

Jadex BDI- Agent System

Übersicht

1. Einleitung

2. Die BDI-Komponenten in Jadex

3. Easy Deliberation Strategy

4. Der BDI-Interpreter in Jadex

5. Ausführungsmodell

6. Realisierung der Plattform

7. Sprache

8. Agentenentwicklung mit Jadex

9. Zusammenfassung, Ausblick, Quellen

Was ist Jadex?

- Software-Framework
- Bau von zielorientierten BDI-Agenten
- reaktive und proaktive (Multi-) Agentensystem
- Ursprung im Projekt „MedPAge“ (Medical Path Agents)

Ziele der Entwicklung

- Die meisten Agentenplattformen wurden mit bestimmten technologischen Zielen entwickelt.
- Vorteile beider Gebiete zusammenzubringen
- BDI-Schwächen anderer Systeme ausbessern
 - explizite Goals und verbesserter Deliberationsprozess
 - konkurrierende Goals zulassen
 - flexibler Interpreter
- einfache Agentenentwicklung, der Programmierer soll kein KI-Experte sein müssen.

BDI-Modell

- von Bratman als Theorie über den menschlichen Entscheidungsprozess entwickelt
- Aktionen basieren auf Wünschen und Absichten
- Rao & Georgeff: Beliefs, Desires & Intentions sind geistige Einstellungen und werden als mögliche Weltzustände dargestellt
- in vielen Systemen gibt es aber Vereinfachungen:
 - nur Beliefs werden explizit definiert
 - Desires sind nur Events
 - Intentions ergeben sich aus den gerade laufenden Plänen

Übersicht

1. Einleitung
- 2. Die BDI-Komponenten in Jadex**
3. Easy Deliberation Strategy
4. Der BDI-Interpreter in Jadex
5. Ausführungsmodell
6. Realisierung der Plattform
7. Sprache
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

Beliefs

- werden in Jadex in einer objektorientierten Form dargestellt
- benannte Fakten oder benannte Faktmengen
- Anfragen und Operationen durch Expressions (OQL)
- Belief-Änderungen können über Events mitgeteilt werden und so Seiteneffekte auslösen

Desires & Goals

- Desires sind Wünsche des Agenten
- viele Systeme bieten keinen Mechanismus positive und negative Beziehungen zwischen Zielen auf der Ebene der Architektur zu behandeln
- Goals sind aktuelle konkrete Instanzen der Desires
- ein Auswahlprozess (Goal-Deliberation) hat die Aufgabe, eine konsistente Teilmenge der Desires zu bestimmen

Pläne

- Möglichkeiten, mit denen der Agent seine Goals erreichen oder auf Events reagieren kann
- Hierarchie: Pläne können auch Subgoals erzeugen
- Header enthält Conditions
- Body ist vordefinierte Abfolge von Aktionen
- wiederverwendbar in anderen Agenten
- Zugriff auf andere Javaklassen möglich
- spezielle API für Nachrichtenversand, Beliefmanipulation und Subgoalerzeugung

