Bubbles of Steel: A Preview of UML 2.0 and MDA

Bran Selic
bselic@rational.com
Part 1: Models, software models, and MDA

- Why and how software models are changing the way we develop software

Part 2: A preview of UML version 2.0

- UML 2.0 = the first major revision of UML
- Important new language features and modeling capabilities
“…bubbles and arrows, as opposed to programs, …never crash”

-- B. Meyer

“UML: The Positive Spin”
American Programmer, 1997
Engineering Models
Before they build the real thing...

...they first build models ...and then learn from them
**Engineering Models**

- **Engineering model:**
  
  *A reduced representation of some system*

- **Purpose:**
  
  *To help us understand a complex problem or solution*
  
  *To communicate ideas about a problem or solution*
Characteristics of Useful Models

- **Abstract**
  - Emphasize important aspects while removing irrelevant ones

- **Understandable**
  - Expressed in a form that is readily understood by observers

- **Accurate**
  - Faithfully represents the modeled system

- **Predictive**
  - Can be used to derive correct conclusions about the modeled system

- **Inexpensive**
  - Much cheaper to construct and study than the modeled system

To be useful, models have to possess all of these characteristics!
How Models are Used

- To detect errors and omissions in designs before committing full resources to full implementation
  - Through (formal) analysis and experimentation
  - Investigate and compare alternative solutions
  - Minimize engineering risk

- To communicate with stakeholders
  - Clients, users, implementers, testers, documenters, etc.

- To drive implementation
A Problem with Models

Semantic Gap due to:
- Idiosyncrasies of actual construction materials
- Construction methods
- Scaling effects
- Skill sets
- Misunderstandings

Can lead to serious errors and discrepancies in the realization
A description of the software which
- Abstracts out irrelevant detail
- Presents the software using higher-level abstractions

case mainState of
  initial: send("I am here");
  end
  Off: case event of
    on: send(oa,5);
    next(On);
    end
    off: next(Off);
    end
end

On: case event of
  off: next(Off);
  end
  done: terminate;
  end
end
Evolving Models

Adding detail to a high-level model:

```
S1
```

```
e1/send(oa, 5);
```

```
e2/
  {printf(q);}
```

```
e1[q=5]/
  {d = msg->data();
   send(oa, 5, d);}
```

```
e2/{printf(q);}
```

```
e32/
```

```
end/{printf("bye");}
```
Software has the rare property that allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods!
Model-Driven Development and MDA
Model-Driven Style of Development

- An approach to software development in which the focus and primary artifacts of development are *models* (as opposed to programs)
  - “The model is the implementation”
Modeling versus Programming Languages

- Cover different ranges of abstraction

**Level of Abstraction**

- **High** ($\Delta_{HI}$):
  - Statecharts, interaction diagrams, architectural structure, etc.
- **Low** ($\Delta_{LO}$):
  - Data layout, arithmetical and logical operators, etc.

**Programming Languages**
- (C/C++, Java, …)

**Modeling Languages**
- (UML, …)
“Action” languages (e.g., Java, C++) for fine-grain detail

- Level of Abstraction
  - Programming Languages (C/C++, Java, ...)
  - Modeling Languages (UML, ...)
  - (Any) Action Language

Fine-grain logic, arithmetic formulae, etc.
Each abstraction level specified using most appropriate form

- Fine-grain logic in a traditional 3G language
- High-level parts described using high-level abstractions

**Advantage:** exploits
- Existing tools
- Code libraries
- Developer experience
By inspection
- mental execution
- unreliable

By formal analysis
- mathematical methods
- reliable (theoretically)
- but: software is very difficult to model accurately!

By experimentation (execution)
- more reliable than inspection
- direct experience/insight
MDD Implications

- Ultimately, it should be possible to:
  - Execute models
  - Translate them automatically into implementations
  - ...possibly for different implementation platforms

Platform independent models (PIMs)

- Modeling language requirements
  - The semantic underpinnings of modeling languages must be precise and unambiguous
  - It should be possible to easily specialize a modeling language for a particular domain
  - It should be possible to easily define new specialized languages
The OMG’s Model Driven Architecture

- The OMG has formulated an initiative called “Model-Driven Architecture” (MDA)
  - A framework for a set of standards in support of a model-driven style of development
  - Inspired by the widespread public acceptance of UML and the availability of mature MDD technologies

- Rational is a pioneer of model-driven development and is one of the principal drivers of MDA
  - Conceived and refined UML (Booch, Rumbaugh, Jacobson)
  - Model-driven development process (RUP)
  - Tools for executable models and automatic code generation (XDE, Rose RealTime, Rose)
The Languages of MDA

- Set of modeling languages for specific purposes

**UML “bootstrap”**

MetaObject Facility (MOF)

- MOF “core”
  - A modeling language for defining modeling languages

General Standard UML

- For general OO modeling

Common Warehouse Metamodel (CWM)

- For exchanging information about business data

Real-Time profile

- EAI profile

Software process profile

- etc.

