Engineering Artifacts for Multi-Agent Systems

(Version 2)

Van Parunak, ERIM CEC (vparunak@erim.org)
Jim Odell, James Odell Associates
(jodell@compuserve.com)

Van Parunak

- Scientific Fellow, ERIM.
- Senior technical direction of marketing initiatives and projects
- Developed and executed a strategy for coherent identification, development, and marketing of ERIM tools
- Responsible for marketing to the research community.
- www.erim.org/~van/
James Odell

- Industrial Consultant in OO, Agents, and Complex systems
- Based in Ann Arbor, MI
- Leadership role in OMG:
  - Co-Chair of Object Analysis and Design Task Force (source of UML)
  - Co-Chair of Agents Working Group
- [www.jamesodell.com](http://www.jamesodell.com)

Overview

- Why Engineering Artifacts?
- OO Artifacts for MAS Development
- Toward Agent-Specific Artifacts
Industry vs. Research

<table>
<thead>
<tr>
<th>Academy</th>
<th>Industry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>100’s of person-years</td>
</tr>
<tr>
<td>One person, 1-3 years</td>
<td></td>
</tr>
<tr>
<td>(Ph.D. project)</td>
<td></td>
</tr>
<tr>
<td>Skills</td>
<td>General SWE</td>
</tr>
<tr>
<td>Latest techniques</td>
<td>Reward for control</td>
</tr>
<tr>
<td>Reward for innovation</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>Satisfy specifications</td>
</tr>
<tr>
<td>Novelty; peer review</td>
<td></td>
</tr>
</tbody>
</table>

Scope: Life Cycle Project Management

- Each stage may involve different people with different skills, perspectives, and objectives.
- Artifacts provide clean interfaces between stages.
Skills:
Artifacts capture “best practice”

- What inputs from previous stages do I need to consider?
- What decisions do I need to make?
- How do these decisions depend on one another?

Success

Intermediate Deliverables
- Problem: How can I track progress through early stages of the life cycle?
- Answer: Define deliverables in terms of artifacts.

Requirements Traceability
- Problem: How can I ensure that requirements are being addressed in the early stages?
- Answer: Map requirements to details of artifacts.
Overview

• Why Engineering Artifacts?
• OO Artifacts for MAS Development
  – Agent = Object++
  – So start with proven OO methods
• Toward Agent-Specific Artifacts

Spreading Localization

<table>
<thead>
<tr>
<th>Machine Language</th>
<th>Structured Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent-Oriented Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>How does a unit behave? (Code)</td>
<td>Diffuse</td>
<td>Focused</td>
<td>Focused</td>
</tr>
<tr>
<td>What does a unit do when it runs? (State)</td>
<td>External</td>
<td>External</td>
<td>Internal</td>
</tr>
<tr>
<td>When does a unit run?</td>
<td>External</td>
<td>External (called)</td>
<td>External (message)</td>
</tr>
</tbody>
</table>

Agents are:
• Objects with Initiative
• Objects with Attitude
• Pro-Active Objects
• Objects++
• Objects on Steroids
• Objects that can say “No” (or “Go”)
Agent Accessories

- Travels
- Represents Human
- Dissipative
- Mentalistic State
- Game-Theoretic
- Social
- Negotiates

(Also: Heterogeneous, Voting, Embodied...)

Two Senses of “Autonomy”

Dynamic Autonomy
- Object that can say “Go”
- Reactive vs. Pro-active
- Determined by agent’s internal structure

(non-)Deterministic Autonomy
- Object that can say “No”
- Predictable vs. Unpredictable
- Determined by external observer
Two Senses of “Autonomy” (2)

- **Proactive**
  - Dynamic: Can say “Go”
  - Ant Colony
  - GM Paint Booth

- **Reactive**
  - Passive: Predictable
    - Ant
  - Unexpected

- **(non-)Deterministic:**
  - Can say “No”

---

Key OO Concepts in the Context of Agents

<table>
<thead>
<tr>
<th>Objects</th>
<th>Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Species, Roles</td>
</tr>
<tr>
<td>Message</td>
<td>Assertion, Query, Command, State change, Event</td>
</tr>
<tr>
<td>Attribute</td>
<td>State, Beliefs, Goals, Second-Order Knowledge</td>
</tr>
<tr>
<td>Method</td>
<td>Plans, Capabilities, Rules, Services, Responsibilities</td>
</tr>
</tbody>
</table>
**OO Concepts: Class**

