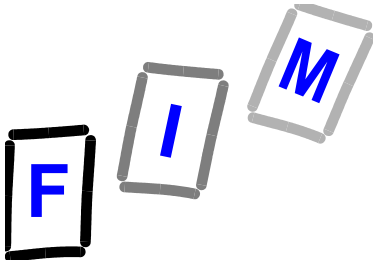
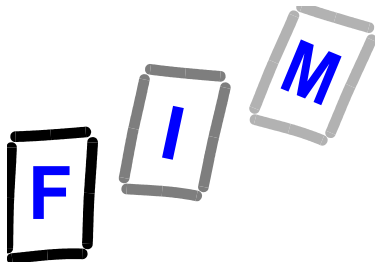


Mag. Dipl.-Ing. Dr. Michael Sonntag

Netzwerke und Agenten

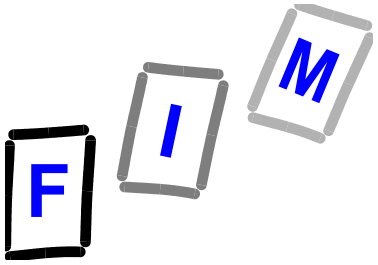
Kommunikation zwischen Agenten, Sicherheit

E-Mail: sonntag@fim.uni-linz.ac.at
WWW: http://www.fim.uni-linz.ac.at/staff/sonntag.htm



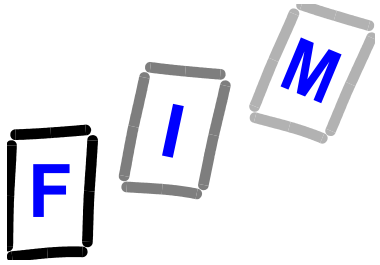
Fragen?

Bitte gleich stellen!



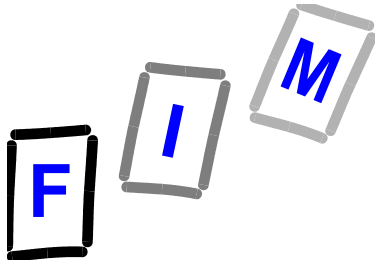
Inhalt

- **Kommunikation zwischen Agenten**
  - ➔Allgemeines, Klassifikation, Ontologien
  - ➔“Eigentliche” Agenten-Kommunikationssprachen
    - » FIPA, KQML
  - ➔Sprachen zur Wissensdarstellung
    - » RDF, (DAML) DAML+OIL
    - » Dublin Core
    - » OWL
- **Sicherheitsaspekte**
  - ➔Wer muß wovor geschützt werden, Sicherheitsmodelle
  - ➔Angriffe auf Agentensysteme
  - ➔Abwehrmöglichkeiten



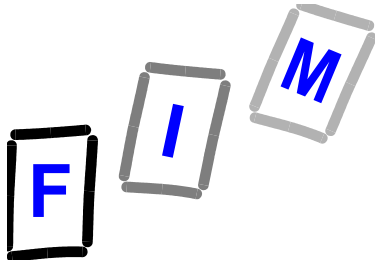
Agenten-Kommunikation Allgemeines (1)

- **Wichtig für Agenten**
  - ➔**Eine Definition: Kommunikativität ist einziges Merkmal**
- **Problem: Gemeinsames Verständnis**
  - ➔**Syntax wichtig, sonst unverständlich**
    - » **Syntax kann abstrakt beschrieben werden**
      - – Meta-Syntax, Meta-Meta-Syntax, …
    - » **Meist kein so großes Problem; Konvertierungen möglich**
  - ➔**Semantik muß aber auch festgelegt werden!**
    - » **Sehr schwer abstrakt zu beschreiben!**
      - – Sehr komplex, vielschichtig, Umgebungsabhängig, ...
    - » **Meist nur eine umgangssprachliche Beschreibung**
      - – Für direkte Verwendung/Prüfung durch Computer ungeeignet!



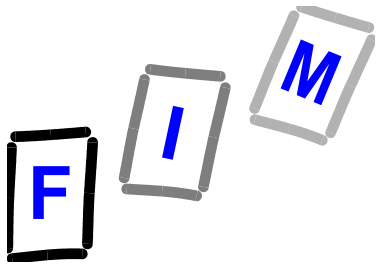
Agenten-Kommunikation Allgemeines (2)

- **Resultat des Semantik-Problems:**
  - ➔Meist genaue Beschreibung der Syntax
  - ➔Zusätzlich Umschreibung der jeweiligen Bedeutung
  - ➔Versuch, Vor-/Nachbedingungen exakt zu definieren
- **Beispiele:**
  - ➔HTML/XML/… - Standards
    - » Syntax exakt; Bedeutung in Worten dabei
  - ➔XML Schema
    - » Teilweise auch Bedeutung dabei (Datentypen)
      - – Das ist aber nur die unterste Schicht: Gut, ein Integer; aber ist die Temperatur jetzt Grad Celsius oder Grad Fahrenheit?



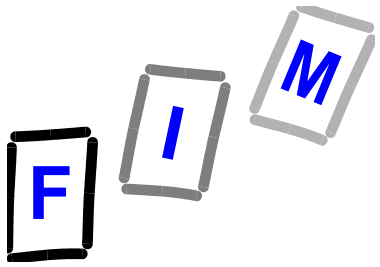
Agenten-Kommunikation Klassifikation (1)

- **“Eigentliche” Agenten-Kommunikationssprachen**
  - ➔**Entworfen speziell mit Agenten im Hintergrund**
  - ➔**Kommen oft aus Entwicklung von Agentensystemen**
  - ➔**Meist prozedural**
  - ➔**Eher Anwendungsbezogen**
- **Sprachen zur Wissensdarstellung**
  - ➔**Kommen aus dem Bereich der KI**
  - ➔**Dienen dazu, Wissen auszutauschen/speichern**
  - ➔**Meist funktional**
  - ➔**Eher abstrakt/mathematisch**



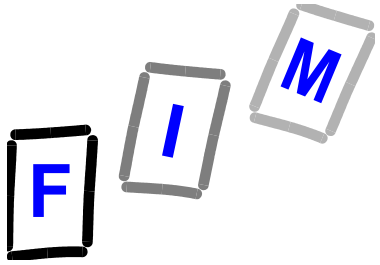
Agenten-Kommunikation Klassifikation (2)

- **Gemischtes**
  - ➔**Allgemeine Protokolle, die auch von Agenten zur Kommunikation benutzt werden**
    - » **Feste Anwendungsprotokolle, die auch von Agenten bedient werden; z. B. SMTP, HTTP, ...**
  - ➔**“Rahmen”-Sprachen**
    - » **Beschreiben nicht den eigentlichen Inhalt, sondern die Art der Kommunikation bzw. den Kommunikationsablauf; z. B. SOAP**



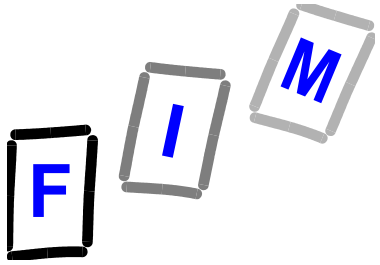
Agenten-Kommunikationssprachen

- **Was ist alles nicht enthalten:**
  - **➔Physische Übertragung**
    - » **Protokoll hängt vom Agentensystem ab (idealerweise!)**
  - **➔Adressierung**
    - » **Namensverzeichnis, …: Agentensystem**
      - – Meist wird eine einheitliche eindeutige ID vorausgesetzt
  - **➔Bedeutung (zumindest nicht die des Inhalts)**
    - » **Hängt vom einzelnen Agenten bzw. der Applikation ab**
    - » **Sollte so allgemein sei, daß alles mitgeteilt werden kann**
  - **➔Schnittstelle**
    - » **Wie werden Nachrichten abgeschickt bzw. empfangen?**



Agenten-Kommunikationssprachen

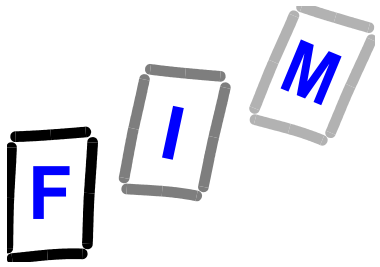
- **Was ist enthalten:**
  - ➔**Syntax der Nachrichten**
    - » **Wie werden Nachrichten ausgebaut, welche Zeichensätze sind erlaubt, wie werden Länge/Ende gekennzeichnet, ...**
  - ➔**Grundlegende Felder**
    - » **Zeit, Weg, … enthalten oder muß dies in die Nachrichten hinein?**
  - ➔**Zusatzfunktionen**
    - » **Protokollierung, garantierte Zustellung, Wiederholungen, …**
    - » **Digitale Signaturen, Verschlüsselung**
  - ➔**Kommunikationsmodell**
    - » **Synchron oder Asynchron?**
- **Allgemeine Abgrenzung aber schwierig!**



Ontologie - Was ist das???

