

Einführung in das Jadex-System

Stefan Tittel

Universität Dortmund

Projektgruppenseminar: Wissen in Multiagentensystemen,
März 2006

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Was ist Jadex?

Was ist Jadex?

„Jadex is an agent-oriented reasoning engine for writing rational agents with XML and the Java programming language.“

Was ist ein Agent?

Ein Programm mit den Eigenschaften:

- autonom
- proaktiv
- reaktiv
- sozial
- lern- und anpassungsfähig

Was ist Jadex?

Was ist Jadex?

„Jadex is an agent-oriented reasoning engine for writing rational agents with XML and the Java programming language.“

Was ist ein Agent?

Ein Programm mit den Eigenschaften:

- autonom
- proaktiv
- reaktiv
- sozial
- lern- und anpassungsfähig

Adapter

Jadex selbst kann keine Agenten ausführen. Dazu bedarf es einer Middleware-Plattform. Es gibt derzeit zwei Adapter:

- Einen Adapter für das **J**ava **A**gent **D**evelopment Framework (JADE),
- einen Standalone-Adapter.

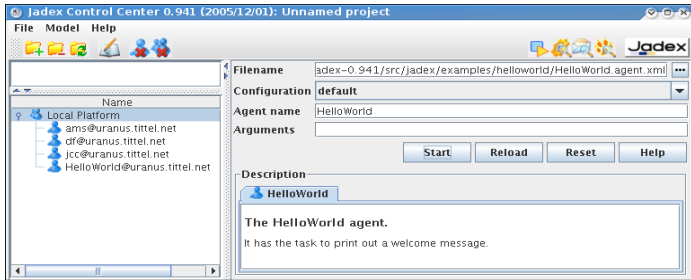


Abbildung: Starter der GUI des Standalone-Adapters

Agentenmodell

Jadex basiert auf dem BDI-Modell. Zur Erinnerung:

- **b**eliefs: Weltwissen (informational state)
- **d**esires: Hauptziele (motivational state)
- **i**ntentions: Absichten/Pläne (deliberative state)

Implementierung in Jadex:

- **b**eliefs: bestehen aus **f**acts (implementiert durch beliebige Java-Objekte); werden durch *beliefbase* zugeordnet
- **d**esires: Realisierung in Form von **g**oals; verschiedene Goal-Typen verfügbar
- **i**ntentions: Realisierung in Form von **p**lans (Java-Prozeduren zzgl. Ausführungsbedingungen)

Agentenmodell

Jadex basiert auf dem BDI-Modell. Zur Erinnerung:

- **beliefs**: Weltwissen (informational state)
- **desires**: Hauptziele (motivational state)
- **intentions**: Absichten/Pläne (deliberative state)

Implementierung in Jadex:

- **beliefs**: bestehen aus **facts** (implementiert durch beliebige Java-Objekte); werden durch *beliefbase* zugeordnet
- **desires**: Realisierung in Form von **goals**; verschiedene Goal-Typen verfügbar
- **intentions**: Realisierung in Form von **plans** (Java-Prozeduren zzgl. Ausführungsbedingungen)

Grundlegende Architektur

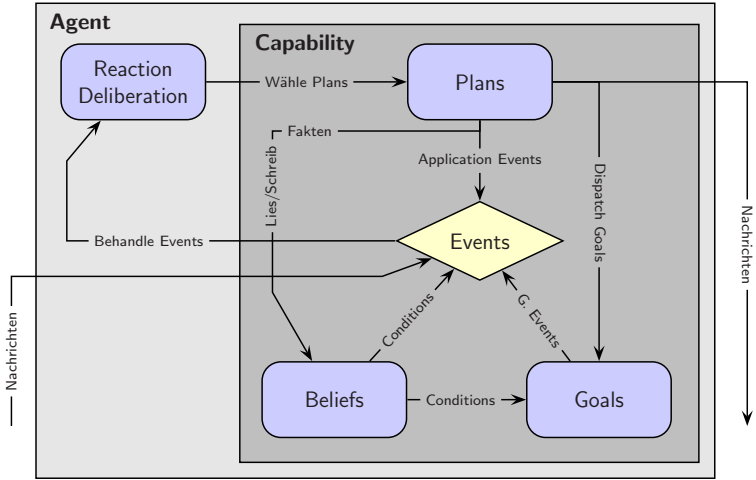


Abbildung: Grundlegende Architektur

Einführung

Komponenten

- Capabilities
- Beliefs
- Goals
- Plans
- Events

Anhang: Beispiele

- Plans
- Beliefs
- Goals

Realisierung von Agenten 1/2

Einführung

Komponenten

Capabilities

Beliefs

Goals

Plans

Events

Anhang:

Beispiele

Plans

Beliefs

Goals

- Jeder Agent besteht aus einer XML-Datei (Agent Definition File – **ADF**) und Java-Klassen.
- Das ADF enthält die eigentliche Spezifikation des Agenten.
- Die Java-Klassen implementieren plans und eventuell benötigte Objekte (z. B. zur Realisierung von beliefs).
- Bei plans: head in ADF, body in Java-Klasse

HelloWorldAgent.xml

```
<agent (Schemaangaben) name="HelloWorld">
  <plans>
    <plan name="hello">
      <body>new HelloWorldPlan()</body>
    </plan>
  </plans>
```

Realisierung von Agenten 1/2

Einführung

Komponenten

Capabilities

Beliefs

Goals

Plans

Events

Anhang:

Beispiele

Plans

Beliefs

Goals

- Jeder Agent besteht aus einer XML-Datei (Agent Definition File – **ADF**) und Java-Klassen.
- Das ADF enthält die eigentliche Spezifikation des Agenten.
- Die Java-Klassen implementieren plans und eventuell benötigte Objekte (z. B. zur Realisierung von beliefs).
- Bei plans: head in ADF, body in Java-Klasse

HelloWorldAgent.xml

```
<agent (Schemaangaben) name="HelloWorld">  
  <plans>  
    <plan name="hello">  
      <body>new HelloWorldPlan()</body>  
    </plan>  
  </plans>
```

Realisierung von Agenten 2/2

```
<initialstates>
  <initialstate name="default">
    <plans><initialplan ref="hello"/></plans>
  </initialstate>
</initialstates>
</agent>
```

HelloWorldPlan.java

```
import jadex.runtime.Plan;
public class HelloWorldPlan extends Plan {
    public void body() {
        System.out.println("Hello world!");
        killAgent();
    }
}
```

Realisierung von Agenten 2/2

```
<initialstates>
  <initialstate name="default">
    <plans><initialplan ref="hello"/></plans>
  </initialstate>
</initialstates>
</agent>
```

HelloWorldPlan.java

```
import jadex.runtime.Plan;
public class HelloWorldPlan extends Plan {
  public void body() {
    System.out.println("Hello world!");
    killAgent();
  }
}
```

Beliefs

- Die beliefbase ordnet Identifizierungsstrings Fakten zu, die wiederum beliebige Java-Klassen sein können.
- Es gibt zwei Arten von beliefs:
 - Einfache single-fact beliefs und
 - beliefsets.
- Jadex unterstützt OQL-ähnliche Anfragen an die beliefbase.
- Änderungen an der beliefbase können plans oder goals automatisch triggern.
- Es gibt keine (logikbasierte) Inferenz in der beliefbase.

