Agent-Oriented Software Engineering with UML

December 1, 2004

Bernhard Bauer
Institute of Computer Science
University of Augsburg

bauer@informatik.uni-augsburg.de
Tel. ++49 821 / 598 2118
Tel. ++49 89 / 612 09 026
Outline

Motivation

Approach
- (Agent-)UML and Model-Driven Architecture
- Current State of the Art
- UML 2.0 and AOSE
- Open Issues

Summary
Motivation

Manufacturing has to deal with

- **Technological Factors**
  - Barriers to interoperability / integration
  - Development / maintenance obstacles
  - Evolving / heterogeneous technology suites

- **Business Factors**
  - Defining / meeting business requirements
  - Complex / changing business processes
  - Shifting enterprise / application boundaries
  - Semantic integration with customers / suppliers / partners

**Possible Solutions**
- for implementation: Agent-Technology
- for modeling: Model-Driven Development / Architecture
Motivation

Why Focus on Modeling?

Because Modeling is the only way to ensure that enterprise IT systems deliver the functionality that a business requires, comprehensive and stable, yet able to evolve in a controlled manner as business needs change over time.

Models built in Modeling Languages represent exactly what a business application - even a complex, multi-platform integrated application - can do, and record it with a clarity and stability that far exceeds that of the applications themselves.

Based on technology-independent representations of their business functionality and behavior, modeled applications last for decades and maximize IT return on investment.
Why UML?

- Unified Modelling Language (UML)
  - Wide acceptance for SE in object-oriented software development
  - Engineering tools and software development process
- However agent technology offers more, e.g.
  - Resource coordination
  - Knowledge and service discovery
  - Ontology management and semantic interoperability
  - Adaptivity
  - Planning and scheduling

Thus

- Build on UML and extend UML where necessary
- Odell and Bauer started Agent-UML in 1999
OMG’s Model-Driven Architecture (MDA)

- A “New” Way to Specify and Build Systems
- Based on modeling and UML
- Supports full lifecycle:
  - analysis and design
  - implementation,
  - deployment,
  - maintenance, and
  - evolution
- Builds in Interoperability and Portability
- Applies directly to the mix of hardware and software:
  - Programming language
  - Operating system and network
  - Middleware
Approach (Agent-)UML and MDA

Model Driven Architecture

- **Computational Independent Model (CIM)**, describes the business (logic) and therefore defines business processes and workflows in detail.
- **Platform-Independent Model (PIM)**, represents only business functionality and behavior, undistorted by technology details.
- **Platform-Specific Model (PSM)** developed/generated from the PIM, depending on the underlying platform.
- **Target platform** implementation mappings to multiple middleware platforms.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Model</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PIM</td>
<td></td>
</tr>
<tr>
<td>CIM</td>
<td>PIM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSM</td>
<td>JADE Impl, Java/EJB Impl, XML/SOAP Impl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other Impl</td>
</tr>
</tbody>
</table>

(c) Bernhard Bauer 2004, all rights reserved
MaSE software engineering process

(c) Bernhard Bauer 2004, all rights reserved
Approach
Current State of the Art

Prometheus process overview

Key
- final design artifact
- intermediate design tool
- crosscheck
- derives

Use cases
- actions and percepts
- Functionality descriptors
- agent grouping
- events
- shared data

Protocols
- Interaction diagrams
- agent acquaintance

System Overview
- Agent descriptors

Agent Overview
- Capability overview
- Event descriptors
- Data descriptions
- Plan descriptors
MDA-based Agent Methodology

CIM
- Use Cases
- Business Processes
- Ontologies
- Organizations
- Regularities
- Security issues
- Trading aspects

Transformation

PIM
- Resource Planing
- Decision making
- Ontology / service mgmt
- Agent classes, behavior
- Agent interaction, protocols
- Architecture overview
- Goals, plans, tasks

Transformation

PSM
- Deployment
- Network
- Data persistence
- Integrating ext. resources
- ...

