Zusammenfassung - Multi-Agenten-Systeme

Andreas Ruscheinski, Marc Eric Meier, CD

26. Februar 2016

Korrektheit und Vollständigkeit der Informationen sind nicht gewährleistet. Macht euch eigene Notizen oder ergänzt/korrigiert meine Ausführungen!

Inhaltsverzeichnis

1 Einführung

Multiagentensysteme bieten eine Alternative zu den schwergewichtigen Expertensystemen und sind in einer Umgebung eingebunden.

1.1 Definition

- Ein Agent ist ein Computer System, welches **selbstständig** Aktionen im Interesse des Benutzers ausführen kann.
- Ein Agent **befindet** sich in einer **dynamischen Umgebung**, mit welcher er interagiert.
- Ein Multi-Agenten-System besteht aus **mehreren Agenten**, welche **miteinander interagieren**.
- In einem Multi-Agenten-System ist es für eine **erfolgreiche Interaktion** notwendig, dass die Agenten miteinander **kooperieren**, **verhandeln** und **sich abstimmen** können.

1.2 Eigenschaften

- Ein Agent hat **keine vollständigen Informationen** über die Umgebung. Er ist begrenzt in Hinblick auf Informationen, Ressourcen, Fähigkeiten → er benötigt die Hilfe von anderen.
- Es gibt **keine globale Kontrolle** der Agenten.
- Die Daten sind **dezentralisiert**.
- Die Berechnung erfolgt **asynchron**.

1.3 Unterscheidung OOP und MAS

- Objekte sind passiv (keine Kontrolle darüber den Methodenaufruf), für den Gemeinnutzen entwickelt und üblicherweise in einen Singlethread integriert
- Agenten sind autonom (und bspw. proaktiv), können unterschiedliche Ziele verfolgen (je nach Organisationsform) und haben ihren eigenen Thread

→ Objects do it for free, agents do it for money

1.4 Gründe für den Einsatz von MAS

- Ein Problem kann nicht zentralisiert gelöst werden, da die **Ressourcen limitiert** sind.
  →*Verteilte Berechnungen*
- **Reduktion der Ausfall-Wahrscheinlichkeit** gegenüber einem zentralisierten System
- **Gewährleistung der Interkonnektion und Interoperation** von verschiedenen Systemen
  →*Migration von veralteter Software*
- Lösung von Problemen, die sich mit einer **Menge von autonomen Komponenten beschäftigen**
  →*Luftfahrkontrolle, Terminkalender*

1.5 Konkrete Anwendungsgebiete

- Cloud-Management
- Ubiquitous Computing
- Grid-Software
- Spiele
- Verschiedene Gebiete der Industrie (Car-Assembly, Factory Management)
- Simulation

MAS befindet sich in der Schnittmenge aus KI, Spieltheorie, Sozialforschung und Verteilten Systemen.

2 Rolle der Logik in MAS

2.1 Gründe für Logik

- Wissensbasis + Aktionen mit Voraussetzung und Auswirkung → Plan für Lösung des Problems
- Verwaltung von Annahmen
- Logik ist ein Framework für das Verstehen von Systemen
- Verifikation, Ausführungsspezifikation, Planung
- Agenten als Theorembeweiser

2.2 Logik-basierende Architektur

- Grundidee: Aufstellen einer Regelmenge zur Beschreibung der besten Aktion bei einem gegebenem Zustand
- Bestandteile:
  - p: Eine Theorie (Menge von Regeln)
  - Δ: Datenbank mit dem aktuellen Zustand der Welt
  - A: Eine Menge von Aktionen, die ein Agent ausführen kann
  - $\Delta \vdash_p \phi$: d.h. ϕ kann aus der Δ unter der Verwendung von p abgeleitet werden, mit ϕ =Do(a) können wir aus dem aktuellen Zustand der Welt auf die bestmögliche Aktion logisch schließen.
- Algorithmus-Bestandteile (unabhängig von einer verwendeter Logik)
  - **see(s,p)**, generiert Beobachtung aus der aktuellen Welt s (schwer siehe Vorlesung Mustererkennung und Kontextanalyse)
  - **next(Δ,p)**, Update der Datenbank mit der Beobachtung
  - **action(Δ)**, ermittelt die auszuführende Aktion aus der Datenbank, entweder ist die Aktion direkt beschrieben oder kann aus den Regeln abgeleitet werden.
- Algorithmus - Ermittelung einer Aktion aus einer Wissensdatenbank:
  1. Überprüfe für jede Aktion a, ob $\Delta \vdash_p Do(a)$ gilt (d.h. a kann direkt abgeleitet werden)
  2. Falls ja: Gib a zurück; sonst
  3. Überprüfe für jede Aktion a, ob $\Delta \nvdash_p \neg Do(a)$ gilt (d.h a ist nicht explizit ausgeschlossen)
  4. Falls ja: Gib a zurück, sonst gib NULL zurück (keine Aktion gefunden)

2.3 Modale Logik

- Erlaubt Ausdrücke wie: wahrscheinlich wahr, geglaubt wahr, wahr in der Zukunft usw.
- Kann genutzt werden, um Informationen für die Agenten abzuleiten und über das Wissen von Agenten zu schließen
- Syntax:
  - Prädikatenlogik mit Erweiterung
  - Prop: Eine Menge von atomaren Formeln
  - Wenn $p, q \in Prob$, dann sind auch $\neg p, p\&q, \diamond p, \Box p$ Formeln
  - $\diamond$ p: möglicherweise p, manchmal p
  - $\Box$p: immer p, notwendigerweise p
- Semantik:
  - Kripke-Struktur: $\mathcal{K} =< W, R, \mu >$
    W eine Menge von Welten
    R eine Menge von binären Relationen, die den Übergang zwischen den Welten beschreiben
    μ Abbildungsfunktion, die jeder Welt Eigenschaften zuordnet ($\mu : W \to 2^{Prop}$)
  - Rahmen: $< W, R >$
    - W und R definiert, wie in Kripke Struktur
    - $R \vdash F$, falls gilt: $\forall \mathcal{K}, s : \mathcal{K}, s \vDash F$
    - Die **Korrespondenztheorie**: Zusammenhang zwischen Axiomensystemen und Rahmen
  - Definition von $\diamond$ und $\Box$ Operator auf Basis von Erreichbarkeit der Welten in einer Kripke-Struktur
  - $\Box p$: p ist wahr in allen Welten, welche von der aktuellen Welt erreichbar sind
  - $< M, w > \vDash \Box p$, wenn für alle $w' \in W$ gilt: wenn $(w, w') \in R$ denn $< M, w' > \vDash p$
  - $\diamond p$: p ist wahr, wenn mindestens eine Welt erreichbar ist, in welcher p wahr ist
  - $< M, w > \vDash \diamond p$, wenn es ein $w' \in W$ existiert mit: wenn $(w, w') \in R$ denn $< M, w' > \vDash p$
  - Für R muss zusätzlich gelten:

reflexiv für jedes $x \in W$ gilt $R(x, x)$, d.h. x ist von x aus erreichbar.

transitiv für jedes $x, y, z \in W$ gilt $R(x, y) \wedge R(y, z) \implies R(x, z)$, d.h. wenn man von x nach y und von y nach z gehen kann, kann man auch von x nach z gehen.

seriell für jedes $x \in W$ existiert ein y, so dass gilt $R(x, y)$, d.h. jede Welt ist mit einer anderen Welt in Relation.

euklidisch wenn für jedes $x, y, z \in W$ mit $R(x, y)$ und $R(x, z)$, dann gilt auch $R(y, z)$, d.h. wenn man von x nach y und von x nach z gehen kann, dann kann man auch von y nach z gehen.

Name	Axiom	Condition on R	First-order characterization
T	$\Box\phi \Rightarrow \phi$	Reflexive	$\forall w \in W : (w, w) \in R$
D	$\Box\phi \Rightarrow \Diamond\phi$	Serial	$\forall w \in W : \exists w' \in W : (w, w') \in R$
4	$\Box\phi \Rightarrow \Box\Box\phi$	Transitive	$\forall w, w', w'' \in W : (w, w') \in R \wedge (w', w'') \in R \Rightarrow (w, w'') \in R$
5	$\Diamond\phi \Rightarrow \Box\Diamond\phi$	Euclidiean	$\forall w, w', w'' \in W : (w, w') \in R \wedge (w, w'') \in R \Rightarrow (w', w'') \in R$

Korrespondenztheorie

- Axiome (Beschreiben Eigenschaften für R):

  $\Box p \Rightarrow p$ Wann immer p wahr ist, folgt daraus, dass aktuell p gilt (**Reflexiv**)

  $\Box p \Rightarrow \Diamond p$ Wann immer p wahr ist, ist p auch in mindestens einer Welt wahr (**Seriell**)

  $\Box p \Rightarrow \Box\Box p$ Wann immer p wahr ist, ist p auch immer wahr, wenn wir einen Übergang machen (**transitiv**)

  $\Diamond p \Rightarrow \Box\Diamond p$ Wenn p in mindestens einer Welt wahr ist, ist p für immer wahr, wenn wir diese Welt erreicht haben (**euklidisch**)

- Anwendung der Modal Logik auf Agenten durch Einführung von Indizes, welche entsprechend für Agent gelten
- Operator - Knowledge: $K_i p$ bedeutet, dass Agent i p weiß
- Der Modale Operator $\Box_i$ wird zu K_i

- Axiome und Agenten:

  $K_i p \Rightarrow p$ Wenn Agent glaubt, dass p wahr ist, ist p auch in Wirklichkeit wahr

  $K_i p \Rightarrow \neg K_i \neg p$ Wenn Agent glaubt, dass p wahr ist, glaub er nicht die Negation

  $K_i p \Rightarrow K_i K_i^p$ Wenn Agent glaubt, dass p wahr ist, weiß er selbst, dass er p glaubt

  $\neg K_i \neg p \Rightarrow K_i \neg K_i \neg p$ Wenn der Agent nicht p glaubt, weiß er dass er nicht p glaubt.