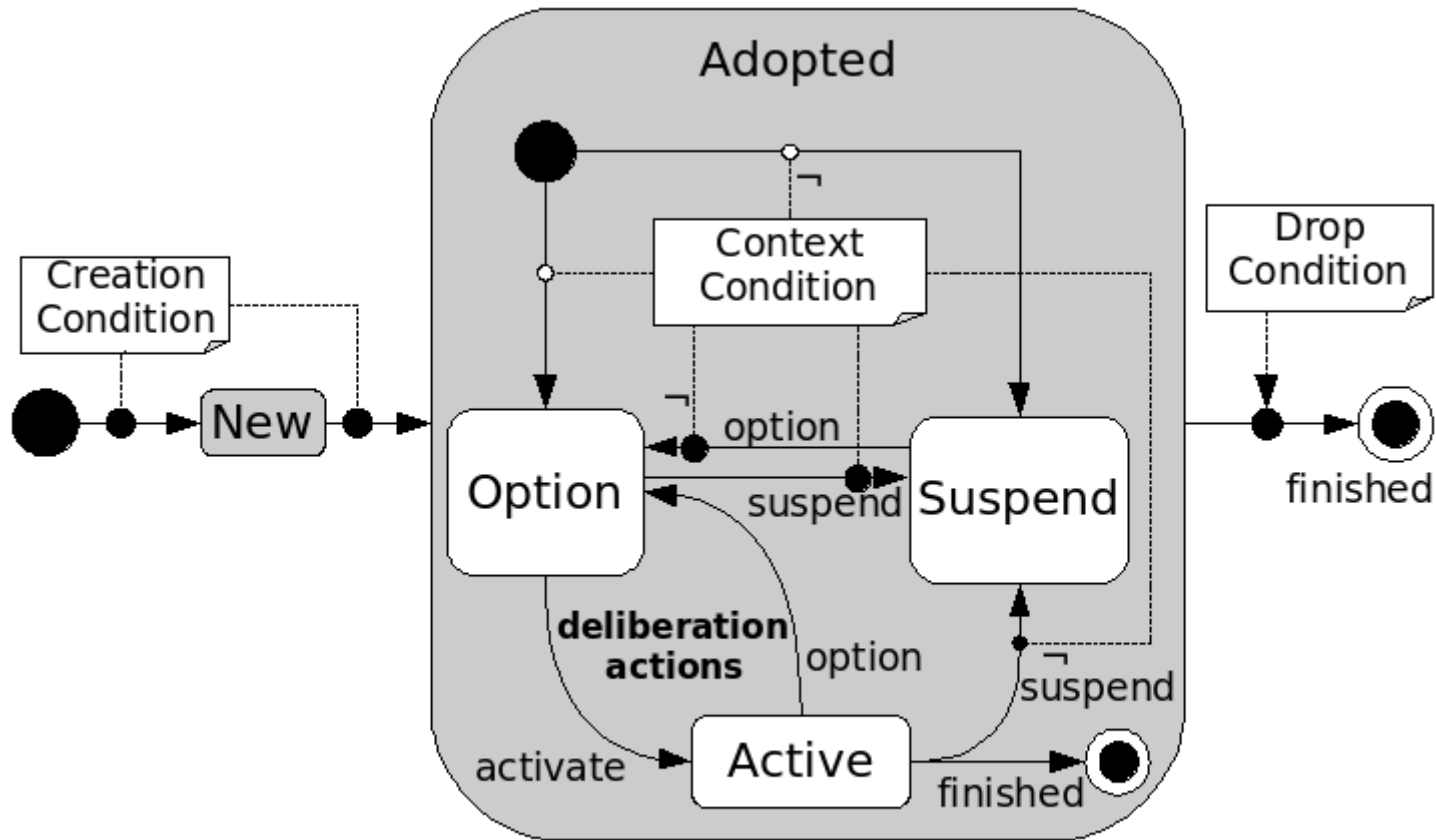
Goals & Pläne

- für jedes Goal werden passende Aktionen ausgeführt
- Goals ohne aktuelle Pläne
- in Jadex gibt es Toplevel-Goals und Subgoals
- das Goalkonzept erlaubt dem Agenten,
 - Pläne zu wiederholen oder andere Pläne auszuprobieren
 - proaktives Verhalten zu entwickeln

Explizite Goals

- Deliberationsprozess kann jederzeit auf Goals zugreifen
- Goal Lifecycle repräsentiert den Zustand des Goals auf Basis seiner Beziehungen zu anderen Goals
- Jedem Goal können verschiedene Bedingungen mitgegeben werden
- zur Laufzeit ändert sich der Zustand des Goals auf Basis dieser Bedingungen
- Agent kann den Zustand aber auch bewußt ändern

Goal-Lifecycle



Goal-Typen in Jadex

- Perform: Aktionen werden ohne Rücksicht auf das Ergebnis ausgeführt
- Achieve: bestimmter Weltzustand soll erreicht werden. Der Agent kann verschiedene Pläne durchprobieren, bis das Goal erreicht ist
- Query: der Agent möchte den internen Zustand so ändern, daß bestimmte Informationen vorhanden sind.
- Maintain: Überwachen eines Weltzustands und ggf. Ausführen von Plänen, um den Zustand zu halten.

Capabilities

- Gruppierungsmechanismus für BDI-Elemente
- separates XML-Dokument
- eigener Namensraum
- explizite Referenzierung mit Scope
- konkrete Elemente (z.B. Beliefs) sind von außen nicht zugreifbar, können aber exportiert werden.
- die äußere Capability muß eine Referenz mit symbolischen Namen auf das Element anlegen
- abstrakte Elemente: Referenz in der inneren Capability auf ein Element der äußeren Capability

Übersicht

1. Einleitung
2. Die BDI-Komponenten in Jadex
- 3. Easy Deliberation Strategy**
4. Der BDI-Interpreter in Jadex
5. Ausführungsmodell
6. Realisierung der Plattform
7. Sprache
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

Easy Deliberation Strategy

- Faktoren, die den Entscheidungsprozess beeinflussen:
 - es werden nur Informationen über Goals benutzt
 - Kardinalität und Ausschlussbeziehungen
- Häufigkeit (auf Anfrage, neues Goal, Kontextänderungen)
- beim Deliberationsprozess werden Teilmengen der vorhandenen Goals betrachtet
- Kardinalität und Ausschlussbeziehungen werden direkt im ADF spezifiziert

Easy Deliberation Strategy

Deliberate new option

- Option kann aktiviert werden, wenn es kein aktives Goal gibt, welches eine Beziehung zur Option hat, d.h. $(A, O) \notin R$
- die Anzahl der aktiven Goals gleichen Typs ist kleiner als in der ADF definiert
- welche aktiven Goals müssen aufgrund von Beziehungen (O, A) deaktiviert werden?
- Aktivierung bzw. Deaktivierung der berechneten Goalmengen

Easy Deliberation Strategy

Deliberate deactivated goal

- Bestimmung der Menge von Optionen gleichen Typs oder mit Beziehung vom deaktivierten Goal zur Option
- Aktivierung der Goals in der berechneten Menge durch die „*Deliberate new option*“-Operation

