UML 2 Activity Modeling for Domain Experts

Conrad Bock
NIST
conrad.bock@nist.gov
Overview

- UML for knowledge capture.
- Input to UML 2 Activities.
- UML 2 Activity elements.
- Systems engineering extensions.
UML For Domain Experts

- UML began as a language for domain experts to record their knowledge.
- Experts in electric motors design expressed concepts in diagrams, …
- … which automatically generated database table definitions.
- 3000% productivity improvement over informal textual descriptions.

Computation-Independent Modeling = Analysis …

… as in “Analysis and Design Task Force”.

Early stages of software development capture end-user concepts (“analysis”) …

… before later stages choose how these are represented in software (“design”).

Many diagrams shared between analysis and design (class, composition, behavior).
UML For System Engineering

- SE specifications are agnostic about how they are implemented, in organizations, hardware, or software.
- Capture domain expert requirements, rather than how they will be satisfied.
- Executable models for over 15 years.
- UML is increasingly the major modeling language used in SE and military architecture communities.
- UML Profile for SE submitted.

UML For Ontology

- Ontology languages becoming popular for expressing domain expert concepts.
- Enable automated consistency checking.
- UML has significant overlap with OL’s …
  
  ... eg, classes, properties, subclasses, subproperties, disjointness, and others.
- Many aspects of OL’s are specialized kinds of constraints.
- UML Profile for RDFS and OWL submitted.

UML for Business Modeling

- **Foundation concepts for BM**
  - Things ("classes", "objects", "entities")
    - For documents, people, resources, etc.
  - Structured/assembled things
    - For organizations, structured entities.
  - Dynamics
    - For processes, collaboration, event monitoring.

- **Continuity with other UML-based knowledge capture, and with IT implementation.**

UML for Process Knowledge

- UML includes three ways to express knowledge about dynamics …
- … each addressing different aspects:
  - Output to input dependencies (Activities)
  - Reaction to events (State Machines)
  - Message-passing (Interactions)
- … but also overlapping:
  - Sequencing, conditionals, concurrency.
- Virtual machines defined for execution.
Integrated Models

Structure
(assembly, organization, interconnections)

Activity / SM
(tasks, process, orchestration)

Interaction
(messages, choreography, collaboration)

Car

1. Serial Function
2. Multi-exit Function
3. Function in Concurrency
4. Function in Multi-exit Construct
5. Function in an Iterate [before third time]
6. Output Function

Data 1
Data 2
Data 3
Data 4
Data 5
Integration with UML 1.5
Action / Procedure Model

- “Action Semantics”.
- Activities fully executable.
- Covers the full range of flow models from flow charts to code.
- More about this later.

More Than Pictures

- Repository provides
  - API’s
  - XML interchange
  - Support for multiple notations

- UML notation stores to repository
  ... and alternate notations can, too.

- Generate systems from repository:
  Notation → Repository → System
Repository-Centered

Amount function Accept_deposit
(a : Account, d : Amount)
{
    Amount nb = a.balance + d;
    a.balance = nb;
    return nb;
}

Presentation

Parsing

Repository

Model compilation

Actual system
Input to UML 2 Activities

- First workflow RFP discussion (HP, FileNet, NIST)
- SAP, Oracle, IBM
- EDOC, BPML, WPDL, BPEL (WSFL, XLANG), ebXML
- CaseWise, Odell and Associates, IntelliCorp.
- And others.
Example from SAP

1. Opening / processing production order
2. Production order is created or changed
3. Preliminary costing for production order
4. Production order is to be released
5. Release of production order
6. Production order is created or changed
7. Purchase Requisition [created]
8. Backorder processing
9. Production order is (partially) released
10. Production order is to be printed
11. Production order is to be printed
12. Production order is to be printed
13. Planned resources (parts, tools and human labor) are to be freed!
14. Execution of production order
15. Not visible (no user interface), if multiple planned orders are converted into production orders
16. Thread is started, if parts missing
17. {Material is produced}
18. {Material is not produced}