- Es gilt: $< M, w > \models K_i p$, genau dann wenn $\forall w' \in W$ gilt: wenn $(w, w') \in R_i$, dann $< M, w' > \models p$ (vgl. Definition von $\Box$)
- Probleme:
  - Agent glaubt alles was wahr ist incl. Tautologien: wenn $\models p$ denn $K_i p$
  - Agent weiß alle Schlussfolgerungen des eigenen Wissens: $\models p \rightarrow q$ denn $\models K_i p \rightarrow K_i q$ (Kripke Axiom: $K_i(p \rightarrow q) \rightarrow (K_i p \rightarrow K_i q)$)
  - Menschliches Wissen ist oft inkonsistent
  - Menschen glauben nicht alle äquivalenten Aussagen

2.4 BDI-Logic

Beliefs
- **Aktuelles Weltwissen** inklusive Umgebungszustand und internem Wissen
- **Informationskomponente** über Systemzustand

Desires
- Beschreiben den **Zustand, den ein Agent zu erreichen versucht**
- Kann als **Anregungszustand des Systems** betrachtet werden

Intentions
- **Der gewählte Weg** eines Agenten um seine Desires zu erreichen.
- Intention beschreibt die Komponente zur **Entscheidungsfindung** des Agenten
- **Committment** beschreibt Grad der Festlegung auf eine Aktionsausführung

- Klassische logische Operatoren: und, oder, Negation usw...
- CTL* Path Quantifikatoren (Computation Tree Logic)
  - $A\phi$: es gilt auf allen Pfaden ϕ
  - $E\phi$: es gilt auf einigen Pfaden ϕ
  - G: global
  - F: zukünftig
- BDI-Funktionen:
  - (Bel i ϕ): i glaubt ϕ
  - (Des i ϕ): i will ϕ
  - (Int i ϕ): i verfolgt ϕ
- Regeln:
  - (Des α) $\rightarrow$ (Bel α) (Belief goal compatibility): Wenn der Agent α erreichen will, folgt daraus, dass der Agent an die Machbarkeit von α glaubt.
  - (Int α) $\rightarrow$ (Des α) (Goal-intention compatibility): Wenn der Agent α verfolgt, folgt daraus, dass der Agent α erreichen will.
  - (Int does(a)) $\rightarrow$ does(a) (Volitional commitment): Wenn der Agent does(a) ausführen möchte, wird er diese als nächstes ausführen.
  - (Des α) $\rightarrow$ (Bel (Des α)) und (Int α) $\rightarrow$ (Bel (Int α)) (Awareness of goals & intentions): Bedingt, dass neue Ziele und Intentionen als Event gepostet werden, müssen in der Wissensbasis...
  - done(a) $\rightarrow$ (Bel (done(a))) (No unconscious actions): Wenn ein Agent eine Aktion ausgeführt hat, weiß er, dass er diese ausgeführt hat.
  - (Int α) $\rightarrow$ AF($\neg$(Int α)) (No infinite deferral): Ein Agent will eine Intention verfolgen oder diese verwerfen.
- reaktive und proaktive Agenten
  - Reaktiv: Aktion wird ausgeführt durch eine Eingabe: Wenn Eingabe = p denn tu a
  - Proaktiv: Planung und Ausführung von Aktionen, um ein Ziel zu erreichen: Tu a, um p zu erreichen

3 Planning

3.1 Einführung - Brooks Vision

- Ziel 1: Intelligentes Verhalten ohne explizite Repräsentation von Wissen
- Ziel 2: Intelligentes Verhalten ohne abstraktes Schließen über die Repräsentation des Wissens
- Idee 1: Echte Intelligenz gibt es nur ein einer Welt und nicht losgelöst von dieser wie, in Theorem Beweisern und Expertensystemen
- Idee 2: Intelligentes Verhalten entsteht erst als Ergebnis der Interaktion mit der Umgebung

3.2 Subsumption Architektur

- Traditionell: Die Intelligenz steht zwischen Beobachtung und Aktion, d.h. aus Beobachtungen wird geschlossen welche Aktion ausgeführt wird.
- Neu: Die Intelligenz entsteht beim Beobachter durch die Aktionen in der Welt, d.h. ein Agent ist nicht an sich intelligent, sondern wirkt intelligent für einen Beobachter
- Entscheidungsfindung durch verschiedene Aufgaben:
  - Verhalten ist eine Abbildung von Zustand auf Aktion
  - Verarbeitung der Sensorwerte mit Schluss auf den Zustand
  - Zustände und Aktionen sind direkt gekoppelt

- Mechanismus zur Auswahl der Aktionen: Prioritäten → d.h Verhalten mit hoher Priorität kann Verhalten mit niedrigeren Prioritäten unterdrücken
- Formales Modell:
  - Ein Verhalten (Behavior) $b \in Beh$ ist ein Tupel (c, a) mit $c \subseteq P, a \in A$ wobei P eine Menge von Beobachtungen und A eine Menge von Aktionen ist.
  - Ein Verhalten wird ausgeführt, wenn die Umgebung sich in dem Zustand $s \in S$ befindet und $see(s) \in c$
  - Subsumption Hierarchie wird realisiert durch eine Hemmungs-Relation $b_1 \prec b_2$ (b_1 hemmt b_2 d.h. b_1 hat eine höhere Priorität)
- Aktionsauswahl-Algorithmus:
  1. Berechne die Menge von aktivierbaren Aktionen $FB = \{(c, a) | (c, a) \in Beh \wedge see(s) \in c\}$
  2. Für jede Aktion in FB überprüfe, ob es eine Aktion in FB gibt welche eine höhere Priorität hat
  3. Wenn keine Aktion mit höherer Priorität gefunden wurde: gib a zurück; sonst: NULL (Keine Aktion gefunden)
- Vorteile:
  - Einfach und hohe Ausdrucksfähigkeit
  - Die Berechnung ist einfach nachzuvollziehen
  - Robust gegen Ausfälle
  - Das gesamte Verhalten entsteht durch Interaktion mit der Umwelt
- Nachteile:
  - Verhalten wird hart kodiert unter der Annahme die Umgebung genau zu kennen
  - Schwierige Entscheidung über das Standardverhalten
  - Langwierige Entscheidungen schwer möglich
  - Skaliert nicht in größeren Systemen

3.3 Planung

- Grundideen:
  - Beschreibung des Zieles (Intention), das erreicht werden soll
  - Beschreibung der Aktionen, die ausgeführt werden können
  - Beschreibung der Umgebung
  - Beschreibungen + Planer = Plan welches das Ziel erreicht
- Umsetzung durch z.B STRIPS Planer
  - Repräsentation der Umgebung durch Ontologie (Begriffe + Relationen)
  - Beschreibung der aktuellen Welt durch Verwendung der Ontologie Begriffe (closed world assumption: alles was nicht angegeben wird, ist falsch)
  - Jede Aktion hat
    - einen Namen,
    - eine Pre-Condition Liste (alle Bedingungen müssen wahr sein bevor Aktion ausgeführt werden kann),
    - eine Delete-Liste (Bedingungen welche nach der Ausführung nicht mehr wahr sind),
    - eine Add-List (Bedingungen, die nach der Ausführung der Aktion gelten. Es können alle Variablen für allgemeine Aussagen erhalten sein.)
- Ein Plan ist eine Liste von Aktionen, mit Variablen ersetzt durch Konstanten. Die Ausführung der Aktionen führt vom aktuellen Zustand in einen Zustand, der das Ziel erfüllt. Der Plan ist vollständig (keine weiteren Aktionen notwendig) und konsistent (alle Pre-Conditions sind erfüllt) und die Schritte können hintereinander ausgeführt werden, ohne dass die Ausführung eines Schrittes beeinflusst wird.
- Formal: partially ordered Plans

  - Schritte eines Plans sind mit partieller Ordnung $\prec$: $S_i \prec S_j$ bedeutet, dass S_i vor S_j ausgeführt werden muss
  - Eine Menge von variablen Zuordnungen $x = t$, mit x ist eine Variable und t ist eine Konstante
  - Eine Menge von kausalen Relationen: $S_i \rightarrow S_j$ bedeutet, dass die Ausführung S_i die Vorbedingungen von S_j wahr macht (impliziert $S_i \prec S_j$)
- Formale Eigenschaften: Konsistenz und Vollständigkeit
  - Vollständigkeit:
    - Es gilt: $\forall S_j$ mit $c \in Precond(S_j)$ und $\exists S_i$ mit $S_i \prec S_j$ und $c \in Effect(S_i)$ (die Vorbedingungen für S_j sind Teil des Effektes von S_i)
    - Für eine Sequenz gilt:$\forall S_k$ mit $S_i \prec S_k \prec S_j, \neg c \notin EffectS_k$
  - Konsistenz: Wenn $S_i \prec S_j$ denn $S_j \prec S_i$ und wenn $x = A$ denn $x \neq B$ für verschiedene A und B für die Variable x.
- Vorlesungsfolien für Beispiel!!!
- Iterative Erstellung eines Plans durch rückwärts Anwenden der Regeln, d.h. in jedem Schritt wird eine offene Bedingung durch die entsprechende Aktion erfüllt.
- Dadurch können Konflikte entstehen:
  - Ein Konflikt ist gdw. wenn S_3 die kausale Ordnung zwischen S_1 und S_2 bedroht.
  - Lösung 1: Demotion - S_3 vor S_1 und S_2 ausführen
  - Lösung 2: Promotion - S_3 nach S_1 und S_2 ausführen
  - Sollte dies wieder zu Konflikte führen, Backtrack und eine andere Lösung ausprobieren in dem S_3 neben S_1 zu S_2 eingeordnet wird.

