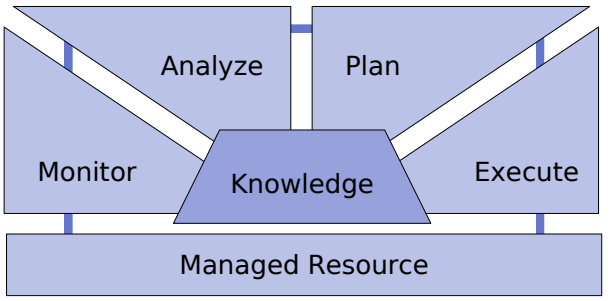
Seminar: Autonomic Computing

Wissensrepräsentation

Kristof Hamann

Vorzutragen am 27.11.2007 (Wintersemester 2007/08).

Einordnung



[IBM05]

Motivation für das Thema „Wissensrepräsentation“ im Bereich „Autonomic Computing“:

In allen Phasen im MAPE-Modell muss auf eine gemeinsame Komponente zurückgegriffen werden können, welche Informationen/Wissen über das zu überwachende System vorhält.

In diesem Vortrag werden zwei Ansätze vorgestellt, die hierbei nützlich sein können: Ontologien und Entscheidungsbäume.

Begriffsklärung

- Daten
  - Nachricht ohne Semantik
- Information
  - Bedeutung einer Nachricht
  - „verstehen“ was gemeint ist
- Wissen
  - Anwenden von bekannten Informationen
  - Fähigkeit, durch Schlussfolgern aus dem Bekannten neues Wissen abzuleiten

Zunächst eine Begriffsklärung, um diese drei häufig verwendeten Begriffe differenzieren zu können. Dies sollte jedem eigentlich aus dem Grundstudium bekannt sein.

Teil I: Ontologien

- Einführung
- Semantic Web
- Beispiel „eAutomation“

Wissen formalisieren

- Konzeptualisierung
  - abstrakte, vereinfachte Sichtweise
  - für einen bestimmten Zweck
  - oft nur implizit
- Gegenstand:
  - Objekte, Konzepte
  - Beziehungen miteinander

Jedes Wissen verarbeitende System konzeptualisiert. Damit sind insbesondere auch Menschen gemeint. Wenn wir beispielsweise von Vögeln reden, so haben wir ein gewisses Konzept im Kopf, nämlich dass dies Tiere sind, die fliegen können.

Quelle: [Gru93]

Beispiel (Zweckbindung):

Für den Statiker ist es wichtig, wieviele Stockwerke ein Haus hat, für den Innenarchitekten ist aber interessanter, welche Farbe die Wände haben.

Ontologie

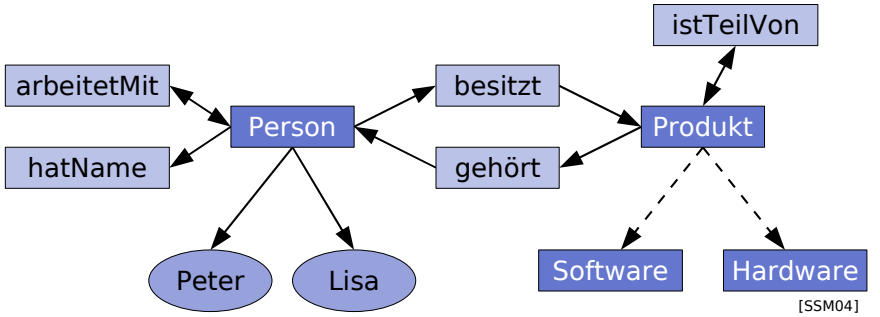
- explizite Spezifikation einer Konzeptualisierung
- besteht aus Termen (Vokabular), welche die Diskurs-Welt wiederspiegeln
- Terme erhalten Bedeutung durch Verknüpfung mit Beschreibung oder Thesaurus
- formale Axiome erlauben Interpretation und Verwendung der Terme

Quelle: [Gru93]

Keine feste Datenstruktur: es existieren verschiedene Formalismen (z.B. OWL, siehe später).

Eine konkrete Ontologie kann immer nur für einen bestimmten Anwendungsbereich verwendet werden, da Konzeptualisierungen immer für einen bestimmten Zweck erfolgen. Eine für einen Anwendungsbereich gut geeignete Ontologie kann möglicherweise für einen anderen Anwendungsbereich auch mit Hilfe von Erweiterungen nicht nützlich sein. Dies hat zur Konsequenz, dass es quasi nicht möglich ist, eine allgemeine Ontologie für alles zu erstellen, auch wenn dies wünschenswert wäre.

Beispiel einer Ontologie



Anhand dieser Ontologie sollen wesentliche Konzepte von Ontologien beispielhaft erklärt werden. Als Ontologie-Sprache wurde hier KAON gewählt. Sie beinhaltet:

- Konzepte

- Eigenschaften/Attribute

Attributwerte können entweder Literale („hatName“) oder eine Menge von Konzepten („arbeitetMit“) sein. Darüber hinaus können Eigenschaften symmetrisch („arbeitetMit“) oder transitiv („istTeilVon“) sein. Zwei Eigenschaften können auch invers zueinander („besitzt“ und „gehört“) sein.

