Roadmap – Course Overview

- Introduction
- Cooperation Models and Mechanisms
- Communication
- Micro-Level Architectures
- Macro-Level Structures

- Development Methods
Roadmap – Chapter Details

Development Methods

- Introductory Remarks
- GAIA
- MASSIVE
(Software) systems development phases (note: linear = idealized!):

The methods we will see in this part had been designed for developing agent-oriented (software) systems, with agents essentially being **cooperative** entities
Development Methods

- Introductory Remarks
- GAIA
  - Overview
  - Role Model
  - Interaction Model
  - Agent Model
  - Services Model
  - Acquaintance Model
- MASSIVE
Overview

Development Methods • GAIA

- **GAIA** = “Generic Architecture for Information Availability”
- **GAIA** distinguishes **two levels of abstraction**
  - macro level: social and communicative dependencies
  - micro level: internal details of cooperating entities (agents)
- **GAIA** covers only **analysis and design**
- **GAIA** is an **iterative refinement** method
- **GAIA** aims at transforming agent-specific abstractions into a lower abstraction level at which traditional (e.g., object-oriented) techniques can be applied
GAIA’s analysis and design models

Systemanforderungen

Rollenmodell

Interaktionsmodell

Agentenmodell

Servicemodell

Beziehungsmodell

ANALYSE

ENTWURF
GAIA at a glance

**Overview (Cont’d)**

**Development Methods • GAIA**

**GAIA at a glance**

- Systemanforderungen
- Identifikation der Rollen
- Liste der Schlüsselrollen
- Identifikation der Protokolle
- Interaktionsmodell (Protokolle)
- Ausarbeitung der Rollen
- Rollenmodell

**Analyse**

**Entwurf**

- Agentenmodell
  - Agententypen
  - Instanzen
- Servicemodell
- Beziehungsmodell
see “Macro-level structures” → “Computational Roles and Groups” → “GAIA role schema”

Components: role name; name of relevant protocols and activities; permissions; liveness and safety responsibilities

Safety responsibilities: formulated as predicates

Liveness responsibilities: formulated as expressions, based on the following operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.y</td>
<td>x followed by y</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>x*</td>
<td>x occurs 0 or more times</td>
</tr>
<tr>
<td>x+</td>
<td>x occurs 1 or more times</td>
</tr>
<tr>
<td>ω</td>
<td>x occurs infinitely often</td>
</tr>
<tr>
<td>[x]</td>
<td>x is optional</td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Interaction Model

Purpose: definition of **interactions** – dependencies and relationships – that occur between roles (role owner)

Basic components of the model: **protocols**, each consisting of the following **attributes**:

- purpose (brief textual description)
- initiator (the role(s) initiating the interaction)
- responder (the role(s) with which the initiator interacts)
- inputs (information used by the initiator)
- outputs (information supplied by/to the responder)
- processing (brief textual description of any processing performed by the initiator during interaction)
The agent model defines the **agent types** that will be used in the system under development, and the **number of agent instances** that will realize these types at run-time.

**Example:**

- **ServerAgent**
  - **Server**: 1
- **UserAgent**
  - **User**: *
- **WrapperAgent**
  - **Wrapper**: *
- **SearcherAgent**
  - **Searcher**: +
This model identifies the services (and their main properties) associated with the roles

- service = coherent block of activity
  - roughly corresponds to a method in object orientation
  - every activity identified during analysis corresponds to a service (not necessarily reversely)

- service is described by the following attributes:
  - inputs (resources needed for service execution)
  - outputs (resources produced)
  - pre- and post-conditions (represent constraints on service, can be derived from the safety properties)
This model defines the communication links that exist between the agent types, with the purpose to identify potential communication bottlenecks.

- NOT defined: what messages are sent or when they are sent.

Example:

```
| SearcherAgent              | WrapperAgent | ServerAgent | UserAgent |
```

Diagram:

- SearcherAgent -> WrapperAgent -> ServerAgent -> UserAgent
- WrapperAgent erzeugt
Development Methods

- Introductory Remarks
- GAIA
- MASSIVE
  - Overview
  - Product Model (Views)
Overview

MASSIVE = “MultiAgent SystemS Iterative View Engineering”

3 central components

- a **product model**, describing different views of the software system to be developed
- a **process model**, describing the developmental steps to be performed
- an **experience factory**, constituting the organizational framework within which learning across individual software projects occurs
Overview (Cont’d)

MASSIVE at a glance:

Round-Trip Schritte

Iterative Verfeinerungen

Verfeinerung

Konstruktion

Produkt-Modell

Implementierung

Re-Engineering

Fehler

speichern

abrufen

Erfahrungswerkstatt
Environment view

- description of the organizational context: where and how the target system should be used
- characterization of the organizational context: static - dynamic; deterministic - nondeterministic; accessible - inaccessible; etc.
- input and output (the target system’s view of the target system)
Product Model – Views (Cont’d)

▶ Task view

- encompasses functional and nonfunctional requirements
- requires identification of actors and use cases
  (self-contained interaction scenarios)

▶ Role view

- identification of roles
- role = logical grouping of atomic activities according to physical constraints of the operational environment
Interaction view

- intent layer: the purpose of interaction
- protocol layer: the protocols to be used for interaction
- transport layer: mapping of abstract messages at protocol level onto concrete agent framework or operating system

(Agent) Society view

- characterizes organizational structure of target agent system
- open - closed; flat - hierarchical; homogeneous - heterogeneous; static - dynamic
Architectural view

- captures fundamental structural attributes of the target system
- use of architectural design patterns
- clarifies what software entities (agents and objects) are part of the overall system
- identifies external resources (e.g. databases) to be integrated into the target system
System view

- user interface (clarity, comprehensibility, control, etc.)
- exception handling (exception = rare and unusual event, typically an error)
- performance engineering (identification and reduction of performance problems after the code has been designed and developed; e.g., useless computation, re-computation; busy waiting for requests, non-scalable algorithms; etc.)
- deployment (Auslieferung/Installation/Einsatz beim Kunden, generally: all activities performed after the target system has been developed)