Goals

- Goals sind konkrete momentane Wünsche des Agenten.
- Der Agent wählt (früher oder später) für jedes goal geeignete Aktionen aus, solange bis
 - das goal erreicht wurde,
 - das goal als unerreichbar gilt oder
 - das goal nicht länger aktuell ist.
- Goals müssen nicht zueinander konsistent sein.
- Jedes goal ist in einem der Zustände:
 - **option** – angenommen, aber derzeit nicht verfolgt
 - **active** – aktuell verfolgt
 - **suspended** – Kontextbedingungen derzeit nicht erfüllt
- Es gibt vier Goal-Typen:
 - **perform goals** – geben zu erledigende Tätigkeiten an
 - **achieve goals** – beschreiben abstrakten Zielzustand
 - **query goals** – geben zu erlangende Informationen an
 - **maintain goals** – geben beizubehaltenen Zustand an

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Capabilities in Jadex

- Agenten ohne Schlussfolgerungsprozess
- aus beliefs, goals und plans bestehend
- können weitere subcapabilities enthalten
- werden eingesetzt um Funktionalität wiederzuverwenden
- Spezifikation analog zu Agenten (in der XML-Datei:
`<capability>` statt `<agent>`)

Beispiel: Agent nutzt capabilities

```
<agent ...> ...  
  <capabilities>  
    <capability name="mysubcap"  
      file="MyCapability.capability.xml"/>  
    <capability name="dfcap"  
      file="jadex.planlib.DF"/>  
  </capabilities> ...  
</agent>
```

Capabilities in Jadex

- Agenten ohne Schlussfolgerungsprozess
- aus beliefs, goals und plans bestehend
- können weitere subcapabilities enthalten
- werden eingesetzt um Funktionalität wiederzuverwenden
- Spezifikation analog zu Agenten (in der XML-Datei:
`<capability>` statt `<agent>`)

Beispiel: Agent nutzt capabilities

```
<agent ...> ...  
  <capabilities>  
    <capability name="mysubcap"  
      file="MyCapability.capability.xml"/>  
    <capability name="dfcap"  
      file="jadex.planlib.DF"/>  
  </capabilities> ...  
</agent>
```

- Sichtbarkeit standardmäßig lokal
- wenn Zugriff von äußerer capability erfolgen soll:
`exported="true"` in `<belief>` etc. in innerer capability
- Referenz in äußerer capability

Beispiel: Innere Capability (Auszug)

```
<belief name="ex" exported="true" class="MyFact"/>
```

Beispiel: Äußere Capability (Auszug)

```
<beliefref name="mysubbelief">  
  <concrete ref="mysubcap.ex"/>  
</beliefref>
```

- Sichtbarkeit standardmäßig lokal
- wenn Zugriff von äußerer capability erfolgen soll:
`exported="true"` in `<belief>` etc. in innerer capability
- Referenz in äußerer capability

Beispiel: Innere Capability (Auszug)

```
<belief name="ex" exported="true" class="MyFact"/>
```

Beispiel: Äußere Capability (Auszug)

```
<beliefref name="mysubbelief">  
  <concrete ref="mysubcap.ex"/>  
</beliefref>
```

- Sichtbarkeit standardmäßig lokal
- wenn Zugriff von äußerer capability erfolgen soll:
`exported="true"` in `<belief>` etc. in innerer capability
- Referenz in äußerer capability

Beispiel: Innere Capability (Auszug)

```
<belief name="ex" exported="true" class="MyFact"/>
```

Beispiel: Äußere Capability (Auszug)

```
<beliefref name="mysubbelief">  
  <concrete ref="mysubcap.ex"/>  
</beliefref>
```

Abstrakte Elemente

- Element hat keine Implementierung und muss von äußerer capability zugewiesen werden
- Definition abstrakter Elementreferenz:
`<beliefref ...><abstract/></beliefref>`
- Implementierung muss nicht erfolgen, wenn
`<abstract required="false">`
- äußere Capability implementiert Element (direkt oder über andere Referenz); Zuweisung mittels `<assignto>`

Beispiel: Innere Capability (Auszug)

```
<beliefref name="abs" exported="true" class="MyF">  
<abstract/> </beliefref>
```

Beispiel: Äußere Capability (Auszug)

```
<belief name="mybelief" class="MyF">  
<assignto ref="mysubcap.abs"> </belief>
```

Abstrakte Elemente

- Element hat keine Implementierung und muss von äußerer capability zugewiesen werden
- Definition abstrakter Elementreferenz:
`<beliefref ...><abstract/></beliefref>`
- Implementierung muss nicht erfolgen, wenn
`<abstract required="false">`
- äußere Capability implementiert Element (direkt oder über andere Referenz); Zuweisung mittels `<assignto>`

Beispiel: Innere Capability (Auszug)

```
<beliefref name="abs" exported="true" class="MyF">  
<abstract/> </beliefref>
```

Beispiel: Äußere Capability (Auszug)

```
<belief name="mybelief" class="MyF">  
<assignto ref="mysubcap.abs"> </belief>
```

Abstrakte Elemente

- Element hat keine Implementierung und muss von äußerer capability zugewiesen werden
- Definition abstrakter Elementreferenz:
`<beliefref ...><abstract/></beliefref>`
- Implementierung muss nicht erfolgen, wenn
`<abstract required="false">`
- äußere Capability implementiert Element (direkt oder über andere Referenz); Zuweisung mittels `<assignto>`

Beispiel: Innere Capability (Auszug)

```
<beliefref name="abs" exported="true" class="MyF">  
<abstract/> </beliefref>
```

Beispiel: Äußere Capability (Auszug)

```
<belief name="mybelief" class="MyF">  
<assignto ref="mysubcap.abs"> </belief>
```

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Spezifikation im ADF

- belief spezifiziert im ADF, Aufruf und Änderung durch plans
- einleitender Tag für beliefs bzw. beliefsets ist `<belief>` bzw. `<beliefset>`
- `<belief>` und `<beliefset>` haben zwei Attribute:
 - `name` – um an anderer Stelle Bezug auf die im belief enthaltenen Fakten zu nehmen
 - `class` – Angabe der Java-Klasse, welche die Fakten realisiert
- direkte Angabe initialer Fakten inmitten `<fact></fact>`
- bei initialen Fakten unbekannter Anzahl:
`<facts></facts>`

Beispiel zur Spezifikation im ADF

```
<beliefs>
  <belief name="my_location" class="Location">
    <fact>new Location("Hamburg")</fact>
  </belief>
  <beliefset name="my_friends" class="String">
    <fact>"Alex"</fact>
    <fact>"Blandi"</fact>
    <fact>"Charly"</fact>
  </beliefset>
  <beliefset name="my_opponents" class="String">
    <facts>Database.getOpponents()</facts>
  </beliefset>
</beliefs>
```

Beispiel zur Spezifikation im ADF

```
<beliefs>
  <belief name="my_location" class="Location">
    <fact>new Location("Hamburg")</fact>
  </belief>
  <beliefset name="my_friends" class="String">
    <fact>"Alex"</fact>
    <fact>"Blandi"</fact>
    <fact>"Charly"</fact>
  </beliefset>
  <beliefset name="my_opponents" class="String">
    <facts>Database.getOpponents()</facts>
  </beliefset>
</beliefs>
```

Beispiel zur Spezifikation im ADF

```
<beliefs>
  <belief name="my_location" class="Location">
    <fact>new Location("Hamburg")</fact>
  </belief>
  <beliefset name="my_friends" class="String">
    <fact>"Alex"</fact>
    <fact>"Blandi"</fact>
    <fact>"Charly"</fact>
  </beliefset>
  <beliefset name="my_opponents" class="String">
    <facts>Database.getOpponents()</facts>
  </beliefset>
</beliefs>
```

Beispiel zur Spezifikation im ADF

```
<beliefs>
  <belief name="my_location" class="Location">
    <fact>new Location("Hamburg")</fact>
  </belief>
  <beliefset name="my_friends" class="String">
    <fact>"Alex"</fact>
    <fact>"Blandi"</fact>
    <fact>"Charly"</fact>
  </beliefset>
  <beliefset name="my_opponents" class="String">
    <facts>Database.getOpponents()</facts>
  </beliefset>
</beliefs>
```

Zugriff auf Beliefs aus Plans

- `getBeliefbase()` liefert `beliefbase (IBeliefbase)`
- `IBeliefbase.getBelief(String name)` bzw. `.getBeliefSet(String name)` liefert `beliefs (Klasse IBelief)` bzw. `beliefsets (Klasse IBeliefSet)`
- Zugriff auf die Fakten mittels `IBelief.getFact()` bzw. `IBeliefSet.getFacts()`
- weiterhin: `containsFact()`, `setFact()`, `addFact(fact)`, `removeFact(fact)`, `updateFact()`