3.4 Planning Agents

- Erster Ansatz für den Planning Agent:
  1. Beobachte die Umgebung
  2. Aktualisiere das interne Modell der Umgebung
  3. Ermittle welche Intention als nächstes erreicht werden soll
  4. Benutze means-end Resoning für die Erstellung des Plans welche die Intention erreicht
  5. Führe Plan aus
- **means-end Resoning**: Gib dem Agenten eine Repräsentation:
  - der Ziele/Intentionen, die erreicht werden sollen
  - Aktionen, die er ausführen kann
  - der Umgebung

  $\rightarrow$ Der Agent nutzt die Repräsentationen, um einen Plan zu generieren.
- Problem: Planung und Ermittelung, welches Ziel als nächstes erreicht werden soll, kosten Zeit.
- Dadurch kann eine Situation entstehen, in der der Agent ein Ziel erreichen will, das nach dessen Ermittelung nicht mehr optimal ist.
- Unter folgenden Annahmen ist die getroffene Entscheidung noch immer optimal: Wenn Planung und Ermittelung schnell genug sind; die Welt sich nicht verändert hat; getroffene Zielentscheidung ist noch immer optimal, wenn Agenten Plan gefunden hat;
- Agenten Algorithmus formal 1:
  1. B = B_0 - Ausgangs-Beliefs
  2. führe unendlich lange aus:
  3. beobachte Umwelt - p
  4. B = brf(B,p) - Aktualisierung des der eigenen Beliefs
  5. I = deliberate(B) - Ermittelung

6. π = plan(B,I) - Plane
7. execute(π) - führe Plan aus

- Ermittelung durch: option-generation (Erstelle mögliche Ziele) und filter (Auswahl des Ziels)
- option-generation: Nutzt aktuelle Beliefs und Intententions und ermittelt daraus eine Menge von Optionen (Desires)
- filter: Der Agent wählt zwischen verschiedenen Alternativen aus den Optionen und beginnt die gewählte Option zu verfolgen
- Agenten Algorithmus formal 2 - BDI Agent:
  1. B = B_0 - Ausgangs-Beliefs
  2. führe unendlich lange aus:
  3. beobachte Umwelt - p
  4. B = brf(B,p) - Aktualisierung des der eigenen Beliefs
  5. D = options(B,i) - Desires (Ziele)
  6. I = filter(B,D,I) - Ermittelung der Intentionen durch Beliefs(Wissen über Umwelt), Desires(Ziele) und Intentions(gewählten Ziel)
  7. π = plan(B,I) - Plane
  8. execute(π) - führe Plan aus
- Begriffserklärungen:
  - Beliefs - Weltwissen (Alles was wir wissen d.h. was wir über die Welt, unsere Fähigkeiten und Zielen glauben)
  - Desires - Ziele (Optionen, die wir gerne erfüllt hätten)
  - Intentions - Absichten (Ziel, die ich jetzt erreichen möchte)

3.5 Commitments

- Strategien:
  - Blind Commitment - Agent will Intention erreichen, bis er glaubt die Intention erreicht zu haben
  - Single-minded Commitment - Agent will Intention erreichen, bis er glaubt diese erreicht zu haben oder es nicht mehr möglich ist
  - Open-minded Commitment - Agent will Intention erreichen, bis er glaubt diese erreichen zu können
- Anpassungen des Agenten Algorithmus
  1. wie bisher $\rightarrow$ Blind Commitment
  2. wie bisher - Ermittle Plan und solange dieser nicht leer ist führe diesen schrittweise aus, nach jedem Schritt hole neuen Percept + Belief Update: wenn Ziel nicht mehr erreicht werden kann ermittle neuen Plan $\rightarrow$ Single-minded da keine neuen Intentionen erwogen werden
  3. wie bisher - Ermittle Plan und solange dieser nicht leer ist führe diesen Schrittweise aus solange Intention nicht erreicht und dieser noch nicht unmöglich ist, hole nach jedem Schritt neuen Percept + Belief Update + Desires ermitteln + Intentions. Wenn das Ziel nicht mehr erreicht werden kann, ermittle neuen Plan.
- Problem: Agent will Intentions verfolgen, selbst wenn schon klar ist, dass das Ziel nicht mehr erreicht werden kann. Der Agent will konstant berücksichtigten, dass evtl. unnötige Zeit verschwendet wurde und deshalb nie das Ziel erreicht werden kann.
- Lösung: Meta-Level Control, das entscheidet, wann der Agent seine Intention verwirft

4 Kooperation von Agenten

4.1 Einführung

- Ein Multi-Agenten-System besteht aus mehren Agenten, die
  - durch Kommunikation interagieren
  - mit der Umgebung interagieren können
  - unterschiedliche Einflussbereiche auf die Umgebung haben
  - in Beziehungen zu einander stehen (Organisation)
- Egoistische Agenten bergen Potential für Konflikte da, jeder seine Ziele verfolgen möchte
- Bei Überlappungen der Umgebungen, können Agenten interagieren: direkt oder indirekt (Bspw. wie Ameisen mittels Endorphinen)
- Lösung: Kooperation mittels Entscheidungstheorie

4.2 Entscheidungstheorie - Spieltheorie

- Wir haben zwei Agenten A_i und A_j
- Jeder Agent hat die gleiche Menge $\Omega = \{\omega_1, \omega_2, \dots\}$ - eine Menge von Ergebnissen, die für den Agenten von Bedeutung sind.
- Die Bedeutung eines Ergebnisses für einen Agent i wird durch eine Utilty-Funktion repräsentiert $u_i : \Omega \rightarrow \mathbb{R}$ (jeder Agent hat eine Utility-Funktion)
- Utility-Funktion bildet Ordnung der Ergebnisse: $\omega \geq_i \omega^{'}$ bedeutet also $u_i(\omega) \geq u_i(\omega^{'})$
- Utility beschreibt den Nutzen, nicht das GELD!
- Modell für mehrere Agenten:
  - Agenten wählen Aktion gleichzeitig aus, $\rightarrow$ als Ergebnis etwas aus Ω
  - Das aktuelle Ergebnis hängt von den Kombinationen der Aktionen ab
  - Jeder Agent hat nur zwei mögliche Aktionen: C und D
  - $\tau : A_c \times A_c \rightarrow \Omega$ eine Funktion, welche die Aktionen von den beiden Agenten auf ein Ergebnis abbildet
  - Eine Welt lässt sich mittels τ beschreiben (unter Verwendung aller möglichen Kombinationen von Aktionen) $\rightarrow$ hat ein Ergebnis
  - Jeder Agent kann die Utility von dem Ergebnis berechnen
- Darstellung mittels Payoff Matrix (Ein Agent ist Spalte (rechter Eintrag), ein Agent ist Zeile (linker Eintrag)

	D	C
D	1,1	1,4
C	4,1	4,4

- Beispiel: $\tau(D, C) = \omega_1$ mit $u_i(\omega_1) = 1$ und $u_j(\omega_1) = 4$

4.3 Strategien

- Dominante-Strategie
  - Aus einer gegebenen Strategie s (Beispiel: D oder C) für Agent i ergeben sich verschiedene Ergebnisse (ω)
  - Strategie s_1 dominiert Strategie s_2, wenn jedes Ergebnis das s_1 spielt gegenüber s_2 bevorzugt wird.
  - Rationale Agenten werden niemals dominierte Strategien spielen, d.h. es gäbe eine Strategie, die besser ist
  - Ziel: dominierte Strategie eliminieren
  - Anmerkung: es ist nicht immer möglich, nur eine nicht-dominierte Strategie zu finden.
- Nash-Equilibrium (Nash-Gleichgewicht):

  - Strategie s_i und s_j sind in einem Nash-Gleichgewicht gdw: Unter der Annahme, dass Agent i s_i spielt und kann Agent j nicht besser sein, als wenn er s_j spielt **UND** unter der Annahme, dass Agent j s_j spielt und kann Agent i nicht besser sein, als wenn er s_i spielt
  - Aber: nicht jede Interaktion hat ein Nash-Gleichgewicht und einige Interaktionen haben mehrere Nash-Gleichgewichte

- Pareto Optimum:
  - Gegeben: eine initialen Zuordnung von Gütern für eine Menge von Agenten
  - Eine Änderung, die für einen Agent besser ist aber für keinen anderen Agenten schlechter nennt sich Pareto-Verbesserung
  - Eine Zuweisung ist Pareto-Optimal gdw. keine weiteren Pareto-Verbesserungen gemacht werden können

- Kompetitive- und Null-Summen-Interaktionen
  - Szenarien in denen Agenten gegenläufige Präferenzen haben, sind streng kompetitiv
  - Null-Summen Spiele gdw. die Summe aller Utilitys der Agenten = 0 sind: $u_i(\omega) + u_j(\omega) = 0, \forall \omega \in \Omega$
  - Null-Summen Spiele sind streng kompetitiv, im Real-Life selten

- Beispiel: Prisoners Dilemma
  - Zwei Männer wurden wegen Diebstahls verhaftet
  - Wenn einer gesteht und der andere nicht, geht der Schweigende 5 Jahre ins Gefängnis und der andere kommt frei
  - Wenn beide gestehen, gehen beide für 3 Jahre ins Gefängnis
  - Wenn keiner gesteht, geht jeder 1 Jahr ins Gefängnis
  - Nash-Equilibirum gdw. jeder gesteht
  - Besser wäre aber wenn jeder schweigt

- Beispiel: Tragedy of the commons - Ein Artikel der das Problem beschreibt, dass keiner verantwortlich ist. Die negativen Konsequenzen für alle sind verzögert, die positiven Effekte wirken für den einzelnen jedoch sofort.

- Grundlegendes Problem von Multi-Agenten Interaktionen: Es wird keine Kooperation entstehen, wenn jeder Agent egoistisch ist

- Lösung: Annahme mein Gegenspieler ist meine Zwilling, oder Shadow of the Future (nochmaliges treffen in der Zukunft)

- Iterated Prison Dilemma
  - Lösung: Spiele das Spiel mehrmals
  - Rückwärts Induktion: Annahme wir spielen n mal, in Runde n-1 wollen wir schweigen für höheren Payoff, dadurch wird n-2 zur letzten richtigen Runde wo wir auch schweigen würden für höheren Payoff ... Prisoners-Dilemma mit fixer Anzahl an Runden ist schweigen immer die beste Strategie
  - Untersuchung in Axelrods Tournament: Iterated Prison Dilemma gegen verschiedene Gegner
  - Verschiedene Strategien: ALLD (always defect), TIT-FOR-TAT (Kooperation in der ersten Runde, danach immer die Aktion welche der Gegner gespielt hat), TESTER, JOSS

4.4 Benevolent Agents - Gutmütige Agenten

- Wenn wir das ganze System kontrollieren, können wir Agenten bauen, die sich gegenseitig unterstützen.

- Es gelten Konventionen über das Verhalten.

- Unter dieser Annahme: Unsere Agenten sind mehr oder weniger gutmütig, d.h. unser bestes Ziel ist deren bestes Ziel.

- Gutmütige Agenten vereinfachen das System-Design.

- Gutmütige Agenten haben eigene Interesse, die aber nicht mit anderen Interessen kollidieren.