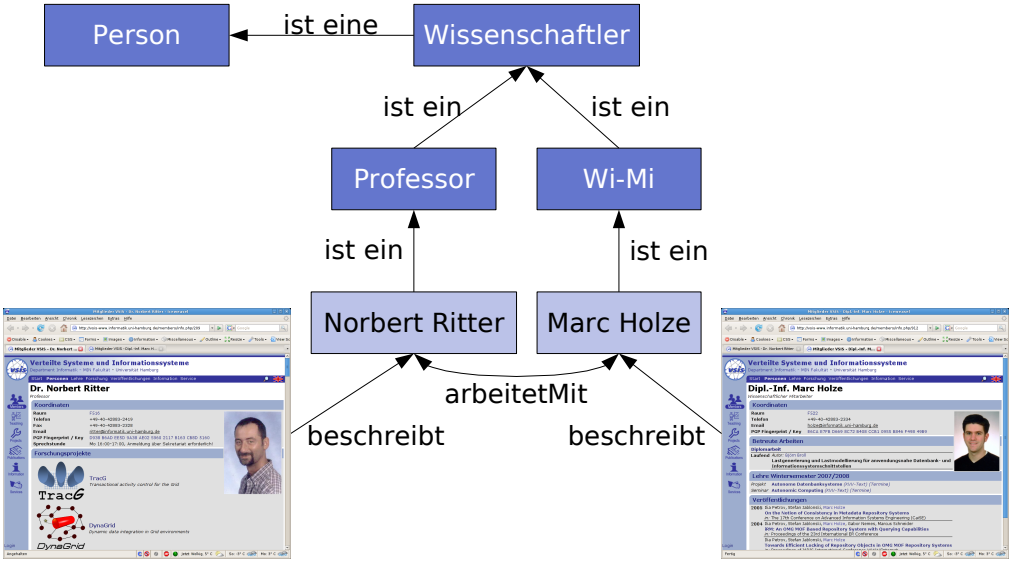
Konzepte können Hierarchien bilden (Vererbung).

Zu jedem Konzept kann es eine Menge von möglichen Ausprägungen („instances“) geben, die ebenfalls zur Ontologie gehören. Ausprägungen halten sich strikt an definierte Wertebereiche für die Attribute.

Lexikalische Einträge (Label, Bezeichner, Synonyme, textuelle Dokumentation) beschreiben durch eine n:m-Beziehung die Bedeutung der Konzepte. Erst so wird der Ontologie eine Semantik gegeben.

Quelle: [SSM04]

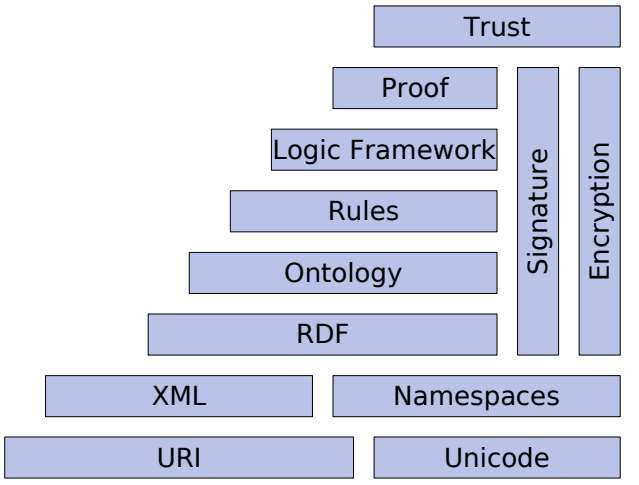
Semantic Web



Im aktuellen World-Wide-Web (WWW) existieren eine Vielzahl von semi- oder unstrukturierten Dokumenten (hauptsächlich Webseiten), die zwar per Hyperlink verknüpft sind. Es wird jedoch nicht deutlich, in welchem Zusammenhang verknüpfte Dokumente zueinander stehen oder was die Dokumente beschreiben. Dies ist derzeit nur per Volltext-Suche möglich.

In der Vision des Semantic-Web wird das herkömmliche WWW um eine semantische Beschreibung erweitert. Dabei existiert zu jedem Dokument eine formale, strukturierte, semantische Beschreibung des Inhalts. Agenten sollen so in die Lage versetzt werden, Zusammenhänge erschließen und aus den Informationen Wissen ableiten zu können.

Ebenen im Semantic Web



[Ber02]

Die Idee des Semantic-Web stammt von Tim Berners-Lee, welcher bereits 1989 mit seinem Vorschlag des World-Wide-Web (WWW) das damalige Internet revolutionierte. Heute werden am W3C diverse Web-Standards entwickelt und standardisiert. Um die Mächtigkeit des Semantic-Web möglich zu machen, wird ein Schichten-Modell umgesetzt.

