "PAM". Im Unternehmen haben sämtliche Eigenprodukte einen einzigartigen dreistelligen Produktcode. Die blauen Rechtecke zeigen, dass es sich um eine Relation handelt. Die Zeilen mit dem farbigen Hintergrund wurden über die transitive Beziehung der gespeicherten Datensätzen geschlussfolgert. "PWD", "PYM", "PYJ" inkludieren PAM und "WSS", "SIN" werden von PAM inkludiert.

Erlaubte Schlussfolgerungen in den Systemanforderungen

Zunächst wurde untersucht, welche Schlussfolgerungen innerhalb der Hierarchie der Systemanforderungen logisch korrekt, also erlaubt sind. Die Abbildung 5.3 zeigt eine mögliche Variante von Produkten (A-E) und den ermittelten Arbeitsspeicher sowie, welche Software unterstützt beziehungsweise erwiesenermaßen nicht unterstützt wird. Die Pfeile "Bottom Up" und "Top Down" helfen bei der Beschreibung der erlaubten Schlussfolgerungen. Generell können ausschließlich minimale Anforderungen berechnet werden.

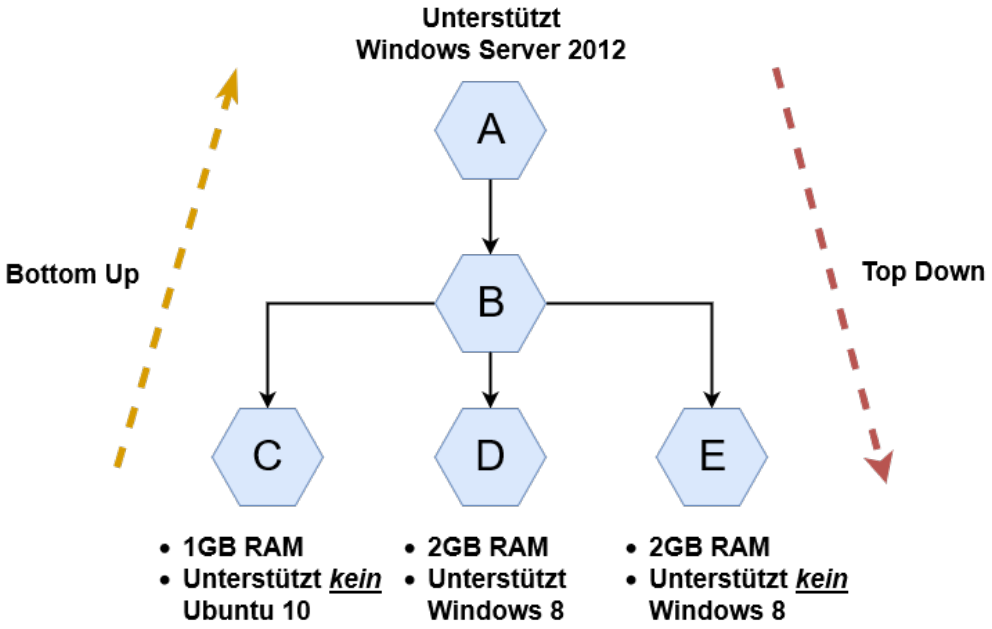


Abbildung 5.3: Beispiel für mögliche Schlussfolgerungen

Software-Schlussfolgerungen
In dem Beispiel unterstützt Produkt D das Betriebssystem Windows 8. Daraus lässt sich keine garantiert richtige Aussage über die Elternprodukte ableiten. Es ist zwar "wahrscheinlicher", dass Produkt B auch Windows 8 unterstützt, weil die Entwickler das Kindprodukt für diese Plattform zur Verfügung stellen, es ist aber nicht garantiert.

Produkt A unterstützt Windows Server 2012. Diese Aussage wiederum propagiert sich durch die Hierarchie (Top Down) und ist ausschließlich möglich, wenn die Aussage von **allen** Kindprodukten geteilt wird. Deshalb lässt sich schlussfolgern, dass Produkt B, C, D und E auch Windows Server 2012 unterstützen, auch wenn es nicht explizit in der Abbildung angegeben wurde. Wenn Kindprodukte das Betriebssystem nicht unterstützen, wäre dies ein logischer Widerspruch und der Datenbestand inkonsistent.

Durch diese Erkenntnis kann auch das Inverse geschlussfolgert werden. Wenn ein Kindprodukt eine Software **nicht** unterstützt, müssen alle Elternprodukte diese Software ebenfalls nicht unterstützen (Bottom Up). Deshalb dürfen die Produkte A und B sowohl "Ubuntu 10" als auch "Windows 8" nicht unterstützen. Ansonsten wäre dies wieder ein logischer Widerspruch. Beide Softwareschlussfolgerungen sind transitiv und vererben sich durch die gesamte Hierarchie.

Hardware-Schlussfolgerungen
Eine sichere Annahme am selben Beispiel ist, dass die Produkte A und B in jedem Fall mehr Arbeitsspeicher benötigen als jedes Kindprodukt einzeln betrachtet. Es kann geschlussfolgert werden, dass Produkt A und B mindestens 2GB RAM beanspruchen werden. Ein kleinerer Eintrag unter A oder B be-

deutet einen logischen Widerspruch. Computerprogramme können Hardwareressourcen "teilen". Ungeachtet davon müssen die Anforderungen an ein Produkt jederzeit größer als die maximalen Anforderungen unter den Kindprodukten sein. Die Annahme, dass der Arbeitsspeicher der Elternprodukte gleich oder größer dem summierten Arbeitsspeicher der Kindprodukte ist, ist nachvollziehbar aber nicht garantiert. Die Summe der Kindprodukte ist trotzdem ein Indiz für Inkonsistenzen und könnte berücksichtigt werden. Beispielsweise sind größere Abweichungen "unwahrscheinlich" und könnten als solche hinterfragt werden. Die Entscheidung, wie das System darauf reagiert, ist eine Designentscheidung. Die gleichen Annahmen zum Arbeitsspeicher treffen analog auch auf Hardwareeigenschaften wie CPU-Taktfrequenz, Anzahl der Prozessorkerne und Festplattenspeicher zu. Bei allen muss der Wert der Elternprodukte größer oder gleich dem Maximum unter den Kindprodukten sein.