5 Anwendung des Prisonersdilemma: Sensornetzwerk Beispiel

5.1 Wireless Sensor Networks

WSN sind Netzwerke aus kleinen, batteriebetriebenen Geräten. Die Knoten bestehen aus folgenden **Komponenten**:

- Low-power Prozessor
- Begrenzter Speicher
- Funk (geringe Reichweite + Datenrate)
- Sensoren
- Stromquelle (Batterie)

Welche **Motivation** und **Anwendung** für WSN gibt es?

- Übergang vom PC in die reale Welt
- *No configuration*
- Große Anzahl von Sensoren in einer großen Umgebung
- Verteiltes *sensing* mittels **Wireless Sensing** und **Data Networking**
- Waldbrandüberwachung, Fahrzeugüberwachung, Flutkontrolle, etc...

Vor welche **Herausforderungen** wird man gestellt?

- Sensoren in Bezug auf Energie und Kommunikation stark eingeschränkt
- Anwendungsfälle setzen gewisse Ansprüche an **Latenz**, **Datendurchsatz**, **Lebenszeit**, welche mit **Netzwerkkapazität** und **Energiehaushalt** in Konflikt geraten

⇒ Nutzung von z.B. S-MAC

5.2 Sensor Medium Acces Protocol (S-MAC)

Stellt die drei Hauptfeatures (siehe Teilüberschriften)

5.2.1 Periodisches Zuhören und Schlafen

- Knoten sparen Energie, indem sie nicht dauerhaft aktiv sind, Verhalten ist abhängig von anwendungsabhängigem **Schedule**.
- Schedules werden mit Nachbarn **synchronisiert**, damit Kommunikation stattfinden kann (sofern nötig). Knoten lernen die Schedules ihrer Nachbarn.
- Initial lauscht ein Knoten, ob er einen Schedule erhält. Ist dies nicht der Fall, wählt er einen (zufälligen) Schedule und broadcasted ihn (**Synchronizer**)
- Erhält er einen Schedule vom Nachbarn, so verfolgt er diesen und broadcasted ihn nach einer zufälligen Zeit. (**Follower**)
- Hat ein Knoten bereits einen eigenen Schedule und erhält einen weiteren, so wacht er nach seinem eigenen und dem Nachbarschedule auf. Schedule wird vor dem Schlafen gebroadcasted.
- Uhren müssen regelmäßig synchronisiert werden (Periodisches `SYNC`-Paket)

5.2.2 Kollisions- und Überhörvermeidung

- Collision Avoidance mittels Ready-to-Send/Clear-to-Send (RTS/CTS), wie bei WLAN (802.11)
- **Kommunikationsschema:** ⇒ `RTS`, ⇐ `CTS`, ⇒ `<DATA>`, ⇐ `ACK`
- Übertragungsdauer wird immer im Voraus angekündigt, also haben andere Konten in der Zeit Funkstille

5.2.3 Nachrichtenübermittlung

Nachrichten sind bedeutsame, zusammengehörige Einheiten von Daten. **Problem:**

- Lange Nachrichten als einzelnes, großes Paket zu übermitteln verursacht hohe Kosten bei Paketverlust (re-transmission)
- Fragmentierung in kleine Pakete bedeutet Overhead (RTS/CTS für jedes Paket)

Die Lösung ist die **Fragmentierung** der Nachricht in kleine Pakete und **Übertragung im Burst-Modus**.

Vorteile

- Reduktion von Latenz
- Reduktion von Overhead

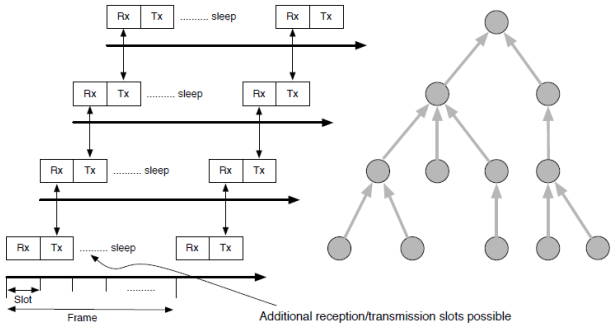
Nachteil

- Knoten mit kleinen Nachrichten müssen warten, bis lange Nachrichten übermittelt sind.

5.3 Win Stay Loose Shift Strategie

Ziel: Weiterleiten sämtlicher Sensordaten an einen Basisknoten mit minimaler Ende-zu-Ende-Latenz. Folgende **Grundidee:**

- Baumstruktur mit Sensoren als Knoten und Basisknoten als Wurzel, gerichtet zur Wurzel
- Schedules sind so abgestimmt, dass ein Paket sofort weitergeleitet werden kann, sobald es von einem Knoten empfangen wurde
- Ein Frame (Active-Sleep-Periode) wird in Slots aufgeteilt; ein *Sende-Slot* ist mit dem *Empfangen-Slot* des Nachfolgers abgestimmt.
- Wenn dieser Stichpunkt hier nicht steht, wird die nachfolgende Überschrift hässlich umgebrochen.



5.3.1 DEcentralized SYnchronization and DEsynchronization (DESYDE)

- Knoten sind Agenten, die ihre Nachrichten mit minimaler Latenz zur Senke (Basis) weiterleiten sollen.
- **Mögliche Aktionen**: `Transmit`, `Listen` und `Sleep`, ausführbar in einem periodischen Frame, bestehend aus mehreren Slots (genau eine Aktivität pro Slot)
- Für jede ausgeführte Aktivität gibt es Feedback von der Umgebung; **Mögliche Feedbacks** (Nutzen in Klammern): `Successful transmission` (1), `successful reception` (1), `overhearing` (0), `idle listening` (0), `unsuccessful transmission` (0) und `collision` (0)
- *Mathematische Definitionen zu Frame, Actions, Reward Signal und Expected Reward etc. ausgelassen*
- Die Knoten lernen das dann *irgendwie* innerhalb von 3-4 Frames.

5.4 Reinforced Learning ?

6 Agenten Kommunikation

6.1 Einführung

- Kommunikation ist der Austausch von Informationen zwischen Agenten.
- Für Kooperation ist es notwendig zu kommunizieren: Speech Acts, KQML & KIF, FIPA ACL

6.2 Speech Acts

- Behandlung der Kommunikation bei Multi-Agenten-Systemen ist häufig an die Speech-Act-Theory angelegt
- Alles wird geäußert, um ein Ziel oder eine Intention zu erfüllen. Da der Effekt der Kommunikation in einem anderen Agenten stattfindet, hat der eigentliche Agent keinen direkten Einfluss.
- Speech Act Theorie: Wie Äußerungen genutzt werden, um Ziele zu erreichen
- Verschiedene Typen von Speech Acts (Searle):

  Representatives: informieren zb. *Es regnet.*

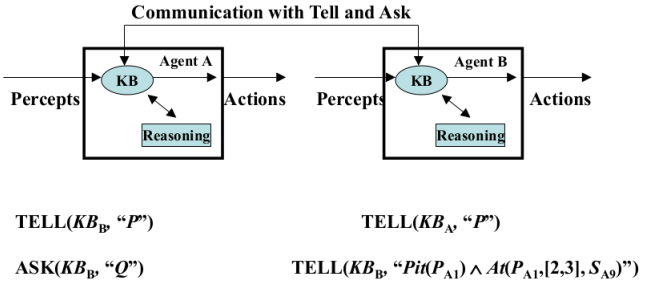
  Directives: versucht den Hörer zu etwas zu bewegen zb. *Bitte mach mir Tee!*

  Commissives: Sprecher verspricht etwas zu tun zb. *Ich verspreche, dir einen Tee zu machen*

Expressives: Sprecher teilt seinen mentalen Zustand mit z.b Danke

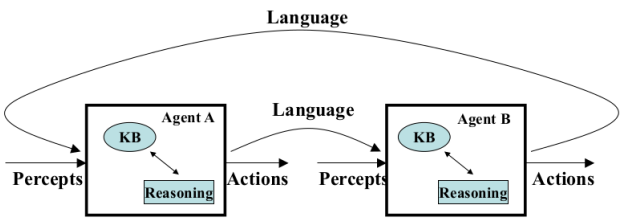
Declarations: wie *Krieg erklären* oder *Taufe*

- Jeder Speech-Act hat zwei Komponenten:
  performative Verb (anfordern, informieren, ...) + propositional content (die Tür ist verschlossen)
- Communication as Action: den Hörer zu einer Aktion zu bewegen
  - Versprechen: Sachen anbieten oder Sachen zu tun (Promise)
  - Anfrage: Anderen Agenten etwas über die Gruppe fragen (Query)
  - Bitte: Anderen Agenten bitten etwas zu tun (Request)
- Probleme mit Kommunikation: Wann soll kommuniziert werden? Welcher Speech-Act ist der richtige für die aktuelle Situation?
- Probleme mit Verstehen: Welche Situation kann dieses Speech-Act erzeugt haben?
- Komponenten der Kommunikation:
  - am Sprecher
    1. Intention: S möchte dass H an P glaubt
    2. Generation: S wählt Worte W
    3. Synthese: S kommuniziert die Worte W
  - am Hörer
    1. Perception: H empfängt W^1 (ideal $W^1 = W$)
    2. Analysis: H schließt aus W^1 mögliche Bedeutungen $P_1, \ldots, P_n$
    3. Disambiguation: H schließt, dass S P_i mitteilen möchte (ideal $P_i = P$)
    4. Intercorporation: H entschließt P_i zu glauben (oder verwirft es, wenn es nicht mit dem aktuellen Glauben zusammen passt)
- Modelle der Kommunikation
  - **Encoded Message Model**: Sprecher kodiert die Nachricht in Wörtern, Der Hörer versucht die Nachricht zu dekodieren; Die Bedeutung der Nachricht, die übertragene Nachricht und die Interpretion sind gleich (falls keine Probleme der Kommunikation auftreten)
  - **Situated Langage Model**: Bedeutung der Nachricht beruht nun auf er Nachricht und der Situation
- Typen von kommunizierenden Agenten
  - Kommunikation unter Verwendung von **Tell und Ask**: Agenten teilen sich eine gemeinsame interne Repräsentation einer Sprache; Kommunikation ohne Verwendung einer externen Sprache



and ask.png

  - Kommunikation unter Verwendung von **Formalen Sprachen**: Agenten machen keine Annahmen über die interne Repräsentation der Sprache; Agenten teilen eine gemeinsame Sprache für die Kommunikation.