Easy Deliberation Strategy

Einschränkungen:

- Konflikte zwischen Subgoals können nicht immer optimal gelöst werden
- Konflikte zwischen Plänen werden nicht erkannt (z.B. bei Resourcezugriffen)
- positive Beziehungen werden nicht erkannt (z.B. Finden von gemeinsamen Subgoals)

Übersicht

1. Einleitung
2. Die BDI-Komponenten in Jadex
3. Easy Deliberation Strategy
- 4. Der BDI-Interpreter in Jadex**
5. Ausführungsmodell
6. Realisierung der Plattform
7. Sprache
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

BDI-Interpreter

```
01 initialize-state();
02 repeat
03   options := options-generator(event-queue); \Suche nach
04   selected-options := deliberate(options);      /passenden Plänen
05   update-intentions(selected-options);         \Ausführung der
06   execute();                                   /Pläne
07   get-new-external-events();
08   drop-successful-attitudes();                 \betroffene
09   drop-impossible-attitudes();                /Absichten updaten
10 end repeat
```

Nachteile:

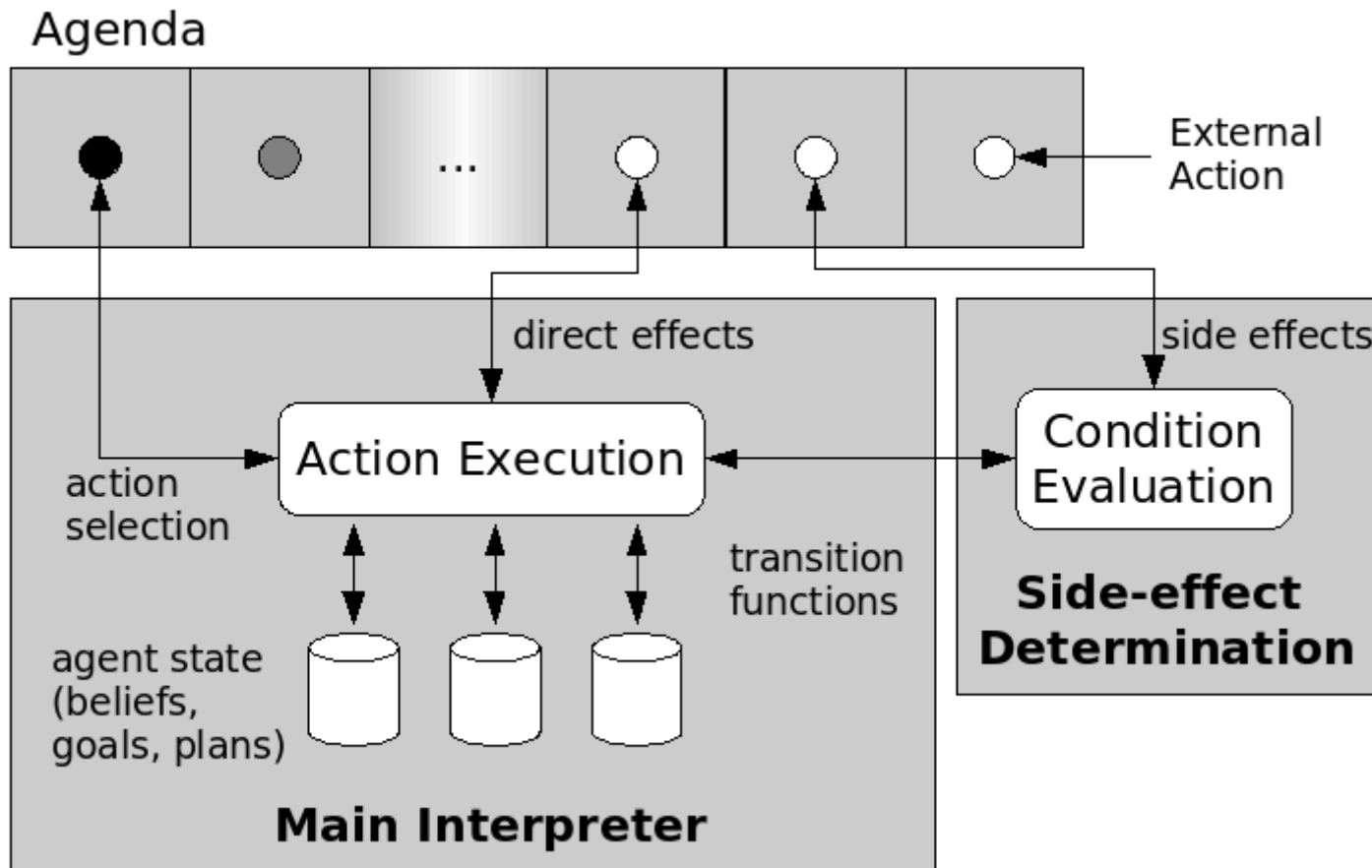
- fester Zyklus, starres Verhalten
- kaum Ansatzpunkte für die Integration zusätzlicher Reasoning-Komponenten