Mögliche Schlussfolgerungen zusammengefasst:

1. Der Arbeitsspeicher, Fesplattenspeicher, Anzahl der Prozessorkerne und CPU-Taktfrequenz der Elternprodukte ist gleich oder größer als der maximale Wert unter den Kindprodukten

2. Unterstützte Software in den Elternprodukten wird auch von den Kindern unterstützt (Top Down)

3. Nicht unterstütze Software in den Kindprodukten wird nicht von den Eltern unterstützt (Bottom Up)

Evaluierung
Die "Evaluierung" hinsichtlich der Korrektheit der Ontologie wird im nächsten Abschnitt "Produktivsetzung in einer Graphdatenbank" durchgeführt. Die Überprüfung, ob die Ontologie in der Entwicklungsumgebung funktioniert, bedeutet nicht zwangsläufig, dass diese auch im Unternehmensumfeld lauffähig ist.

6 PRODUKTIVSETZUNG IN EINER GRAPHDATENBANK

6.1 UNZULÄNGLICHKEITEN VON PROTÉGÉ

Die bisher verwendete Modellierungsumgebung Protégé, ist optimiert für die Erstellung und Validierung von Ontologien. Protégé integriert Inferenzmaschinen und kann Datensätze speichern. Abseits dessen ist es für einen realen Betrieb im Unternehmen ungeeignet, da es keine Serveranwendung ist und für diesen Einsatz nicht entwickelt wurde. Für einen realen Betrieb muss eine Datenbank aufgesetzt werden, welche die erstellte Ontologie integriert.

6.2 VERGLEICH VON GRAPH- UND RELATIONALEN DATENBANKEN

Das bisher eingesetzte Datenbanksystem hat Probleme mit zyklischen und hierarchischen Abfragekonstrukten (vgl. Abschnitt 3.3). Die eingesetzte Traversierung des Graphen in einer Graphdatenbank verzichtet gänzlich auf Rekursion oder komplexe Abfragekonstrukte. In einer Graphdatenbank werden die Beziehungen nicht über Schlüssel implizit hergestellt, sondern physisch mit den Knoten gespeichert. Dies ermöglicht ein Verzicht von Joins und ist Voraussetzung für die effiziente Traversierung des Graphen, welche unter diesen Bedingungen performanter ist (vgl. [12]).
Die relationale Datenbank benötigt ein Datenbankschema zur Abspeicherung von Daten. Dieses Datenbankschema gewährleistet die Datenintegrität und dass Daten ausschließlich schemakonform gespeichert werden können (Closed-World-Assumption). Das Datenmodell wird in der Regel initial bei der Entwicklung der Datenbank und mit den bis dato erkannten Anforderungen modelliert. Mitunter bedingt durch die Aufteilung der Daten über verschiedene Tabellen, ist eine nachträgliche Änderung des Schemas ohne Beschädigung der gespeicherten Daten unkalkulierbar aufwändig.
Mit dem tabellenlosen Konzept einer Graphdatenbank kein Schema notwendig, da Daten in beliebiger Form hinzugefügt werden können (Open-World-Assumption). Die Verbindung durch Kanten mit anderen Knoten stellt letztlich die Relevanz des Datums dar. Durch das schemalose Konzept ist eine Graphdatenbank nach der Erstellung und Befüllung mit Datensätzen im höchsten Maß flexibel.
Triple-Stores sind Graphdatenbanken, welche zur Speicherung der Daten das RDF-Schema verwenden. Die Ontologiebeschreibungssprachen RDF-S und OWL verwenden zur Speicherung der Ontologie das RDF-Format. Das bedeutet einen nahtlosen Übergang zwischen der Speicherung der Daten und der Beschreibung in der Wissensrepräsentation.

6.3 RAHMENBEDINGUNGEN FÜR GRAPHDATENBANKEN

Im Unternehmen wird bereits eine NEO4J Graphdatenbank eingesetzt. Dabei handelt es sich um einen Labeled Property Graph (siehe Abschnitt 2.4.2). Untersuchungen im Unternehmen haben gezeigt, dass die Integration einer Ontologie ohne Weiteres nicht möglich ist. Zur Lösung dieses Problems, wird der Lösungsvorschlag von Jesús Barrasa herangezogen (vgl. [3]). Dieser hat anschaulich demonstriert, dass mittels seines Plugins ein Import der Ontologie möglich ist. Beim Umsetzen im Unternehmen konnte dies auch reproduziert werden. Indes konnten damit keine Integritätsprüfungen durchgeführt werden. Zur Lösung wurde stattdessen eine GraphDB-Instanz eingesetzt. Dabei handelt es sich um einen RDF-Triple-Store, konzipiert und optimiert für die Aufgabengebiete im Bereich der semantischen Technologien.

6.4 IMPORTIEREN DER DATENSÄTZE

Für die Migration der Daten aus einer relationalen Datenbank in GraphDB müssen die Daten vorher in die durch die Ontologie vorgegebenen Form überführt werden. Dies geschieht in der Regel über ein Mapping und erfordert eine spezielle Beschreibungssprache. Für eine automatische Migration der Daten müsste demzufolge ein Service, eine Anwendung oder sonstige technische Lösungen konzipiert werden, welche zwischen GraphDB und der relationalen Datenbank vermittelt. Zur Machbarkeitsuntersuchung wurden deshalb exemplarisch die Daten aus einem Softwarerelease verwendet. Dabei handelt es sich um circa zwei Prozent der vorhandenen Softwaredaten aus dem jetzigen System. Die Daten zu den Hardwareanforderungen existieren aktuell nicht in der Datenbank. Diese Daten liegen "kompiliert" in der jeweiligen Dokumentation der Produkte vor und lassen sich zum jetzigen Zeitpunkt nicht ohne Weiteres extrahieren. Deshalb wurden zum Testen der Funktionalität Pseudozufallswerte berechnet. Entscheidend sind nicht die verwendeten Werte, sondern dass deren Berechnung valide und korrekt ist. Das Mapping für den Release wurde mit dem "Cellfie"-Plugin innerhalb von Protégé durchgeführt. Dieses erfordert die Daten in Form einer Exceltabelle. Deshalb wurden die Daten aus der Datenbank als CSV exportiert und in Excel überführt. Das Plugin benötigt formulierte Regeln, um die Exceldaten den richtigen Ontologie-Instanzen zuzuordnen. Das Vorgehen wird beispielhaft an einem Ausschnitt der gespeicherten Betriebssysteme durchgeführt (siehe Tabelle 6.1).