- Größtes Problem: Kommunikation ist zweideutig; Lösung unter Berücksichtigung des Kontextes und der vorigen Kommunikationen
- Definition der Semantik auf Basis von Planung: bestimmte Prädikate, die in der jeweiligen precondition-delete-add Listen verwaltet werden
- Beispiel: request(s,h,ϕ)
  - pre: s glaubt h kann ϕ tun; s glaubt h glaubt h kann ϕ; s glaubt s will ϕ
  - post: h glaubt s glaubt s will ϕ

6.3 KQML und KIF

- Nun Betrachtung von Agent Communication Languages (ACLs) d.h. dem Standardformat für den Austausch von Nachrichten
- Bekannteste ACL: KQML (Knowledge Query and Manipulation Language)
- Zwei Bestandteile: Wissens-Abfrage und -Manipulations Sprache (KQML) und Wissens-Austausch Format (KIF)
- KQML ist eine äußere Sprache, die verschiedene akzeptierbare Verben (performatives) definiert (Bsp: ask-if, perform, tell, reply)
- KIF (knowledge interchange format)ist eine Sprache für die Repräsentation des Nachrichteninhaltes (meistens ähnlich zu Common Lisp)
- Für die Kommunikation müssen die Agenten das gleiche Verständnis von Ausdrücken haben (Was bedeutet Schalter1?), formale Definition in einer Ontologie

6.4 FIPA

- Neue Entwicklung der Agenten Standards incl. ACL
- Basisstruktur ist ähnlich zu KQML: Permative + Housekeeping + Content
- INFORM und REQUEST sind die zwei Standard Permative in FIPA; der Rest wird auf Basis dieser definiert
- Bedeutung von INFORM und REQUEST hat zwei Parts: pre-condition (muss wahr sein damit der Speech Act erfolgreich ist), rational effect (Ziel, das der Sender der Nachricht erreichen möchte)
- Beschreibung der pre-condtions usw als logischer Ausdruck

6.5 Ontologien

- Situation: Service Oriented Computing
- Alle Teilnehmer haben ein gemeinsames, gleiches Verständnis von Begriffen über einer Domäne (Ontologie) d.h. jeder Teilnehmer weiß genau, was unter einem bestimmten Begriff zu verstehen ist
- Eine Ontologie beschreibt Begriffe und deren Relationen zu einander
- Ontologien unterstützen die Interaktion. Dadurch, dass sie Begriffen eine Bedeutung geben
- Ontologie-Sprachen erlauben dem Nutzer eine explizite und formale Kozeptionierung der Domäne
- Problem: je umfangreicher die Sprache desto schwieriger wird es aus diesen zu schließen
- Schließen in einer Ontologie
  - Klassenzugehörigkeit: Wenn x in der Klasse C und C eine Unterklasse von D, ist x auch in der Klasse D
  - Klassenäquivalenz: Wenn Klasse A ist äquivalent zu B und B zu C denn ist A äquivalent zu C
  - Konsistenz: Wir haben X Instanzen von A und B aber A und B sind disjunkt d.h. Fehler in der Ontologie
  - Klassifikation: gegeben Eigenschaft-Werte-Paare für Klasse A, wenn x die Kriterien erfüllt ist x in A

- Schließen ist wichtig für: Überprüfung der Ontologie und des Wissens; Überprüfung, ob ungewollte Beziehungen der Klassen entstanden sind; automatische Zuordnung von Instanzen zu Klassen
- Überprüfungen sind wertvoll für: den Entwurf von großen Ontologien mit mehren Autoren; Integration und Verteilung anderer Ontologien an/von verschieden Quellen

7 Wie sollten Agenten kommunizieren? Strategien und Protokolle

7.1 Einführung

- Kommunikation ist wichtig für die Verteilung von Aufgaben(Aufteilung von Teilaufgaben) und Ergebnissen (Teilergebnisse)
- Protokolle regeln die Interaktion zwischen Agenten

7.2 Contract Net

- Realisierung von Multi-Cast Kommunikation in Agenten-basierten Systemen
- Zwei Rollen: Selector und Contractor
- Contract Net: Aufgabenverteilungs-Protokoll für Aufgabenzuweisung
- Schritte:
  1. Recognition: Feststellung des Problems
  2. Announcement: Mitteilung des Problems an alle Teilnehmer
  3. Bidding: Teilnehmer antworten mit den erwarteten Kosten
  4. Awarding: Auftragsgeber teilt einem Teilnehmer die Aufgabe zu
  5. Expediting: Ausführung
- Recognition
  - Agent nimmt Problem wahr
  - Agent hat Ziel und merkt, dass er das Ziel nicht alleine erreichen kann (Nicht genügend Kapazität) bzw. dass er dieses Ziel nicht alleine erreichen möchte (Deadline, Lösungsqualität)
- Announcement
  - Agent sendet Aufgabe incl. Spezifikation der Aufgabe an alle Teilnehmer
  - Spezifikation beinhaltet: Aufgabenbeschreibung, Anforderungen (Deadlines, Qualität der Lösung), Meta-Informationen
  - Announcement ist ein Broadcast
- Bidding
  - Agenten empfangen Announcement und entscheiden sich, ob sie für die Aufgabe bieten möchten
  - Faktoren: Agent muss entscheiden, ob er in der Lage ist die Aufgabe zu bewältigen; Agent muss die Contraints ermitteln ggf. Preis
  - Wenn Agent mitbieten möchte, sendet er ein Gebot (tender) an Aufgabensteller
- Awarding & Expediting
  - Aufgabensteller muss aus den Geboten auswählen und entscheiden wer belohnt werden soll
  - Ergebniss wird den teilnehmenden Agenten mitgeteilt
  - Erfolgreicher Contractor (Teilnehmer) führt die Aufgabe aus
  - Ggf. Erweiterung der Contractor Beziehung um weitere Unterbeziehungen: sub-contracting
- Problem: Spezifikation von ...
  - Aufgaben: gemeinsame Sprache der Agenten
  - Lösungsqualität: Anforderungen (Ontologien), Werbungen und Reputation
  - Auswahl von konkurrierenden Angeboten: Präferenzen

8 Verhandlungen

8.1 Einführung

- Wie können Agenten eine Übereinkunft treffen, wenn sie sich egoistisch verhalten?
- Worst-Case: Null-Summen-Spiele d.h. keine Übereinkunft möglich, ABER meistens besteht eine Möglichkeit zum Treffen einer kurzzeitigen Übereinkunft - zum allgemeinen Interesse.
- Fähigkeiten der Verhandlung notwendig
- Verhandlungen werden durch einen Mechanismus (Protokoll) gesteuert
- Mechanismus = Regeln für das Treffen von Agenten
- Mechanismus-Design: Entwicklung von Mechanismen mit bestimmten Eigenschaften
- Mechanismus-Design Eigenschaften:
  - Konvergenz und garantierter Erfolg
  - Maximierung des Gemeinwohles
  - Pareto-Effizienz
  - Individuelle Rationalität
  - Stabilität
  - Einfachheit
  - Verteilung

8.2 Verhandlung - Auktionen

- Auktionen finden zwischen einem Auktionator und einer Menge von Bietern statt
- Ziel: Auktionator möchte eine Zuweisung von Gütern (Ressourcen) an einen Bieter erreichen
- Meistens möchte Auktionator den maximalen Preis erreichen, die Bieter den minimalen
- Auktions-Parameter
  - Güter haben: Privaten Wert, Allgemeinen Wert, Korrelierten Wert
  - Gewinnermittelung durch: erstes Gebot, zweites Gebot
  - Gebote sind: für alle sichtbar (open cry); verborgen (sealed bid)
  - Gebotsabgabe: einmalig (one shot), aufsteigend, absteigend

8.2.1 Englische Auktion

- Höchstes Gebot gewinnt, für alle sichtbar, aufsteigend
- Dominante Strategie: minimale Erhöhung des höchsten Gebotes bis Obergrenze erreicht, falls diese überschritten: Rückzug
- Anfällig: Fluch des Gewinners (Bezahlt meistens zu viel), Lockvögel (Agent arbeitet mit Auktionator zusammen und treibt den Preis künstlich in die Höhe)

8.2.2 Holländische Auktion

- Offene Gebote, absteigend
- Auktionator startet mit hohen Startgebot
- Auktionator senkt den Preis, bis ein Agent ein Gebot zu dem Preis abgibt
- Gewinner: Agent mit der Preisabgabe

8.2.3 First-Price Sealed-Bid Auction

- Einmaliges verborgenes Gebot
- Eine Runde
- Bieter sendet Gebot
- Bieter mit höchstem Gebot gewinnt
- Gewinner bezahlt Preis des höchstem Gebotes
- Beste Strategie: biete weniger als den eigentlichen Wert

8.2.4 Vickrey Auction

- Second preis, sealed bit
- Gewinner mit dem höchstem Gebot zum Preis vom zweit-höchstem Gebot
- Beste Strategie: Preisabgabe zum wirklichen Wert
- Überbieten wird dominiert durch das Bieten des echten Wertes

→ Wenn ein Bieter mit einem höheren Wert als die anderen gewinnt, hat dieser wohl überboten; wenn Bieter eines niedrigeren Wertes als andere Bieter antritt, hat er verloren obwohl überboten hat oder nicht

- Unterbieten wird dominiert durch Bieten des echten Wertes

→ Wenn ein Bieter zu gering bietet verliert er; wenn ein Bieter zu hoch bietet wird er gewinnen

- Anfällig für antisoziales Verhalten
  - Agent A beziffert den Wert bei 90$ und weiß, dass Agent B 100$ bieten würde.
  - Agent A kann Zuschlag also nicht bekommen.
  - Stattdessen bietet Agent A 99$ um den Preis für Agent B in die Höhe zu treiben und diesem so zu schaden.