Beispiel

```
public void body { ...
    IBelief h = getBeliefbase().getBelief("hungry");
    h.setFact(new Boolean(true)); ...
    Food[] food = (Food[])getBeliefbase()
        .getBeliefSet("food").getFacts(); ... }
```

Zugriff auf Beliefs aus Plans

- `getBeliefbase()` liefert `beliefbase (IBeliefbase)`
- `IBeliefbase.getBelief(String name)` bzw. `.getBeliefSet(String name)` liefert `beliefs (Klasse IBelief)` bzw. `beliefsets (Klasse IBeliefSet)`
- Zugriff auf die Fakten mittels `IBelief.getFact()` bzw. `IBeliefSet.getFacts()`
- weiterhin: `containsFact()`, `setFact()`, `addFact(fact)`, `removeFact(fact)`, `updateFact()`

Beispiel

```
public void body { ...
    IBelief h = getBeliefbase().getBelief("hungry");
    h.setFact(new Boolean(true)); ...
    Food[] food = (Food[])getBeliefbase()
        .getBeliefSet("food").getFacts(); ... }
```

Auswertung und Propagation

Auswertung initialer Fakten nur beim Start des Agenten, es sei denn explizite Angabe von:

- `<fact evaluationmode="dynamic">` – Auswertung bei jedem Zugriff
- `<belief updatarate="n">` – Auswertung zusätzlich alle n Sekunden

Propagation von Belief-Änderungen:

- Beliefs können als Bedingungen dienen, z. B. für goals.
- Änderungen an beliefs müssen deshalb propagiert werden.
 - Propagation in einfachen Fällen automatisch, z. B. bei Zuweisung neuen Faktums zum belief oder Änderungen durch Belief-Abhängigkeiten
 - bei Änderungen eines komplexen Fakt-Objektes explizites Feuern von events

Auswertung und Propagation

Auswertung initialer Fakten nur beim Start des Agenten, es sei denn explizite Angabe von:

- `<fact evaluationmode="dynamic">` – Auswertung bei jedem Zugriff
- `<belief updatarate="n">` – Auswertung zusätzlich alle n Sekunden

Propagation von Belief-Änderungen:

- Beliefs können als Bedingungen dienen, z. B. für goals.
- Änderungen an beliefs müssen deshalb propagiert werden.
 - Propagation in einfachen Fällen automatisch, z. B. bei Zuweisung neuen Faktums zum belief oder Änderungen durch Belief-Abhängigkeiten
 - bei Änderungen eines komplexen Fakt-Objektes explizites Feuern von events

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Allgemeines und Spezifikation 1/2

- bereits erwähnt: perform goals, achieve goals, query goals, maintain goals
- zusätzlich: **meta-level goals**, z. B. „finde einen auszuführenden plan“
- Unterscheidung zwischen:
 - **top-level goals** – zu Beginn verfügbar oder später angenommen
 - **subgoals** – nur von laufenden plans dispatcht
- Spezifikation von goals im ADF
- Goals können **Parameter** haben (Spezifikation ähnlich beliefs).
 - Attribut **direction** $\in \{\text{in}, \text{out}, \text{inout}\}$
 - Attribut **optional** – Parameter ist optional
 - Parameterwert mittels **<value>** oder **<bindingoptions>**

Allgemeines und Spezifikation 2/2

- **<unique/>** – Es wird nur eine Goal-Instanz gleichen Typs und gleicher Parameter instantiiert.
- **<exclude>** kann angeben, welche Parameter dabei nicht verglichen werden sollen.
- autom. Instantiierung durch **<creationcondition>**
- autom. Suspendierung durch **<contextcondition>**
 - Terminiere alle zum goal gehörenden plans und subgoals.
 - Sind die Kontext-Bedingungen wieder erfüllt: Instantiiere neue plans und subgoals.
- autom. Verwurf durch **<dropcondition>**
 - Verworfen goals können nicht reaktiviert werden.

BDI-Flags

- Attribute von Elementen (hier: goals)
- Spezifikation im ADF in `<goal>` oder für einzelne Goal-Instanzen mittels `set`-Methode

Name	Default	mögliche Werte
<code>retry</code>	<code>true</code>	<code>{true, false}</code>
<code>retrydelay</code>	<code>0</code>	positive long value
<code>exclude</code>	<code>"w._t."</code>	<code>{"when_tried"</code> <code>"when_succeeded"</code> <code>"when_failed"</code> <code>"never"}</code>
<code>posttoall</code>	<code>false</code>	<code>{true, false}</code>
<code>randomselection</code>	<code>false</code>	<code>{true, false}</code>
<code>metalevelreasoning</code>	<code>true</code>	<code>{true, false}</code>

Tabelle: allen Goal-Typen gemeine BDI-Flags

BDI-Flags

- Attribute von Elementen (hier: goals)
- Spezifikation im ADF in `<goal>` oder für einzelne Goal-Instanzen mittels `set`-Methode

Name	Default	mögliche Werte
<code>retry</code>	<code>true</code>	<code>{true, false}</code>
<code>retrydelay</code>	<code>0</code>	positive long value
<code>exclude</code>	<code>"w._t."</code>	<code>{"when_tried"</code> <code>"when_succeeded"</code> <code>"when_failed"</code> <code>"never"}</code>
<code>posttoall</code>	<code>false</code>	<code>{true, false}</code>
<code>randomselection</code>	<code>false</code>	<code>{true, false}</code>
<code>metalevelreasoning</code>	<code>true</code>	<code>{true, false}</code>

Tabelle: allen Goal-Typen gemeine BDI-Flags

Spezielle Bedingungen

Achieve Goals:

- `<targetcondition>` – gibt an, wann goal erreicht ist
- `<failurecondition>` – gibt an, wann goal fehlschlägt
- wenn nicht angegeben: Erfolg der plans bestimmt Erfolg des goals

Query Goals:

- implizite Zielbedingungen:
 - alle Parameter mit `direction=out` \neq null
 - `parametersets` verfügen über mindestens einen Wert

Maintain Goals:

- `<maintaincondition>` (verpflichtend) – gibt zu erhaltenden Zustand an
- `<targetcondition>` – Zielzustand bei Verletzung der `<maintaincondition>`

Spezielle Bedingungen

Achieve Goals:

- `<targetcondition>` – gibt an, wann goal erreicht ist
- `<failurecondition>` – gibt an, wann goal fehlschlägt
- wenn nicht angegeben: Erfolg der plans bestimmt Erfolg des goals

Query Goals:

- implizite Zielbedingungen:
 - alle Parameter mit `direction=out` \neq null
 - `parametersets` verfügen über mindestens einen Wert

Maintain Goals:

- `<maintaincondition>` (verpflichtend) – gibt zu erhaltenden Zustand an
- `<targetcondition>` – Zielzustand bei Verletzung der `<maintaincondition>`

Spezielle Bedingungen

Achieve Goals:

- `<targetcondition>` – gibt an, wann goal erreicht ist
- `<failurecondition>` – gibt an, wann goal fehlschlägt
- wenn nicht angegeben: Erfolg der plans bestimmt Erfolg des goals

Query Goals:

- implizite Zielbedingungen:
 - alle Parameter mit `direction=out` \neq null
 - parametersets verfügen über mindestens einen Wert

Maintain Goals:

- `<maintaincondition>` (verpflichtend) – gibt zu erhaltenden Zustand an
- `<targetcondition>` – Zielzustand bei Verletzung der `<maintaincondition>`

Perform Goal

```
<performgoal name="patrol" retry="true"
  exclude="never">
  <contextcondition>
    !$beliefbase.is_loading & &
    !$beliefbase.daytime
  <contextcondition>
</performgoal>
```