Jedes Objekt soll durch eine URI identifizierbar sein.

Dokumente werden in der Sprache XML erstellt, welche aufgrund ihrer Erweiterbarkeit mittels „Namespaces“ eine Einbettung von semantischen Beschreibungen in bestehende Dokumente erlaubt.

Das Resource Description Framework (RDF) ist die Sprache, mit welcher Dokumente im Web beschrieben werden sollen. Die Sprache ist recht einfach gehalten, um sie flexibel anwenden zu können.

Damit Agenten die verschiedenen RDF-Beschreibungen verstehen können, wir eine einheitliche bzw. zumindest kompatible Ontologie benötigt. Diese soll mittels der Web Ontology Language (OWL) definiert werden.

Regeln sollen Schlussfolgerungen erlauben, die durch ein Logik-System verarbeitet werden können. Auf diese Weise können Beweise geführt werden und unter Verwendung von kryptographischen Signaturen und Verschlüsselung wird ein Vertrauen erreicht.

Semantic Web: Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE owl [ [...]
    <!ENTITY vsis "http://vsis-www.informatik.uni-hamburg.de/members/info.php/">
]>
<rdf:RDF xmlns=[...]>
        <owl:Ontology rdf:about="" />
        <owl:Class rdf:ID="Person" />
        <owl:Class rdf:ID="Wissenschaftler">
                <rdfs:subClassOf rdf:resource="#Person" />
        </owl:Class>
        <owl:ObjectProperty rdf:ID="arbeitetMit">
                <rdf:type rdf:resource="&owl;SymmetricProperty" />
                <rdfs:domain rdf:resource="#Person" />
                <rdfs:range rdf:resource="#Person" />
        </owl:ObjectProperty>
        <rdf:Description rdf:about="&vsis;912">
                <rdf:type rdf:resource="#Wissenschaftler"/>
        </rdf:Description>
        <rdf:Description rdf:about="&vsis;209">
                <rdf:type rdf:resource="#Wissenschaftler"/>
                <arbeitetMit rdf:resource="&vsis;912" />
        </rdf:Description>
</rdf:RDF>
```


Umsetzung des zuvor dargestellten Beispiels der Beschreibung der Webseiten der VSIS-Mitarbeiter. Aus Platzgründen wurde nur ein kleiner Ausschnitt realisiert.

Wichtig ist hierbei, dass die Eigenschaft „arbeitetMit“ als symmetrisch definiert wurde. Ein wissensverarbeitendes System sollte daher schließen können, dass Marc Holze (&vsis;912) mit Norbert Ritter (&vsis;209) arbeitet.

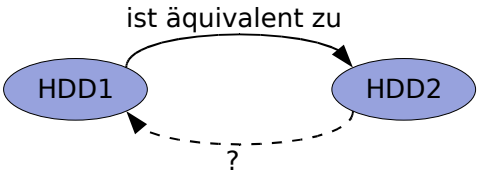
Die verwendete XML-Entity &vsis; wurde lediglich aus Platzgründen eingeführt. Leider mussten aus dem selben Grund auch weiterer Code weggelassen werden.

eAutomation

- „correlation engine“
  - analysiert Ereignisse verschiedener Systeme
  - findet Zusammenhänge
  - Ziel: Erkennung von Angriffen, Systemfehlern und Behebung derselbigen
- „eAutomation“ ist Teil von „IBM Tivoli System Automation for Linux on xSeries and zSeries“