- Basis-Vokabular
- Bedeutung der Wörter
- Regeln für Bedeutung von Wort-Kombinationen

- Ermöglicht Unabhängigkeit von konkreten Systemen, da einmal allgemein definiert
  - ➔Es existieren aber viele Alternativen zu jedem Gebiet!
- Befaßt sich immer nur mit einem relativ kleinen Bereich für eine einzelne Anwendung
- Beschreibt die Interpretation einer Nachricht



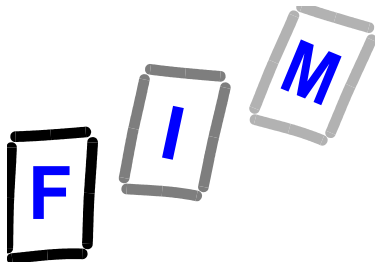
Ontologie-Beispiel

- **Ontologie 1 (Beleuchtung):**
  - ➔**Birne** (=Leuchtkörper)
  - ➔**Einsetzen** (=montieren)
  - ➔**Birne einsetzen** (=in die Fassung einschrauben; geht nur, wenn Fassung vorhanden ist und keine Strom)
- **Ontologie 2 (Garten):**
  - ➔**Birne** (=Obstbaum)
  - ➔**Einsetzen** (=pflanzen)
  - ➔**Birne einsetzen** (=Baum pflanzen; geht nur zu bestimmter Jahreszeit)

Bedeutung

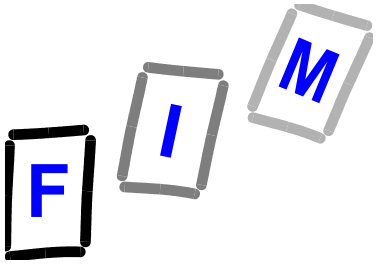
Bedeutung von Kombinationen (z. B. Nebenbedingungen)

Vokabular: Birne, Einsetzen



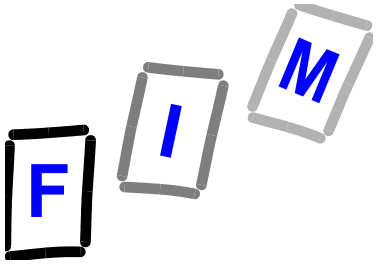
KQML Grundstruktur (1)

- **Basiert auf “performatives”: Anweisungen, etwas bestimmtes zu tun**
  - **➔Dies sind die erlaubten Einflußmöglichkeiten auf andere Agenten**
  - **➔Sprech-Akt-Theorie**
  - **➔Unterteilt in Gruppen**
- **Zur Unterstützung der Zusammenarbeit dienen “Facilitators”:**
  - **➔Verzeichnis von Agenten**
  - **➔Nachrichten-Weiterleitung**
  - **➔Finden von Agenten mit bestimmten Fähigkeiten**



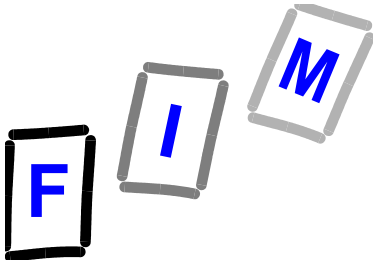
KQML Grundstruktur (2)

- **Basiert auf dem BDI-Modell**
  - ➔**Performatives sollten “beliefs” oder “goals” sein**
- **Kann mit jeder Ontologie verwendet werden**
- **Syntax ist ähnlich zu LISP**
- **Protokolle werden NICHT beschrieben, nur die Nachrichten, die verwendet werden**
  - ➔**Ausnahme: Fehler-Rückmeldung (“Nicht verstanden”)**
- **Benötigt zusätzlich eine Sprache (Ontologie) für den Anwendungs-Inhalt der Nachrichten**
  - ➔**Z. B. KIF (Knowledge Interchange Format)**



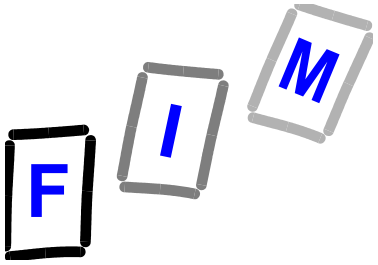
KQML Grundstruktur (3)

- **Eigentlicher Nachrichten-Inhalt: Ausdrücke**
  - ➔In einer nicht näher spezifizierten Sprache, Syntax, Semantik!
  - ➔Ontologie ist explizit bei jeder Nachricht anzugeben
- **Keine globale Wahrheit**
  - ➔Jeder Agent besitzt seine eigene “Weltanschauung”
  - ➔Diese kann sich von der anderer Agenten unterscheiden
  - ➔Gegenseitige Beeinflussung durch Mitteilungen
    - » **Müssen aber nicht ernstgenommen werden!**



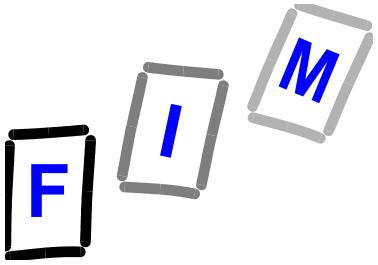
KQML
Performatives - Gruppen

- **Discourse**
  - ➔Anfragen: ask-if, ask-one, ask-all, ...
  - ➔Mitteilungen: tell, untell, insert, uninsert, …
  - ➔“Befehle”: achieve, unachieve, …
  - ➔Ereignisse: advertise, unadvertise, subscribe, ...
- **Intervention and Mechanics**
  - ➔Fehler, Warten, Bereit, ...
- **Facilitation and Networking**
  - ➔Registrieren bei Namensserver (ID + Fähigkeiten)
  - ➔Anfragen nach passenden Agenten



KQML
Einzelne Performatives

- **ask-if:**
  - ➔Fragt den Empfänger ob der Ausdruck für ihn wahr ist
  - ➔Entweder als Faktum oder mit seinem Wissen beweisbar
- **tell:**
  - ➔Teilt dem Empfänger mit, daß der Ausdruck für den Sender wahr ist
- **insert (rein intern):**
  - ➔Empfänger **soll** Ausdruck als wahr akzeptieren
- **achieve (Umgebungs-Beeinflussung):**
  - ➔Empfänger soll den Ausdruck zu erreichen versuchen



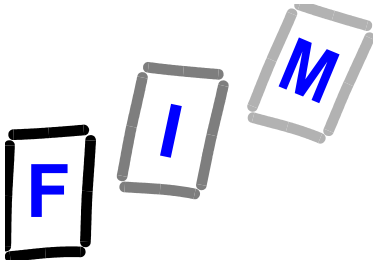
KQML Beispiel

- **Drehzahl neu stellen (Lüfter 1, 3500 U/min)**

```
(achieve  :sender        A
          :receiver      B
          :in-reply-to   id1
          :reply-with    id2
          :language      Prolog
          :ontology      motors
          :content       “speed(fan1,35)” )
```

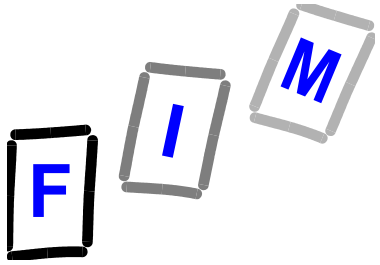
- **Antwort (nach Vollzug)**

```
(tell     :sender        B
          :receiver      A
          :in-reply-to   id2
          :reply-with    id3
          :language      Prolog
          :ontology      motors
          :content       “speed(fan1,35)” )
```



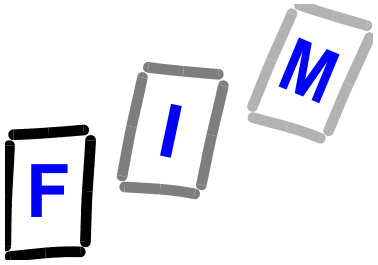
FIPA Grundstruktur

- **FIPA: Umfassende Menge an Spezifikationen**
  - ➔Analog zu KQML, basiert auch auf performativen
  - ➔String-Darstellung fast identisch!
  - ➔Darstellungen in XML, Binärformat, … definiert
- **Jedoch andere Performative**
  - ➔Bedeutung wird auch mathematisch beschrieben (eigenes sematisches Modell)
- **Grundidee: Agentensystem-Interoperabilität**
- **Namensdienst, etc. erfolgt NICHT über performatives, sondern über das API!**
  - » **D. h. kein eigener Agent dafür; sondern im System integriert**



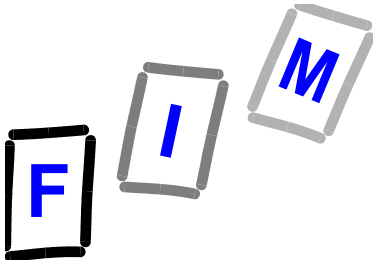
FIPA Nachrichtenelemente (1)

- performative: Zweck/Typ der Nachricht
- sender, receiver: Absender, Empfänger
  - ➔Spezifikation für Agenten-Benennung existiert
- reply-to: An wen geantwortet werden soll
- content: Eigentlicher Nutzinhalt
- language: Sprache des Inhalts
  - ➔Welcher Parser für den Inhalt benötigt wird (Prolog, …)
- encoding: Codierung des Inhalts
  - ➔Wie Inhalt aufgebaut ist: Unicode, Zeichensatz, Binhex, ...
- ontology: Bedeutungszusammenhang des Inhalts