	Betriebssysteme (A)	Version (B)	ID (C)
1	Windows Server	2012 R2	1Win
2	Suse Linux	12 SPx	1Linux
3	Solaris	11	1Solaris

Tabelle 6.1: Ausschnitt aus Betriebssystemtabelle

Tabellenprogramme benutzen als Spaltenbezeichner standardmäßig das Alphabet (A-Z). Die Spalte "ID" (C) ist ein Primary-Key und deshalb garantiert einzigartig. Eine Regel zum Hinzufügen der Datensätze kann wie folgt formuliert werden (siehe Beispiel 7.6):

Listing 6.1: Beispiel für eine Importregel

```
1 Individual: @C*
2 Types: OperatingSystem
3 Facts: SoftwareName @A*
4 Facts: VersionNumber @B*
```

Die zu selektierenden Spalten werden in der "Manchester-Schreibweise" mit einem "@"-Zeichen angeführt. Die erste Zeile beginnend mit "Individual" legt Instanzen mit einem Identifier aus der Spalte C an. Dadurch lässt sich in der Graphdatenbank später jede Instanz eindeutig adressieren. Das Asterisk (*) beschreibt die Verwendung aller vorhandenen Zeilen in der Tabelle. Das "Types" definiert die spätere Zuordnung zu der Klasse "OperatingSystem" aus der Ontologie (siehe Abschnitt 5.1). "Facts" beschreiben die Attribute der jeweiligen Instanzen. In diesem Fall wird der Inhalt der Spalte A als "SoftwareName" verwendet. Mit Protégé wird die Ontologie im RDF-Format exportiert. GraphDB unterstützt das Importieren der Daten als RDF-Tripel-Store nativ.

6.5 UMSETZUNG IN GRAPHDB

Zum späteren Vergleich der aus der Ontologie geschlussfolgerten Daten wurde eine Webanwendung implementiert. Diese Anwendung berechnet anhand von Algorithmen die vorher definierten Schlussfolgerungen. Unterschiedliche Werte zwischen der Anwendung und GraphDB bedeuten Unstimmigkeiten und müssen untersucht werden. In Protégé wurde die Beziehung "includes" als transitiv gekennzeichnet. Dabei handelt es sich um eine Standard-Inferenzregel, welche in OWL definiert ist. GraphDB unterstützt diese vordefinierten Regeln, aber bietet die Möglichkeit, eigene Regeln zu definieren. Im Abschnitt 5.5 wurden drei garantiert richtige Schlussfolgerungen vorgestellt, diese gilt es als Regeln zu definieren.

Listing 6.2: Regel zum Unterstützen von Software (Top-Down)

```
Id: asupportsb

  a <sysr:includes> b
  a <sysr:supports> c
-------------------------------

  b <sysr:supports> c
```

Die vorangestellte Regel (siehe Beispiel 6.2) besagt, dass wenn ein Produkt "a" das Produkt "b" inkludiert und gleichzeitig das Produkt "a" die Software "c" unterstützt. So muss das Produkt "b" ebenso Software "c" unterstützen. Im Anhang befindet sich das Regelsystem mit allen Regeln zum Schlussfolgern der im Abschnitt 5.5 entwickelten Aussagen. Die Abbildung 6.1 zeigt ein mögliches Produkt ohne die Inferenzmaschine. Alle Produkte unterstützen jeweils eine Software und unterstützen gleichzeitig explizit nicht eine Software aus derselben Kategorie. Das hilft die verschiedenen Inferenzen auseinanderzuhalten. Besonderes Augenmerkt gilt Produkt "P1" und "P2", welche einen Zyklus bilden.

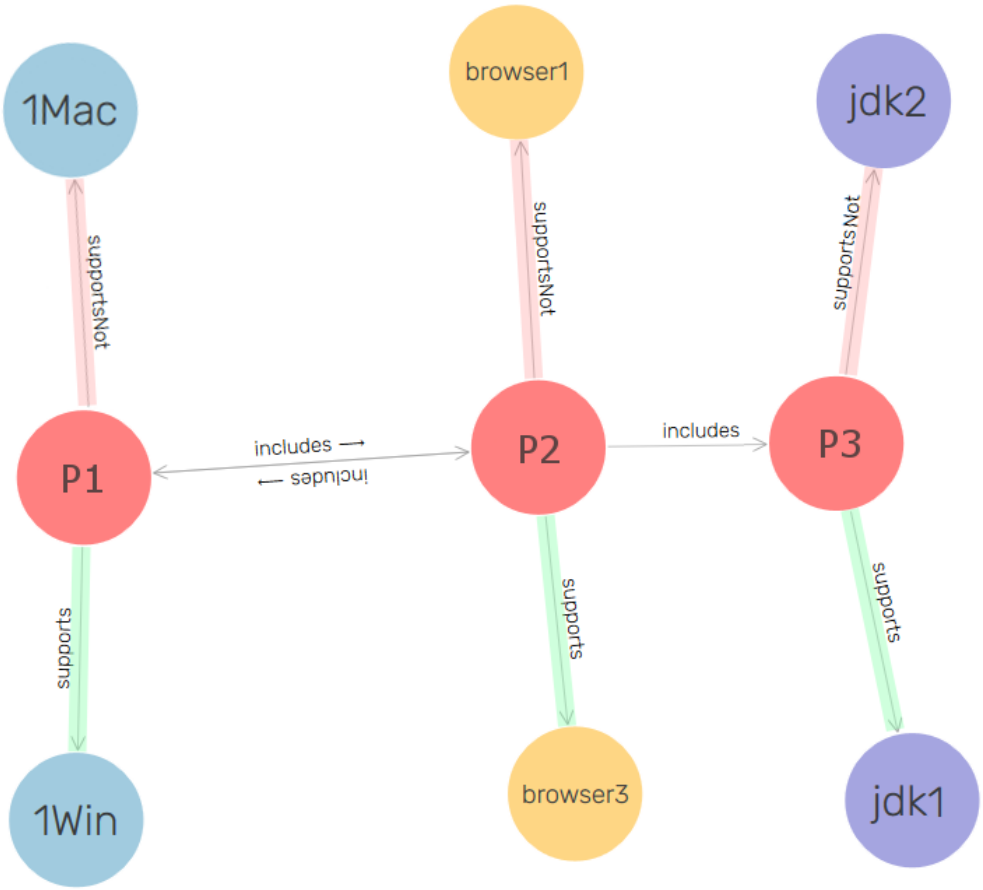


Abbildung 6.1: Screenshot aus GraphDB - Testprodukte

Zur Orientierung wurde die “supportsNot” rot und “supports” Beziehung grün gefärbt. Die Abbildung 6.2 zeigt die gleichen Datensätze mit der Inferenzmaschine. Wie gewünscht propagiert sich die Beziehung “supportsNot” von unten nach oben (Bottom-Up) und die Beziehung “supports” von oben nach unten (Top-Down). Deshalb wird “jdk1” ausschließlich von Produkt “P3” unterstützt. Auffällig ist, dass die zyklisch behafteten Produkte “P1” und “P2” jeweils die gleiche Software unterstützen beziehungsweise nicht unterstützen. Dies ist zu erklären, da beide zusammen eine Top-Down- und Bottom-Up-Hierarchie bilden.