Bei „eAutomation“ handelt es sich um ein Produkt der Firma IBM, welches in [SSM04] untersucht wurde. Es handelt sich dabei um einen Teil einer „correlation engine“. Ein solches System analysiert Ereignisse verschiedener System und findet Zusammenhänge um Angriffe und Systemfehler zu erkennen und diese möglicherweise auch zu beseitigen.

Motivation

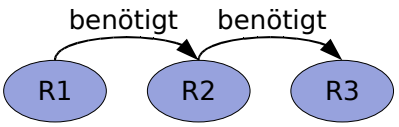


- ohne explizite Programmierung keine Aussage möglich
- Wissen liegt implizit in den Komponenten versteckt

In diesem Beispiel ist nur die Information „Festplatte 1 ist äquivalent zu Festplatte 2“ gespeichert. Woher weiß das System nun, ob auch gilt: „Festplatte 2 ist äquivalent zu Festplatte 1“?

Ohne Ontologien muss dieser Fall explizit programmiert werden. Der „Fragende“ muss dabei darauf achten, die Wissensbasis auch hiernach zu fragen.

Motivation



- Anordnung: Starten von Ressource R1
- was ist als nächstes zu tun?

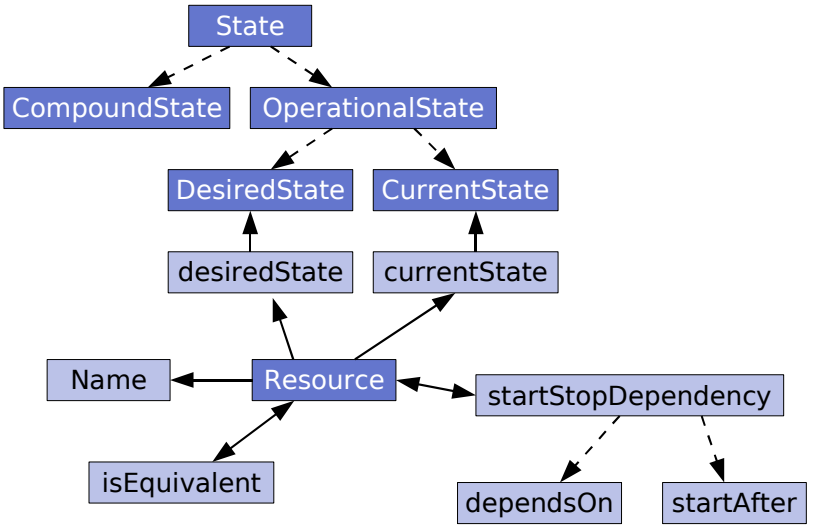
Folgende Aussagen sind bekannt:

„R1 muss nach R2 gestartet werden“

„R2 muss nach R3 gestartet werden“

R1 ist derzeit nicht gestartet, besitzt aber als „desiredState“ den Wert „online“. Aufgrund der Eigenschaft „startAfter“, kann R1 allerdings nicht direkt gestartet werden. Herauszufinden, welche Ressource als nächstes gestartet werden muss, erfordert viele Abfragen.

eAutomation-Ontologie



[SSM04]

Diese Ontologie dient als Beispiel dafür, wie Ontologien im Bereich des Autonomic Computing eingesetzt werden können. Ausgangspunkt war eine Correlation-Engine von IBM (eAutomation). Es geht nun nicht darum, dass beim Autonomic Computing so eine Ontologie aussehen muss. An dieser Stelle sollen nur ein paar Punkte gezeigt werden, die den Einsatz von Ontologien im Autonomic Computing motivieren.

Transitive Relation: „startStopDependency“

Werte für „DesiredState“: online, offline

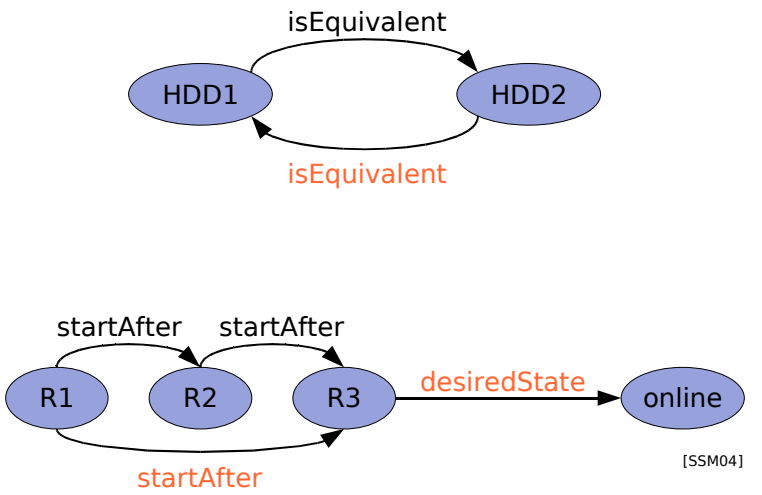
Werte für „CurrentState“: offline, online, starting, stopping, ...

Beispiel:

Ressource R1 hat den Wert currentState=offline und desiredState=online. R1 kann nicht einfach gestartet werden, da über die Start-Stop-Abhängigkeiten definiert ist, dass vor dem Starten von R1 zunächst R2 gestartet werden muss („startAfter“).

Quelle: [SSM04]

Abgeleitetes Wissen



Festplattenbeispiel:

In einer Ontologie wäre die Eigenschaft „isEquivalent“ als symmetrisch definiert. Daher kann die Information „Festplatte 2 ist äquivalent zu Festplatte 1“ automatisch abgeleitet werden.

Das Wissen wird dabei explizit durch die Symmetrie-Eigenschaft in der Ontologie repräsentiert. Im ersten Fall wäre das Wissen lediglich implizit im Programm versteckt.

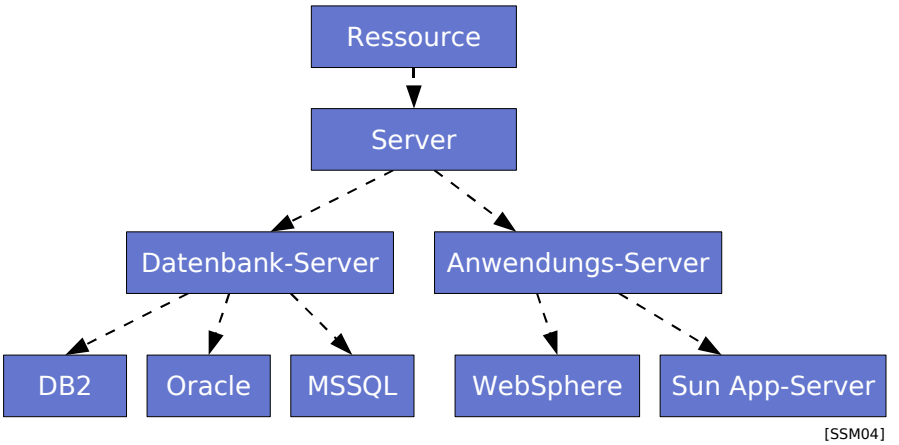
Beispiel: Starten von Ressourcen:

Dank der Transitivität dieser Eigenschaft kann mit Hilfe der Ontologie festgestellt werden, dass zunächst R3 gestartet werden muss. Erst wenn dies erfolgt ist, kann R2 gestartet werden und schließlich R1.