Achieve Goal

```
<achievegoal name="moveto">
  <parameter name="loc" class="Location"/>
  <targetcondition>
    $beliefbase.my_loc.isNear($goal.loc)
  </targetcondition>
</achievegoal>
```

Perform Goal

```
<performgoal name="patrol" retry="true"
  exclude="never">
  <contextcondition>
    !$beliefbase.is_loading & &
    !$beliefbase.daytime
  <contextcondition>
</performgoal>
```

Achieve Goal

```
<achievegoal name="moveto">
  <parameter name="loc" class="Location"/>
  <targetcondition>
    $beliefbase.my_loc.isNear($goal.loc)
  </targetcondition>
</achievegoal>
```

Query Goal

```
<querygoal name="qw" exclude="never" retry="true">
  <parameter name="result" class="Wastebin"
    direction="out">
    <value evaluationmode="dynamic">
      select one $wastebin
      from $beliefbase.wastebins
      where !$wastebin.isFull()
      order by $beliefbase.my_location
        .getDistance($wastebin.getLocation())
    </value>
  </parameter>
</querygoal>
```

Maintain Goal

```
<maintaingoal name="battery_loaded">
  <maintaincondition>
    $beliefbase.my_chargestate > 0.2
  </maintaincondition>
  <targetcondition>
    $beliefbase.my_chargestate == 1.0
  </targetcondition>
</maintaingoal>
```

Easy Deliberation

- paralleles Verfolgen aller derzeitig möglichen goals nicht immer sinnvoll \Rightarrow Sperrbeziehungen festlegen
- `<deliberation cardinality="n">` – nur n Instanzen gleichen Goal-Typs zulassen
- `<inhibits ref="gn">` (innerhalb von `<deliberation></deliberation>`) – goal sperrt „gn“
 - Attribut `inhibit="when_in_process"` – Sperrung nur wenn goal gerade verarbeitet wird

Beispiel

```
<maintaingoal name="maintainbatteryloaded"> ...  
  <deliberation>  
    <inhibits ref="performpatrol"  
      inhibit="when_in_process"/>  
  </deliberation> ...  
</maintaingoal>
```

Easy Deliberation

- paralleles Verfolgen aller derzeitig möglichen goals nicht immer sinnvoll \Rightarrow Sperrbeziehungen festlegen
- `<deliberation cardinality="n">` – nur n Instanzen gleichen Goal-Typs zulassen
- `<inhibits ref="gn">` (innerhalb von `<deliberation></deliberation>`) – goal sperrt „gn“
 - Attribut `inhibit="when_in_process"` – Sperrung nur wenn goal gerade verarbeitet wird

Beispiel

```
<maintaingoal name="maintainbatteryloaded"> ...  
  <deliberation>  
    <inhibits ref="performpatrol"  
      inhibit="when_in_process"/>  
  </deliberation> ...  
</maintaingoal>
```

Meta-Level Goals

- falls mehrere passende Pläne zur Behandlung eines goals (oder events) bereitstehen \Rightarrow Plan-Auswahl durch **meta-level reasoning**
- Erzeugung von **meta-level goal** für aufgetretenes goal (oder event)
- Ausführung von **meta-level plans** für meta-level goal
- danach Ergebnis (tatsächlich auszuführende plans) im Parameter **result** des meta-level goals
- **<metagoal>** muss **<trigger>** haben
- **<metagoal>** muss mindestens in-Parameter **"applicables"** und out-Parameter **"result"** (beide vom Typ `jadex.runtime.ICandidateInfo`) haben

Überblick

① Einführung

② Komponenten

Capabilities

Beliefs

Goals

Plans

Events

③ Anhang: Beispiele

Plans

Beliefs

Goals

Trigger und Bedingungen

- **<trigger>** gibt an, wann eine neue Plan-Instanz erzeugt werden soll
 - eventgetrieben: **<internalevent>**, **<messageevent>**, **<goal>**
 - Verfeinerung durch **<parameter>** möglich
 - Übereinstimmung aller angegebenen Parameter
 - datengetrieben: **<condition>**, **<beliefchange>**, **<beliefsetchange>**, **<factadded>**, **<factremoved>**
- Vorbedingungen: **<precondition>**
- Kontextbedingungen: **<contextcondition>**

Beispiel: Reparaturplan

```
<trigger><condition> $beliefbase.out_of_order  
</condition></trigger>  
<contextcondition> $beliefbase.repairable  
</contextcondition>
```

Trigger und Bedingungen

- **<trigger>** gibt an, wann eine neue Plan-Instanz erzeugt werden soll
 - eventgetrieben: **<internalevent>**, **<messageevent>**, **<goal>**
 - Verfeinerung durch **<parameter>** möglich
 - Übereinstimmung aller angegebenen Parameter
 - datengetrieben: **<condition>**, **<beliefchange>**, **<beliefsetchange>**, **<factadded>**, **<factremoved>**
- Vorbedingungen: **<precondition>**
- Kontextbedingungen: **<contextcondition>**

Beispiel: Reparaturplan

```
<trigger><condition> $beliefbase.out_of_order
</condition></trigger>
<contextcondition> $beliefbase.repairable
</contextcondition>
```

Waitqueue und Cleanup

Waitqueue:

- Zeitverzögerung zwischen Eventauslösung und Ausführung des dadurch ausgelösten Planschrittes möglich
- in der Zwischenzeit ausgelöstes Event, das erst nach Ausführung des Planschrittes relevant ist, wird nicht weitergegeben
- Lösung: Spezifiziere, welche Events in die waitqueue sollen innerhalb von `<waitqueue></waitqueue>` oder über `getWaitqueue()` im body.

Cleanup:

- Plan **erfolgreich**, wenn Beendigung ohne exception; Spezifikation des Aufräumcodes mittels `passed()`
- Plan **fehlgeschlagen**, z. B. wenn exception geworfen; Spezifikation des Aufräumcodes mittels `failed()`
- Plan **abgebrochen**, z. B. wenn goal nicht mehr aktuell; Spezifikation des Aufräumcodes mittels `aborted()`

Waitqueue und Cleanup

Waitqueue:

- Zeitverzögerung zwischen Eventauslösung und Ausführung des dadurch ausgelösten Planschrittes möglich
- in der Zwischenzeit ausgelöstes Event, das erst nach Ausführung des Planschrittes relevant ist, wird nicht weitergegeben
- Lösung: Spezifiziere, welche Events in die waitqueue sollen innerhalb von `<waitqueue></waitqueue>` oder über `getWaitqueue()` im body.

Cleanup:

- Plan **erfolgreich**, wenn Beendigung ohne exception; Spezifikation des Aufräumcodes mittels `passed()`
- Plan **fehlgeschlagen**, z. B. wenn exception geworfen; Spezifikation des Aufräumcodes mittels `failed()`
- Plan **abgebrochen**, z. B. wenn goal nicht mehr aktuell; Spezifikation des Aufräumcodes mittels `aborted()`

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Events in Jadex

- Jadex ist eventbasiert; nichts geschieht ohne event.
- `<goalevent>` – wird automatisch ausgelöst
 - process event – active goal soll verfolgt werden
 - info event – Verarbeitung des goals beendet
- `<internalevent>` – explizite Einwege-Kommunikation innerhalb des Agenten
- `<messageevent>` – Nachricht von/nach außen
- `<parameter>` bzw. `<parameterset>` möglich
 - Attribut `direction` $\in \{in, out, inout\}$
- alle events haben Attribute `posttoall`, `metalevelreasoning`, `randomselection`
- weitere benutzertransparente events (z. B. wenn beliefs plans auslösen)

Beispiel: Internal Event

Auszug aus ADF

```
<events>
  <internalevent name="gui_update">
    <parameter name="content" class="String">
  </internalevent>
</events>
```

Auszug aus plan

```
public void body() {
  String update_info; ...
  IInternalEvent event =
    createInternalEvent("gui_update");
  event.getParameter("content")
    .setValue(update_info);
  dispatchInternalEvent(event); ...
}
```