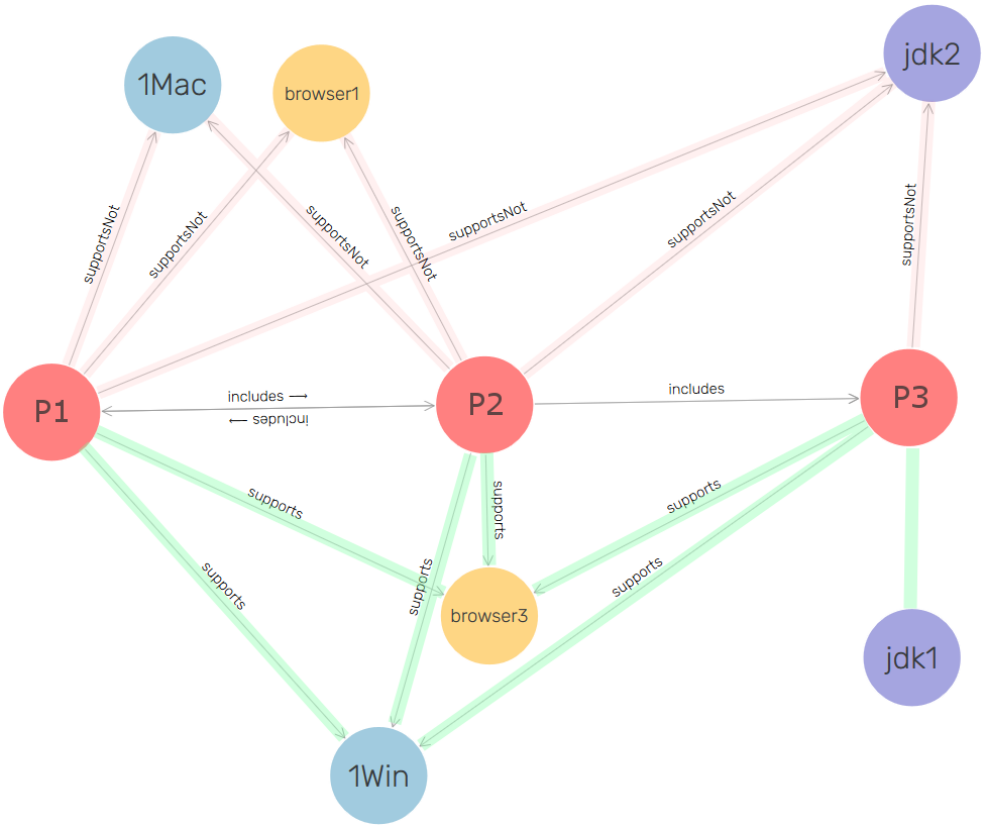


Abbildung 6.2: Screenshot aus GraphDB - Testprodukte mit Inferenzmaschine

Die bisherigen Regeln ermöglichen das Schlussfolgern von neuen Datensätzen. In dem Regelsystem müssen darüber hinaus logische Konsistenzprüfungen erstellt werden. Zum Beispiel ist es mit den bisherigen Bezeichnungen möglich, dass ein Produkt eine Software sowohl unterstützt, als auch nicht unterstützt (Open-World). Dieses Verhalten lässt sich wie folgt verbieten (siehe Beispiel 6.3).

Listing 6.3: Beispiel für Konsistenzprüfung

```
Consistency: supports_supportsNot

  a <sysr:supports> b
  a <sysr:supportsNot> b
```

Das "Consistency" Keyword bedeutet, dass wenn die beschriebene Regel zutrifft, das Modell **nicht** konsistent ist.
GraphDB unterstützt keine Regeln, die das Vergleichen von einzelnen RDF-Werten zulassen. Dies hat zur Folge, dass **keine** arithmetischen Operationen wie "Addition", "Subtraktion", "größer als" oder "kleiner als" als Regel definiert werden können. Die Hardwareschlussfolgerungen erfordern den Vergleich von Werten der Eltern- und Kindprodukten über Arithmetische Operatoren. Für die Schlussfolgerung der Hardwareanforderungen muss folglich eine Abfragesprache wie "**S**PARQL-**P**rotocol-**a**nd-**R**DF-**Q**uery-**L**anguage" (SPARQL) verwendet werden. In den Datensätzen wurden bewusst drei Inkonsistenzen eingebaut, um die Abfragen zu testen. Neben den Abfragen zu den Hardwareanforderungen wurden im Anhang (siehe Anhang 7) auch Abfragen entwickelt, die sämtliche formulierten Regeln simulieren. Das ermöglicht auch ohne Inferenzmaschine das Ermitteln der Werte, aber ohne Persistierung. Zwischen den Ergebnissen der Abfragen und der Webanwendung sowie den Ergebnissen der Inferenzmaschine wurden keine Unterschiede festgestellt. In den SPARQL-Abfragen wurden Bewertungen eingebaut, um signifikante Unterschiede festzustellen. Zum Beispiel, wenn das Elternprodukt weniger RAM benötigt als die Summe der Kindprodukte, ist dies zwar möglich, aber unwahrscheinlich (siehe Abschnitt 5.5). Wenn das Elternprodukt weniger Arbeitsspeicher benötigt als jedes Kind einzeln betrachtet, bedeutet dies eine Inkonsistenz. Das Abbild 6.3 zeigt ein Beispiel für eine SPARQL-Abfrage die den benötigten Arbeitsspeicher auswertet. Das Maximum unter den Kindsprodukten wird als "MINRAM" angegeben. Der Arbeitsspeicher der Elternprodukte muss größer sein als MINRAM ansonsten wäre dies eine logische Inkonsistenz. Die Summe unter den Kindsprodukten bildet die Spalte "SUMRAM". Wenn das Elternprodukt weniger Arbeitsspeicher benötigt als die Summe der Kinder, ist dies zu hinterfragen.
Die Ontologie beantwortet damit alle Competency Questions (siehe Abschnitt 5.1). Die erste Frage, aus welchen anderen Produkten sich ein Produkt zusammensetzt, ist mittels der transitiven Beziehung "includes" gelöst. Die zweite Question, welche andere Software darüber hinaus installiert sein muss, ist mit den definierten Regeln auf Konsistenzebene sowie auf Schlussfolge-

	product	RAM	MINRAM	SUMRAM	result
1	:EVS	"89548"	"18850"	"28275"	RAM is OK
2	:MSC	"89548"	"84823"	"113098"	RAM is unlikely
3	:PAM	"100"	"89548"	"212070"	Inconsistency!
4	:PZI	"89548"	"65973"	"122622"	RAM is unlikely