BDI-Interpreter

Meta-Aktionen:

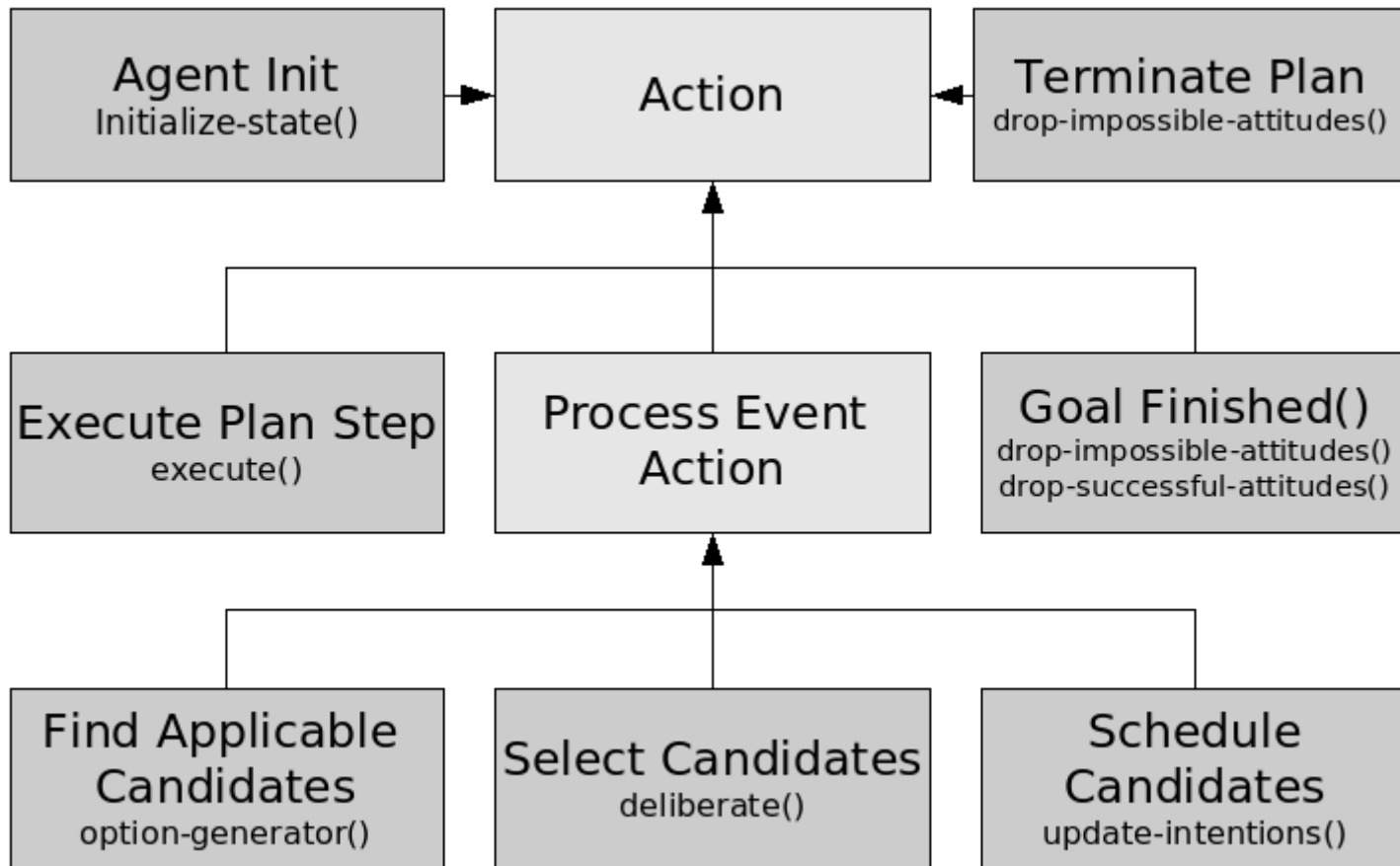
- kleine Teilaktionen
- jede Aktion kann bei Bedarf ausgeführt werden
- jede Meta-Aktion bezieht sich auf einen entsprechenden Teilaspekt
- bestehende Operationen des BDI-Zyklus bilden die Grundlage der neuen Architektur

BDI-Interpreter



BDI-Interpreter

Die neuen Meta-Aktionen:



BDI-Interpreter

Meta-Aktionen, die das BDI-System erweitern:

- UpdateBelief (automatisches Update eines Wertes, z.B. Sensor)
- CreateGoal, SwitchContext, DropGoal
- DeliberateNewOption und DeliberateDeactivatedGoal (Easy Deliberation)

BDI-Interpreter

Implementierung:

- für jede Meta-Aktion gibt es eine Javaklasse
- Vorbedingung durch isValid() Methode
- Zustandsänderungen in einer execute() Methode
- System sammelt alle Zustandsänderungen
- wenn eine Creation-, Context-, Drop-Bedingung nach einer Zustandsänderung wahr geworden ist, wird die zugehörige Aktion in die Agenda eingefügt
- bei internen Nachrichten oder Zeittriggern werden Aktionen automatisch in die Agenda eingefügt