Beispiel: Internal Event

Auszug aus ADF

```
<events>
  <internalevent name="gui_update">
    <parameter name="content" class="String">
  </internalevent>
</events>
```

Auszug aus plan

```
public void body() {
  String update_info; ...
  IInternalEvent event =
    createInternalEvent("gui_update");
  event.getParameter("content")
    .setValue(update_info);
  dispatchInternalEvent(event); ...
}
```

Message Events

- Spezifikation aller zu sendenden und zu empfangenden message event types im ADF notwendig
- **direction** \in {send, receive, send_receive}
- Nachrichtentyp (Attribut type von <event>) gibt mögliche Parameter an; derzeit nur FIPA¹ möglich
- message event type nur lokal bekannt, Vorgehensweise bei eintreffender Nachricht?
 - Parametervergleich: Nachricht \Leftrightarrow message event type
 - nur Berücksichtigung wenn **direction="fixed"**
- Erzeugung: **createMessageEvent(String type)**
- Empfänger: **jadex.adapter.fipa.AgentIdentifier**
- Setze Nachrichteninhalt: **setContent(Object content)**
- Senden: **sendMessage(IMessageEvent me)**
- nutze Parameter **conversion-id** oder **reply-with** zur Zuordnung von Antworten
- direkte Antwort mit **createReply()**

¹Foundation for Intelligent Physical Agents

Beispiel: Message Event

```
<events>
  <messageevent name="request_carry" type="fipa"
    direction="send">
    <parameter name="performative"
      class="String" direction="fixed">
      <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="reply-with" class="String">
      <value>SFipa.createUniqueId(...)</value>
    </parameter>
  </messageevent>
  ...
</events>
```

Beispiel: Message Event

```
<events>
  <messageevent name="request_carry" type="fipa"
    direction="send">
    <parameter name="performative"
      class="String" direction="fixed">
      <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="reply-with" class="String">
      <value>SFipa.createUniqueId(...)</value>
    </parameter>
  </messageevent>
  ...
</events>
```

Beispiel: Message Event

```
<events>
  <messageevent name="request_carry" type="fipa"
    direction="send">
    <parameter name="performative"
      class="String" direction="fixed">
      <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="reply-with" class="String">
      <value>SFipa.createUniqueId(...)</value>
    </parameter>
  </messageevent>
  ...
</events>
```

Beispiel: Message Event

```
<events>
  <messageevent name="request_carry" type="fipa"
    direction="send">
    <parameter name="performative"
      class="String" direction="fixed">
      <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="reply-with" class="String">
      <value>SFipa.createUniqueId(...)</value>
    </parameter>
  </messageevent>
  ...
</events>
```

Literatur

- ▶ Alexander Pokahr, Lars Braubach, Andrzej Walczak. *Jadex User Guide (Release 0.941)*. Internet: <http://tinyurl.com/r4dly>, 2005 (Stand: 2006-03-25).
- ▶ Lars Braubach, Alexander Pokahr, Andrzej Walczak. *Jadex Tutorial (Release 0.941)*. Internet: <http://tinyurl.com/h657v>, 2005 (Stand: 2006-03-25).
- ▶ Diverse Autoren. *Software-Agent*. Internet: <http://de.wikipedia.org/wiki/Software-Agent> (Stand: 2006-03-25).

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

TranslationB1.agent.xml

```
<agent ...>
  <plans>
    <plan name="egtrans">
      <body>new EGTP1B1()</body>
      <waitqueue>
        <messageevent ref="req_t"/>
      </waitqueue>
    </plan>
  </plans>
  <events>
    <messageevent name="req_t"
      direction="receive" type="fipa">
      <parameter name="performative"
        class="String" direction="fixed">
```

TranslationB1.agent.xml

```
<agent ...>
  <plans>
    <plan name="egtrans">
      <body>new EGTP1B1()</body>
      <waitqueue>
        <messageevent ref="req_t"/>
      </waitqueue>
    </plan>
  </plans>
  <events>
    <messageevent name="req_t"
      direction="receive" type="fipa">
      <parameter name="performative"
        class="String" direction="fixed">
```

TranslationB1.agent.xml

```
<agent ...>
  <plans>
    <plan name="egtrans">
      <body>new EGTP1B1()</body>
      <waitqueue>
        <messageevent ref="req_t"/>
      </waitqueue>
    </plan>
  </plans>
  <events>
    <messageevent name="req_t"
      direction="receive" type="fipa">
      <parameter name="performative"
        class="String" direction="fixed">
```

```
        <value>
            jadex.adapter.fipa.SFipa.REQUEST
        </value>
    </parameter>
</messageevent>
</events>
<initialstates>
    <initialstate name="default">
        <plans>
            <initialplan ref="egtrans"/>
        </plans>
    </initialstate>
</initialstates>
</agent>
```

```
<value>
    jadex.adapter.fipa.SFipa.REQUEST
</value>
</parameter>
</messageevent>
</events>
<initialstates>
  <initialstate name="default">
    <plans>
      <initialplan ref="egtrans"/>
    </plans>
  </initialstate>
</initialstates>
</agent>
```

```
        <value>
            jadex.adapter.fipa.SFipa.REQUEST
        </value>
    </parameter>
</messageevent>
</events>
<initialstates>
    <initialstate name="default">
        <plans>
            <initialplan ref="egtrans"/>
        </plans>
    </initialstate>
</initialstates>
</agent>
```

EGTP1B1.java

```
import java.util.*;
import jadex.runtime.*;

public class EGTP1B1 extends Plan {
    HashMap h = new HashMap();
    public EGTP1B1() {
        h.put("firefly", "Gluehwuermchen");
        h.put("serenity", "Gelassenheit");
    }

    public void body() {
        while(true) {
            IMessageEvent me
                = waitForMessageEvent("req_t");
```

EGTP1B1.java

```
import java.util.*;
import jadex.runtime.*;

public class EGTP1B1 extends Plan {
    HashMap h = new HashMap();
    public EGTP1B1() {
        h.put("firefly", "Gluehwuermchen");
        h.put("serenity", "Gelassenheit");
    }

    public void body() {
        while(true) {
            IMessageEvent me
                = waitForMessageEvent("req_t");
```

EGTP1B1.java

```
import java.util.*;
import jadex.runtime.*;

public class EGTP1B1 extends Plan {
    HashMap h = new HashMap();
    public EGTP1B1() {
        h.put("firefly", "Gluehwuermchen");
        h.put("serenity", "Gelassenheit");
    }

    public void body() {
        while(true) {
            IMessageEvent me
                = waitForMessageEvent("req_t");
```

Service Plan 3/4

EGTP1B1.java

```
import java.util.*;
import jadex.runtime.*;

public class EGTP1B1 extends Plan {
    HashMap h = new HashMap();
    public EGTP1B1() {
        h.put("firefly", "Gluehwuermchen");
        h.put("serenity", "Gelassenheit");
    }

    public void body() {
        while(true) {
            IMessageEvent me
                = waitForMessageEvent("req_t");
```

Service Plan 4/4

```
String eword = (String)me.getContent();
if (h.containsKey(eword)==true) {
    String gword = (String)h.get(eword);
    System.out.println(eword+" - "+gword);
}
else {
    System.out.println("Not in database");
}
}
}
}
```

Passive Plan 1/2

TranslationB2.agent.xml

TranslationB1.agent.xml mit folgenden Änderungen:

- kein `<initialstates></initialstates>`
- `<trigger>` statt `<waitqueue>`, also:

```
<plans>
  <plan name="egtrans">
    <body>new EGTP1B2()</body>
    <trigger>
      <messageevent ref="req_t"/>
    </trigger>
  </plan>
</plans>
```

Passive Plan 1/2

TranslationB2.agent.xml

TranslationB1.agent.xml mit folgenden Änderungen:

- kein `<initialstates></initialstates>`
- `<trigger>` statt `<waitqueue>`, also:

```
<plans>
  <plan name="egtrans">
    <body>new EGTP1B2()</body>
    <trigger>
      <messageevent ref="req_t"/>
    </trigger>
  </plan>
</plans>
```

Passive Plan 2/2

EGTP1B2.java

EGTP1B1.java mit folgenden Änderungen:

- Wegfall der `while(true)`-Schleife
- `IMessageEvent me = (IMessageEvent)getInitialEvent();`
statt
`IMessageEvent me = waitForMessageEvent("req_t");`

Plan-Auswahl 1/2

EGTP1B3.java

EGTP1B2.java zzgl. public boolean containsWord(...)