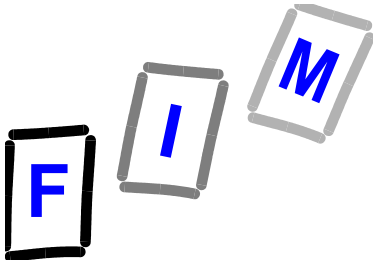
FIPA Nachrichtenelemente (2)

- **protocol: Spezifiziert das verwendete Protokoll**
  - ➔**z. B. sind bestimmte Auktionen fest definiert**
- **conversation-id: Eindeutige Instanz-Nummer**
  - ➔**Gleiches Protokoll mit gleichem Agenten: Neue Nummer**
- **reply-with, in-reply-to: Nachrichten-ID für Verkettung**
  - ➔**Vereinfacht Verfolgung gleichzeitiger Kommunikationen**
- **reply-by: Spätester Zeitpunkt für eine Antwort**
  - ➔**Spätere Antworten werden ev. nicht mehr berücksichtigt**
  - ➔**Zeit aus Sicht des Senders (keine globale Zeit)!**



FIPA Nachrichtentypen

- Agree: Zustimmung, eine Aktion auszuführen
- Call for proposal: Anfrage nach Lösungen für einen Ausdruck (Art öffentl. Ausschreibung)
- Confirm: Information, daß ein Ausdruck wahr ist
  - ➔Sender ist überzeugt: Daß es wahr ist, daß Empfänger unsicher, und daß Empfänger akzeptieren sollte
- Inform: Ähnlich wie confirm
  - ➔Aber Empfänger-Unsicherheit wird nicht vorausgesetzt
- Not understood: Nachricht unverständlich
- Request: Bitte zur Ausführung einer Aktion



FIPA Beispiel

- **Anfrage: Welche Dienste bietet Agent j an?**

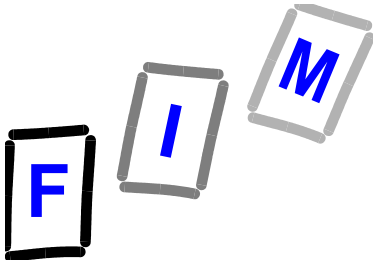
```
(query-ref      :sender (agent-identifier :name i)
                :receiver (set (agent-identifier :name j))
                :content "((all ?x (available-service j ?x)))")
```

- **Antwort: Agent j kann Flüge, Zugplätze und Autos reservieren**

```
(inform         :sender (agent-identifier :name j)
                :receiver (set (agent-identifier :name i))
                :content "((= (all ?x (available-service j ?x))
                              (set    (reserve-ticket train)
                                      (reserve-ticket plane)
                                      (reserve automobile))))")
```

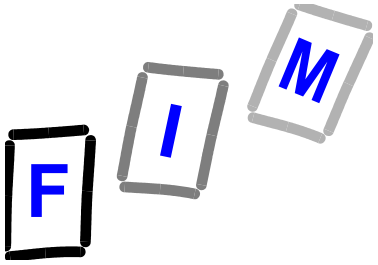
- **Ereignis-Abonnement:**

```
(subscribe      :sender (agent-identifier :name i)
                :receiver (set (agent-identifier :name j))
                :content "((iota ?x (= ?x (xch-rate EUR USD))))")
```



RDF
Grundstruktur (1)

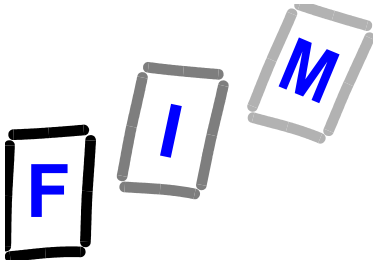
- **Format für Metadaten**
  - ➔**Hauptsächlich für WWW gedacht, aber universell einsetzbar; jetzt WWW nur mehr ein kleiner Teil**
- **Hängt stark mit XML zusammen (=Repräsentation)**
  - ➔**XML + Schema (Datentypen) + Namespaces**
- **Formale Semantik**
  - ➔**Erlaubt Einsatz von Inferenz-Regeln (Ableitung neuen Wissens aus Summe von bekanntem)**
- **RDF Schema: KEIN Vokabular!**
  - ➔**Typsystem, um Vokabulare zu definieren!**



RDF Grundstruktur (2)

- **Basiert vollkommen auf Tripeln**
  - ➔**Subject, Predicate, Object**

- **Subjekt und Objekt werden durch URI identifiziert**
  - ➔**URL (Uniform Resource Locator): Webadresse**
  - ➔**URI (Uniform Resource Identifier): Kann alles bezeichnen; muß keine Webadresse sein!**
    - » **Einzelne Knoten können auch unbenannt bleiben**
- **Container: Um mehrere Objekte einfach zusammenzufassen (Bag, Sequence, Alternative)**



RDF Aufbau

- **Aufbau von Statements:**
  - **➔<rdf:Description about="……">**

    <…….>…… </…….>

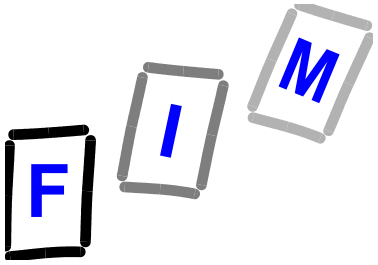
    </rdf:Description>

Subjekt

Objekt

Eigenschaft

- **Statements können zusammengefaßt werden**
  - **➔Verkürzung der Schreibweise**
  - **➔Können jederzeit wieder expandiert werden**



RDF Beispiel

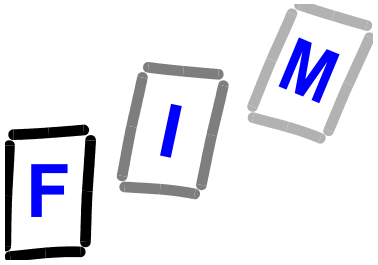
Die Resource “http://www.fim.uni-linz.ac.at/” wurde von der Person mit der Nummer 8154711 erstellt. Der Name dieser Person ist Michael Sonntag und diese hat die E-Mail Adresse sonntag@fim.uni-linz.ac.at

```
<rdf:RDF>
  <rdf:Description about="http://www.fim.uni-linz.ac.at">
    <s:Creator rdf:resource="http://www.uni-linz.ac.at/staffID/8154711"/>
  </rdf:Description>
  <rdf:Description about="http://www.uni-linz.ac.at/staffId/8154711">
    <v:Name>Michael Sonntag</v:Name>
    <v:Email>sonntag@fim.uni-linz.ac.at</v:Email>
  </rdf:Description>
</rdf:RDF>
```

S P O S P O

Verkürzte Form:

```
<rdf:RDF>
  <rdf:Description about="http://www.fim.uni-linz.ac.at">
    <s:Creator rdf:resource="http://www.uni-linz.ac.at/staffID/8154711"
      v:Name=“Michael Sonntag”
      v:Email=“sonntag@fim.uni-linz.ac.at” />
  </rdf:Description>
</rdf:RDF>
```



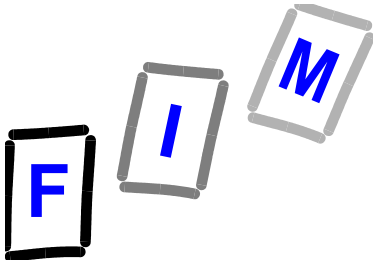
RDF Reification

- **Statements über Statements:**
  - **➔Aussage: “Michael hat diese Webseite erstellt”**

```
<rdf:RDF>
  <rdf:Description about="http://www.fim.uni-linz.ac.at">
    <s:Creator v:Name="Michael Sonntag"/>
  </rdf:Description>
</rdf:RDF>
```

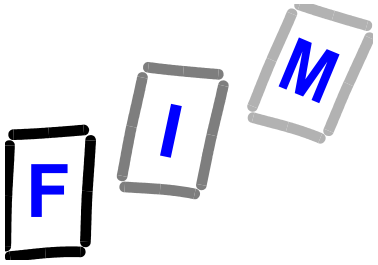
  - **➔“Susi sagt, daß Michael diese Webseite erstellt hat”**

```
<rdf:Description>
  <rdf:subject resource="http://www.fim.uni-linz.ac.at/" />
  <rdf:predicate resource="http://description.org/schema/Creator" />
  <rdf:object>Michael</rdf:object>
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
  <a:attributedTo>Susi</a:attributedTo>
</rdf:Description>
```



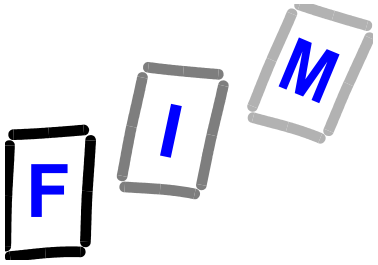
RDF Schema (1)