Abbildung 6.3: SPARQL-Abfrage über benötigten Arbeitsspeicher

rungsebene definiert. Die letzte Frage, welchen Hardwareanforderungen das System mindestens genügen muss, ist mit SPARQL-Abfragen durchgeführt worden. Zusammengefasst ist festzuhalten, dass die Ontologie die Struktur bietet, die Systemanforderungen abzuspeichern und diverse Schlussfolgerungen automatisch zu ermitteln.

7 FAZIT

In dieser Arbeit wurde untersucht, inwieweit folgende Probleme im Zusammenhang mit den Systemanforderungen gelöst werden können:

1. Eingeschränkte Anpassbarkeit des Systems
2. Möglichkeit Unvorhersehbarer Abfragekonstrukte
3. Keine Überprüfung von logischer Integrität
4. Keine Schlussfolgerung von neuen Datensätzen

Dabei hat sich herausgestellt, dass die Probleme hinsichtlich der Anpassbarkeit der relationalen Datenbank hauptsächlich dem Datenbankschema geschuldet sind. Dies konnte mit der Überführung in eine Graphdatenbank gelöst werden, da diese kein Schema besitzen und die Daten über die Ontologie gesteuert werden. Die Graphdatenbank hat auch die Probleme hinsichtlich der komplexen und rekursiven Abfragen gelöst, indem es für Beziehungen und deren effizienten Traversierung optimiert ist.
Die logischen Zusammenhänge und das Abspeichern des Wissens über die Systemanforderungen sind zentrale Aspekte dieser Arbeit. Dabei wurde unterschieden zwischen logischen Konsistenzprüfungen und dem Ermitteln von neuen Datensätzen durch Schlussfolgerungen. Die entwickelte Ontologie ist in der Lage, die vorhandenen Daten der Systemanforderungen abzuspeichern und die entwickelten Inferenzregeln können Datensätze automatisch schlussfolgern. Aus den ca. 3.000 explizit angebenen Datensätzen wurden ca. 14.000 geschlussfolgert (siehe Abb. 7.1).

Abbildung 7.1: Screenshot aus GraphDB - Expansionsrate der Datenbank

Die Abbildung 7.2 zeigt das Produkt "AIC" vor und nach der Schlussfolgerung von neuen Datensätzen. Dabei ist auffällig, dass vor den Inferenzen **keine** Datensätze bekannt waren. Mittels den in der Arbeit entwickelten Schlussfolgerungen konnten diverse Anforderungen ermittelt werden.
Der Name des Produkts "AIC" lautet "Software AG Installer Always Installed Components" und unterstützt alle angebotenen Plattformen, da dieses in jedem Produkt vorhanden ist. Das bisherige Datenbanksystem würde bei

einer Abfrage zu dem Produkt AIC kein Ergebnis zurückliefern, da keine Datensätze dazu gespeichert sind. Neben dem Schlussfolgern neuer Datensätze ist die Integritätsprüfung ein wichtiges Kriterium. In diesem Zusammenhang wurden Regeln entwickelt, die das Eintragen von logisch inkonsistenten Datensätzen verhindern. In der Arbeit konnten alle entwickelten Schlussfolgerungen mit der Inferenzmaschine gelöst werden. Die einzige Ausnahme bilden die Hardwareanforderungen, da diese arithmetische Operationen erfordern, die so mit der Inferenzmaschine nicht umzusetzen sind. Zur Lösung wurden deshalb Abfragen entwickelt, die die fehlenden Schlussfolgerungen simulieren beziehungsweise logische Inkonsistenzen aufdecken.

AUSBLICK

Das System wurde mit einem Bruchteil der Datensätze getestet, wodurch die Antwortzeiten entsprechend performant waren. Interessant wäre, zu untersuchen, welche Auswirkungen eine Vergrößerung der Datenbasis auf die Antwortzeit hat. Vor allem die Verwendung der Inferenzmaschine und das automatische Schlussfolgern könnten die Anwendbarkeit im Unternehmen negativ beeinflussen. Gegebenenfalls müssten die Abfragen sowie das Regelsystem optimiert werden.
Weitere Forschungsmöglichkeit bietet die Entwicklung einer Verbindungsontologie zur "Enterprise Cloud Architecture Ontology". Dies könnte mit einer Inferenzmaschine und einem IT-Planungssystem die Migration einer kompletten IT-Landschaft automatisch simulieren. Dadurch könnten die Risiken eines Systemausfalls durch eine Migration gesenkt werden.
Ein weiterer Aspekt könnte die Anbindung des Systems mit dem "Unternehmens-Installer" sein. Dieser wird verwendet, um neue Unternehmensprodukte zu installieren und könnte vor der Installation eine Systemanalyse durchführen. Dies gewährleistet, dass das Zielsystem in der Lage ist, die Anwendung ordnungsgemäß auszuführen. Damit könnte das System die Arbeit der Kollegen beim Kunden unterstützen, welcher aktuell den Abgleich manuell durchführt und darüber hinaus die unvollständigen Daten aus dem bisherigen System verwendet.