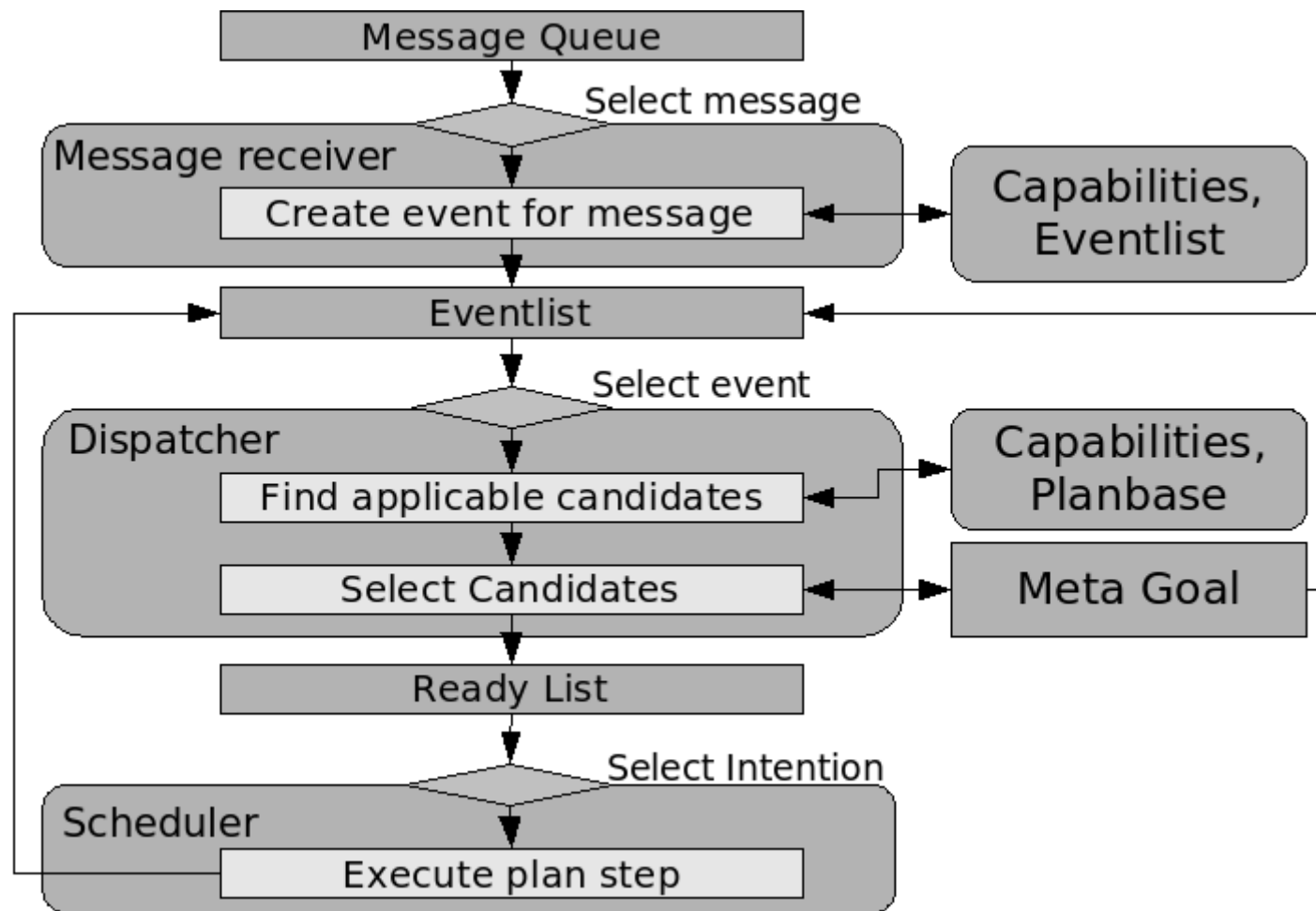
Übersicht

1. Einleitung
2. Die BDI-Komponenten in Jadex
3. Easy Deliberation Strategy
4. Der BDI-Interpreter in Jadex
- 5. Ausführungsmodell**
6. Realisierung der Plattform
7. Sprache
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

Ausführungsmodell

- Auswahl von Plänen zur Behandlung von Nachrichten oder internen Events
- hierfür gibt es drei Komponenten:
 - Messagereceiver
 - Dispatcher
 - Scheduler
- drop-impossible/successful-attitudes() kommen nicht vor, weil sie automatisch bei Bedarf ausgeführt werden