- **Etwas anderes als XML Schema!**
  - ➔**Spezifiziert nicht die Struktur, sondern die Bedeutung!**
- **Objektorientierter Aufbau: Klassen und Vererbung**
  - ➔**Basis: Resource (ALLES ist eine Ressource)**
  - ➔**Class: Eine Klasse**
    - » **Ableitungen: subClassOf**
  - ➔**Property: Eigenschaften einer Klasse**
    - » **Spezialisierung: subPropertyOf**
      - – Bsp: biologischerVater ist Spezialisierung von biologischemElternteil
- **Grundsätzlich Mehrfachvererbung möglich**



RDF Schema (2)

- **Einschränkungen (Constraints)**
  - ➔**range: Aus welcher/n Klasse(n) die Werte einer Eigenschaft stammen können**
  - ➔**domain: Zu welchen Klassen eine Eigenschaft gehören kann**
- **Kommentare möglich (comment, label)**
- **Modell-Elemente (Reification):**
  - ➔**Statement, subject, predicate, object, Container, ...**
- **Erweiterungen: Zusätzliche Eigenschaften werden festgelegt**
  - ➔**Beispiel: DAML+OIL**



RDF Schema Beispiel

- **Klasse:**

S

O

P

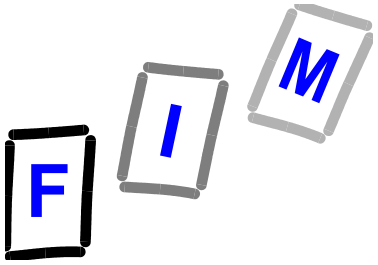
```
<rdfs:Class rdf:ID="Product">
  <rdfs:label>Product</rdfs:label>
  <rdfs:comment>An item sold by ACME Widgets Inc.</rdfs:comment>
</rdfs:Class>
```

- **Eigenschaft:**

```
<rdfs:Property rdf:ID="productNumber">
  <rdfs:label>Product Number</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdfs:Property>
```

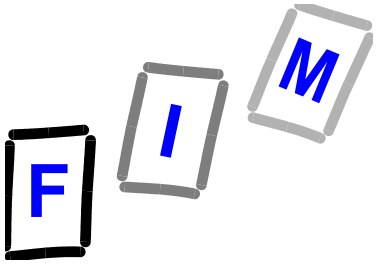
- **Instanz:**

```
<Product rdf:ID=”BestWidget">
  <rdfs:label>Mega Widget V10</rdfs:label>
  <productNumber>08/15</productNumber>
</Product>
```



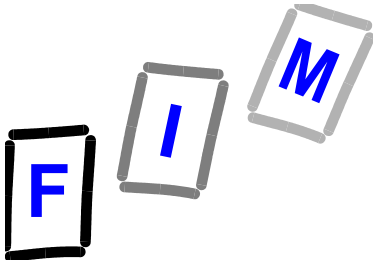
DAML Grundstruktur

- DAML = DARPA Agent Markup Language
- Erweiterung von XML und RDF
- Grundgedanke: Aus einzelnen Fakten und Regeln lassen sich automatisch neue Fakten ableiten
  - ➔Beispiel (Fakten):
    MutterVon *subpropertyOf* ElternteilVon
    Maria *MutterVon* Herbert
  - ➔Beispiel (Resultat):
    Maria *ElternteilVon* Herbert
- In XML würde dies von der Applikation abhängen und wäre NICHT in XML codiert!



DAML+OIL
Was kommt dazu?

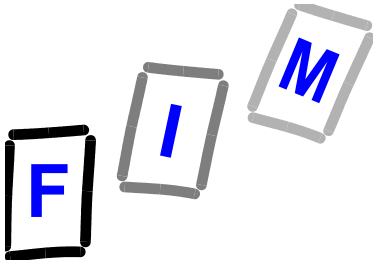
- **OIL = Ontology Inference Layer**
  - **➔Kardinalität: Eine Person kann nur einen Vater haben**
    - minCardinality, maxCardinality, ...
  - **➔Transitivität von Eigenschaften (Vorfahre)**
    - TransitiveProperty, UniqueProperty
  - **➔Mehrere Elemente bezeichnen dasselbe**
    - **» Äquivalente Klassen und äquivalente Instanzen**
      - sameClassAs, samePropertyAs, sameIndividual As
  - **➔Neue Klassen aus Vereinigung/Schnittmenge von Klassen**
    - unionOf, intersectionOf, complementOf, ...



DAML+OIL Datentypen (1)

- **Basis: “Thing”**
  - ➔**Jedes Objekt ist davon abgeleitet (Ur-Basisklasse)**
- **Eigentliche Datentypen werden nicht definiert; XML-Schema hierzu verwenden**
  - ➔**Sowie zusätzlich die selbst definierten!**
- **Beispiel oben (RDF): Produktnummer könnte beliebiger String sein; sollte aber Nummer sein!**

➔
```
<daml:DatatypeProperty rdf:ID="productNumber">
  <rdfs:label>Product Number</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:range rdf:resource=
          "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>
```

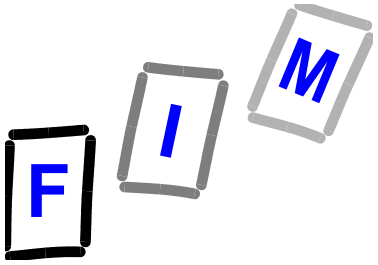


DAML+OIL Datentypen (2)

- **Enumeration:**

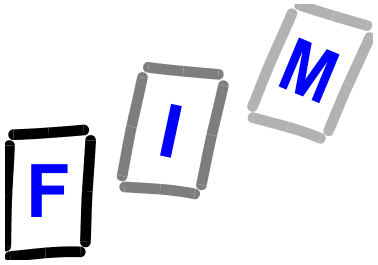
  ➔Verfügbarbeit ist entweder “InStock”, “BackOrdered” oder “SpecialOrder”, aber nichts anderes!

```
<daml:Class ID="Availability">
  <daml:oneOf parseType="daml:collection">
    <daml:Thing rdf:ID="InStock">
      <rdfs:label>In stock</rdfs:label>
    </daml:Thing>
    <daml:Thing rdf:ID="BackOrdered">
      <rdfs:label>Back ordered</rdfs:label>
    </daml:Thing>
    <daml:Thing rdf:ID="SpecialOrder">
      <rdfs:label>Special order</rdfs:label>
    </daml:Thing>
  </daml:oneOf>
</daml:Class>
```



DAML+OIL Klassen Erweiterungen (1)

- **“Klassen” haben hier wenig mit Objektorientierung zu tun; sie sind vielmehr ähnlich der Mengenlehre!**
  - **➔Es gibt Vererbung (oben nach unten; Spezialisierung)**
    - » **Agent; Subklassen: LokalerAgent, MobilerAgent**
    - » **Acess; Subklassen: HardwareAccess, LimitedAccess, NoAccess**
  - **➔Es gibt aber auch Aggregation (unten nach oben)!**
    - » **HardwareInterface = LokalerAgent UND HardwareAccess**
- **Mehrfachvererbung möglich und wichtig!**
  - » **LokalerAgent Subklasse von Agent und (ev.) von HardwareInterface**

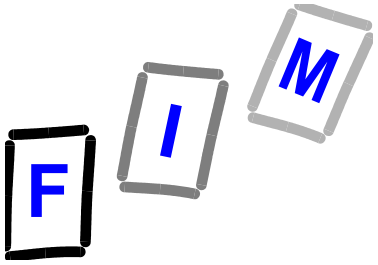


DAML+OIL Klassen Erweiterungen (2)

- **Disjoint Classes (Klassen mit getrennten Instanzen)**

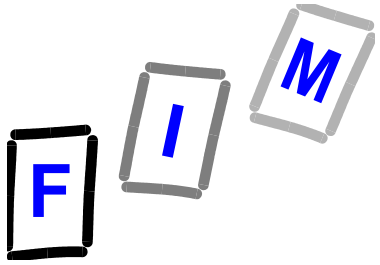
  ➔Ein Agent ist entweder lokal oder mobil, aber niemals beides!

```
<daml:Class rdf:ID=”Agent">
  <rdfs:label>Agent</rdfs:label>
  <rdfs:comment>An software agent in our agentsystem.</rdfs:comment>
  <daml:disjointUnionOf parseType="daml:collection">
    <daml:Class rdf:ID=”MobileAgent">
      <rdfs:label>Mobile agent</rdfs:label>
      <rdfs:comment>An agent able to move between
                    agentsystems.</rdfs:comment>
    </daml:Class>
    <daml:Class rdf:ID=”LocalAgent">
      <rdfs:label>Local agent</rdfs:label>
      <rdfs:comment>An immobile agent, which always stays within the
                    same agentsystem</rdfs:comment>
    </daml:Class>
  </daml:disjointUnionOf>
</daml:Class>
```



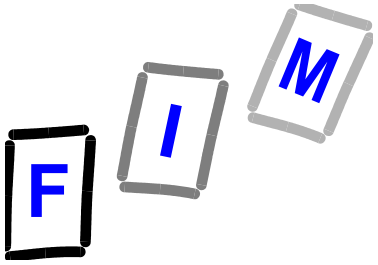
DAML+OIL Klassen Erweiterungen (3)