It might be irrelevant (depending on the responsible person and not on the IT system) whether the backorder processing succeeds in getting the missing parts: Parts may still be missing when the execution of this production order is started. This execution may last for a week, or even longer. In between the missing parts may arrive or be produced.
“HP EER Workflow Example,” Matheson, D.,
<table>
<thead>
<tr>
<th>Feature</th>
<th>U2P AM</th>
<th>CaseWise</th>
<th>BPML</th>
<th>EDOC</th>
<th>WPDL-XML</th>
<th>WSFL</th>
<th>XLANG</th>
<th>ebXML/ebTWG</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical notation</td>
<td>Y</td>
<td>Y</td>
<td>Not yet</td>
<td>Nonnormative</td>
<td>N</td>
<td>N</td>
<td>Y (UML 1.x activity graphs)</td>
<td></td>
</tr>
<tr>
<td>Metamodel</td>
<td>Y</td>
<td>N (not exportable anyway)</td>
<td>Not yet</td>
<td>Y (not in UML)</td>
<td>N (but spec is written so that it could)</td>
<td>N</td>
<td>Y (Stereotypes of UML 1.x)</td>
<td></td>
</tr>
<tr>
<td>XML - any</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y (XML, presumably)</td>
</tr>
<tr>
<td>Human usable textual notation</td>
<td>N</td>
<td>N</td>
<td>Y (XML)</td>
<td>Not yet</td>
<td>Y (tags not too complex)</td>
<td>Y (tags not too complex)</td>
<td>Y (XML)</td>
<td>N</td>
</tr>
<tr>
<td><strong>General Process Features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message or control/data flow model?</td>
<td>Control/Data</td>
<td>Control/Data</td>
<td>Message</td>
<td>Data</td>
<td>Control (with parameters)</td>
<td>Control/Data (message data only)</td>
<td>Message</td>
<td>Control/Data (UML 1.x activities)</td>
</tr>
<tr>
<td>Data and control on same diagram/model</td>
<td>Y (shown by usage in notation)</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA (shown by solid/dotted in spec)</td>
<td>NA</td>
<td>Y</td>
</tr>
<tr>
<td>Business-specific features</td>
<td>N</td>
<td>Y (eg. location)</td>
<td>N</td>
<td>N</td>
<td>Y (eg. responsible party, manual activity, cost, etc)</td>
<td>N (but relates to WSDL)</td>
<td>N (but extends WSDL)</td>
<td>Y (stereotypes)</td>
</tr>
<tr>
<td>Activity - actor link</td>
<td>N (but has hook)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Activity - artifact link</td>
<td>N (but has hook)</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplified subsets of functionality defined</td>
<td>Y (well-nested, flowchart)</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y (well-nested, acyclic)</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Simulation-specific information</td>
<td>N</td>
<td>Y (a little for branching)</td>
<td>N</td>
<td>N</td>
<td>Y (timing attributes)</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Transaction model</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N (very little)</td>
<td>N (activities are atomic)</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Detailed Process Features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin model</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Objectflow “in the middle” model</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Explicit control constructs</td>
<td>Y</td>
<td>N (guards for conditionals)</td>
<td>Y (uses message consumption for conditionals)</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y (switch only, uses “Qname” for conditionals)</td>
</tr>
<tr>
<td>“else” functionality for conditionals</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Explicit merge construct</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Optional inputs</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Optional outputs</td>
<td>Y (part of asych outputs)</td>
<td>Y (in control only)</td>
<td>Y (alternate output sets)</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Alternative input/output sets</td>
<td>N</td>
<td>Y (in control only)</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Asynchronous inputs/outputs</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Fork/join functionality</td>
<td>Y</td>
<td>Y</td>
<td>Y (flexible join using named spawn)</td>
<td>Y</td>
<td>Y</td>
<td>Y (part of invocation, guards)</td>
<td>Y (part of invocation, guards)</td>
<td>Y (well-nested only)</td>
</tr>
</tbody>
</table>
Exclusive/complex join