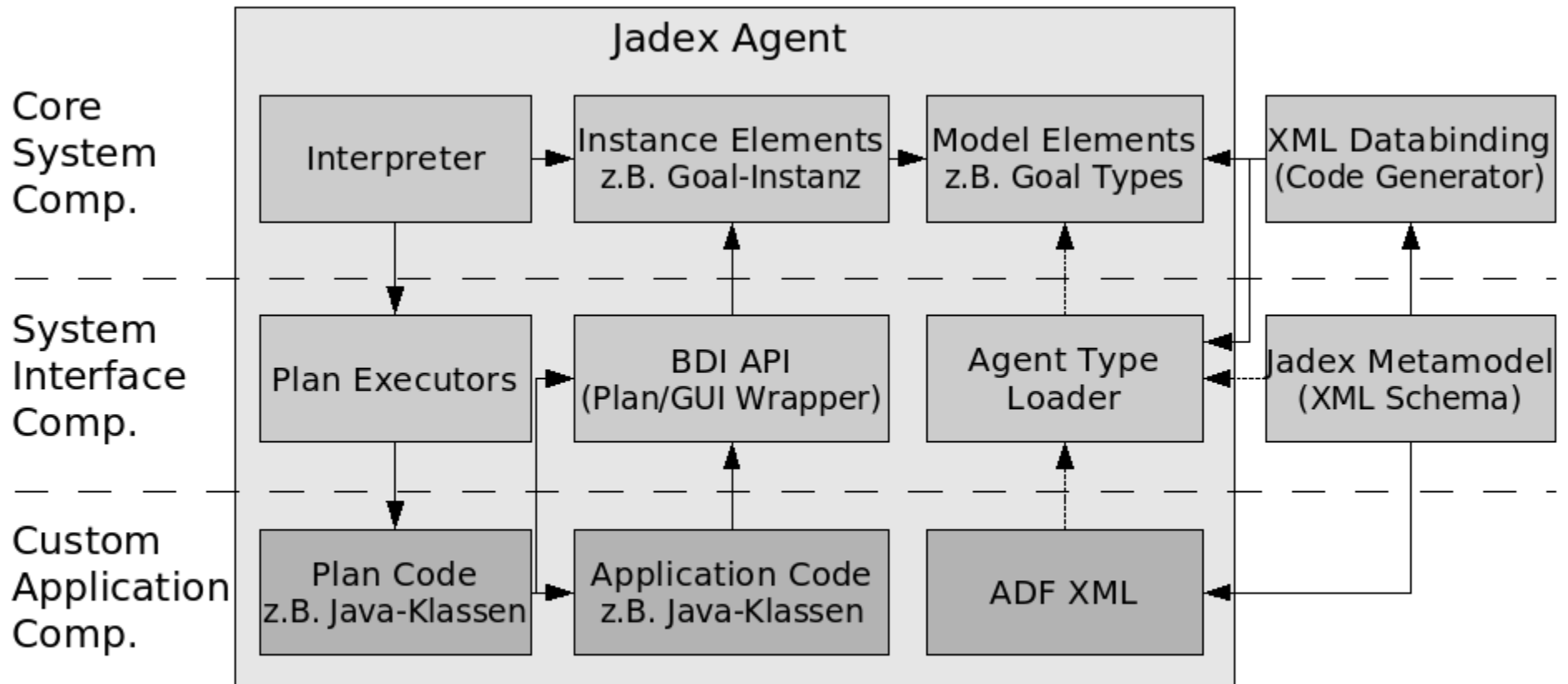
Ausführungsmodell



Übersicht

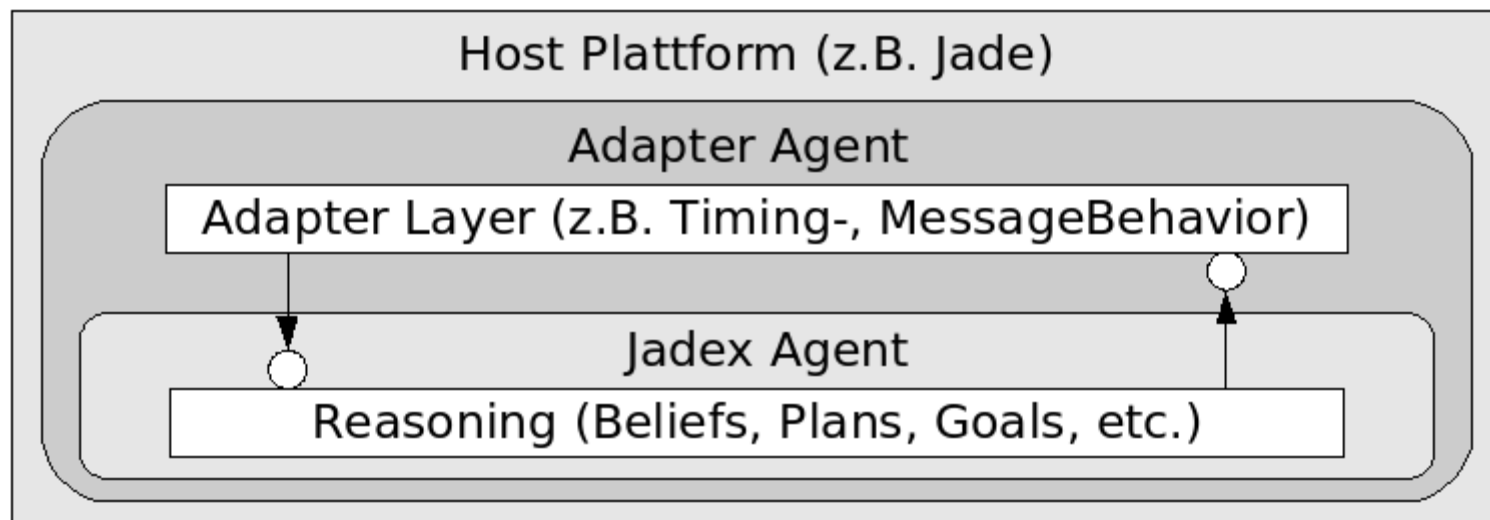
1. Einleitung
2. Die BDI-Komponenten in Jadex
3. Easy Deliberation Strategy
4. Der BDI-Interpreter in Jadex
5. Ausführungsmodell
- 6. Realisierung der Plattform**
7. Sprache
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