8.3 Probleme

- Auktionen sind anfällig für Lügen vom Auktionator und Absprache von Bietern
- Alle Auktionen sind können durch Absprache der Bieter manipuliert werden
- Ein böser Auktionator kann bei der Vickrey Auktion bezüglich es zweit höchsten Gebots lügen
- Shills (Lockvögel) können bei der Englischen Auktion den Preis in die Höhe treiben
- Anwendung von Auktionen: Lastverteilung, Routing, Koordination

8.4 Heterogene und selbst-motivierende Agenten

- Kein zentrales Design
- Es gibt keinen globalen Nutzen
- Sind Dynamisch d.h. neue Typen von Agenten können leicht hinzugefügt werden
- Agenten sind nicht wohlwollend solange sie es nicht sein wollen d.h. Agenten kooperieren erst, wenn es nötig ist
- Notwendig: Entwickler einigen sich auf Standards, wie Agenten in Domäne zu agieren haben
- Abstimmung von Möglichkeiten und Tradeoffs für Protokolle, Strategien und die sozialen Regeln der Agenten
- Eigenschaften von Standards:
  - Effizienz: Pareto Optimal
  - Stabil: kein Grund vom optimal abzuweichen

  - Einfach: gerade Berechnungs- und Kommunikationskosten
  - Verteilt: kein zentraler Koordinator
  - Symmetrisch: jeder Agent spielt äquivalente Rolle
- MAS: Gruppe von Nutzen-maximierende heterogenen Agenten, die koexistieren in der gleichen Umgebung, evtl. Konkurrenz
- Auktionen als Verhandlungsmechanismus für die Teilung von Ressourcen
- In Abhängigkeit von Auktion wählt jeder Agent eine andere Strategie
- Möglich: andere Szenarien für Verhandlungen außer die Verteilung von Ressourcen

8.5 Verschiedene Domänen

- Aufgaben orientierte Domäne: Agenten wollen Aufgaben erledigen → Aufgabenverteilung
- Zustand orientierte Domäne: Ziele sind bestimmte finale Zustände der Welt → gemeinsamen Plan
- Wert orientierte Domäne: Zustände haben einen Wert → gemeinsamen Plan- und Ziel-Realisierung

8.6 Aufgaben orientierte Domäne

- Tripe: $< T, Ag, c >$
- T ist eine endliche Menge von möglichen Aufgaben
- $Ag = \{1, \ldots, n\}$ ist eine Menge von beteiligten Agenten
- $c = p(T) \rightarrow R^+$ definiert die Kosten für die Ausführung jeder Teilmenge von Aufgaben (p ist Potenzmenge)
- Ein Encounter ist eine Menge von Aufgaben: $< T_1, \ldots, T_n >$ mit $T_i \subseteq T$ für jedes $i \in Ag$
- Bestandteile: Domäne, Verhandlungsprotokoll, Verhandlungsstrategie
- Gegeben ein Encounter, dann ist $< T_1, T_2 >$ ein Deal für die Zuweisung $T_1 \cup T_2$ bezüglich der Agenten 1 und 2
- Die Kosten für den Agenten i eines Deals: $\beta =< D_1, D_2 >$ ist $c(D_i)$ (nachfolgend als $cost_i(\beta)$)
- Utility eines Deals β des Agenten i ist mit $c(T_i)$ die ursprüngliche Zuweisung einer Aufgabe für Agent i: $utility_i(\beta) = c(T_i) - cost_i(\beta)$
- Conflict deal Ω ist ein Deal $< T_1, T_2 >$ mit der ursprünglichen Zuweisung an Aufgaben: $utility_i(\Omega) = 0, \forall i \in Ag$
- Ein Deal ist individuell rational, wenn dieser den conflict deal schwach dominiert
- Agenten nutzen Produkt-maximierende Verhandlungsprotokolle
- The Monotonic Concession Protocol ???
  - Verhandlung in Runden
  - Runde 1: Agenten schlagen gleichzeitig einen Deal aus dem Verhandlungsset vor
  - Übereinkunft, wenn ein Agent einen Deal findet, der mindestens so gut ist wie die von anderen vorgeschlagenen
  - Wenn keine Übereinkunft nächste Runde
  - In Runde $u + 1$ darf kein Agent einen schlechteren Deal machen als der in der vorigen Runde
  - Wenn nach einigen Runden keine Übereinkunft gefunden wird: Dann wird die Verhandlung mit dem Conflict-Deal beendet
- Zeuthen Strategie
  - **Was sollte der erste Vorschlag sein?** — der beste Deal aus Sicht des Agenten
  - **Welcher Agent sollte ein Zugeständnis machen?** —
    * Agent mit der kleinsten Risikobereitschaft bezüglich Konflikt (der, der am meisten bei einem Konflikt zu verlieren hat)

* $r_i = \frac{\text{Nutzenminderung durch Zugeständnis}}{\text{Nutzenminderung durch Konflikt}} = \frac{u_i(d_i) - u_i(d_j)}{u_i(d_i)}$

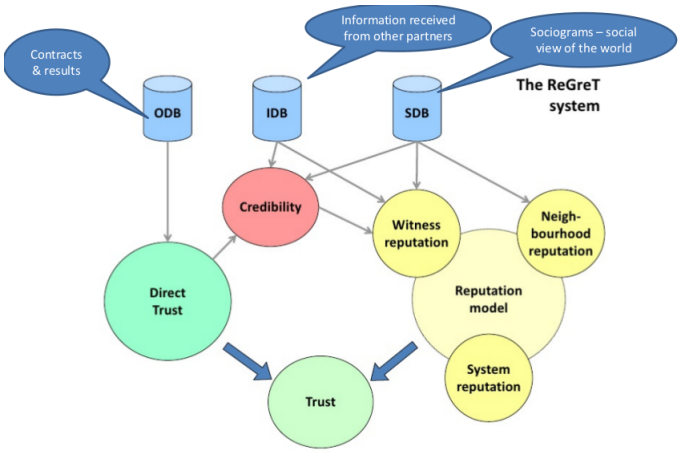
- **Wie groß sollte das Zugeständnis sein?** — gerade groß genug, so dass er beim nächsten Mal nicht der mit kleinster Risikobereitschaft ist
- Die Zeuthen Strategie ist ein **Nash-Gleichgewicht**: Wenn ein Agent ihr folgt, kann der andere nicht besser handeln als ihr auch zu folgen.

9 Trust and Reputation

9.1 Einführung

- Trust: Der Glaube, dass ein anderer Agent eine Aktion tun wird, **ohne** dass **explizite Garantien** vergeben werden, um ein Ziel in einer riskanten Situation zu erreichen.
- Reputation: Was andere Agenten bezüglich seines Verhaltens sagen; *opinion or view of one about something* → global.
- Reputation erlaubt die Bildung von Trust und hat so eine soziale Komponente, die nicht nur für einen Agenten sondern für alle nützlich ist. Um Reputation aufzubauen ist Kommunikation notwendig und ähnlich wie Ontologien erleichtert diese den Alltag.
- Jeder Agent wird durch andere Agenten beobachtet, keine zentrale Autorität
- Trust und Reputationsmechanismen basieren auf den zwei Layern: Sicherheit (Integrität, etc.) und Institution (Organisationsaspekte wie Abläufe oder Protokolle)
- Subjective vs. Globale Reputation
  - Subjektiv: Jeder Agent hat eine eigene Vorstellung von der Reputation anderer Agenten
  - Global: Reputation als eine zentrale Ressource, auf die alle Agenten Zugriff haben und die gleichen Reputationswerte erhalten.
  - Vorteil global: Reputation ist auch für neue Agenten verfügbar; einfacher für Agenten, da diese Werte nicht berechnet werden müssen
  - Nachteil global: Funktioniert nur unter der Annahme, dass alle Agenten denken und sich ähnlich verhalten, da interne Zustände des Agenten nicht berücksichtigt werden; Es ist nicht immer erwünscht, dass Agenten Information public machen oder diese an eine zentrale Verwaltungsstelle senden
  - Ein hoher Trust in die zentralen Organisation ist notwendig
- Vgl. Ebay: komplett zentral, Käufer hinterlassen Kommentare + Bewertung nach Käufen, jeder Teilnehmer hat einen Wert als Summe der Bewertungen, viele Nutzer, einfacher Identitätenwechsel
- Beispiel: Regret System

9.2 Regret System



- Modulares Trust und Reputations-System für komplexe E-Commerce Umgebungen, in denen soziale Relationen zwischen den Teilnehmern eine wichtige Rolle spielen

- Es kann auch auf andere Eigenschaften als Trust/Reputation ankommen bspw. auf Aspekte wie Qualität oder schnelle Lieferung
- Die Folien und die beiden Papers beschreiben die Ansätze teilweise ziemlich unterschiedlich und widersprechen sich zum Teil sogar:
  - Anschauliches Paper: `http://www.iiia.csic.es/files/pdfs/591.pdf`
  - Mathematisches Paper: `http://www.iiia.csic.es/~sierra/articles/2001/reputation.pdf`
- Agenten stehen in Relationen zueinander, z.B.

  Competition — Agenten verfolgen dasselbe Ziel und benötigen gleiche Ressourcen. Es wird versucht, gegenüber dem Anderen einen Vorteil zu gewinnen, beispielsweise durch Lügen oder das Verstecken von Informationen.

  Cooperation — Agenten haben das Interesse, zusammenzuarbeiten. Daher wird oft angenommen, dass der Andere vertrauenswürdig ist. Schließt sich mit *Competition* aus

  Trade — Es existieren kommerzielle Transaktionen. Kompatibel mit *Competition* und *Cooperation*
- Jeder Agent kennt zu jeder Relation ein **Soziogramm**, also einen gewichteten Graphen mit Agenten als Knoten und Kanten, wenn eine entsprechende Beziehung besteht. Die Gewichte der Kanten geben an, wie stark die jeweilige Relation (vermutlich) ausgeprägt ist ($w \in [0, 1]$).
- **Kernkonzept:** Regret unterscheidet bei der Reputation zwischen der **Individuellen Dimension**, der **sozialen Dimension** und der **ontologischen Dimension**.

9.2.1 Ontologische Dimension

Statt *der* Reputation eines Agenten, gibt es mehrere Typen der Reputation. So kann ein Agent die Reputation haben, ein *Schwindler* zu sein oder *häufig die Qualitätsansprüche zu übertreffen*. Diese Typen können auch zusammengesetzt sein. Ein Agent, der *nicht zahlt* fällt ebenso in die Kategorie *Schwindler*, wie einer der *nicht liefert, obwohl er bezahlt wurde*.