- **Unions (Vereinigungen):**
  - ➔Ein Objekt gehört einer Klasse an, wenn es entweder der einen oder der anderen Klasse angehört
    - » TrustedAgent = LokalerAgent unionOf CertificateChecked
- **Intersection (Schnittmenge):**
  - ➔Ein Objekt gehört genau dann einer Klasse an, wenn es allen Teilklassen zugehört
    - » HardwareInterface = LokalerAgent intersectionOf HardwareAccess



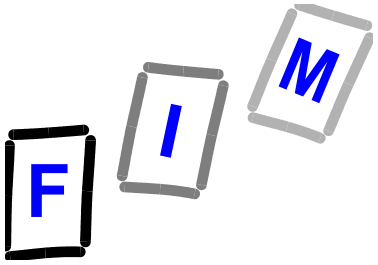
DAML+OIL Eigenschaften Erweiterungen

- **Inverse Properties:**
  - ➔Nur eines muß spezifiziert (bei Instanzen) werden, das andere wird automatisch abgeleitet
    - » ElternteilVon inversePropertyOf KindVon
- **Transitive Properties:**
  - ➔Gilt über mehrere Stufen (wie subclassOf)
    - » P *MitgliedVon* VereinA, VereinA *MitgliedVon* VereinB
    - » Daher ist P *MitgliedVon* VereinB
- **Restrictions (besser: Klassifizierungen):**
  - ➔Alle Objekte mit best. Eigenschaften gehören zu Klasse
    - » Alles was das Attribut “hasGUI” hat und bei dem dieser Wert “true” ist, gehört zur (jetzt definierten) Klasse “interactive”



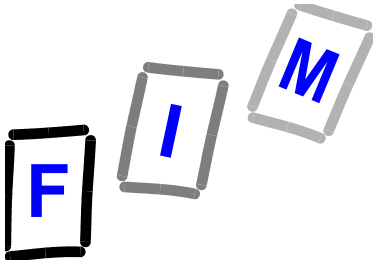
OWL: Nachfolger von DAML+OIL

- **OWL Web Ontology Language**
  - ➔**Bisher nur Entwurf**
  - ➔**Sehr ähnlich zu DAML+OIL**
    - » **Nur leichte Änderungen**
    - » **Daher eine Erweiterung von RDF Schema**
  - ➔**Vereinfachte Version (OWL Lite) existiert zur leichteren Implementierung**
    - » **Beschwerden über zu hohe Komplexität von DAML+OIL, die Implementierungen sehr schwer macht!**
  - ➔**Ziel: Trennung Semantik-Definition ⇔ Applikation**
  - ➔**3 Unterarten: Lite, DL (Description Logic), Full**



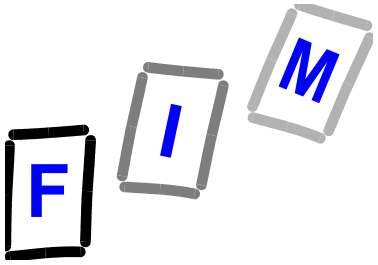
OWL: Datentypen

- Datentypen werden aus RDF übernommen
  - ➔Dort wiederum aus XML Schema übernommen
- Zusätzlicher Datentyp: rdfs:Literal
  - ➔Aus RDF Schema übernommen
- Sowie natürlich die konstruierten (zusammengesetzten) Datentypen!
- Eigener OWL-Enumerationstype
  - ➔Nicht in OWL-Lite!



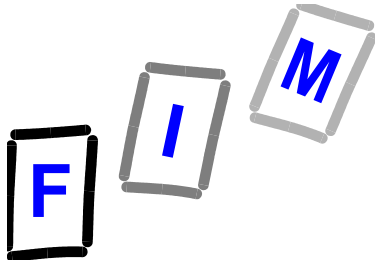
OWL Lite

- **Aus RDF Schema (über DAML+OIL) übernommen:**
  - ➔**Class: Eine Menge von Individuen mit gemeinsamen Eingenschaften. Organisation über eine Spezialisierungs-Hierarchie mittels "subClassOf"**
  - ➔**rdfs:subClassOf: Spezialisierung in Klassenhierarchien:**
    - » **Alle Individuen der Subklasse sind auch Elternklassen-Individuen**
  - ➔**rdfs:property: Beschreibt Verhältnisse zwischen Individuen (ähnlich zu Operatoren; z. B. hasRelative) oder Individuen und Daten (ähnlich zu Attributen der Individuen; z. B. hasAge)**
  - ➔**rdfs:subPropertyOf: Hierarchien von Verhältnissen**
    - » **hasSibling ist subproperty von hasRelative**
    - » **isOfAge ist subproperty von hasAge**



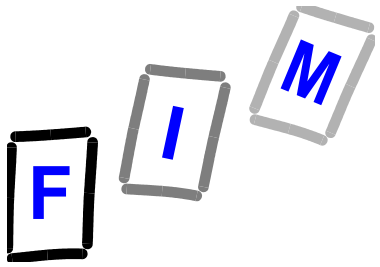
OWL Lite

- **Aus RDF Schema (über DAML+OIL) übernommen:**
  - ➔**rdfs:domain: Einschränkung der Individuen, denen eine Eigenschaft (Property) zugeordnet werden kann (ähnlich der Zurdnung von Methoden zu Objekten)**
    - » **Welche Klasse kann diese Eigenschaft haben**
  - ➔**rdfs:range: Einschränkung der Individuen, die eine Eigenschaft (Property) als Wert haben kann**
    - » **Welche Werte kann die Eigenschaft annehmen**
  - ➔**Individuen: Instanzen von Klassen (ähnlich Objekten)**



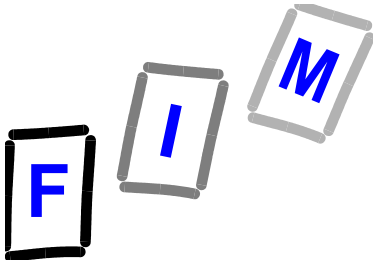
OWL Lite
Gleichheit

- ➔equivalentClass: Gleiche Klassen (=nur anderer Name). Alle Inidividuen der einen Klasse sind auch Individuen der anderen Klasse
- ➔equivalentProperty: Wie equivalentClass, aber für Eigenschaften
- ➔sameAs: Zwei Individuen sind identisch (nicht gleich!)
  - » Verschiedene Namen, aber selbes Objekt
- ➔differentFrom: Explizit erklären, daß zwei Individuen nicht identisch sind
  - » Manchmal wichtig; wegen der Existenz von sameAs
- ➔allDifferent: Verkürzung von differentFrom für alle möglichen Kombinationen aus einer Menge
  - » Erlaubt die Eindeutigkeit von Namen festzulegen



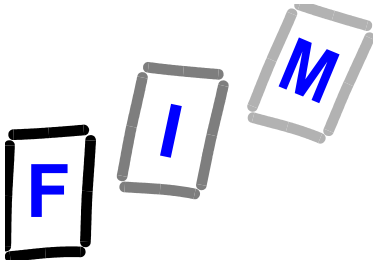
OWL Lite
Property-Qualifikationen

- ➔**inverseOf: Gegenläufige Eigenschaft**
  - » **"X *prop1* Y" und "prop1 *inverseOf* prop2" dann "Y *prop2* X"**
  - » **Beispiel: hatKind *inverseOf* hatEltern**
- ➔**transitiveProperty: Transitivität von Eigenschaften**
  - » **"X *prop1* Y", "Y *prop1* Z" und "*transitiveProperty* prop1" dann "X *prop1* Z"**
- ➔**symmetricProperty: Symmetrie von Eigenschaften**
  - » **"X *prop1* Y" und "*symmetricProperty* prop1" dann "Y *prop1* X"**
- ➔**funtionalProperty: Hat für jedes Individuum genau 0 oder einen Wert**
  - » **Kurzfassung für minCardinality=0 und maxCardinality=1 !**
- ➔**inverseFuntionalProperty: Inverse Eigenschaft ist funktional**
  - » ***inverseFunctionalProperty* hatSozialversicherungsnummer**
  - » **Wer eine Sozialversicherungsnummer hat, hat maximal eine!**



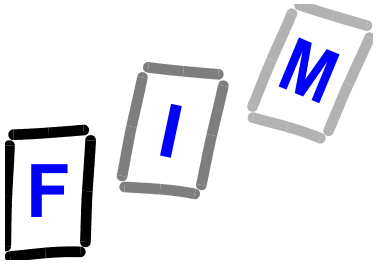
OWL Lite Einschränkungen

- **Einschränkungen welche Werte Eigenschaften annehmen können**
  - **➔allValuesFrom: Wert der Eigenschaft kann nur aus einer bestimmten Klasse stammen**
    - » **"prop1 *allValuesFrom* class1" = Wenn prop1 vorhanden ist, dann muß es Element von class1 sein (oder einer Subklasse)**
      - – Kann aber auch die Eigenschaft prop1 gar nicht haben!
  - **➔someValuesFrom: Zumindest ein Wert der Eigenschaft stammt aus einer bestimmten Klasse**
    - » **"prop1 *someValuesFrom* class1" = Jedes Objekt hat zumindest eine Eigenschaft prop1, deren Wert Elemente von class1 ist (oder einer Subklasse)**
      - – Muß die Eigenschaft prop1 zumindest ein Mal besitzen!