- Class defined by static identity
- Roles defined by patterns of interaction
- Example: a company agent in a supply chain may assume different roles as
  - Purchaser of supplies
  - Vendor of finished products
  - Employer of workers
  - Football for government regulatory agencies

<table>
<thead>
<tr>
<th>Class</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>One per object</td>
<td>Many per agent</td>
</tr>
<tr>
<td>Immutable</td>
<td>Mutable</td>
</tr>
<tr>
<td>Focus on Implementation</td>
<td>Focus on Function</td>
</tr>
</tbody>
</table>

**OO Concepts: Message**

- OO: <Object, method, parameters>
- Agent:
  - Any change in the environment that the agent is equipped to detect.
  - At the heart of *Dynamic Autonomy*
OO Concepts: Attributes

- **OO:**
  - Internal state variables.
  - “Variable” ↔ The mutable part of an object’s code.
- **Agents:** Much more is “mutable”
  - Beliefs about the agent and the external world
    » Can include beliefs about agent capabilities (“methods”)
  - Desires about later states
  - Intentions (plans) for getting from here to there
  - At the heart of Deterministic Autonomy

<table>
<thead>
<tr>
<th>OO:</th>
<th>Agents:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Beliefs, Desires</td>
</tr>
<tr>
<td>Methods</td>
<td>Intentions</td>
</tr>
</tbody>
</table>

Subroutine < Method < Service < Responsibility

- **Method** = Subroutine + Polymorphism
- **Service** = Method + Discretion (Deterministic Autonomy; “Can say No”)
- **Responsibility** includes
  - Services (initiated by requests other agents)
  - Responses to environmental changes
  - Self-initiated activity (Dynamic Autonomy; “Can say Go”)
Agent Capabilities

<table>
<thead>
<tr>
<th>Internal</th>
<th>What I know how to do</th>
<th>Psychology</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>Laws of Nature</td>
<td>Physics</td>
</tr>
<tr>
<td>Both</td>
<td>Resources at my disposal</td>
<td>Economics</td>
</tr>
</tbody>
</table>

Implemented with
• Subroutines
• Rules
• Constraints
• ...

Unified Modeling Language:
A Proven Language for OO Artifacts

Integration and rationalization of methods promoted by a number of OO gurus
• Booch
• Rumbaugh
• Jacobson
• Odell
• ...

Developed through OMG process
NB: a good example of a standard following technology rather than preceding it.
Under UML’s Hood

Object Constraint Language (FOL)

Static (Class Diagrams, Packages, Components)

Dynamic*

Graphical Language

UML Metamodel

*Under development; first vendor proposals due 23 Nov 1999.


Object Constraint Language (OCL)

- Language developed at IBM.
- Side-effect free, constraint language used to specify well-formedness rules (static semantics).
- The rules specify constraints over attributes and associations defined in the metamodel.
- Basic types - Boolean, Integer, String, Set, Sequence, ...
- Operations on the basic types:
  - and, or, not, implies, ...
  - +, -, >, =, ...
  - aSet.union (anotherSet)
  - aSet.forAll ( e | Pred (e) )
  - aSet.select ( e | Pred (e) )
  - aSequence.at (index)
- User-defined types: the metaclasses
- Properties: attributes, association roles, and operations
  - anInstance.property
- Operation
  - anOp ( p : ParType ) : RetType;
  - anOp ( p ) = ... function of p ...