<table>
<thead>
<tr>
<th>Feature</th>
<th>U2P AM</th>
<th>CaseWise</th>
<th>BPML</th>
<th>EDOC</th>
<th>WPDL-XML</th>
<th>WSFL</th>
<th>XLANG</th>
<th>ebXML/ebTWG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive join functionality</td>
<td>Y, (late flows ignored)</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y? (spec is ambiguous)</td>
<td>Y (late flows ignored)</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Complex joins</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Error handling

<table>
<thead>
<tr>
<th></th>
<th>U2P AM</th>
<th>CaseWise</th>
<th>BPML</th>
<th>EDOC</th>
<th>WPDL-XML</th>
<th>WSFL</th>
<th>XLANG</th>
<th>ebXML/ebTWG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>N</td>
<td>Y (compensation for completed activity called by an aborted activity)</td>
<td>Y (exception outputs)</td>
<td>N</td>
<td>Y (as part of data)</td>
<td>Y (compensation for completed activity called by an aborted activity)</td>
<td>Y</td>
</tr>
</tbody>
</table>
Asynchronous invocation

<table>
<thead>
<tr>
<th>U2P AM</th>
<th>CaseWise</th>
<th>BPML</th>
<th>EDOC</th>
<th>WPDL-XML</th>
<th>WSFL</th>
<th>XLANG</th>
<th>ebXML/ebTWG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y (for operations, signals, not subactivities)</td>
<td>N</td>
<td>Y (implements synch as two asych)</td>
<td>N</td>
<td>Y (for subactivities only)</td>
<td>N</td>
<td>Y</td>
<td>Y (WSDL operations)</td>
</tr>
</tbody>
</table>
## Process instances

![Process definition](image)

![Process Execution](image)

<table>
<thead>
<tr>
<th>U2P AM</th>
<th>CaseWise</th>
<th>BPML</th>
<th>EDOC</th>
<th>WPDL-XML</th>
<th>WSFL</th>
<th>XLANG</th>
<th>ebXML/ebTWG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y (structural features only, but extensible)</td>
<td>N</td>
<td>N</td>
<td>Y (scenario only)</td>
<td>N (but may through interop standard)</td>
<td>N</td>
<td>Y (identifier only, with mapping to messages)</td>
<td>N</td>
</tr>
</tbody>
</table>
Activity Modeling

- Activity modeling emphasizes the output/input dependencies, sequencing, and conditions for coordinating other behaviors.
- Uses secondary constructs to show which classifiers are responsible for those behaviors.
- Focus is on what tasks need to be done, with what inputs, in what order, rather than who/what performs each task.
Activity Modeling

- Tasks and ordering ...

Diagram:
- Receive Order
- Fill Order
- Ship Order
- Close Order
- Send Invoice
- Invoice
- Make Payment
- Accept Payment

[order accepted]
Activity Modeling

- ... plus resource assignments.

Partition (notation is called a “swimlane”)
Activity Elements

Action (uses of other activities / tasks)

Object Node (queuing outputs and inputs)

Edge/Flow (execution dependencies)

Control Node (routing control and objects)
"Flow” semantics

Activity execution defined in terms of flow of control and objects/data.
Actions and Object Nodes

- Accept inputs, start behaviors, provide outputs.

Action starts when input arrives.

Output provided when action is complete.

Sequencing: control “flows” when action is complete.
Actions and Object Nodes

- Alternate object node notation (pin).

Action provides values to output pin.

Action accepts values arriving at pin.

No pins for control (usually).

Must use this notation if the output type is different than the input type. The underlying repository stores pins.
Tokens can
- stack up in “in/out” boxes
- backup in network
- prevent upstream behaviors from taking new inputs

Applicable to systems with significant resource constraints, such as physical or manual processes.
Queuing

- Tokens can be
  - Stored temporarily
  - Divided between flows

- Tokens cannot
  - Flow in more than one direction, unless copied.
Activity Parameter Nodes

Parameter nodes accept and provide values to/from whatever behavior uses this activity.
Streaming Parameters

- Values accepted and provided while action is executing.
Exception Parameters

- Outputs that are exclusive of others, and aborts the activity.
Parameter Sets

- Parameters accepting input or providing output exclusive of each other (for each execution).
Control Nodes