IBM Software Group
UML: The Foundation of MDA

- UML 1.1 (OMG Standard)
  - Rumbaugh
  - Booch
  - Jacobson
  - Foundations of OO (Nygaard, Goldberg, Meyer, Stroustrup, Harel, Wirfs-Brock, Reenskaug,...)

- UML 1.3 (extensibility)
- UML 1.4 (action semantics)
- UML 1.4.1
- UML 2.0 (MDA)
The Unified Modeling Language – version 2.0: Fundamentals
The technical material described here is still under development and is subject to modification prior to adoption by the OMG
Formal RFP Requirements

- Infrastructure – UML internals
  - More precise conceptual base for better MDA support

- Superstructure – User-level features
  - New capabilities for large-scale software systems
  - Consolidation of existing features

- OCL – Constraint language
  - Full conceptual alignment with UML

- Diagram interchange standard
  - For exchanging graphic information (model diagrams)
Approach: Evolutionary

- Improved precision of the infrastructure
- Small number of new features
- New feature selection criteria
  - Required for supporting large industrial-scale applications
  - Non-intrusive on UML 1.x users (and tool builders)
- Backward compatibility with 1.x
A core language + optional “sub-languages”

- Enables flexible subsetting for specific needs
- Users can “grow into” more advanced capabilities

**Basic UML**

(Classes, Basic behavior, Internal structure, Use cases...)

**Intermediate Level**

- Structured Classes and Components
- Activities
- Interactions

**Complete Level**

- State Machines
- Detailed Actions
- Flows
Infrastructure: Consolidation of Concepts

- Breakdown into fundamental conceptual primitives

- Eliminates semantic overlap
- Better foundation for a precise definition of concepts and semantics
Infrastructure: Behavior Harmonization

- Common semantic base for all behaviors
  - Choice of behavioral formalism driven by application needs

```
Classifier 0..1 ClassifierBehavior 0..1 Behavior

Class  UseCase  Component  ...  Action  Activity  Interaction  Statemachine

classifierBehavior
```
Structure Modeling:
UML as an Architectural Description Language
Structured Classes: External View

- Distributed active (concurrent) objects with
  - Full two-way encapsulation
  - Multiple interaction points: **ports**
Boundary objects that
- help separate different (possibly concurrent) interactions
- fully isolate an object’s internals from its environment

“There are very few problems in computer science that cannot be solved by adding an extra level of indirection”
Port Semantics

- A port can support multiple interface specifications
  - Provided interfaces (what the object can do)
  - Required interfaces (what the object needs to do its job)

```
interface MasterIF
    stateChange(s:state):void
    ...

interface SlaveIF
    start():void
    stop():void
    queryState():state
    ...
```

© IBM Software Group

Incoming signals/calls

`class X`

Outgoing signals/calls

`p1`
Protocols: Reusable Interaction Sequences

- Communication sequences that
  - always follow a pre-defined dynamic order
  - occur in different contexts with different specific participants

- Important architectural tool
  - Defines valid interaction patterns between architectural elements
Modeling Protocols with UML 2.0

- Modeled by a set of interconnected interfaces whose features are invoked according to a formal behavioral specification
  - Based on the UML collaboration concept
  - May be refined using inheritance

**Operator Assisted Call**

![Diagram showing the interaction between a Caller, Callee, and Operator interfaces, with interaction specs and a state machine specification.](image)
Associating Protocols with Ports

- **Ports play individual protocol roles**
  - Ports assume the protocol roles implied by their provided and required interfaces

![Diagram showing the relationship between interfaces and roles](image)
Assembling Communicating Objects

- Ports can be joined by connectors to create peer collaborations composed of structured classes.

Connectors model communication channels
A connector is constrained by a protocol
Static typing rules apply (compatible protocols)
Structured classes may have an internal structure of (structured class) parts and connectors.

Delegation connector

sender:Fax

remote

receiver:Fax

remote

Part

FaxCall
Structure Refinement Through Inheritance

- For product families with a common architecture
Modeling Complex Interactions
Overview of New Features

- Interactions focus on the communications between collaborating instances communicating via messages
  - Both synchronous (operation invocation) and asynchronous (signal sending) models supported

- Multiple concrete notational forms:
  - sequence diagram
  - communication diagram
  - interaction overview diagram
  - timing diagram
  - interaction table
All interactions occur in structures of collaborating parts

- the structural context for the interaction
Interaction Occurrences

Interaction Frame

Lifeline is one object or a part

Interaction Occurrence

sd GoHomeSetup

:ServiceUser

:ServiceBase

ref SB_GoHomeSetup

:ServiceTerminal

ref

Authorization

opt

FindLocation

SetHome

SetInvocationTime

SetTransportPreferences

sd Authorization

:ServiceUser

:ServiceBase

ref SB_Authorization

:ServiceTerminal

Code

OK

OnWeb

OK

Asynchronous message (signal)