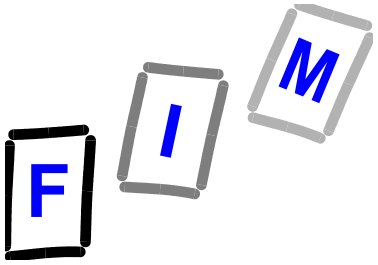
OWL Lite Anzahl-Einschränkungen

- **Einschränkungen wieviele Werte Eigenschaften annehmen können**
  - ➔**OWL Lite: Jeweils nur 0 und 1 erlaubt!**
  - ➔**minCardinality: Wie oft die Eigenschaft mindestens bei jeder Instanz vorkommen muß**
  - ➔**maxCardinality: Wie oft die Eigenschaft höchstens bei jeder Instanz vorkommen darf**
  - ➔**Cardinality: Wie oft genau die Eigenschaft bei jeder Instanz vorkommen muß**
    - » **Entspricht gleichzeitige Angabe von minCardinality und maxCardinality mit demselben Wert**



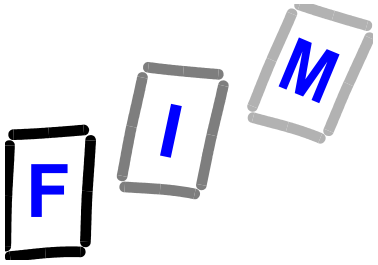
OWL Lite Klassen-Überschneidung

- **intersectionOf: Definieren einer neuen Klasse aus der Schnittmenge zweier Klassen**
  - ➔**Alle Individuen die Elemente der ersten und der zweiten Klasse sind, sind Elemente der neuen Klasse**
  - ➔**Beispiel:**
    - » **Klasse1: Studenten**
    - » **Klasse2: Österreicher**
    - » **Klasse1 *intersectionOf* Klasse2: Alle österreichischen Studenten**
      - – Wenn jemand Student ist und Österreicher ist, dann ist er ein Element dieser Klasse
    - » **Alternative: Klasse "ÖsterreichischeStudenten"**
      - – Wer Element der Klasse ist, ist Student und ist Österreicher
      - – Aber nicht aller Österreicher, die Studenten sind, müssen auch Element dieser Klasse sein!



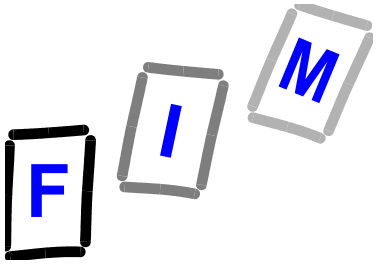
OWL DL und OWL Full (1)

- ➔oneOf: Eine Klasse durch die exakte Aufzählung der Individuen definieren (diese und nur diese gehören dazu)
  - » Beispiel: Klasse "Wochentage"
- ➔hasValue: Einschränkung auf alle Individuen, deren Eigenschaft einen bestimmten Wert besitzt
  - » Beispiel: Klasse "Österreicher" kann bestimmt werden als alle Personen, deren Eigenschaft "Staatsbürgerschaft" den Wert "Österreich" besitzt
- ➔disjointWith: Zwei Klassen besitzen keine gemeinsamen Individuen (d. h. kein Individuum ist Element beider)
  - » Beispiel: Mann *disjointWith* Frau
- ➔minCardinality, maxCardinality, cardinality: Kann beliebige positive Werte annehmen (0...N)



OWL DL und OWL Full (2)

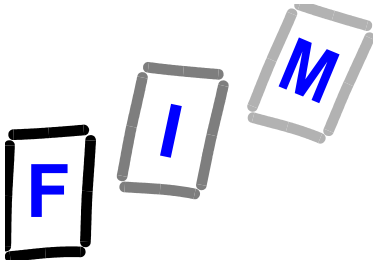
- ➔unionOf, complementOf, intersectionOf: Definition von Klassen über Mengen-Operationen:
  - » Vereinigungsmenge
  - » Komplementärmenge
  - » Schnittmenge (in allgemeinerer Form als in OWL Lite)
- ➔Komplexe Klassen: OWL Lite erlaubt vielfach nur benannte Klassen und nur einzelne Klassen
  - » subclassOf nur von 1 Klasse, daher in Lite keine Mehrfachvererbung
  - » OWL Full erlaubt es auch, Klassen als Individuen zu behandeln
    - – OWL kann über OWL sprechen



OWL Lite Beispiel

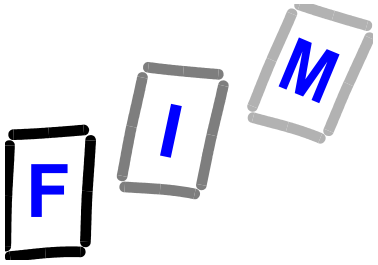
- **Modellierung von Studenten und Vortragenden**
  - ➔**Minimal-Modell, daß jeweils nur ein paar Dinge modelliert**
  - ➔**Nur zur Darstellung von OWL Lite geeignet, nicht für praktischen Einsatz**

OWL_Lite_Example.owl



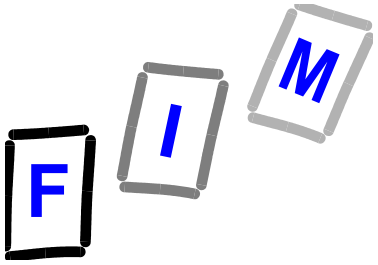
Dublin Core Grundstruktur

- **Webbasierte Beschreibung von Dokumenten**
  - ➔Metadaten über Dokumente; im Web suchbar
  - ➔Die Dokumente selbst müssen **NICHT** im Web sein!
- **Besteht aus 15 Elementen mit definierter Semantik**
- **Web-Dokumente: Oft als “META” Tag im Header**
  - ➔Oder als eigene XML-Dateien
- **Unabhängig von RDF**
  - ➔Können aber gemeinsam verwendet werden
- **Spezialisierungen für einige Elemente möglich**
  - ➔z. B. Description: TableOfContents, Abstract
  - ➔z. B. Date: Created, Valid, Available, Issued, Modified



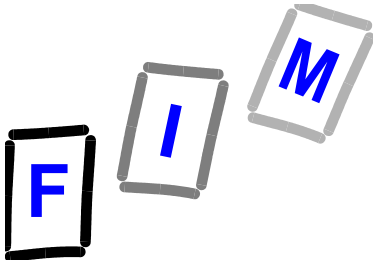
Dublin Core Elemente (1)

- Title: Offizielle Benennung der Ressource
- Creator: Person, die hauptsächlich für die Erstellung verantwortlich war
- Subject: Inhalt der Ressource (Schlüsselwörter)
- Description: Beschreibung des Inhalts
- Publisher: Verantwortlich für Zugänglichmachung
- Contributor: Wer Beiträge zum Inhalt geliefert hat
- Date: Datum im Lebenszyklus der Ressource
  - ➔In der Regel das Erstellungsdatum
  - ➔Format nach ISO 8601 (YYYY-MM-DD)



Dublin Core Elemente (2)

- Type: Software, Text, Audio, …
- Format: Medientyp, Abmessungen, … (z. B. MIME-Typ)
- Identifier: Eindeutige Referenz (z. B. URI) in Kontext
  - ➔Kontext selbst wird aber nicht spezifiziert!
- Source: Wovon die Res. abgeleitet ist (Vorversion)
- Language: Sprache (z. B. “de-at”)
- Relation: Verwandte (wie auch immer) Ressource
- Coverage: Gültigkeitsbereich
  - ➔ Geograph. Bereich, Zeitperiode, Zuständigkeit, ...
- Rights: IPR, Copyright, DRMS, ...

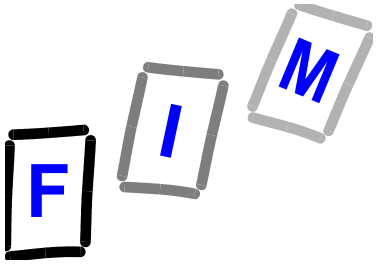


Dublin Core Beispiel (1)

- **Metadaten zum WeLearn-System auf den FIM-Webseiten:**

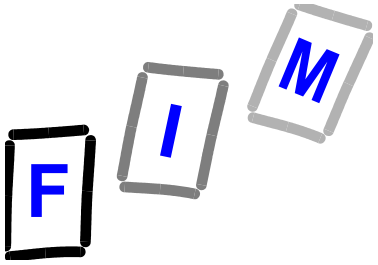
```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.fim.uni-linz.ac.at/WeLearn/">
      <dc:title>WeLearn - Web Environment for Learning</dc:title>
      <dc:creator>The WeLearn Team</dc:creator>
      <dc:subject>Online learning platform, E-Learning</dc:subject>
      <dc:description>An online platform for learning. Handling is
   similar to webpages. Contains mainly resources and communication
   elements (e. g. forums).</dc:description>
      <dc:date>2002-11-05</dc:date>
      <dc:type>InteractiveResource</dc:type>
      <dc:language>de-at</dc:language>
  </rdf:Description>
</rdf:RDF>
```