- Route objects/data
- At beginning and end of activity:

**Initial Node**
- Gets control when containing activity starts. Flows out immediately.

**Activity Final**
- Accepts input, aborts containing activity.

**Flow Final**
- Accepts input, does nothing.
Control Nodes

- Route objects/data
- In middle of activity:

**Decision**
- Flows out in exactly one direction.

**Merge**
- Flows through immediately.
- Does not combine the tokens.

**Fork**
- Copies inflow to multiple multiple outflows.

**Join**
- Flows out when all inflows arrive. Combine tokens when possible.
Interruptible Region

- Flows abort region.

![Diagram showing the flow of orders and payments with INTERRUPTIBLE REGION and INTERRUPTING EDGE highlighted.]
Reentrant Activities

- No token interaction.
- For domains without resource constraint, such as computation.

```plaintext
Amount function update_account
   (a : Account, d : Amount)
{
    Amount nb = a.balance + d;
    a.balance = nb;
    send_notice (a.customer, a, nb);
    return nb;
}
```
Reentrant Activities

update_account

Deposit → Account

Get Balance

Set Balance

Get Customer

Send Notice

Amount
Time Model

- Can be used to state constraints on processes:

   Close Doors → Move Elevator < 2 sec
First-class Behavior Model

- Object-orientation not required to model dynamics ...
- ... but supported when needed.
- Flexibility in using/not using:
  - Behaviors owned by objects.
  - Messages and Polymorphism
- Integrate with OO for:
  - Relating internal execution to exchanges with between partners.
  - Transformation to implementation
First-class Behavior Model

POEmployee

- sortMail()
- deliverMail()

Deliver Mail

- Keys
- Check Out Truck
- Put Mail In Boxes

Deliver Mail

- HowLong : Time
- Abort()

Truck

resource

0..1
1
Full Action Model

- Kinds of actions include:
  - Invoking behaviors/functions.
  - Creating/destroying objects.
  - Getting/setting property values.
  - Structured nodes (conditionals, etc).
  - Exception handling.

- For fully-executable models and simulations.
SE Extensions

- **Control as Data**
  - Enabling and disabling control values.
  - Output from activities to turn other behaviors “on” and “off”.

- **Rate of flow, on edges and streaming parameters.**

- **Reduce buffering**
  - Overwrite values already in buffer
  - Turn off buffering

- **Probability on decisions, parameter sets, competing outflows from object node.**

- **Behavior decomposition.**
Rate and Buffer Reduction

«controlOperator»
Monitor Temperature

Heat Air

{ rate < 1 per 10 minutes }

ControlValue
{ stream }

«controlOperator»
Monitor Temperature

Measure Temperature
{ stream }

Temperature

[ above threshold ]

«ValueSpecificationAction»
enable

ControlValue
{ stream }

[ else ]

«ValueSpecificationAction»
disable

Measure Temperature

Receive Temperature
«continuous»

Temperature
«noBuffer»

CalculateFeels-like Temperature

Temperature

Receive Temperature

{ rate = 1 per second }

Humidity

Receive Humidity
Activity Decomposition

- «activity» Maintain Temperature
  - maintainTemperature [1..1]
  - monitorTemperature [0..1]
    - calculateTraction [0..1]
  - heating [0..1]

- «activity» Monitor Temperature
  - monitorTemperature [1..1]

- «activity» Heat Air
  - maintainTemperature [1..1]

- «class» Temperature
  - temperature [0..*]
  - humidity [0..*]
Validation

- Systems Engineering
  - UML 2 developed completely separately from SE...
  - ... SE execution semantics matched UML 2 activities almost exactly.

- High-throughput data flow applications
  - Concurrent/pipeline hybrid.
  - Optimized concurrent flow rate.
  - Used for coordinating networks and business applications in telecom and financial applications.
More Information

- UML 2 specification:  
  http://doc.omg.org/formal/05-07-04

- UML 2 Activity articles:  
  http://www.conradbock.org/#UML2.0

- SysML submission:  
  http://doc.omg.org/ad/05-11-01