TranslationB3.agent.xml

TranslationB2.agent.xml mit folgenden Änderungen:

- `<precondition>` für plan „egtrans“:

```
<precondition>
```

```
EGTP1B3.containsWord((String)$event.getContent())
```

```
</precondition>
```

- Definition neuen plans zur Online-Suche:

```
<plan name="searchonline" priority="-1">
```

```
<body>new SearchOnline()</body>
```

```
<trigger>
```

```
<messageevent ref="req_t"/>
```

```
</trigger>
```

```
</plan>
```

Plan-Auswahl 1/2

EGTP1B3.java

EGTP1B2.java zzgl. public boolean containsWord(...)

TranslationB3.agent.xml

TranslationB2.agent.xml mit folgenden Änderungen:

- `<precondition>` für plan „egtrans“:

```
<precondition>
```

```
EGTP1B3.containsWord((String)$event.getContent())
```

```
</precondition>
```

- Definition neuen plans zur Online-Suche:

```
<plan name="searchonline" priority="-1">
```

```
  <body>new SearchOnline()</body>
```

```
  <trigger>
```

```
    <messageevent ref="req_t"/>
```

```
  </trigger>
```

```
</plan>
```

Plan-Auswahl 1/2

EGTP1B3.java

EGTP1B2.java zzgl. public boolean containsWord(...)

TranslationB3.agent.xml

TranslationB2.agent.xml mit folgenden Änderungen:

- `<precondition>` für plan „egtrans“:

```
<precondition>
```

```
EGTP1B3.containsWord((String)$event.getContent())
```

```
</precondition>
```

- Definition neuen plans zur Online-Suche:

```
<plan name="searchonline" priority="-1">
```

```
  <body>new SearchOnline()</body>
```

```
  <trigger>
```

```
    <messageevent ref="req_t"/>
```

```
  </trigger>
```

```
</plan>
```

SearchOnline.java

```
public void body() {
    IMessageEvent me =
        (IMessageEvent)getInitialEvent();
    String eword = (String)me.getContent();
    try {
        URL dict = new URL("http://wolfram.
            schneider.org/dict/dict.cgi?query="+eword);
        BufferedReader in = new BufferedReader
            (new InputStreamReader(dict.openStream()));
        // Bearbeite in und gibt Uebersetzung aus
    }
    catch (Exception e) {}
}
```

Plan-Auswahl 2/2

SearchOnline.java

```
public void body() {
    IMessageEvent me =
        (IMessageEvent)getInitialEvent();
    String eword = (String)me.getContent();
    try {
        URL dict = new URL("http://wolfram.
            schneider.org/dict/dict.cgi?query="+eword);
        BufferedReader in = new BufferedReader
            (new InputStreamReader(dict.openStream()));
        // Bearbeite in und gibt Uebersetzung aus
    }
    catch (Exception e) {}
}
```

Plan-Auswahl 2/2

SearchOnline.java

```
public void body() {
    IMessageEvent me =
        (IMessageEvent)getInitialEvent();
    String eword = (String)me.getContent();
    try {
        URL dict = new URL("http://wolfram.
            schneider.org/dict/dict.cgi?query="+eword);
        BufferedReader in = new BufferedReader
            (new InputStreamReader(dict.openStream()));
        // Bearbeite in und gibt Uebersetzung aus
    }
    catch (Exception e) {}
}
```

Plan-Auswahl 2/2

SearchOnline.java

```
public void body() {
    IMessageEvent me =
        (IMessageEvent)getInitialEvent();
    String eword = (String)me.getContent();
    try {
        URL dict = new URL("http://wolfram.
            schneider.org/dict/dict.cgi?query="+eword);
        BufferedReader in = new BufferedReader
            (new InputStreamReader(dict.openStream()));
        // Bearbeite in und gibt Uebersetzung aus
    }
    catch (Exception e) {}
}
```

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Belief 1/4

- neues Nachrichtenformat: `<action> <language(s)> <content>`, also z. B. „translate english_german doughnut“ oder „add english_german doughnut Berliner“
- Wörterbuch nun als belief realisiert

TranslationC1.agent.xml

```
<agent ...>
  <beliefs>
    <belief name="egw" class="Map">
      <fact>EGTP1C1.getDictionary()</fact>
    </belief>
  </beliefs>
  <plans>
    <plan name="addword">
      <body>new EGAddPlC1()</body>
    </plan>
  </plans>
</agent>
```

- neues Nachrichtenformat: `<action> <language(s)> <content>`, also z. B. „translate english_german doughnut“ oder „add english_german doughnut Berliner“
- Wörterbuch nun als belief realisiert

TranslationC1.agent.xml

```
<agent ...>
  <beliefs>
    <belief name="egw" class="Map">
      <fact>EGTP1C1.getDictionary()</fact>
    </belief>
  </beliefs>
  <plans>
    <plan name="addword">
      <body>new EGAddP1C1()</body>
```

- neues Nachrichtenformat: `<action> <language(s)> <content>`, also z. B. „translate english_german doughnut“ oder „add english_german doughnut Berliner“
- Wörterbuch nun als belief realisiert

TranslationC1.agent.xml

```
<agent ...>
  <beliefs>
    <belief name="egw" class="Map">
      <fact>EGTP1C1.getDictionary()</fact>
    </belief>
  </beliefs>
  <plans>
    <plan name="addword">
      <body>new EGAddPlC1()</body>
```

- neues Nachrichtenformat: `<action> <language(s)> <content>`, also z. B. „translate english_german doughnut“ oder „add english_german doughnut Berliner“
- Wörterbuch nun als belief realisiert

TranslationC1.agent.xml

```
<agent ...>
  <beliefs>
    <belief name="egw" class="Map">
      <fact>EGTP1C1.getDictionary()</fact>
    </belief>
  </beliefs>
  <plans>
    <plan name="addword">
      <body>new EGAddP1C1()</body>
```

```
<trigger>
  <messageevent ref="req_add"/>
</trigger>
</plan>
<plan name="egtrans">
  <body>new EGTP1C1()</body>
  <trigger>
    <messageevent ref="req_t"/>
  </trigger>
</plan>
</plans>
<events>
  <messageevent name="req_add"
    direction="receive" type="fipa">
    <parameter name="performative"
      class="String" direction="fixed">
```

```
<trigger>
  <messageevent ref="req_add"/>
</trigger>
</plan>
<plan name="egtrans">
  <body>new EGTP1C1()</body>
  <trigger>
    <messageevent ref="req_t"/>
  </trigger>
</plan>
</plans>
<events>
  <messageevent name="req_add"
    direction="receive" type="fipa">
    <parameter name="performative"
      class="String" direction="fixed">
```

```
<trigger>
  <messageevent ref="req_add"/>
</trigger>
</plan>
<plan name="egtrans">
  <body>new EGTP1C1()</body>
  <trigger>
    <messageevent ref="req_t"/>
  </trigger>
</plan>
</plans>
<events>
  <messageevent name="req_add"
    direction="receive" type="fipa">
    <parameter name="performative"
      class="String" direction="fixed">
```

```
<trigger>
  <messageevent ref="req_add"/>
</trigger>
</plan>
<plan name="egtrans">
  <body>new EGTP1C1()</body>
  <trigger>
    <messageevent ref="req_t"/>
  </trigger>
</plan>
</plans>
<events>
  <messageevent name="req_add"
    direction="receive" type="fipa">
    <parameter name="performative"
      class="String" direction="fixed">
```

```
        <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="content-start"
        class="String" direction="fixed">
        <value>"add english_german"</value>
    </parameter>
</messageevent>
<messageevent name="req_t"
    direction="receive" type="fipa">
    <parameter name="performative"
        class="String" direction="fixed">
        <value>
            jadex.adapter.fipa.SFipa.REQUEST
        </value>
    </parameter>
```

```
        <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="content-start"
        class="String" direction="fixed">
        <value>"add english_german"</value>
    </parameter>
</messageevent>
<messageevent name="req_t"
    direction="receive" type="fipa">
    <parameter name="performative"
        class="String" direction="fixed">
        <value>
            jadex.adapter.fipa.SFipa.REQUEST
        </value>
    </parameter>
```

```
        <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="content-start"
        class="String" direction="fixed">
        <value>"add english_german"</value>
    </parameter>
</messageevent>
<messageevent name="req_t"
    direction="receive" type="fipa">
    <parameter name="performative"
        class="String" direction="fixed">
        <value>
            jadex.adapter.fipa.SFipa.REQUEST
        </value>
    </parameter>
```