Realisierung der Plattform



Realisierung der Plattform

- Reasoning Engine ist eigenständige Komponente
- unterschiedliche Plattformen (z.B. Jade, DIET, J2EE)
- Adapter besteht aus zwei Interfaces:
 - Jadex Agent Interface -> bietet Reasoning
 - Adapter Agent Interface -> Notification & Messages



Übersicht

1. Einleitung
2. Die BDI-Komponenten in Jadex
3. Easy Deliberation Strategy
4. Der BDI-Interpreter in Jadex
5. Ausführungsmodell
6. Realisierung der Plattform
- 7. Sprache**
8. Agentenentwicklung mit Jadex
9. Zusammenfassung, Ausblick, Quellen

Sprache

- Jadex enthält keine eigene neue Sprache
- Typ-Spezifikation (statisch) / Verhalten (dynamisch)
- AgentDefinitionFile (ADF) in XML definiert Beliefs, Goals und Plans sowie Initialwerte
- Erhöhung der Ausdruckskraft durch Expressions
- Programmcode hat durch API Zugriff auf BDI-Einrichtungen

Expressions

- Definition von Objekten (z.B. Initialwerte)
- Beschreibung von Bedingungen oder Queries
- Platzhalter für Werte, die erst zur Laufzeit genau bestimmt werden
- Syntax folgt den Java Expressions + eine Erweiterung, die einen Teil der OQL beinhaltet
- Expressions können überall benutzt werden, sind aber besonders nützlich, um Teilansichten der Beliefbase zu erzeugen

Beispiel OQL Expression

```
01 select_expression ::= „SELECT“ („ALL“ | „ANY“ | „IOTA“)?  
02 (  
03 (expression „FROM“ („$“ identifier „IN“ expression) („“ „$“ identifier „IN“ expression)* )  
04 | („$“ identifier „FROM“ expression)  
05 )  
06 („WHERE“ expression)?  
07 („ORDER“ „BY“ expression („ASC“ | „DESC“)? )?
```

```
SELECT $block FROM $beliefbase.blocks WHERE $block.isClear()
```

- OQL-Query in EBNF:
 - Anfrage in Select-From-Where Struktur
 - IOAT, FROM, WHERE, ORDER BY
 - Java Methodenaufrufe möglich

Elemente der ADF

- Agent: Name, Package, Beschreibung
- neben den Subtags für das BDI-Konzept können noch weitere definiert werden:
 - Languages, Ontologies
 - Servicedescription & Agentdescription (Yellow Page)
- Imports & Expressions
- Properties (z.B. Debugging/Logging)

Beliefs

- Darstellung in einer objektorientierten Weise
- jedes Belief kann Name, Beschreibung und Exportflag enthalten
- Angabe der entsprechenden Javaklasse
- ggf. Initialwert als Expression
- Wert kann statisch oder dynamisch deklariert sein
- dynamische Werte werden beim Zugriff neu berechnet und ggf. zusätzlich in bestimmten Zeitintervallen

Beispiel-ADF: Beliefs

```
...  
<beliefs>  
  <belief name="geburtstag" class="Date">  
    <fact>new Date()</fact>  
  </belief>  
  
  <beliefset name="geburtstage" class="Date">  
    <fact>new Date()</fact>  
    <fact>new Date()</fact>  
    ...  
  </beliefset>  
</beliefs>  
...
```

Goals (ADF)

- vier Goal-Typen mit vielen gleichen Eigenschaften:
 - Creation, Drop & Context-Conditions als Expression
 - Angabe von Parametern bzw. Bindings
 - weitere Flags (z.B. retry, posttoall, ...)
- Performgoal: keine weiteren Angaben
- Achievegoal: Target- und Failure-Bedingung
- Querygoal: Target- und Failure-Bedingung
- Maintaingoal: Maintain- und Target-Bedingung, sowie Retry und Delay im Fehlerfall.
- Initialgoals

Beispiel-ADF: Goals

```
...  
<goals>  
  
  <achievegoal name="emptyList">  
    <parameter name="liste" class="List">  
      <targetcondition>$goal.liste.isEmpty()</targetcondition>  
    </achievegoal>  
  
  ...  
</goals>  
  
...
```

Pläne (ADF)