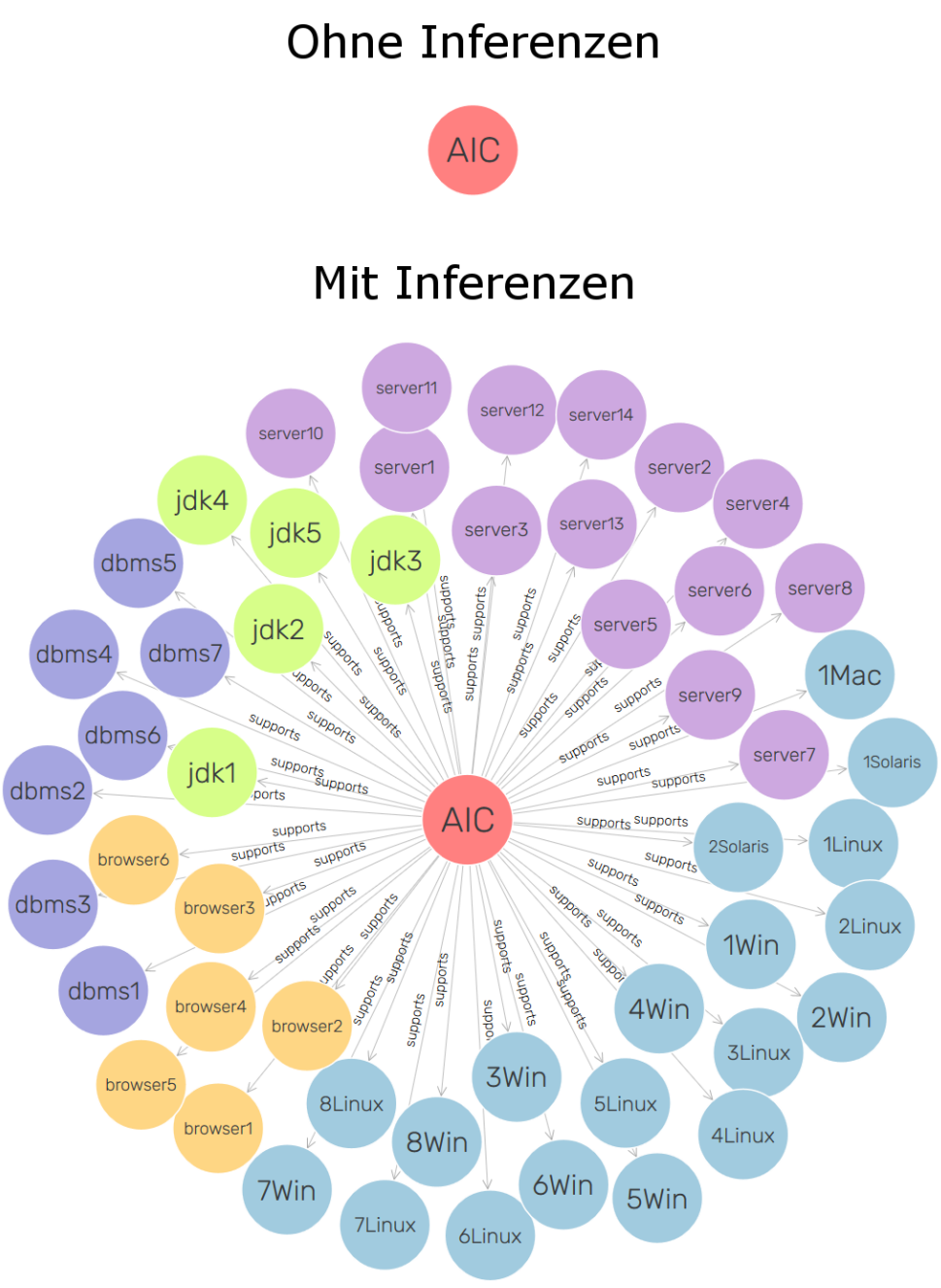


Abbildung 7.2: Screenshot aus GraphDB - Produkt AIC

LITERATUR

[1] Renzo Angles und Claudio Gutierrez. "Survey of graph database models". In: *ACM Computing Surveys* 1 (2008), S. 1–39.

[2] Patrick Arnold. "Information und Wissen". In: *Universität Leipzig* (2009). URL: http://www.informatik.uni-leipzig.de/~graebe/Texte/Arnold-09.pdf.

[3] Jesús Barrasa. *Building a semantic graph in Neo4j*. 2016. URL: https://jbarrasa.com/2016/04/06/building-a-semantic-graph-in-neo4j/.

[4] Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlüter, Bernhard Seiler, Erwin Tegtmeier und Thomas Zeh. "Was bedeutet eigentlich Ontologie?" In: *Informatik-Spektrum* 4 (2014), S. 286–297.

[5] Thomas H. Davenport und Laurence Prusak. *Working knowledge: How organizations manage what they know*. [Nachdr.] Boston, Mass.: Harvard Business School Press, 2010.

[6] Ramez Elmasri und Sham Navathe. *Database systems: Models, languages, design, and application programming*. 6th ed. Boston: Addison-Wesley, 2011.

[7] Frank Geisler. *Datenbanken: Grundlagen und Design*. Verlagsgruppe Hüthig Jehle Rehm, 2014.

[8] Thomas R. Gruber. "A translation approach to portable ontology specifications". In: *Knowledge Acquisition* 2 (1993), S. 199–220.

[9] Peter Haase, Tobias Mathäß, Michael Schmidt, Andreas Eberhart und Ulrich Walther. "Semantic Technologies for Enterprise Cloud Management". In: *The Semantic Web – ISWC 2010*. Hrsg. von Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks und Birte Glimm. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 98–113.

[10] Matthias Haun. *Handbuch Wissensmanagement: Grundlagen und Umsetzung, Systeme und Praxisbeispiele*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[11] Steffen Jung. "Erarbeitung eines wissensbasierten Systems". In: (2002), S. 12–17. URL: https://sundoc.bibliothek.uni-halle.de/diss-online/02/02H162/prom.pdf.

[12] Klöckner Kevin. "Im Vergleich NoSQL vs relationale Datenbanken". In: *Universität Siegen* (2009).

[13] Mark A. Musen. "The Protégé Project: A Look Back and a Look Forward". In: *AI matters* 4 (2015), S. 4–12.

[14] Natalya F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. URL: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

[15] Klaus North, Andreas Brandner und MScThomas Steininger, Hrsg. *Wissensmanagement für Qualitätsmanager*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016.

[16] Heiner Stuckenschmidt. *Ontologien: Konzepte, Technologien und Anwendungen*. Berlin und Heidelberg: Springer, 2009.

[17] Rudi Studer, V.Richard Benjamins und Dieter Fensel. "Knowledge engineering: Principles and methods". In: *Data & Knowledge Engineering* 1-2 (1998), S. 161–197.

[18] TSB. *We're getting back on track and remain committed to putting things right*. Hrsg. von TSB. 2018. URL: https://www.tsb.co.uk/news-releases/half-year-2018-results/.

[19] Tish, Chungoora. *Practical Knowledge Modelling*. URL: https://www.udemy.com/practical-knowledge-modelling/learn/v4/overview.