OCL Examples

**Association**

[1] The `AssociationRoles` must have a unique name within the `Association`.

\[
\text{self.allRoles.getAll(} \ r1, \ r2 | \ r1.name = r2.name \ \implies \ r1 = r2 \ \text{)} \quad \text{(e.g.,}
\text{can't be reflexive: "spouse" can't be both association role and opposite association role for same association)}
\]

[2] At most one role in a given association may be an aggregation.

\[
\text{self.allRoles.select(} \text{aggregation } \neq \text{none).size } \leq 1
\]

**Additional Operations**


\[
\text{allRoles : Set(AssociationRole);}
\]

\[
\text{allRoles = self.role}
\]

---


---

UML Resources

- **Metamodel and Object Constraint Language**

- **Static Diagrams**
  - Class diagrams
  - Packages
  - Component diagrams

- **Dynamic Diagrams**
  - Statechart
  - Collaboration diagrams
  - Sequence diagrams
  - Activity diagrams
  - Deployment diagrams
  - Use Case diagrams
CLASS DIAGRAMS
Entity-attribute-relationship approach (ERA)

- ERA was introduced to the DB community and refined by Bachman, Chen, and others.
- Relationships are usually binary (dyadic); some variants allow n-ary.
- Some variants allow relationships to have attributes.

CLASS DIAGRAMS
Binary relationship approach (BRA)

- Roots in AI and linguistics; introduced to DP community by Abrial, Bracchi, Nijssen, and Senko.
- Does not distinguish relationships from attributes.
- Relationships not restricted to binary (dyadic).
- Relationships can be component types.
Order Processing Types (Agents and/or Objects) using a Class Diagram

- All classes in this example are <<type>> (i.e., conceptual rather than implementational)
- Possible heuristic for identifying agents

PACKAGES

For grouping classes (and other packages).

Basic syntax:
COMPONENT DIAGRAMS

- For visualizing physical modules of code
- Shows dependencies between components and their permitted operations
- Corresponds to package diagrams
- Used with deployment diagrams

Basic syntax:

```
component name  operation
```

```
component name  contained class  contained component  operation
```
COMPONENT DIAGRAMS

Example

Scheduler
  reservations

Planner
  update

GUI

Dictionary
  spell-check synonyms

Mailer
  Mailbox
  RoutingList

UML Diagrams

- Object Constraint Language
- Static Diagrams
  - Class diagrams
  - Packages
  - Component diagrams
- Dynamic Diagrams
  - Statechart
  - Collaboration diagrams
  - Sequence diagrams
  - Activity diagrams
  - Deployment diagrams
  - Use Case diagrams
STATECHARTS

- Expresses state machine behavior
- For visualization of an object life cycle
- Can be used in analysis or design
- Often used for selected types or classes, i.e., those whose objects exhibit interesting behaviors

Basic syntax:

```
Superstate Name

State Name
  event, (parameters),
  [guard condition],
  operation

Transition

State Name
  event, (parameters),
  [guard condition],
  operation
```

A State-Based Speech-Act Version of the Winograd-Flores Protocol

After Smith and Cohen 1995
A State-Based Speech-Act Version of the Winograd-Flores Protocol in UML Statechart Notation

Proposed → Requested → Committed → Shipped

- A: REQUEST
- B: PROPOSE
- B: REFUSE

Closed
- A: PAY
- B: SHIP
- B: RENEGE

Aborted → Reneged → Rejected

A: ASSERT Bad
B: COMMIT
A: ASSERT Out

A: PAY
B: ASSERT Out

COLLABORATION DIAGRAM

- Overall interactions among objects
- Primarily useful for design
- Can provide a higher level visualization between state-transition diagrams
- When to use collaboration diagrams:
  - for visualizing process from a class diagram perspective
  - for expressing simple behavior; concurrency, ambiguous sequences, and conditional behavior require additional documentation

Basic syntax:

:class

n: operation name

(object list)

attribute
attribute=value

Synchronous and flat flow of control (typical of objects)