(c) Bernhard Bauer 2004, all rights reserved
UML 2.0 Specification consists of

- **Infrastructure**
  - defines foundational language constructs required for UML 2.0.
  - achieved results
    - adjustment between UML, MOF and XMI
    - re-structuring of language definition (meta-model as well as notation) with the goal to increase understandability and extensibility
    - first class extensibility mechanisms

- **Superstructure**
  - defines user level constructs (diagrams) required for UML 2.0.
  - achieved results
    - Modeling of pattern, e.g. component based development, specification of run-time architectures
    - support for scalability and encapsulation
    - unique definition of semantics for relations, like generalization, dependencies and associations
Approach
UML 2.0 and AOSE

UML 2.0 Specification consists of

- **OCL (Object Constraint Language)**
  - a formal language used to describe expressions on UML models
  - achieved results
    - meta-model-based definition of OCL
    - increased expressability of OCL
    - formal semantics of OCL

- **Diagram Interchange**
  - enable a smooth and seamless exchange of documents
    compliant to the UML standard between different software tools
**UML Diagrams**

- **Foundation: Structural Diagrams** – static structure
  - Class Diagram
  - Object Diagram
  - Composite Structure Diagram

- **Behavior: Behavioral Diagrams** – dynamic behavior
  - Use Case Diagram
  - Sequence Diagram
  - Interaction Overview Diagram
  - Communication Diagram

- **Model Management Diagrams** – organization
  - Packages
  - Subsystems
  - Models
  - Component Diagram
  - Deployment Diagram
  - State Machine Diagram
  - Activity Diagram
  - Timing Diagram
UML 2.0 and AOSE

Class Diagram

Policy Coverage Type

Policy

Insurance Line

Group Policy

Individual Policy

Life Policy

Health Policy

Property/Casualty Policy

association

generalization

properties of generalization set

issue policies

issue policies

coverage type

1

(*)

{disjoint, complete}

:Policy Coverage Type

:Insurance Line

(c) Bernhard Bauer 2004, all rights reserved
Composite Structure Diagram

<<agent>>
ABC Ltd.

<<agent role>>
Manager

<<agent role>>
ABC Buyer

<<agent role>>
Broker
UML 2.0 and AOSE

Composite Structure Diagram - Ports

- Provided interface
- Required interface
- Public port
- Multiplicity
- Port name
- Port type
Composite Structure Diagram - Pattern

Sale

broker
buyer
seller
producer
consumer

wholesale: Sale

retail: Sale

BrokeredSale
Activity Diagrams

Process Order

Requested Order

Receive Order

Fill Order

Ship Order

Close Order

{plan rule = (When orderPaymentType = invoice)}

[order rejected]

[order accepted]

Send Invoice

Make Payment

Accept Payment

Invoice

(c) Bernhard Bauer 2004, all rights reserved
Sequence Diagrams

FIPA-ContractNet-Protocol

Initiator

Participant

cfp

n

alt

refuse

j

propose

k

alt

k-a

reject-proposal

k-r

accept-proposal

alt

cancel

inform

failure

(c) Bernhard Bauer 2004, all rights reserved
Interaction Overview Diagrams
UML 2.0 has no “off-the-shelf” constructs to express:

- goals,
- agents,
- groups,
- multicasting,
- generative functions, such as cloning, birthing, reproduction, parasitism and symbiosis,
- emergent phenomena, and
- many other nature-based constructs from a methodological point of view.
Organic Computing Aspects:

- **Modeling techniques for the self-x-properties, i.e.**
  - defining appropriate abstractions and models for understanding, controlling, and designing emergent behavior based on self-organization, i.e. defining the properties of
    - self-configuration,
    - self-optimization,
    - self-healing & self-protecting,
    - self-describing,
    - self-maintenance, and
    - context-awareness (adaptivity, context-sensitivity)

- **Modeling techniques for the life cycle of an autonomic element,**
  - modeling the installation,
  - configuration, and optimization,
  - upgrading,
  - monitoring,
  - problem determination, and
  - recovery
Summary

AOSE
- A lot of work has been done the last years
- UML 2.0 supports several diagrams useful for AOSE
- MDA-based approach seems appropriate
- Model transformations have to be elaborated

In the context of manufacturing
- Take standards into account
- Have a look at organic/autonomic computing
- Develop methodology
  - independent of application domain
  - Customizable to specific needs
  - Generate code when HW changes
  - Embedded agents vs. SCM agents
Thank you…

… for your interest.

I‘ll be happy to answer your questions!