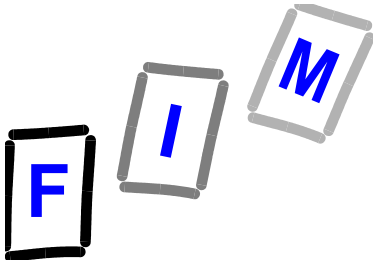
Dublin Core Beispiel (2)

- Direkter Einbau von DC in Webseiten:
- Siehe RFC 2731
  - ➔Meta-Tags
  - ➔Name: DC.<DC-Element-Name>
  - ➔Value: <Element-Wert>
- Beispiel:
  - ➔<meta name="DC.Creator" content="Michael Sonntag">
- Element-Präfix assoziieren:
  - ➔<link rel="schema.DC" href="http://purl.org/DC/elements/1.0/">
- Externe DC-Daten in XML-Form:
  - ➔<link rel="meta" href="index.dcxml">



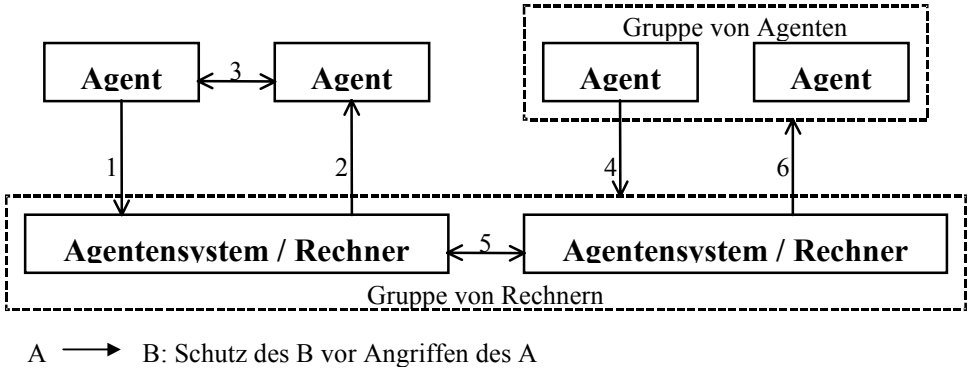
Dublin Core Probleme

- **Probleme erkennbar am obigen Beispiel:**
  - **➔Nur ein Creator möglich (Aber: Erweiterungen!)**
  - **➔Was bedeutet das Datum?**
- **Spezialisierung könnte helfen**
  - **➔Deren Codierung ist aber noch nicht spezifiziert!**
  - **➔Vorschlag dafür existiert**
    - » **Ausgiebige Verwendung von RDF; recht komplex**
- **Sonstige Probleme:**
  - **➔Wie bringen wir das ins Web?**
    - » **Als Meta-Tags in der Webseite: Wartung?**
    - » **Eigene Datei: Webseite abfragen und dann erst Verweis auf Metadaten erhältlich (Verweis ist selbst Meta-Tag)**
  - **➔Teile der Daten stehen auch so ev. bereits auf Webseite!**

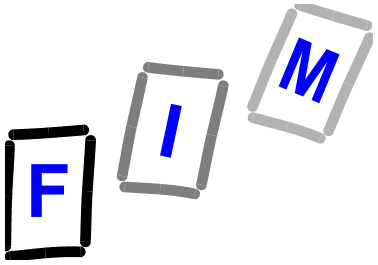


Sicherheitaspekte: Mobilität (1)

Mobilität bedeutet Gefahren für beteiltigte Komponenten:



- ❶ **Schutz des Rechners vor Agenten**
  - ➔ Sandbox, Code-Verifikation
- ❷ **Schutz der Agenten vor dem Rechner**
  - ➔ SEHR schwierig; derzeit keine Lösungen
    - » Ansätze: Verschleierung des Programms, spätere Nachprüfung



Sicherheitaspekte: Mobilität (2)

❸ **Schutz der Agenten voreinander**

➔ Meist kein Problem; ergibt sich aus Rechnerschutz

❹ **Schutz einer Gruppe von Rechnern**

➔ Angriffe: z. B. Ressource-Exhaustion

➔ Abhilfe: Verzeichnisdienste und übergreifende Kontrolle

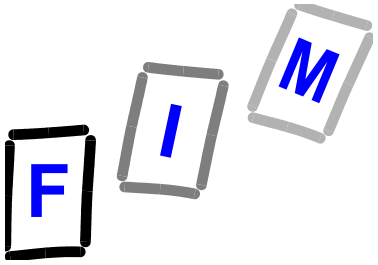
❺ **Schutz von Rechnern untereinander**

➔ Kein Agenten-spezifisches Problem; Agentensystem / Kommunikation kann aber weiterer Angriffspunkt sein!

❻ **Schutz einer Gruppe von Agenten**

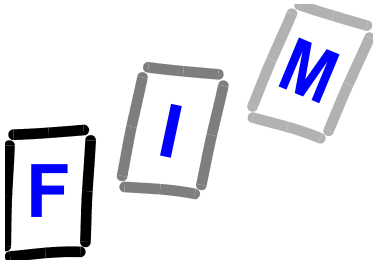
➔ Angriffe: Diskriminierung (einzelnen Verzögern)

➔ Erkennung: Kommunikation zwischen den Agenten



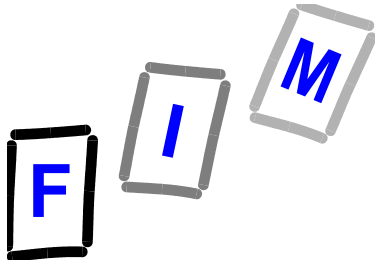
Angriffsarten: Bereicherung/Schädigung

- **Bereicherung (Rationale Angreifer; “Diebe”)**
  - ➔Rückabwicklung eventuell möglich
  - ➔Kosten des Angriffs wichtig
  - ➔Erkennung oft ausreichend
  - ➔Sicherung relativ einfach (⇒ Angriff verteuern)
- **Schädigung (Irrationale Angreifer; “Terroristen”)**
  - ➔Rückabwicklung unmöglich
  - ➔Angriffskosten großteils egal
  - ➔Echte Vorbeugung/Verhinderung erforderlich
  - ➔Sicherung sehr schwer (⇒ **Jede** Lücke stopfen)
- **Praxis: Abschätzung Gefahr / Kosten**



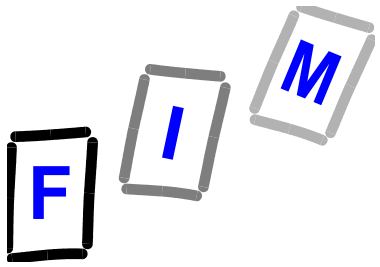
Angriffsarten: Extern / Intern

- **Extern:**
  - ➔Kein Vertrauen gegenüber dem Angreifer
  - ➔Erringen neuer Rechte
  - ➔Identifikation schwer
  - ➔Kein Zugriff auf die Person
- **Intern:**
  - ➔Dem Angreifer wird vertraut
  - ➔Ausnützen/erweitern bestehender Rechte
  - ➔Identifikation eher einfach
  - ➔Haftung aus bestehendem Verhältnis (z. B. Arbeitsvertrag)
  - ➔Typisch: Mitarbeiter



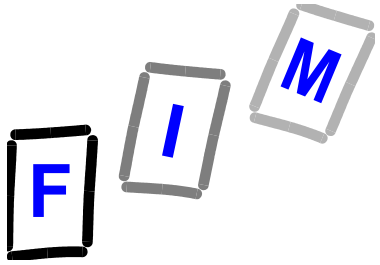
Angriffsarten: Datenveränderung

- **Mit Datenveränderung**
  - ➔**Prüfsummen/Signaturen zur Erkennung**
    - » **Einfach bei statischen Daten (Programmcode, Initialisierung)**
    - » **Schwierig bei dynamischen Daten (Agenten-Zustand)**
  - ➔**Nachweis oft einfach**
  - ➔**Schwierigkeit bei mobilen Agenten: Urheber**
    - » **Hand-off von Agenten**
- **Ohne Datenveränderung**
  - ➔**Beispiel: Zeitverzögerung, Ausspähen, Agenten-Kopien, Kommunikationsunterbrechung, ...**
  - ➔**Erkennung sehr schwer**
  - ➔**Nachweis oft fast unmöglich**



Angriffsarten: Kurzfristig / Langfristig

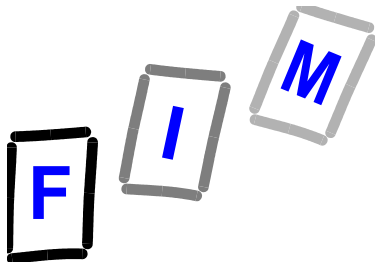
- **Kurzfristige Angriffe**
  - ➔Abhilfe: Anomalitäten erkennen
    - » Schwierig, da keine Historie vorhanden!
    - » Insgesamt leichter, da meist große Anomalitäten nötig
  - ➔Meist eher mit Gewalt oder recht offensichtlich
- **Langfristige Angriffe**
  - ➔Zuerst Vertrauen erreichen, dann zuschlagen
  - ➔Versuchen, Spuren zu verwischen
  - ➔Abhilfe: Anomalitäten erkennen
    - » Leichter, da viel Historie vorhanden
    - » Trotzdem viel schwerer erkennbar als oben, da genau geplant und meist nur kleine Anomalitäten erforderlich



Angriffsarten: Einzeln- / Gruppenangriffe

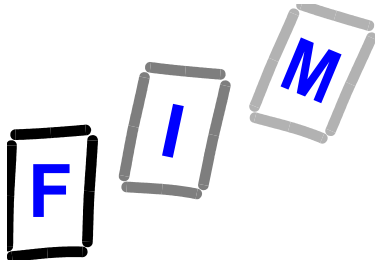
➔Sowohl bezüglich Agenten als auch Rechner!