Man könnte auch eine Relation „startBefore“ als invers zu „startAfter“ definieren und so automatisch weitere Informationen aus dem Modell erhalten.

Quelle: [SSM04]

Hierarchiebildung



Vorteile der Hierarchiebildung sind:

Wiederverwendbarkeit: Bilden einer Bibliothek aus Modulen von Ontologien. Ein Modul kann dabei allgemeine, abstrakte Konzepte oder aber konkrete Ausprägungen enthalten. Die allgemeinen Konzepte können durch Vererbung konkretisiert werden.

Erweiterbarkeit: Jedes Konzept ist in sich abgeschlossen, kann aber durch Vererbung erweitert werden.

Quelle: [SSM04]

Vorteile bei der Modellierung

- Wiederverwendbarkeit
- Erweiterbarkeit
- Anwendbarkeit
- Verifikation
- Integration
- Evolution
- Visualisierung
- offene Standards

Wiederverwendbarkeit: siehe Hierarchiebildung

Erweiterbarkeit: siehe Hierarchiebildung

Anwendbarkeit: erleichtert die Suche nach Diensten, wenn der exakte Name nicht bekannt ist, durch die Speicherung von Synonymen, Abkürzungen usw. (beispielsweise On-Demand-Computing)

Verifikation: Prüfen auf Widersprüche, z.B. durch inverse Beziehungen („besitzt“ und „gehört“). Erkennung von Redundanz, die beispielsweise durch Transitivität von Eigenschaften entsteht. Begründung von Entscheidungen des Systems durch Ableitungsbäume.

Integration: Ontologien als Vermittler zwischen verschiedenen Systemen

Evolution: Anpassung an sich ändernde Bedürfnisse

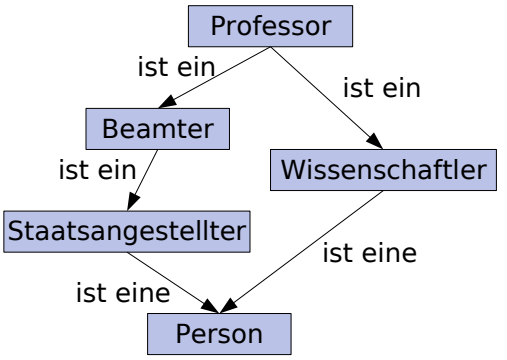
Visualisierung: Mit einer hierarchischen graphischen Darstellung ist es viel einfacher Beziehungen und Kategorien zu erkennen.

offene Standards: Ontologien als Standard für das Semantic-Web erleichtern die Interoperabilität zwischen verschiedenen Systemen.

Quelle: [SSM04]

Vorteile bei der Anwendung

- Begründen einer Antwort
- Bewertung der Relevanz
- Suche nach Fehlern



Trifft ein Autonomic System eine Entscheidung, so kann eine nachvollziehbare Begründung für diese Wahl geliefert werden, indem beispielsweise dem Administrator der Ableitungsbaum übermittelt wird. So kann nachvollzogen werden, wo das Problem lag um ggfs. händisch zu optimieren.

Um den nächsten Schritt zu planen können möglicherweise mehrere Optionen zur Wahl stehen. Durch Analyse der Komplexität der Ableitungsbäume der Ergebnisse, können diese unterschiedlich gewichtet werden, um zunächst einfachere Lösungen durchzuführen.

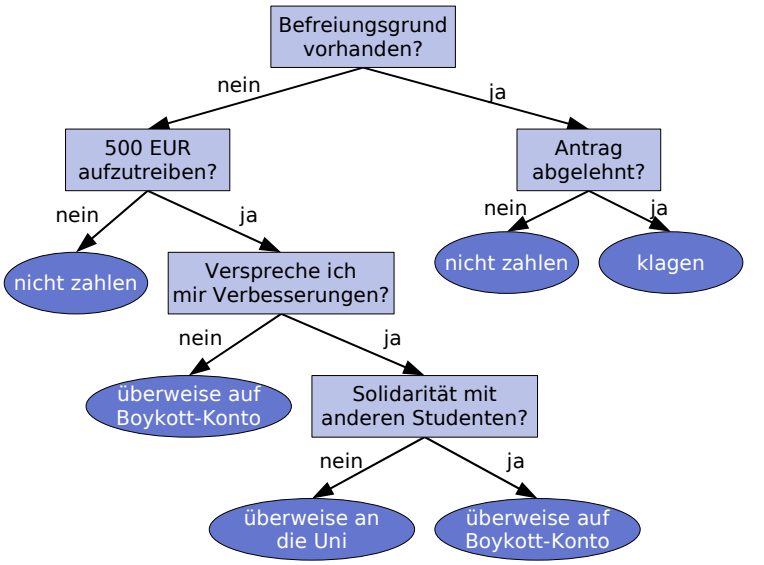
Falls keine Lösung für ein Problem gefunden werden kann, gibt es weder einen Fehler noch ein Ergebnis. In der Regel ist die Ursachenfindung daher sehr schwierig. Auch hier kann der Ableitungsbaum helfen, denn durch dessen Analyse kann festgestellt werden, an welcher Stelle nicht fortgefahren werden konnte, beispielsweise weil entsprechende Inferenzregeln fehlten.