ANHANG

VERWENDETE INFORMATIONSBASIS

Information acquisition

Legend:

- Noun, Object
- *Predicate*
- ✓ Transfered to Excel

General

- **Computer** *consist of* **hardware components** ✓
- **Computer** *run* **computer software** ✓
- **Computer software** may contain **documentation** ✓
- **Company** *develops* **computer software** ✓
- **Companys** *have a* ***name*** ✓

Software

- **Computer software** *requires* **hardware components** ✓
- **Computer software** *requires* other **computer software** ✓
- **Computer software** *is outdated at* **End-of-Maintanance** date ✓
- **Computer software** will *be released on* **release date** ✓
- **Computer software** *has a* **price** ✓
- **Computer software** *has a* **description** ✓
- **Computer software** *has a* **version number** ✓
- **Computer software** *is divided* into different **software categories** ✓
- **Computer software** *has a* **name** ✓

Hardware

Processor *is a* **hardware component** ✓
  - **Instruction set** *is a characteristic of* **Processor** ✓
  - **Clock rate** *is a characteristic of* **Processor** ✓
  - **Caching size** *is a characteristic of* **Processor** ✓
  - **Core number** *is a characteristic of* **Processor** ✓

RAM *is a* **hardware component** ✓
  - **RAM types** *is characteristic of* **RAM** ✓
  - **RAM size** *is a characteristic of* **RAM** ✓

Hard disk *is a* **hardware component** ✓
  - **Hard disk capacity** *is characteristic of* *an* **hard disk** ✓
  - **SSD** *is a* **hard disk type** ✓
  - **HDD** *is a* **hard disk type** ✓

Mainboard *is a* **hardware component** ✓

Graphics card *is a* **hardware component** ✓
  - **Graphics card memory** *is characteristic of* *a* **graphics card** ✓

Interface *is an* **hardware component** ✓

VERWENDETE INFORMATIONSBASIS

Peripherals *are* **hardware components**

Proprietary products – properiatry products

- **Proprietary product** *is a* **computer software**
- **Proprietary product** *have a* **product code**
- **Proprietary product** *includes* **proprietary product**

Software categories

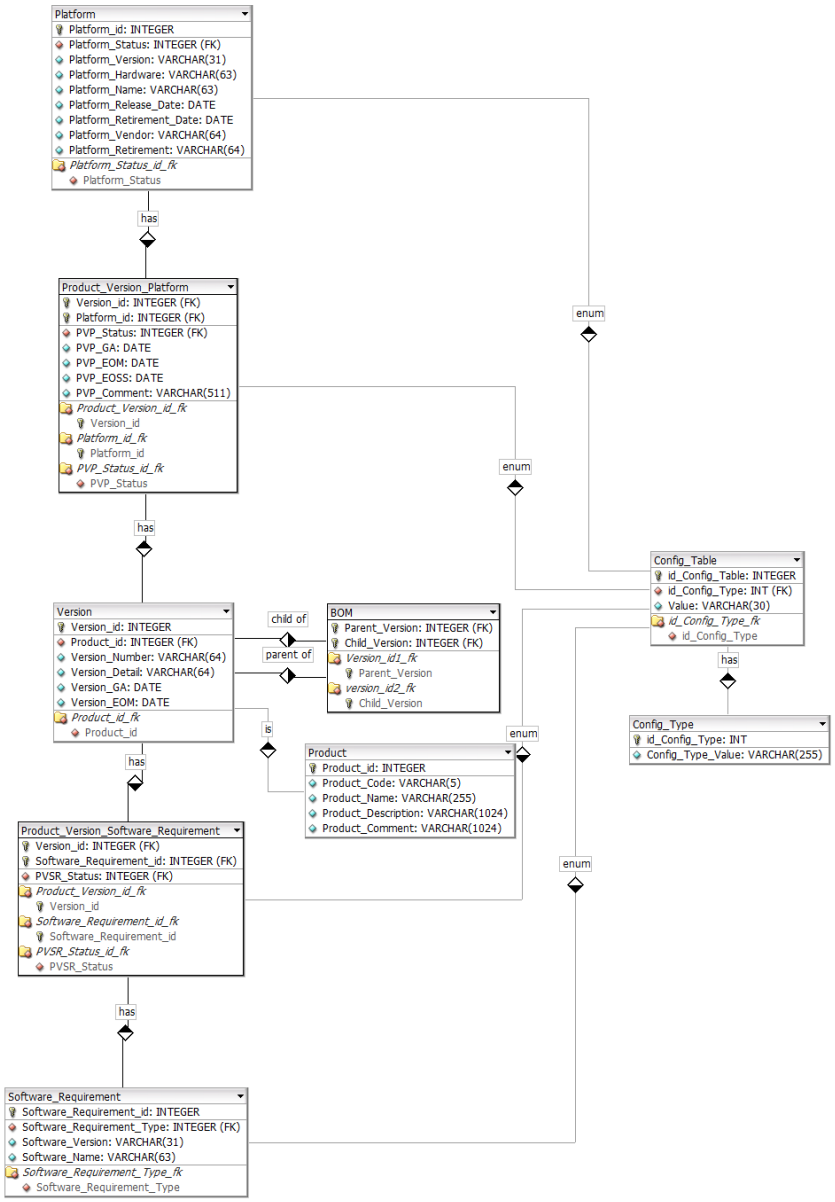
- **Security software** *is a* **software category**
- **Graphics software** *is a* **software category**
- **Application software** *is a* **software category**
- **Standard software** *is a* **software category**
- **Web browser** *is a* **software category**
- **Web server** *is a* **software category**
- **Operating system** *is a* **software category**
- **Java** *is a* **software category**
- **DBMS** *is a* **software category**

Computer

- **PC** *is a* **computer**
- **Server** *is a* **computer**
- **Laptop** *is a* **computer**
- **Game console** *is a* **computer**
- **Embedded system** *is a* **computer**
- **Smartphone** *is a* **computer**
- **Super computer** *is a* **computer**
- **Mainframe** *is a* **computer**

VERWENDETE INFORMATIONSBASIS

Current System with ER-Diagram



Not Captured Information from ER-Diagram:

Operating Systems: **Retirement date, Vendor, Release date**
Properiatry products: **product code**