Belief 4/4

```
<parameter name="content-start"  
  class="String" direction="fixed">  
  <value>  
    "translate english_german"  
  </value>  
</parameter>  
</messageevent>  
</events>  
</agent>
```

EGTP1C1.java / EGAddPlC1.java

- EGTP1C1.java ähnlich EGTP1B2.java
- statische Methode getDictionary() liefert Map
- Zugriff auf belief durch Map wordtable = (Map) getBeliefbase().getBelief("egw").getFact();
- Auswahl des Wortes durch StringTokenizer

```
<parameter name="content-start"  
  class="String" direction="fixed">  
  <value>  
    "translate english_german"  
  </value>  
</parameter>  
</messageevent>  
</events>  
</agent>
```

EGTP1C1.java / EGAddPlC1.java

- EGTP1C1.java ähnlich EGTP1B2.java
- statische Methode getDictionary() liefert Map
- Zugriff auf belief durch Map wordtable = (Map) getBeliefbase().getBelief("egw").getFact();
- Auswahl des Wortes durch StringTokenizer

TranslationC2.agent.xml

TranslationC1.agent.xml mit folgenden Änderungen:

```
<beliefs>
  <beliefset name="egw" class="Tuple">
    <fact>new Tuple("scope", "Weite")</fact>
    <fact>new Tuple("tranquility", "Ruhe")</fact>
  </beliefset>
</beliefs>
<expressions>
  <expression name="query_egw">
    select one $wordpair.get(1)
    from Tuple $wordpair in $beliefbase.egw
    where $wordpair.get(0).equals($sword)
    <parameter name="$sword" class="String"/>
  </expression></expressions>
```

TranslationC2.agent.xml

TranslationC1.agent.xml mit folgenden Änderungen:

```
<beliefs>
  <beliefset name="egw" class="Tuple">
    <fact>new Tuple("scope", "Weite")</fact>
    <fact>new Tuple("tranquility", "Ruhe")</fact>
  </beliefset>
</beliefs>
<expressions>
  <expression name="query_egw">
    select one $wordpair.get(1)
    from Tuple $wordpair in $beliefbase.egw
    where $wordpair.get(0).equals($sword)
    <parameter name="$sword" class="String"/>
  </expression></expressions>
```

TranslationC2.agent.xml

TranslationC1.agent.xml mit folgenden Änderungen:

```
<beliefs>
  <beliefset name="egw" class="Tuple">
    <fact>new Tuple("scope", "Weite")</fact>
    <fact>new Tuple("tranquility", "Ruhe")</fact>
  </beliefset>
</beliefs>
<expressions>
  <expression name="query_egw">
    select one $wordpair.get(1)
    from Tuple $wordpair in $beliefbase.egw
    where $wordpair.get(0).equals($sword)
    <parameter name="$sword" class="String"/>
  </expression></expressions>
```

EGTP1C2.java

```
import java.util.*; import jadex.runtime.*;
public class EGTP1C2 extends Plan {
    protected IExpression queryword;
    public EGTP12() {
        this.queryword = getExpression("query_egw");
    }
    public void body() {
        MessageEvent me =
            (IMessageEvent)getInitialEvent();
        StringTokenizer stok =
            new StringTokenizer((String)me.getContent());
        stok.nextToken(); stok.nextToken();
        String eword = stok.nextToken();
        String gword =
            (String)queryword.execute("$eword", eword);
        ...
    }
}
```

EGTP1C2.java

```
import java.util.*; import jadex.runtime.*;

public class EGTP1C2 extends Plan {
    protected IExpression queryword;

    public EGTP12() {
        this.queryword = getExpression("query_egw");
    }

    public void body() {
        MessageEvent me =
            (IMessageEvent)getInitialEvent();
        StringTokenizer stok =
            new StringTokenizer((String)me.getContent());
        stok.nextToken(); stok.nextToken();
        String eword = stok.nextToken();
        String gword =
            (String)queryword.execute("$eword", eword);
        ...
    }
}
```

EGTP1C2.java

```
import java.util.*; import jadex.runtime.*;

public class EGTP1C2 extends Plan {
    protected IExpression queryword;
    public EGTP12() {
        this.queryword = getExpression("query_egw");
    }
    public void body() {
        MessageEvent me =
            (IMessageEvent)getInitialEvent();
        StringTokenizer stok =
            new StringTokenizer((String)me.getContent());
        stok.nextToken(); stok.nextToken();
        String eword = stok.nextToken();
        String gword =
            (String)queryword.execute("$eword", eword);
        ...
    }
}
```

EGTP1C2.java

```
import java.util.*; import jadex.runtime.*;

public class EGTP1C2 extends Plan {
    protected IExpression queryword;
    public EGTP12() {
        this.queryword = getExpression("query_egw");
    }
    public void body() {
        MessageEvent me =
            (MessageEvent)getInitialEvent();
        StringTokenizer stok =
            new StringTokenizer((String)me.getContent());
        stok.nextToken(); stok.nextToken();
        String eword = stok.nextToken();
        String gword =
            (String)queryword.execute("$eword", eword);
        ...
    }
}
```

Belief Condition 1/2

Gratuliere jedem 10. Anfrager!

TranslationC3.agent.xml

TranslationC2.agent.xml mit folgenden Ergänzungen in den jeweiligen Sektionen:

```
<belief name="transcnt" class="int">  
<fact>0</fact> </belief>
```

```
<plan name="thankyou">  
  <body>new ThankYouPlanC3()</body>  
  <trigger>  
    <condition>  
      $beliefbase.transcnt>0 &&  
      $beliefbase.transcnt%10==0  
    </condition>  
  </trigger></plan>
```

Belief Condition 1/2

Gratuliere jedem 10. Anfrager!

TranslationC3.agent.xml

TranslationC2.agent.xml mit folgenden Ergänzungen in den jeweiligen Sektionen:

```
<belief name="transcnt" class="int">  
<fact>0</fact> </belief>
```

```
<plan name="thankyou">  
  <body>new ThankYouPlanC3()</body>  
  <trigger>  
    <condition>  
      $beliefbase.transcnt>0 &&  
      $beliefbase.transcnt%10==0  
    </condition>  
  </trigger></plan>
```

Belief Condition 1/2

Gratuliere jedem 10. Anfrager!