- **Einzelangriffe**
  - ➔Einfach; klassischer Fall
  - ➔Einmaliger “Fehler” kann auch echter Fehler sein
- **Gruppenangriffe**
  - ➔Keine Fluchtmöglichkeiten, keine Vergleichswerte
    - » Auf allen Rechnern gleich; alle Agenten agieren gleich
  - ➔Komplex zu bewerkstelligen
  - ➔Gehäuftes Fehlverhalten fällt leichter auf
    - » Aber bei vielen Mitwirkenden reichen geringe Fehler



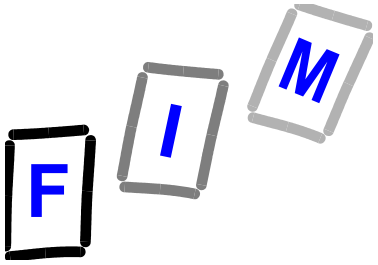
Sicherheitaspekte: Ausgewählte Angriffe (1)

- **Abhören des Agenten-Transfers**
  - ➔Agent inaktiv; auf Agentensystem angewiesen
  - ➔Abhilfe: Verschlüsselung der Übertragung
    - » Problem: Man-in-the-middle Angriff: Gegenstelle erkennen
- **Kopieren von Agenten**
  - ➔Durch externen Angreifer (Replay)
    - » Seriennummern in Übertragung einbauen
    - » Verschlüsselung
  - ➔Durch Agentensystem
    - » Wie Veränderungen des Agenten selbst
    - » Erlaubt spätere Analyse und “nachspielen”/experimentieren



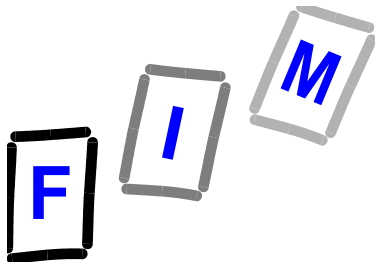
Sicherheitaspekte: Ausgewählte Angriffe (2)

- **Behinderung / Denial of Service**
  - ➔“Derzeit keine Ressourcen verfügbar”
    - » **Praktisch nicht zu erkennen!**
  - ➔Gefahr besonders bei Zeitlimits: Verzögern
  - ➔Vorteil von mobilen Agenten: Ausweichen ev. möglich
- **In-/Output Modifikation**
  - ➔Für Agenten nur zu erkennen, wenn von wirklicher Quelle signiert
  - ➔Behinderung: Eigenen Output verschlüsseln/signieren
    - » **Rechner muß zuerst den Schlüssel aus dem Programmcode extrahieren bevor er fälschen kann!**
  - ➔Vor-Filterungs Agenten sorgfältig auswählen!



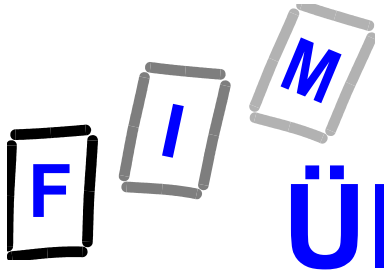
Abwehrmöglichkeiten: Sichere Hardware

- "Tamper-proof box": Sehr selten, extrem teuer
  - ➔Wird beim öffnen zerstört (zumindest Teile davon)
- In praktisch keinem Rechner vorhanden
- Einziger wirkliche Schutz von Agenten vor Rechner
  - » Aber nicht gegen Ein-/Ausgabe-Modifikation!
- Besseres Software-Äquivalent:
  - ➔Agenten nur auf solche Rechner verlagern lassen, deren Besitzern vertraut wird!
    - » Zertifikate sagen idR über so ein Vertrauen NICHTS aus!
  - ➔Dann sind fast keine Sicherheitsvorkehrungen mehr nötig
    - » Die Vorhandenen sind dann meistens eher Schutz vor Fehlern!



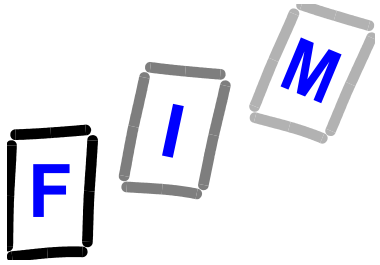
Abwehrmöglichkeiten: Time-limited security

- Programmcode und/oder Daten werden verschleiert (Obfuscation)
- Bei bestimmter Rechnerleistung kann meist eine Untergrenze angegeben werden, die für die Entschlüsselung mindestens nötig ist
- Bis dahin ist der Agent sicher
  - ➔Spätestens hier muß er von dem Server wegwandern
- Probleme:
  - ➔Um so öfter verwendet, desto kürzer!
  - ➔Keine Implementierung bekannt
  - ➔Kein Schutz geheimer Daten (später IST Lesen möglich!)



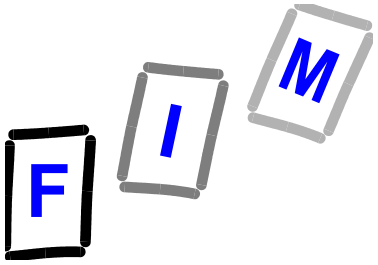
Abwehrmöglichkeiten: Übetragungs-Verschlüsselung

- **Für Kommunikation und Agenten-Transfer**
- **Meist kann eigener Transfer nicht überwacht werden**
  - **➔Vertrauen in Agentensystem nötig!**
- **Man-in-the-middle Angriffe beachten!**
  - **➔Zertifikate austauschen, diese überprüfen**
  - **➔Kenntnis des privaten Schlüssels prüfen**
  - **➔Gemeinsamen Schlüssel berechnen (z. B. Diffie-Hellman)**
  - **➔Agent transferieren**
- **Problem: Was, wenn Empfänger nicht verschlüsseln kann/will?**
  - **➔Dem Agenten bleibt nur mehr die Heimkehr!**



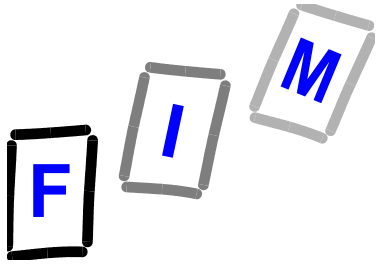
Abwehrmöglichkeiten: Code-Signaturen

- **Code kann signiert werden: Feststellung, vom wem**
  - **➔Wird das Besitzer-Zertifikat mitsigniert, so kann auf Übereinstimmung geprüft werden**
    - **» Vor-/Nachteil: Jeder kann sich selbst zum Signator machen!**
  - **➔Funktioniert auch für Initialisierungs-Daten**
- **Überprüfung muß durch Agentensystem erfolgen**
- **Problem bei Laufzeit-Daten (bzw. Ergebnissen)**
    - **» Privater Schlüssel müßte mitgeführt werden: SEHR gefährlich!**
  - **➔Hand-off: Agentensystem signiert Daten beim Verlassen**
  - **➔Spätere Änderungen nicht möglich, nur Hinzufügen**
  - **➔Feststellung möglich, wo Fehler/Modifikation passiert ist**



Abwehrmöglichkeiten: Sandbox

- **Analog zu Java Sandbox**
  - » **Und vielfach darauf basierend!**
- **Möglichkeiten an Aktionen für Agenten werden stark eingeschränkt**
  - ➔**z. B. kein ungeprüfter Datei-/Netzwerkszugriff**
  - ➔**Verlassen des Sandbox für Agenten selbst unmöglich**
- **Können recht sicher erstellt werden (da rel. klein)**
- **Problem: Vieles benötigt dennoch Sonderrechte**
  - ➔**Wie überprüfe man, wem man Rechte gibt?**
  - ➔**Wer bekommt welche Rechte?**
  - ➔**Wie können diese Rechte auf Teile beschränkt werden?**



Sicherheitaspekte: Zusammenfassung

- **Sandbox + Kommunikations-Verschlüsselung + Partner-Identifikation = Ausreichend für Rechner**
  - ➔**Eventuell: Code-Signaturen + Hand-off zusätzlich**
- **Vertrauenswürdiger Recher = Meist ausreichende Sicherheit für Agenten**
- **Erforderlich:**
  - ➔**PKI (um Zertifikate prüfen zu können)**
  - ➔**Sandbox (muß Teil des Betriebssystems oder der Sprache sein)**
  - ➔**Verschlüsselungs-/Signatur Software**
  - ➔**Vertrauens-Informationen**