AUSZUG MAPPING TABELLE

Term	Source	Suggested Type	In Diagram	Diagram Commentar	Protege Type	In Protege
HardwareComponent		Class	yes		Class	yes
ComputerSoftware	ER-Diagram	Class	yes		Class	yes
Documentation		Not Relevant		Not relevant for competency-qu	Not Included in Protege	-
SoftwareCategory		Class	yes		Class	yes
Processor		Class	yes		Class	yes
RAM		Class	yes		Class	yes
HardDisk		Class	yes		Class	yes
Mainboard		Class	yes		Class	yes
GraphicsCard		Class	yes		Class	yes
Interface		Class	yes		Class	yes
Peripherals		Class	yes		Class	yes
ProperietaryProduct	ER-Diagram	Class	yes		Class	yes
SercuritySoftware		Class	yes		Class	yes
GraphicsSoftware		Class	yes		Class	yes
ApplicationSoftware		Class	yes		Class	yes
StandardSoftware		Class	yes		Class	yes
WebBrowser	ER-Diagram	Class	yes		Class	yes
WebServer	ER-Diagram	Class	yes		Class	yes
OperatingSystem	ER-Diagram	Class	yes		Class	yes
Java	ER-Diagram	Class	yes		Class	yes
Company		Class	yes		Class	yes
Computer		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
consistsOf		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
run		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
PC		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
Server		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
Computer		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
Laptop		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
GameConsole		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
EmbeddedSystem		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
Smartphone		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
SuperComputer		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
Mainframe		~~Part Of ECO~~		Is part of an existing Ontology	Not Included in Protege	-
EndOfMaintanance	ER-Diagram	Property	yes		Data-Property	yes
ReleaseDate	ER-Diagram	Property	yes		Data-Property	yes
Price		Property	yes		Data-Property	yes
Description	ER-Diagram	Property	yes		Data-Property	yes
VersionNumber	ER-Diagram	Property	yes		Data-Property	yes
InstructionSet		Property	yes		Data-Property	yes
ClockRate		Property	yes		Data-Property	yes
CachingSize		Property	yes		Data-Property	yes
CoreNumber		Property	yes		Data-Property	yes
RAMType		Property	yes		Data-Property	yes
RAMSize		Property	yes		Data-Property	yes
Capacity		Property	yes		Data-Property	yes
Type		Property	yes		Data-Property	yes
Memory		Property	yes		Data-Property	yes

SÄMTLICHE DISJUNKTE KLASSEN

In Abbildung gibt die erste Zeile die Elternklassen an. Jeweils die in der Spalte vorhanden Klassen sind zueinander disjunkt. Am Beispiel der ersten Spalte "-" sind die Klassen "HardwareComponent", "Company", "ComputerSoftware" zueinander Disjunkt. Diese haben keine Elternklassen.

-	HardwareComponent	ComputerSoftware
HardwareComponent	GraphicsCard	ApplicationSoftware
Company	HardDisk	DBMS-Software
ComputerSoftware	Interface	GraphicsSoftware
	Mainboard	Java
	Periphals	OperatingSystem
	Processor	SecuritySoftware
	RAM	StandardSoftware
		WebBrowser
		WebServer

Tabelle 7.1: Spaltenweise: die miteinander disjunkten Klassen

INFERENZREGELN

Listing 7.1: Auszug aus Inferenzregeln

```
Id: asupportsb

  a <sysr:includes> b
  a <sysr:supports> c
-------------------------------

  b <sysr:supports> c

Id: asupportsNotb

  a <sysr:includedBy> b
  a <sysr:supportsNot> c
-------------------------------

  b <sysr:supportsNot> c


Consistency: supports_supportsNot

  a <sysr:supports> b
  a <sysr:supportsNot> b
-------------------------------


-----------------------------------
```

Listing 7.2: Berechnung von Arbeitsspeicher

```
PREFIX this: <http://bachelor.chrismey.com/light-weight.owl#>
SELECT
  ?product
  ?RAM (MAX(?ramsize) AS ?MINRAM)
  (SUM(?ramsize) AS ?SUMRAM)
  (IF(?RAM < ?MINRAM, "Inconsistency!", IF(((?RAM > ?MINRAM) &&
      (?RAM < ?SUMRAM)) && (?SUMRAM != 0), "RAM is unlikely", "
      RAM is OK")) AS ?result)
WHERE {
  ?product (this:includes*) ?childProduct.
  ?childProduct this:requires ?c2.
  ?c2 this:RAMSize ?ramsize.
  ?product this:requires ?c4.
  ?c4 this:RAMSize ?RAM.
  FILTER(?ramsize != ?RAM)
}
GROUP BY ?RAM ?product
```

Listing 7.3: Berechnung von Kindprodukten

```
PREFIX this: <http://bachelor.chrismey.com/light-weight.owl#>
SELECT * WHERE {
  ?products ?b this:ProprietaryProduct .
  ?products this:includes* ?childrenProduct .
}
```

Listing 7.4: Berechnung von Elternprodukten

```
PREFIX this: <http://bachelor.chrismey.com/light-weight.owl#>
SELECT * WHERE {
  ?products ?b this:ProprietaryProduct .
  ?products this:includedBy* ?parentProduct .
}
```

Listing 7.5: Berechnung von nicht-unterstützter Software der Kindprodukte (Bottom-Up)

```
PREFIX this: <http://bachelor.chrismey.com/light-weight.owl#>
SELECT ?products ?productSupports ?childrenSupportsNot
(IF(?productSupports = ?childrenSupportsNot, "Inconsistency with
    children", "OK") AS ?result) WHERE {
  ?products ?b this:ProprietaryProduct .
  ?products this:supports ?productSupports.
  ?products this:includes* ?childrenProduct .
  ?childrenProduct this:supportsNot* ?childrenSupportsNot .
}
```

Listing 7.6: Berechnung von Softwareanforderungen

```
PREFIX this: <http://bachelor.chrismey.com/light-weight.owl#>
SELECT
  ?products
  ?productSupports
  ?productSupportsNot
  ?parentSupports
  ?childrenSupportsNot
  (IF(?productSupports = ?childrenSupportsNot, "Inconsistency w
      children",
  IF(?productSupportsNot = ?parentSupports, "Inconsistency w Parent
      ",
  IF(?parentSupports = ?childrenSupportsNot, "Inconsistency between
      p and c", "OK"))) AS ?result)
WHERE {
  ?products ?b this:ProprietaryProduct .
  ?products this:supportsNot ?productSupportsNot.
  ?products this:supports ?productSupports.
  ?products this:includedBy* ?parentProduct .
  ?parentProduct this:supports ?parentSupports .
  ?parentProduct this:supportsNot ?parentSupportsNot .
  ?products this:includes* ?childrenProduct .
  ?childrenProduct this:supports ?childrenSupports .
  ?childrenProduct this:supportsNot ?childrenSupportsNot .
}
```