Quelle: [SSM04]

Teil II: Entscheidungsbäume

- Einführung
- Lernen von Entscheidungsbäumen
- Einsatz zur Fehlerdiagnose

Studiengebühren – was tun?



Entscheidungsbäume sind gerichtete, kantenbewertete Bäume im Sinne der Graphentheorie. Den inneren Knoten sind Tests zugeordnet, welche i.A. Abfragen von Attributwerten darstellen. Für mögliche Antworten der Tests gibt es eine abgehende Kante von diesem Knoten zu einem weiteren Knoten (mit neuem Test) oder einen Blatt. Den Blättern sind Klassifikationswerte zugeordnet, die das getestete Objekte in eine Klasse einordnen.

Vorteil von Entscheidungsbäumen ist, dass diese sowohl maschinell verarbeitet als auch vom Menschen verstanden werden können.

Quelle: [Gri02]

Lernen von Entscheidungsbäumen

- benötigt bereits klassifizierte Lernmenge
- Wurzel ist der für die Lernmenge am besten geeignete Test
- für jeden möglichen Testausgang wird rekursiv ein Teilbaum eingefügt
- spätestens bis Lernmenge gleich klassifiziert

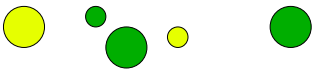
Diese Lernmethode nennt man „Top-Down-Induktion of Decision Trees“ (TDIDT). Der gezeigte Algorithmus ist das Grundschema. In der Praxis gibt es verschiedene Algorithmen, wie z.B. ID3 und C4.5, die sich unterscheiden in ihrer Strategie, einen Test zu wählen, das Ende-Kriterium, ihre Pruning-Methode (siehe nächste Folie). Kriterium für die Wahl des „besten“ Tests kann beispielsweise die Entropie des Attributes sein (s.u.).

Neben TDIDT gibt es noch Algorithmen, die inkrementell Lernen.

Quelle: [Gri02]

Die Entropie gibt an, wieviel interessante/neue Information eine Datenquelle hat bzw. wie zufällig die daraus kommenden Daten sind. Ein Test ist dann geeignet, wenn die resultierende Teilmenge der Lernmenge eine möglichst geringe Entropie aufweist, d.h. die Elemente möglichst wenig Klassifkationen angehören.

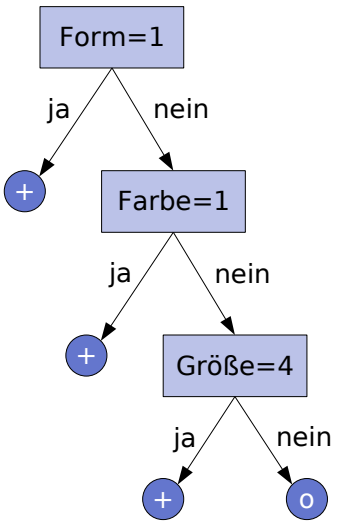
Beispiel



Objekte (Erbsen) mit drei Eigenschaften:
- Farbe: *grün* (1) oder *gelb* (0)
- Form: *rund* (1) oder *runzelig* (0)
- Größe: kontinuierlich

Klassifikator: Fruchtbarkeit

	+	+	+	+	+	+	+	o	+	o
Farbe	1	1	0	0	0	1	1	0	0	0
Form	1	1	1	1	1	0	0	0	0	0
Größe	6	2	8	1	9	5	3	7	4	2



Dieses Beispiel ist frei erfunden und der biologische Hintergrund ist mehr als fraglich. Darum geht es jedoch nicht. Das Beispiel zeigt eine bereits klassifizierte Lernmenge, anhand der ein Entscheidungsbaum „gelernt“ wird.

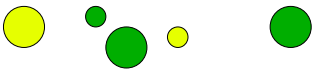
Auftretende Probleme

- Lernmenge wird korrekt klassifiziert
- Ausreißer führen zu einem falschen Baum
- Baum ist zu genau für Lernmenge und klassifiziert Daten falsch (*„overfitting“*)
- vereinfachen des Baumes (*„pruning“*)
  - Lernen vorzeitig abbrechen
  - Knoten im fertigen Baum zusammenfassen

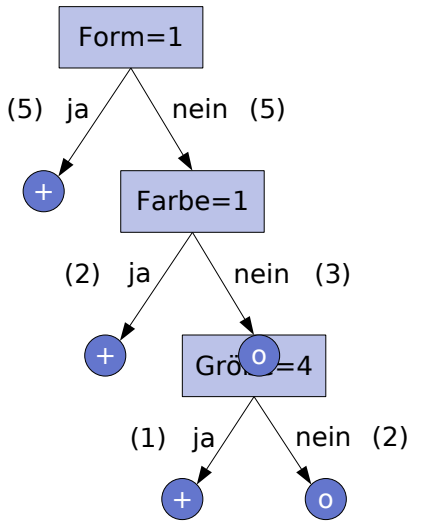