Asynchronous and flat flow of control (typical of agents)
An Agent Conversation

<table>
<thead>
<tr>
<th>Seq</th>
<th>Snr</th>
<th>Rcvr</th>
<th>Utterance</th>
<th>Rspnds to</th>
<th>Replies to</th>
<th>Resolves</th>
<th>Completes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B.C.D</td>
<td>REQUEST: Please send me 50 widgets at your catalog price by next Thursday.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>C</td>
<td>QUESTION: Are you bidding on A’s RFQ?</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>B</td>
<td>INFORM: Yes, I am.</td>
<td>2 2 2 2 2</td>
<td>2 2 2 2 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>A</td>
<td>REFUSE</td>
<td>3 1 1 1 1</td>
<td>3 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>A</td>
<td>PROPOSE (INFORM + REQUEST): How about 40 widgets at catalog price by next Friday?</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>C</td>
<td>REQUEST: Please send me 40 widgets at catalog price by next Friday.</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>A</td>
<td>COMMIT: I plan to send you 40 widgets at catalog price by next Friday.</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>D</td>
<td>A</td>
<td>COMMIT: I plan to send you 50 widgets at catalog price by next Thursday.</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>C</td>
<td>ASSERT: I’ve found a better supplier, and am not relying on your COMMIT.</td>
<td>7 8 7 7 7</td>
<td>7 8 7 7 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C</td>
<td>A</td>
<td>REFUSE: I’m abandoning my COMMIT.</td>
<td>9 9 9 9 9</td>
<td>9 9 9 9 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>D</td>
<td>A</td>
<td>SHIP: Here are your widgets. Please pay me.</td>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>A</td>
<td>D</td>
<td>ASSERT + REQUEST: You’re five short. Please send the difference.</td>
<td>11 11 11 11 11</td>
<td>11 11 11 11 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>A</td>
<td>SHIP: Here are five more widgets. Please pay me.</td>
<td>12 12 12 12 12</td>
<td>12 12 12 12 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A</td>
<td>D</td>
<td>PAY</td>
<td>13 13 13 13 13</td>
<td>13 13 13 13 13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Agent Conversation using a Collaboration Diagram

1.1: request  
2. question  
3. inform  
4. refuse  
5. propose  
6. request  
7. commit  
8. commit  
9. assert  
10. refuse  
11. ship  
12. assert + request  
13. ship  
14. pay
**SEQUENCE DIAGRAM**

- Explicit sequence of operations (messages)
- Particularly useful for real-time systems
- Sequence is basically vertical
- Objects and/or classes arranged horizontally
- Useful for visualizing process as an interacting sequence of classes/objects

**Basic syntax:**

```
object:operation
: class
operation name

[constraint-1]
[constraint-2]
```

---

**Sealed Bid Auction using a Sequence Diagram**

```
auction sell agent: request Action Sale
auctioneer agent: request Action Process
buy agent 1: subscribe
buy agent 2: call For Bid
buy agent 3: bid
admin agent: inform of Auction

item:
<<create>>
accept Auction Item

inform of Auction Results
propose
reject Proposal
```
Agent Conversation: Sequence Diagram

A: request → B: question → C: inform → A: refuse
B: propose → B: request → C: commit → D: commit
C: assert → C: refuse
D: ship → B: assert + request → B: ship → B: pay

ACTIVITY DIAGRAMS

- Process flow (or workflow) visualization
- Can be used in analysis or design
- Expresses the method of an operation

When to use activity diagrams:
- for visualizing process as a workflow
- for expressing complex behavior, such as concurrency, complex synchronization, and conditional behavior

Cf. Petri nets; Grafcet; CEC's XSpec; Ferber's BRIC
Order Processing:
Activity Diagrams

Place Order → Process Order → Accept Order → Accept Quote → Create Quote → Match Order and Quote → Close Order → Update Quote → Settle Order

Order Processing:
Activity Diagrams with Swimlanes

Customer
Place Order → Process Order → Accept Order → Match Order and Quote → Close Order → Settle Order

Broker
Process Order → Accept Quote

ECN
Accept Order → Match Order and Quote

Market Maker
Create Quote → Update Quote
DEPLOYMENT DIAGRAMS

- Depicts the physical relationship of software and hardware components
- For visualizing runtime configurations
- Shows communication between system nodes
- Nodes are instances
- Can be used with component diagrams

Basic syntax:

```
node name -- link -- node name
```
DEPLOYMENT DIAGRAMS
Example

Example:

DEPLOYMENT DIAGRAMS WITH COMPONENTS
USE CASE DIAGRAMS
for context specification

Expresses the relationship between a given system and its environment.


Both are UML compliant.

A context diagram expressing the operations that will provide the interface between the system and its environment. UML refers to these operations as use cases (discussed in the next section).

USE CASE DIAGRAMS

Shows the relationships among actors and use cases within a system.
includes relation – A “base” use case incorporates the behavior of another use case, i.e., mandatory.