Combined (in-line) Fragment
Combined Fragments and Data

sd GoHomeInvocation(Time invoc)

:ServiceUser

:Clock

:ServiceBase

:ServiceTerminal

InvocationTime

FindLocation

TransportSchedule

ScheduleIntervalElapsed

GetTransportSchedule

TransportSchedule

FetchSchedule

[Now>invoc]

[Now>interv+last]

[pos-lastpos>dist]

[Now>invoc]
Interaction Overview Diagram

An interaction with the syntax of activity diagrams
Dynamic Process Modeling Capabilities (Activities)
Activities: New Semantic Foundation

- Petri Net foundation (vs. statecharts) enables
  - Un-structured graphs (graphs with “go-to’s”)
  - Richer models of concurrency

Pre- and post-conditions

Input pin

ProcessOrder
RequestedOrder:Order

<<precondition>> Order complete
<<postcondition>> Order entered

Requested Order → Receive Order → [order accepted] → Bill Order → Skip Order → Close Order

Send Invoice → Invoice

Make Payment → Accept Payment
Hierarchical Partitions

Order Department

Receive Order → Fill Order → Ship Order → Close Order

Acctg Department

Send Invoice → Accept Payment

Invoice

Customer

Make Payment

(order accepted)
**Extended Concurrency Model**

- Fully independent concurrent streams (“tokens”)

```
Concurrency fork

A → B → C

X → Y → Z

Concurrency join
```

Trace: A, \{ (B,C) || (X,Y) \}, Z

“Tokens” represent individual execution threads (executions of activities)

NB: Not part of the notation
Activities: Token Queuing Capabilities

- **Tokens can**
  - queue up in “in/out” pins.
  - backup in network.
  - prevent upstream behaviors from taking new inputs.

- …or, they can flow through continuously
  - taken as input while behavior is executing.
  - given as output while behavior is executing.
New Statechart Modeling Capabilities
State Machine Improvements

- **New modeling constructs:**
  - Modularized submachines
  - State machine specialization/redefinition
  - State machine termination
  - “Protocol” state machines
    - transitions pre/post conditions
    - protocol conformance

- **Notational enhancements**
  - action blocks
  - state lists
Modular Submachines: Definition

- **ENTRY point**: selectAmount
- **EXIT point**: EnterAmount
- **Submachine definition**

The diagram shows the flow of control through the submachine, starting at the **ENTRY point** and proceeding through **selectAmount** and **otherAmount** to the **EXIT point**. The transition paths include:

- **selectAmount** to **otherAmount**
- **otherAmount** to **amount**
- **amount** to **ok**
- **ok** to **again**
- **again** to **EnterAmount**

The **aborted** state is reached through the **aborted** transition from the **EXIT point**. The diagram also highlights the **abort** transitions from **selectAmount** and **EnterAmount**.
Specialization

- Redefinition as part of standard class specialization

```
<table>
<thead>
<tr>
<th>ATM</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptCard()</td>
<td></td>
</tr>
<tr>
<td>outOfService()</td>
<td></td>
</tr>
<tr>
<td>amount()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FlexibleATM</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>otherAmount()</td>
<td></td>
</tr>
<tr>
<td>rejectTransaction()</td>
<td></td>
</tr>
</tbody>
</table>
```

<<Redefine>>

IBM Software Group | Rational software
Example: State Machine Redefinition

- State machine of ATM to be redefined

```
ATM

<table>
<thead>
<tr>
<th>Transition</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>VerifyCard</td>
<td>{final}</td>
</tr>
<tr>
<td>acceptCard</td>
<td></td>
</tr>
<tr>
<td>ReadAmount</td>
<td></td>
</tr>
<tr>
<td>selectAmount</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td></td>
</tr>
<tr>
<td>outOfService</td>
<td>{final}</td>
</tr>
<tr>
<td>outOfService</td>
<td></td>
</tr>
<tr>
<td>VerifyTransaction</td>
<td>{final}</td>
</tr>
<tr>
<td>releaseCard</td>
<td></td>
</tr>
<tr>
<td>ReleaseCard</td>
<td>{final}</td>
</tr>
</tbody>
</table>
```
Summary

- The “next generation” UML represents a significant evolutionary step:
  - Balance of consolidation and feature extensions
  - Modularized (core + optional specialized sub-languages)
  - Increased semantic precision and conceptual clarity
  - Supports full diagram interchange
  - Full alignment with MOF
  - Suitable MDA foundation (executable models, full code generation)

- New modeling features chosen for modeling large-scale systems

- Expected availability: 2003
QUESTIONS?

(bselic@rational.com)