Das Problem ist, dass die Lernmenge nicht unbedingt repräsentativ für den Klassifikator ist. Insbesondere durch Ausreißer können die Algorithmen völlig falsche Bäume aufbauen. Dies soll durch „pruning“ vermieden, bzw. zumindest abgeschwächt, werden.

Quelle: [Gri02]

Beispiel



	+	+	+	+	+	+	+	o	+	o
Farbe	1	1	0	0	0	1	1	0	0	0
Form	1	1	1	1	1	0	0	0	0	0
Größe	6	2	8	1	9	5	3	7	4	2



Wenn wir uns die Daten genau angucken, stellen wir fest, dass die Klassifikation für die vorletzte Erbse ungewöhnlich ist. Die beiden anderen Erbsen, die gelb und runzelig sind, haben die Klassifikation „o“ erhalten, während diese die Klassifikation „+“ besitzt. Das einzige Attribut, welches diese Erbsen unterscheidet ist die Größe. Jedoch liegt 4 zwischen 2 und 7. Möglicherweise ist bei der Datenaufnahme ein Fehler passiert oder diese Erbse ist einfach ein Sonderfall (Ausreißer), der jedoch nicht berücksichtigt werden sollte.

Solche Ausreißer kann man auf der einen Seite durch spezielle Ausreißer-Erkennung berücksichtigen (Data Mining, siehe DIS-Vorlesung). Einfacher ist es jedoch, den gelernten Baum mittels „pruning“ von diesen Spezialfällen zu beseitigen oder die Verzweigung gar nicht erst entstehen zu lassen. Man kann beispielsweise festlegen, dass keine weitere Unterteilung vorgenommen wird, wenn die derzeitige oder resultierende Menge einen bestimmten Schwellwert (in Abhängigkeit von der Kardinalität der gesamten Lernmenge) unterschreitet.

Fehlerdiagnose

- Fehler-Erkennung vs. Fehlerdiagnose
- große Internet-Dienste besitzen i.d.R. sichtbare Fehler in der Funktionalität
- Quelle liegt oft in der Anwendungslogik, daher schwer zu erkennen
- Auffinden kann ¾ der Behebungszeit kosten
- Wunsch: Unterstützung bei der Suche

Fehler-Erkennung: Erkennen, dass ein Fehler aufgetreten ist.

Fehlerdiagnose: Herausfinden, wo die Ursache des Fehlers liegt.

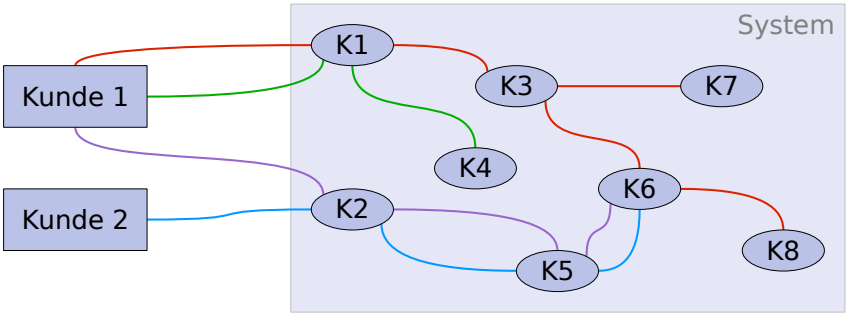
Laut [KF05] wurde in einer Studie festgestellt, dass 72% der Top40-Websites sichtbare Fehler in der Funktionalität besitzen. Diese Fehler betreffen nur kleine Teile und sind nicht auf Low-Level-Fehler (z.B. Ausfall einer Festplatte) zurückzuführen, sondern liegen auf Anwendungsebene. D.h. es liegt ein semantischer Fehler vor – der Webserver antwortet beispielsweise weiterhin auf Pings, liefert keinen HTTP-Fehler sondern eine Seite mit gültigem HTML. Fehler dieser Art sind daher schwer zu erkennen.

Es wurden zwei Ansätze betrachtet, die sich diesem Problem annehmen und im Folgenden vorgestellt werden.

Quelle: [KF05], [CZL04]

Idee und Voraussetzungen

- System besteht aus Komponenten
- Zugriffe als Anfrage-Antwort-Zyklus
- Auswerten der Interaktionen zwischen den Komponenten



Voraussetzungen für den Einsatz:

Kompentenbasiertes System: die einzelnen Komponenten können dabei Software-Objekte (Klassen, EJBs, ...) oder separate Programme (Datenbank) sein, aber auch Hardware-Einheiten.