9.2.2 Individuelle Dimension

- Modelliert die Reputation aus der direkten Interaktion zwischen zwei Agenten. Dies ist die beste Methode, um eine Reputation zu einem anderen Agenten zu berechnen, ist jedoch nicht immer möglich, wenn keine oder nicht genügend Interaktionen stattgefunden haben.
- Beide Papers scheinen unterschiedliche Ansichten zu haben, wie man auf die individuelle Reputation kommt.
- Agenten speichern **Outcomes**, also Paare von Verträgen und den dazugehörigen tatsächlichen Ergebnissen in einer **Outcome Database (ODB)**.
- Outcomes werden vom Agenten bezüglich bestimmter Eigenschaften (z.B. Lieferzeit eingehalten, Qualitätsansprüche erfüllt etc.) bewertet. Beide Paper beschreiben hierfür unterschiedliche Ansätze(Impressions, Issues und Grounding Relation). **Hinweis**: Die im *mathematischen Paper* genutzte **Impression Database (IDB)** steht im Widerspruch zur IDB in der Grafik aus den Folien!
- Die Reputation eines Agenten bezüglich bestimmter Eigenschaften ist (vereinfacht) die Summe der relevanten Bewertungen von Outcomes.
- Die **Verlässlichkeit** dieser Reputation ist Abhängig von der *Anzahl* und der *Streuung* der individuellen Erfahrungen.

9.2.3 Social Dimension

- Wenn keine (oder nicht genügend) eigene Erfahrungen verfügbar sind, um eine Reputation zu einem Agenten zu berechnen, muss das soziale Umfeld befragt werden. Da dieses sehr groß sein kann, muss eine Teilmenge aller Agenten ermittelt werden, die befragt wird. Kurzfassung:

  Witness Reputation — Befragen von Agenten, die idealerweise schon Kontakt mit dem Agenten hatten und damit schon eine Reputation über diesen kennen.
  - Zeugen können lügen oder Informationen auslassen, um dem anfragenden Agenten zu schaden oder den Zielagenten zu schützen.

- Weiterhin tritt *correlated evidence* auf, weil Agenten aufgrund derselben Ereignisse bewerten und sich gegenseitig beeinflussen.
- Es müssen weiterhin nicht alle Zeugen befragt werden. Ein vorgeschlagener (stark vereinfachter) Algorithmus zum Finden von geeigneten Zeugen nimmt ein Soziogramm aller Zeugen, identifiziert die zusammenhängenden Komponenten und wählt aus diesen jeweils die Cut-Points (Knoten, deren Wegnahme die Komponentenzahl erhöht) und, wenn in einer Komponente keine Cut-Points vorhanden sind, die Knoten mit der höchsten Gradzahl (Central Points).
- Das Vertrauen in den Zeugen spielt eine Rolle und muss ermittelt werden. Hierbei gibt es **social trust** und **outcome trust reputation**.
  * Beim social trust werden die Beziehungen zwischen anfragenden Agenten, Ziel und Zeuge betrachtet und anhand von Fuzzy-Logik bewerter: `IF` $coop(w_i, b)$ `is high THEN socialTrust is very_bad`
  * Die bessere Alternative, sofern möglich, ist die Bewertung des Zeugen anhand eigener Outcomes mit diesem.

Neighbourhood Reputation — ?

System Reputation — Standardwert anhand der Rolle des Agenten, Vorurteile (z.B. *Studenten kommen zu spät*)

9.3 Fuzzy-Logik

Kommt dann ins Spiel, wenn etwas unscharf ist. Fuzzy Logik ist eine Theorie, welche vor allem für die Modellierung von Unsicherheit und Vagheit von umgangssprachlichen Beschreibungen entwickelt wurde. In der Vorlesung wurde als Beispiel die Einteilung in große bzw. kleine Körpergröße gemacht. Es ist unklar ab wann jemand groß ist und wann klein.

Fuzzy Logik tritt dann auf, wenn eine Einschätzung unklar/verwischt/verschwommen/unbestimmt ist. Gegenbeispiel: Tot oder lebendig

10 Blackboard

Hilfreiche Papers:

- `http://mas.cs.umass.edu/pub/paper_detail.php/218` (im Skript angegeben)
- `http://www.aaai.org/ojs/index.php/aimagazine/article/view/537` (yay, Koalas!)

10.1 Die Blackboard Metapher

Steht so nicht in den Folien, jedoch in der Zusammenfassung der Folien und sollte ausreichend sein, um die Grundidee ausführlicher zu beschreiben. Ein **Blackboardsystem** ist durch drei Hauptkomponenten charakterisiert: Eine Gruppe von Experten (**Knowledge Sources — KS**), einer gemeinsam verwendeten Tafel mit (teilweise gelösten) Problemen (**Blackboard — BB**) und einem Moderator (**Controller — C**).

Es kann viele, unterschiedliche, unabhängige und asynchron arbeitende KS geben. Die KS sollen ein definiertes Problem lösen. Dazu suchen sie die Tafel nach für sie lösbaren Problemen ab, lösen diese und schreiben die Lösung an die Tafel.
Corkill beschreibt folgende Charakteristiken:

Independence of expertise (I think, therefore I am.) — KS arbeiten eigenständig und sind daher voneinander unabhängig.

Diversity in problem-solving techniques (I don't think like you do.) — KS sind Spezialisten auf ihren Gebieten. Das Problem wird in verschiedene Fachgebiete unterteilt (z.B. Fahrzeugbau → Maschinenbauingenieure, Elektroingenieure....) und die KS sollten entsprechend gewählt sein.

Flexible representation of blackboard information (If you can draw it, I can use it.)

Common interaction language (What'd you say?) — Gemeinsame Sprache ist verfügbar, was ein KS auf das BB schreibt, kann von einem anderen KS verstanden werden. Mitunter kann es vorkommen, dass eine Sprache nur von einer Teilmenge von KS verstanden wird.

Positioning metrics (You could look it up.) — Inhalt des BB kann von den KS effizient erfasst werden. So können etwa Bereiche des BB thematisch aufgeteilt sein, sodass eine KS nur einen bestimmten Bereich beobachten muss.

Event-based activation (Is anybody there?) — KS können Ereignisse (sowohl auf BB, als auch extern) wahrnehmen und darauf reagieren.

Need for control (It's my turn.) — Es wird eine Kontrollinstanz C benötigt. Diese lenkt die Aktionen der KS, damit das Ziel letztendlich erreicht werden kann.

Incremental solution generation (Step by step, inch by inch...) — KS nehmen ein Problem vom BB, lösen es und schreiben die Lösung auf das BB zurück. Die Lösung wird von anderen KS als Input verwendet. So wird die Lösung an der Tafel inkrementell verbessert, bis letztendlich die Lösung des großen Problems an der Tafel steht.

10.2 Model of Problem Solving

Knowledge Sources Unabhängige Systeme zum Lösen von Problemen aus bestimmten Domänen. Verstehen den aktuellen Zustand im Problem-Lösungsprozess und die relevanten Informationen am BB. Kennen außerdem die Vorbedingungen für ihre möglichen Aktionen (*Was muss an der Tafel stehen, damit ich aktiv werden kann?*)

Blackboard Globale Datenstruktur, die für alle KS verfügbar ist. Auf ihr werden Rohdaten, Teillösungen, Alternativen, Endlösungen uns Steuerinformationen festgehalten. Dadurch ermöglicht das BB Kommunikation und Aktivierung von KS. Kann einen Flaschenhals darstellten.

Control Component Entscheidet, welche KS auf Veränderungen auf dem BB reagieren dürfen und lenkt so die Lösungsfindung.

10.3 Aus den Folien gekratztes Wissen

- Kommunikation über ein gemeinsames Medium hatten wir vorher schon einmal: Ameisen
- Schema zum kooperativen Problemlösen über eine gemeinsame Datenstruktur
  Im Gegensatz zur direkten Kommunikation via *casts und Protokolle, sind Blackboard-Systeme eine indirekte Kommunikationsform.
- Merkmale
  - BB können als Datenspeicher, Experten Systeme oder als Programmierumgebung genutzt werden
  - Agenten können auf dem BB Teillösungen schreiben/lesen, da es eine geteilte Datenstruktur ist. Es werden Teillösungen auf das BB geschrieben.
  - Da kein zeitgleicher Zugriff möglich ist, ist gegenseitiger Ausschluss notwendig.
  - Unnötige Datenduplikationen werden vermieden und es kann entschieden wer die Kontrolle über das BB hat.
  - Eine Kommunikation der KS (mit Hinblick auf Daten) erfolgt via BB.
  - Nachteil ist jedoch, dass das BB sich zu einem Flaschenhals entwickeln kann
- Subscribe/Notify Pattern
  - Um Ergebnisse mit anderen Agenten zu teilen wird dieses OO-Pattern verwendet.
  - Ein Objekt kann sich bei einem anderen anmelden, um über ein Ereignis informiert zu werden.
- Grundsätzlich ist ein BB eine multidimensional Datenstruktur. Es ist aus Knoten aufgebaut, die wiederum Datenfelder beinhalten.
- Die Kooperation erfolgt durch **Hypothese-und-Test**: Teillösungen sind die Hypothesen, die anschließend getestet werden.
- Alternativen für das Problem
  - Pipe and Filter
    - Filter benötigen kein Wissen darüber, an was sie angebunden sind.
    - Sie können parallel laufen
    - Das Verhalten des Systems basiert auf dem Verhalten der einzelnen Filter
  - Direkte Kommunikation ist erheblich aufwändiger und führt zu:
  - Objektorientierte Architektur

    - Interessante Eigenschaften: Data Hiding (Datenrepräsentation für Clients nicht einsehbar), Wahl zwischen single- und multi-threaded und Probleme lassen sich zerlegen und an eine Menge von Agenten delegieren
    - ABER miteinander agierende Objekte müssen sich gegenseitig kennen
    - Lässt sich verfeinern zu Client/Server/Brocker → Schnittstelle/Middleware
  - Layered System: Wenn die Struktur es zulässt. Durch verschiedene voneinander gelöste Layer, kann der Designprozess, die Wiederverwendung und weitere Verbesserungen unterstützt werden. Es können Interfaces standardisiert werden. Aber es können nicht immer klare Layer erkannt werden.
  - Event-Based
    - Die Sender von Events müssen sich nicht damit auseinandersetzen, wer damit wie umgeht. Es ermöglicht eine einfachere Wiederverwendung und Weiterentwicklung des Systems - einfaches Hinzufügen von weiteren Agenten.
    - ABER: Die Komponenten haben keine Kontrolle über das Ordnen der Berechnungen.
  - Model View Controller: Es gibt ein zentrales Model und viele Views. Zu jedem View gehört ein Controller, der die Eingaben des Nutzers auf dem View an das Model updatet. Änderungen am Model werden an alle Views verteilt.