- extends relation – A “base” use case adds to the behavior specified in another use case, i.e., optional.

- generalization – A “base” use case that has more specialized use cases.

```
<table>
<thead>
<tr>
<th>Schedule Job</th>
<th>Schedule Priority Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;include&gt;&gt;</td>
<td>&lt;&lt;include&gt;&gt;</td>
</tr>
<tr>
<td>Review Work Order</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;include&gt;&gt;</td>
<td>&lt;&lt;include&gt;&gt;</td>
</tr>
<tr>
<td>Select Technician</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;include&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Confirm with Customer</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;extend&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Approve Overtime</td>
<td></td>
</tr>
</tbody>
</table>
```

"Uses" has been replaced by include and generalization.

Actors are roles played when interacting with the use case. They may be human, hardware, or software.

- The symbol can be confusing: an actor does not have to be a person.
- It represents the external interfaces to the system.
- Identifying the actor helps the team in understanding the stakeholder of a use case.
- Since actors are types, they may be subtyped.
Overview

- Why Engineering Artifacts?
- OO Artifacts for MAS Development
- Toward Agent-Specific Artifacts

Some Deficiencies of UML Diagrams for Agents

UML has no “off-the-shelf” constructs for:

- mobility
- role changes
- generative functions, such as cloning, birthing, reproduction
- parasitism and symbiosis
- agents on “cell membranes” versus within membranes
- emergent phenomena
- ...
**UML Extension Mechanisms**

- Labels in ```<<...>>``` ("stereotypes")
- Extensions may become normative in later versions of the language.
- These are only suggestions; help us develop the right notations.
  - What do we need to represent?
  - What are the most helpful ways to represent them?