Eine Anfrage an das System verbleibt nur kurze Zeit im System und kann in einem Pfad/Baum dargestellt werden, der aus den einzelnen Komponenten besteht, welche an der Bearbeitung der Anfrage beteiligt waren.

Um gute Ergebnisse liefern zu können, müssen sehr viele, unabhängige Anfragen an das System gestellt werden. Nur so kann sichergestellt werden, dass die beobachteten Zugriffspfade auch einigermaßen repräsentativ sind.

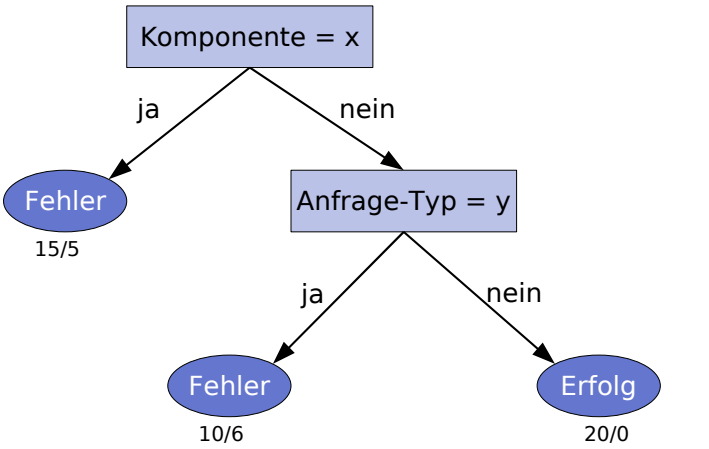
Quelle: [KF05]

Lernen als Mittel zum Zweck

- „Ziel“: Entscheidungsbaum, der Zugriffspfade klassifiziert (Erfolg/Fehler)
- benötigt klassifizierte Trainingsdaten
  - Name der Komponente
  - Typ der Anfrage
  - Name des Servers
- aus der Struktur des Baumes schließen, welche Komponente fehlerhaft ist

Quelle: [CZL04]

Beispiel



Um fehlerhafte Interpretation durch Rauschen zu vermeiden, werden nur Blätter berücksichtigt, deren Fehlerrate über einem gewissen Prozentsatz liegt.

Subsumieren: Wenn auf Komponente x keine Anfragen vom Typ y verarbeitet werden, so können die beiden Knoten zusammengefasst werden.

Um die Wichtigkeit der resultierenden Pfade einschätzen zu können, werden diese nach Fehlerrate sortiert.

Quelle: [CZL04]

Zusammenfassung

- Wissensrepräsentation ist zentraler Bestandteil im Autonomic-Computing
- Ontologien
  - beschreiben Struktur der „Managed Resource“
  - Inferenzmethoden leiten implizites Wissen ab
  - die alles beschreibende Ontologie fehlt
- Entscheidungsbäume
  - können per Hand erstellt werden um bekannte Probleme zu erkennen und lösen
  - durch automatisches Lernen für Fehlerdiagnose von unbekannten Fehlern „missbrauchen“

Ende des Vortrages.

Literatur

[Ber02] Tim Berners-Lee, The Semantic Web, Academic discussion, Japan Prize 2002, http://www.w3.org/2002/Talks/04-sweb/slide12-0.html

[CZL04] Chen et al., Failure Diagnosis Using Decision Trees, ICAC, 2004

[Gri02] Gunter Grieser, Selbsteinschätzende Lernverfahren: Möglichkeiten und Grenzen, Berlin: Akad. Verl.-Ges. Aka, 2002

[GH04] OWL Web Ontology Language Overview, McGuinness & Harmelen, W3C Recommendation 10 February 2004

[Gru93] Thomas R. Gruber, A Translation Approach to Portable Ontology Sepcifications, Knowledge Acquisition, 1993

[IBM05] IBM Corporation, An architectural blueprint for autonomic computing, White Paper June 2005

[KF05] Kiciman et al., Detecting Application-Level Failures in Component-Based Internet Services, IEEE TRANSACTIONS ON NEURAL NETWORKS, 2005

[SSM04] Stojanovic et al., The role of ontologies in autonomic computing systems, IBM Systems Journal, 2004

Literatur, die zwar während der Recherche gelesen, später aber nicht in diesem Vortrag eingeflossen ist, taucht in dieser Liste nicht auf.