- Hearsay II
  - Eins der ersten akademischen BB-Systeme zur Spracherkennung.
  - Da viele Team parallel zu einander gearbeitet haben, sollen diese partiellen Lösungen auf dem BB arbeiten und daraus eine globale Lösung generieren. So entwickelte sich ein umfangreiches System zur Spracherkennung.
  - Es ist kein Expertensystem, das alles kann - dafür die Subsysteme.

- Sinnvolle Anwendung von BB
  - Entkopplung: Bspw. Sensoren in einem Raum
  - Mehere Akteure in einer Welt: So können Bedingungen global gespeichert werden, wenn sie mehrere betreffen
  - Gut zur Koordination zwischen Agenten → NOLF2 Beispiel

- Probleme von verteilten Netzwerken: Wie oft muss aktualisiert werden? Mit wie viel falschen Wissen kann noch gearbeitet werden?

- Probleme bei der Koordination zwischen Agenten:
  - Agenten machen Sachen gleichzeitig
  - Agenten machen Sachen zu oft
  - Spezielle Bedingungen für Taktiken
  - Agenten nehmen die gleichen Wege
    Lösung: Rasterung/Einteilung des Gebietes. A erhöht Kosten, wo er langläuft und verhindert so dass B dort auch lang läuft.s
  - Agents clump at destinations

11 Organisationskontrolle

11.1 Allgemein

- Organisationskontrolle (Wer kann mit wem interagieren) kann Kommunikation und Kooperation vereinfachen (kürzere Entscheidungshierarchie, weniger Kommunikation)
- Strukturierung als zentrale Aufgabe reflektiert die Aufgabe und sorgt für eine effizientere Lösung der Aufgabe
- Organisationskontrolle funktioniert, wenn ein Problem zerlegbar und häufig wiederholbar ist
- Framework für die Skalierung der Aufgaben
- Ziel: Wissen lokal zu halten, damit nicht zusätzlicher Aufwand durch Verwaltung fremden Wissens entsteht
- Organisations Tradeoff
- Die Rollenzuteilung zur Steigerung der Effizienz kann von der Natur abgeschaut werden. Bspw. das Flugkonzept von Vögeln → Arbeitslastverteilung

11.2 Selbstorganisierende Rollenänderungen

- Beim Senken-organisierten Verfahren pollt die Senke regelmäßig die Knoten und übernimmt die Zuweisung von Aufgaben
- Bei der Selbstorganisation benachrichtigt ein Knoten den Clusterhead, sobald sein Energielevel unter einen bestimmten Wert gefallen ist. Dieser verteilt anschließend die Aufgabe an Knoten mit einem höheren Energielevel.
- Ergebnis: Erhöhrung der Lebenserwartung um 40%
  (Limitierte Faktoren sind Senden, Messen und Rechnen)

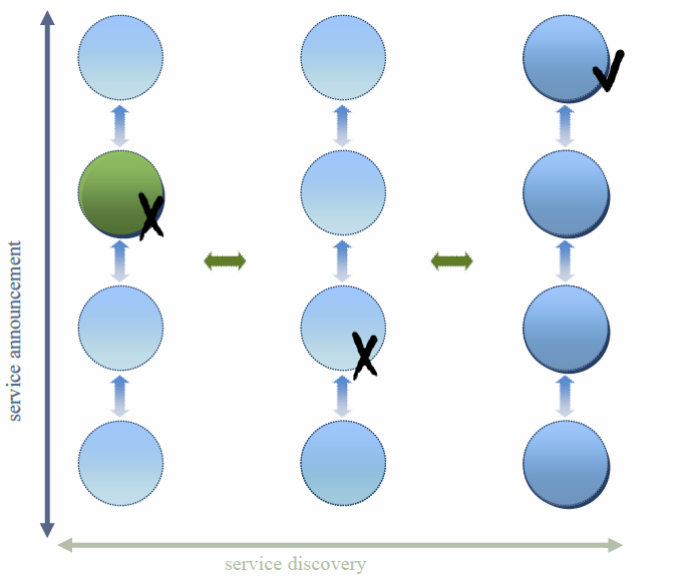
11.3 Kontrolle von Distributed Sensor Networks)

- Wenn bei vielen Sensoren/Agenten keine Organisation dahintersteht, dann kann viel via Broadcasts gesendet werden; Wer entscheidet, ob ein Ziel neu ist?; Wer verfolgt das Ziel? → Diese Entscheidungen können zwar individuell getroffen werden, sind aber durch eine globale Organisation einfacher.
- Rollenzuteilung der Knoten: Sensor, Sector-Manager, Track Manager
- Die Umgebung wird in Sektoren eingeteilt und je Sektor wird ein Sektor-Manager bestimmt. Die anderen Knoten eines Sektors senden ihre Kapazitäten an den Sektor-Manager, der daraufhin Scan-Pläne entwickelt.
- Knoten die für das Scannen eingeplant sind, machen selbiges und teilen dem Sektor-Manager Beobachtungen mit. Es wird ein Track-Manager bestimmt.
- Die Aufgabe des Track-Manager ist es die Tracking-Nodes zu finden und zu koordinieren. Es kann hierbei zu Konflikten mit anderen Aufgaben kommen. Der Track-Manager sollte ab einem bestimmten Punkt in einen anderen Sektor migrieren.

11.4 Content-Based Hierarchical Agent Organizations

- Agenten mit ähnlichem Inhalt werden gruppiert. Dabei ist die Größe der zu durchsuchenden Teilgruppen limitiert. Es werden Querverweise gesetzt, um schnell Inhalt zu finden.
- Die Organisation vergrößert sich inkrementell, wenn neue Agenten joinen.
- Die Suche läuft in 2 Phasen ab: Cluster mit gesuchten Inhalt lokalisieren → Cluster durchsuchen

11.5 Lane-Protokoll



- ???

Um **Fehlverhalten** zu analysieren, muss zuerst definiert werden, was ein Fehlverhalten ist. Außerdem kann Fehlverhalten nach schwere des Vergehens klassifiziert werden. Als nächstes muss das Fehlverhalten detektiert werden - dies verursacht jedoch Aufwand, der vermieden werden sollte.
Detektion:

1. Identifizierung von kritischen Nachrichten
2. angemessene Reaktionen überprüfen
3. Trust-Wert anpassen

11.6 How to Create an Organization

- Top-Down **oder** Emergent/Self-Organizing **oder**
- Kombination
- Typen von Agenten (kooperativ, eigennützig, semi-kooperativ)
- Rollen (Gruppen, Regeln, Rollen)

12 Agenten und Mobilität

- Rückblick - Gebiete zusammengefasst
  - KI (Planung, Kommunikation, Wissen, Logik)
  - Soziologie (Wie Menschen sich verhalten)
  - **Jetzt:** Verteilte Systeme (unabhängig, own Thread of Computiation, Garbage Collection in MAS nicht möglich, distributed Location)
- Verteilte Orte: Wie können diese miteinander reden?
- Verteilte Systeme: verschiedene Möglichkeiten um Methoden von anderen Koten (Remote Procedure Call) aufzurufen
- Lösungen: Stub, Skeleton. Marshalling (Serialisierung von Objekten), IDL
- Marshalling: Serialisieren, sodass darauf referenziert werden kann
- Stub hat Referenz zu Skeleton-Methodes
- Probleme:
  - Client kann blockieren
  - Verbindung kann nicht sicher sein (fällt ggf aus), Implementierung von Timeouts usw (unreliable links)
  - Übertragung von großen Daten kann ein Problem sein. Es ist besser, wenn der Server Berechnungen mit Daten vornimmt - ohne zu verschicken ABER möglicherweise kennt der Server den Algorithmus nicht (Lösung: Mobile Agents)
- Agenten-basierter Ansatz: Alternative zu RPC ist RP (remote programming). Der Client macht hierfür Methoden und Klassen verfügbar. Die Methode wird zusammen mit dem Zustand des Agenten auf dem Server ausgeführt. (?)
- Client macht Code für Prozeduren und Klassen bekannt
- Prozedur + Aktuellen Zustand präsentiert einen Mobiler Agent
- Weak migration: ist heutzutage Standard
  (Code&Date werden migriert. Es ist notwendig einen Startpunkt zu definieren und es ist komplizierter damit zu arbeiten)
- Strong migration
  (Code&Data&State(Programming Stack) Das Programm kehrt zu einem Interrupt-Point zurück. Da ein Agentenzustand groß sein kann, ist dies ein langsamer und teurer Prozess)
- Remote Programming: Konventionen zwischen Client und Server für Befehle und Datentypen bilden eine Sprache
- Vorteile von mobilen Agenten: Reduzierung der Netzwerklast, Offline-Ausführung von Agenten möglich, Höhere Clientflexibilität, Bessere Lastverteilung bei Berechnungen

- Notwendige Infrastruktur: transportierbare Sprache in der die Agenten entwickelt wurden (Java), Interpreter, Kommunikation zum Agentenaustausch, Agenten
- **Gründe für Java:** Plattformunabhängigkeit, Mechanismus zum Laden von Klassen, Multi-Threading, Serialisierung, RMI, Reflektion **ABER** keine strong migration
- Migration darf nicht im Selbstmord enden: Daher wird ein „Shadow“ hinterlassen. Außerdem müssen Nachrichten an die neue Position weitergeleitet werden (neue Location wird in der alten gespeichert). Erst nach erfolgreichen Migration wird das alte Objekt gelöscht, im Fehlerfall wird es re-installiert
- Agenten können reflektieren, wer sie zu was auffordert - Objekte können das nicht
- Agenten müssen selbstständig festlegen, wann sie garbage collected werden können.
- Sicherheitsaspekte -¿ Authentizität (Spoofing), Integrität(Veränderungen), DOS, ...
  → Authentifikation (User, Host, Agent, Code), Access Control, Vertraulichkeit
- Hostsicherung: Safe Code Interpretation (interpretierte Sprachen), Authentifizierung, Authorisierung, Ressourcen Allocation, Path-History
- Mobile-Agenten-Sicherung: safe Environment, Signaturen, shared Secret