**DEPLOYMENT DIAGRAMS**

**Extending UML for Mobile Agents**

*Example:*

- **Shopper Agent**
  - ```<<at-home>>```
- **Salesperson's laptop**
- **Internet**

*Innovations:*

- **Class (not just component)** in deployment box
- **```<<at-home>>```** to indicate origin
- **```<<mobile>>```** link to show possible movement
Application-oriented view

- How to define agents?
  - Class diagrams
  - Activity diagrams via swimlanes
  - Sequence diagrams defined first at the agent level, then abstracted to the role level
- How to derive roles?
  - Dooley analysis from collaboration diagram
  - Time analysis from sequence diagram

Multi-Role Agents

1. Defining Roles in the UML Metamodel:

   ![Object_Class_diagram]

   - Other roles the object can fill
   - Primary role of the object

2. Detecting Roles:
   - From Dooley analysis
   - From sequence diagrams
3. Representing Roles
An Agent Conversation

<table>
<thead>
<tr>
<th>Seq</th>
<th>Snr</th>
<th>Rcvr</th>
<th>Utterance</th>
<th>Rspnds to</th>
<th>Replies to</th>
<th>Re-solves</th>
<th>Completes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B,C,D</td>
<td>REQUEST: Please send me 50 widgets at your catalog price by next Thursday.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>C</td>
<td>QUESTION: Are you bidding on A’s RFQ?</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>B</td>
<td>INFORM: Yes, I am.</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>A</td>
<td>REFUSE</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>A</td>
<td>PROPOSE (INFORM + REQUEST): How about 40 widgets at catalog price by next Friday?</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>C</td>
<td>REQUEST: Please send me 40 widgets at catalog price by next Friday.</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>A</td>
<td>COMMIT: I plan to send you 40 widgets at catalog price by next Friday.</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>D</td>
<td>A</td>
<td>COMMIT: I plan to send you 50 widgets at catalog price by next Thursday.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>C</td>
<td>ASSERT: I've found a better supplier, and am not relying on your COMMIT.</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C</td>
<td>A</td>
<td>REFUSE: I'm abandoning my COMMIT.</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>D</td>
<td>A</td>
<td>SHIP: Here are your widgets. Please pay me.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>A</td>
<td>D</td>
<td>ASSERT + REQUEST: You're five short. Please send the difference.</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>A</td>
<td>SHIP: Here are five more widgets. Please pay me.</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A</td>
<td>D</td>
<td>PAY</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Role-Less Agent Conversation using a Collaboration Diagram

1.1: request
1.2: request
1.3: request
1.4: refuse
1.5: propose
1.6: request
1.7: commit
1.8: commit
1.9: assert
2: question
3: inform
4: refuse
5: propose
6: request
7: commit
8: commit
9: assert
10: refuse
11: ship
12: assert + request
13: ship
14: pay
Detecting Roles with Dooley Analysis

Dooley-Based Roles in a Collaboration Diagram
Detecting Roles as Time Intervals in a Sequence Diagram (only A’s roles labeled)

A:
- request
  \[\langle\text{role: customer}\rangle\]
- refuse
  \[\langle\text{role: negotiator}\rangle\]
B:
- request
  \[\langle\text{role: customer}\rangle\]
- inform
  \[\langle\text{role: negotiator}\rangle\]
C:
- propose
  \[\langle\text{role: negotiator}\rangle\]
- refuse
  \[\langle\text{role: customer}\rangle\]
D:
- commit
  \[\langle\text{role: negotiator}\rangle\]
- ship
  \[\langle\text{role: customer}\rangle\]
- assert + request

Representing Roles: Role Stereotypes

\[\langle\text{role: [role-name]}\rangle\]
WHEN \(<\text{events}>\)
UNTIL \(<\text{events}>\)
\[; \text{IMMUTABLE; MANDATORY}\] >>

\[\langle\text{role change (classify): FROM [type(s)] TO [type(s)]}\rangle\]

- MANDATORY: all instances of the agent must have this role.
- IMMUTABLE: once an agent receives this role, it can never lose it.
  For instances, MANDATORY \[\rightarrow\] IMMUTABLE
Role Changes using a Class Diagram

---

Role Changes using an Activity Diagram

Kinds of events: {create, terminate, classify, declassify, reclassify, connect, disconnect, reconnect, coalesce, decoalesce}
Roles on Messages in a Collaboration Diagram (only roles for A shown)

1.3: request <<role: customer>>
12: assert + request <<role: customer>>
14: pay <<role: customer>>

Roles as Parallel Branches in a Sequence Diagram
Roles as Types in a Sequence Diagram

Roles as Types without individual agents
Generative Functions: Cloning

- **Class Diagram**
  - Dr_Frankenstein: Scientist
  - Dolly: Sheep
  - Dolly2: Sheep

- **Sequence Diagram**
  - get specs
  - <<clone>>

Generative Functions: Mitosis and Bisexual

- **Mitosis**
  - A: Amoeba
  - A1: Amoeba
  - A2: Amoeba
  - <<mitosis>>

- **Bisexual**
  - John: Starfish
  - Janet: Starfish
  - Junior: Starfish
  - <<reproduction>>

Innovations:
- stereotypes
- synchronization bars (native to Activity Diagrams) on Sequence and Collaboration Diagrams
Extending Class Diagrams to Express Parasitism and Symbiosis

Dog
0..*
<<host/parasite>>
Flea
0..*
<<symbiosis>>
Tree
1
Epiphyte
0..*

Extending Packages for Agent (vs. Object) Interface

Example: AMIS
telescoping agent packages

Manufacturing Cell
Common Function Agents
Resource Agent
Resource
0..*
1..*
Negotiator Agent

Innovations:
- Agents can be either on or within cell "membranes"
- Replace interface methods with interface agents (and their responsibilities)
Expressing Emergence by Extending Aggregation in Class Diagrams

Role Changes

- Concurrent vs. sequential roles
  - Concurrent on sequence type
  - Sequential on dashed arrow in collaboration diagram
- Gets into branching temporal logics; multiple world semantics;
- Need either
  - extension to state or activity diagram
  - new “role diagram”
  - class diagram with dashed line and name of process that causes the change
- Need examples and list of issues.
Generative and Collaborative

- Currently, use sequence diagram with arrow going to a box. But not joint effort. Could add synch bar from activity diagrams.

Overview

- Why Engineering Artifacts?
- OO Artifacts for MAS Development
- Toward Agent-Specific Artifacts
  - What do we need to represent?
  - What are the most helpful ways to represent them?
  - vparunak@erim.org