TranslationC3.agent.xml

TranslationC2.agent.xml mit folgenden Ergänzungen in den jeweiligen Sektionen:

```
<belief name="transcnt" class="int">  
<fact>0</fact> </belief>
```

```
<plan name="thankyou">  
  <body>new ThankYouPlanC3()</body>  
  <trigger>  
    <condition>  
      $beliefbase.transcnt>0 &&  
      $beliefbase.transcnt%10==0  
    </condition>  
  </trigger></plan>
```

Belief Condition 2/2

EGTP1C3.java

EGTP1C2.java mit folgender Ergänzung:

```
int cnt = ((Integer)getBeliefbase().
    getBelief("transcnt").getFact()).intValue();
getBeliefbase().getBelief("transcnt").
    setFact(new Integer(cnt+1));
```

ThankYouPlanC3.java

Gib Gratulationsnachricht aus!

Belief Condition 2/2

EGTP1C3.java

EGTP1C2.java mit folgender Ergänzung:

```
int cnt = ((Integer)getBeliefbase().
    getBelief("transcnt").getFact()).intValue();
getBeliefbase().getBelief("transcnt").
    setFact(new Integer(cnt+1));
```

ThankYouPlanC3.java

Gib Gratulationsnachricht aus!

Überblick

1 Einführung

2 Komponenten

Capabilities

Beliefs

Goals

Plans

Events

3 Anhang: Beispiele

Plans

Beliefs

Goals

Achieve Goal 1/5

- Zusätzliche Übersetzungsrichtung Englisch → Französisch
- neuer Plan ProcessT soll Übersetzungsanfragen entgegennehmen und goals dispatchen

TranslationE1.agent.xml

basiert auf TranslationC2.agent.xml mit Änderungen:

```
<goals>
  <achievegoal name="translate">
    <parameter name="direction" class="String"/>
    <parameter name="word" class="String"/>
    <parameter name="result" class="String"
      direction="out"/>
  </achievegoal>
</goals>
```

Achieve Goal 1/5

- Zusätzliche Übersetzungsrichtung Englisch → Französisch
- neuer Plan ProcessT soll Übersetzungsanfragen entgegennehmen und goals dispatchen

TranslationE1.agent.xml

basiert auf TranslationC2.agent.xml mit Änderungen:

```
<goals>  
  <achievegoal name="translate">  
    <parameter name="direction" class="String"/>  
    <parameter name="word" class="String"/>  
    <parameter name="result" class="String"  
      direction="out"/>  
  </achievegoal>  
</goals>
```

Achieve Goal 2/5

```
<plans>
  <plan name="process">
    <body>
      new ProcessT()
    </body>
    <waitqueue>
      <messageevent ref="req_t"/>
    </waitqueue>
  </plan>
  <plan name="egtrans">
    <parameter name="word" class="String">
      <goalmapping ref="translate.word"/>
    </parameter>
    <parameter name="result" class="String">
      <goalmapping ref="translate.result"/>
    </parameter>
  </plan>
</plans>
```

Achieve Goal 2/5

```
<plans>
  <plan name="process">
    <body>
      new ProcessT()
    </body>
    <waitqueue>
      <messageevent ref="req_t"/>
    </waitqueue>
  </plan>
  <plan name="egtrans">
    <parameter name="word" class="String">
      <goalmapping ref="translate.word"/>
    </parameter>
    <parameter name="result" class="String">
      <goalmapping ref="translate.result"/>
    </parameter>
  </plan>
</plans>
```

Achieve Goal 2/5

```
<plans>
  <plan name="process">
    <body>
      new ProcessT()
    </body>
  </plan>
  <plan name="egtrans">
    <parameter name="word" class="String">
      <goalmapping ref="translate.word"/>
    </parameter>
    <parameter name="result" class="String">
      <goalmapping ref="translate.result"/>
    </parameter>
  </plan>
</plans>
```

Achieve Goal 3/5

```
<body>new EGTP1E1()</body>
<trigger>
  <goal ref="translate">
    <parameter ref="direction">
      <value>"english_german"</value>
    </parameter>
  </goal>
</trigger>
</plan> <!-- Franzoesisch analog -->
</plans>
<initialstates>
  <!-- Starte service plan ProccesT wie in
    TranslationB1.agent.xml -->
</initialstates>
```

Achieve Goal 3/5

```
<body>new EGTP1E1()</body>
<trigger>
  <goal ref="translate">
    <parameter ref="direction">
      <value>"english_german"</value>
    </parameter>
  </goal>
</trigger>
</plan> <!-- Franzoesisch analog -->
</plans>
<initialstates>
  <!-- Starte service plan ProceST wie in
        TranslationB1.agent.xml -->
</initialstates>
```

Achieve Goal 3/5

```
<body>new EGTP1E1()</body>
<trigger>
  <goal ref="translate">
    <parameter ref="direction">
      <value>"english_german"</value>
    </parameter>
  </goal>
</trigger>
</plan> <!-- Franzoesisch analog -->
</plans>
<initialstates>
  <!-- Starte service plan ProceST wie in
        TranslationB1.agent.xml -->
</initialstates>
```

ProcessT.java (Auszug)

```
while (true) {  
    // hole Nachricht und zerlege sie in action,  
    // dir und word mittels StringTokenizer  
    IGoal goal = createGoal("translate");  
    goal.getParameter("direction").setValue(dir);  
    goal.getParameter("word").setValue(word);  
    try {  
        dispatchSubgoalAndWait(goal);  
        System.out.println(  
            goal.getParameter("result").getValue());  
    }  
    catch(GoalFailureException e) {  
        System.out.println("Word not in database!")  
    }  
};  
}
```

ProcessT.java (Auszug)

```
while (true) {  
    // hole Nachricht und zerlege sie in action,  
    // dir und word mittels StringTokenizer  
    IGoal goal = createGoal("translate");  
    goal.getParameter("direction").setValue(dir);  
    goal.getParameter("word").setValue(word);  
    try {  
        dispatchSubgoalAndWait(goal);  
        System.out.println(  
            goal.getParameter("result").getValue());  
    } catch (GoalFailureException e) {  
        System.out.println("Word not in database!")  
    };  
};  
}
```

ProcessT.java (Auszug)

```
while (true) {  
    // hole Nachricht und zerlege sie in action,  
    // dir und word mittels StringTokenizer  
    IGoal goal = createGoal("translate");  
    goal.getParameter("direction").setValue(dir);  
    goal.getParameter("word").setValue(word);  
    try {  
        dispatchSubgoalAndWait(goal);  
        System.out.println(  
            goal.getParameter("result").getValue());  
    } catch (GoalFailureException e) {  
        System.out.println("Word not in database!")  
    };  
};  
}
```

ProcessT.java (Auszug)

```
while (true) {  
    // hole Nachricht und zerlege sie in action,  
    // dir und word mittels StringTokenizer  
    IGoal goal = createGoal("translate");  
    goal.getParameter("direction").setValue(dir);  
    goal.getParameter("word").setValue(word);  
    try {  
        dispatchSubgoalAndWait(goal);  
        System.out.println(  
            goal.getParameter("result").getValue());  
    } catch (GoalFailureException e) {  
        System.out.println("Word not in database!")  
    };  
};  
}
```

ProcessT.java (Auszug)

```
while (true) {  
    // hole Nachricht und zerlege sie in action,  
    // dir und word mittels StringTokenizer  
    IGoal goal = createGoal("translate");  
    goal.getParameter("direction").setValue(dir);  
    goal.getParameter("word").setValue(word);  
    try {  
        dispatchSubgoalAndWait(goal);  
        System.out.println(  
            goal.getParameter("result").getValue());  
    }  
    catch(GoalFailureException e) {  
        System.out.println("Word not in database!")  
    };  
};  
}
```

Achieve Goal 5/5

EGTP1E1.java

- grundsätzlich analog zu EGTP1C2.java
- Anfrage wird nicht mehr durch StringTokenizer zerlegt, sondern liegt als Parameter vor, also:

```
String eword =
```

```
    (String)getParameter("word").getValue();
```

- Ausgabe findet nun durch ProcessT statt, EGTP1C2 setzt daher Ergebnis als Rückgabeparameter:

```
getParameter("result").setValue(gword);
```

- wirft exception, wenn keine Übersetzung gefunden:

```
throw new PlanFailureException();
```