- Plan-Header
- Angabe von Name, Beschreibung
- Trigger, Pre- & Context-Bedingung (Expression)
- parameterisierbar durch Bindings
- weitere Angaben möglich:
 - ob Plan direkt beim Start instanziiert werden soll
 - Priorität
 - Exported
- Planbody wird als Expression angegeben

Beispiel-ADF: Pläne

```
...  
<plans>  
  
  <plan name="walk">  
    <body>new WalkPlan($beliefbase.standort)</body>  
    <trigger><goal ref="GoAway" /></trigger>  
  </plan>  
  
  ...  
  
</plans>  
  
...
```

Pläne (Klasse)

- extends Plan
- implementiert body() -> Planverhalten
- optional:
 - passed()
 - failed() -> bei unbehandelten Exceptions
 - aborted() -> Plan wurde vorzeitig abgebrochen

Beispiel Planklasse

```
01 public class SomePlan extends jadex.runtime.Plan {
02
03     public void body() {
04         // plan code
05     }
06
07     public void passed() {
08         // optionaler cleanup, wenn Plan erfolgreich
09     }
10     public void failed() {
11         // optionaler cleanup, wenn Plan fehlschlägt
12     }
13     public void aborted() {
14         // optionaler cleanup, wenn Plan abgebrochen wurde
15     }
16 }
```

Übersicht

1. Einleitung
2. Die BDI-Komponenten in Jadex
3. Easy Deliberation Strategy
4. Der BDI-Interpreter in Jadex
5. Ausführungsmodell
6. Realisierung der Plattform
7. Sprache
- 8. Agentenentwicklung mit Jadex**
9. Zusammenfassung, Ausblick, Quellen

Agentenentwicklung

- leichte Programmierung und trotzdem komplexe Agentensysteme möglich
- Übergang von OO -> Agentenprogrammierung
 - Einsatz gängiger Techniken (Java und XML)
 - Entwicklungsumgebungen und Tools wiedernutzen
- wiederverwendbare Programmteile
- umfangreiche Dokumentation, Tutorial, User Guide und Beispielagenten
- Mailinglist, Forum, Bugreports

Agentenentwicklung

- Runtime-Tools im Jadex Control Center:
 - Jadex Starter
 - Introspector
 - Conversation Center
 - BDI Tracer
 - DF Browser (ab 0.95)
- Development-Tools:
 - Beanyner
 - Jadexdoc

Agentenentwicklung

Jadex Starter

- starten und beenden von Agenten
- Überprüfung des Agenten-Modells
- Jadexdoc erzeugen
- Modeltree

Agentenentwicklung

Introspector

- beobachten und verändern des internen Zustands eines Agenten
- Anzeige der Belief-, Goal- und Planbase
- Debugger zeigt die Eventverarbeitung an

Conversation Center

- direktes Versenden und Empfangen von Nachrichten

Agentenentwicklung

BDI Tracer

- zeichnet Zustandsänderungen und Nachrichten eines Agenten auf
- wertet die Aufzeichnungen aus und stellt diese graphisch dar

DF Browser

- Durchsuchen und Anzeige des DF

Agentenentwicklung

Development-Tools:

- Beanyner
 - Plug-In für Protégé
 - erzeugt JavaBeans oder Jade Ontology Dateien aus der modellierten Ontologie
- Jadexdoc Tool
 - Dokumentationstool ähnlich Javadoc
 - erzeugt HTML aus ADFs

Zusammenfassung & Ausblick

- Framework und Laufzeitumgebung für BDI-Agenten
- Erweiterung der Reasoning-Engine und des Interpreters
- Ausblick:
 - Erweiterungen der internen Konzepte
 - Teams
 - Goals auf Interagent-Ebene
 - Entwicklung von weiteren Tools
 - graphische Modellierung eines Agenten

Quellen

- L. Braubach, A. Pokahr, W. Lamersdorf. Jadex: A BDI Agent System Combining Middleware and Reasoning. In M. Klusch, R. Unland, M. Calisti, editors, *Software Agent-Based Applications, Platforms and Development Kits*. Birkhäuser, 2005.
- A. Pokahr, L. Braubach, W. Lamersdorf. Jadex: A BDI Reasoning Engine. In J. Dix, R. Bordini, M. Dastani, A. Seghrouchni, editors, *Multi-Agent Programming*. Kluwer, 2005.
- A. Pokahr, L. Braubach, W. Lamersdorf. A Goal Deliberation Strategy for BDI Agent Systems. In *Proc. of the third German Multi-Agent conference (MATES 2005)*, 2005.
- A. Pokahr, L. Braubach, W. Lamersdorf. A Flexible BDI Architecture Supporting Extensibility. In *The 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, 2005.
- M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts, 1987.
- A. Pokahr, L. Braubach, A. Walczak. *Jadex User Guide, Release 0.941*, 2005.
- A. Pokahr, L. Braubach, A. Walczak. *Jadex Tutorial, Release 0.941*, 2005.
- A. Pokahr, L. Braubach, R. Leppin, A. Walczak. *Jadex Tool Guide, Release 0.941*, 2005.