An Overview of UML 2.0

Bran Selic
IBM Distinguished Engineer
IBM Rational Software – Canada
bselic@ca.ibm.com
The technical material described here is still under development and is subject to modification prior to full adoption by the Object Management Group.
The Essence of MDA

- Support for model-driven development through open industry standards

![Diagram with Abstraction, Automation, and Open Standards connected by MDA]

IBM Software Group | Rational Software
The Languages of MDA

- Set of modeling languages for specific purposes

**MetaObject Facility (MOF)**

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

**UML “bootstrap”**

- General Standard UML
  - For general OO modeling

- Common Warehouse Metamodel (CWM)
  - For exchanging information about business data

- etc.

- Real-Time profile
- EAI profile
- Software process profile
- etc.
UML: The Foundation of MDA

UML 2.0 (MDA)
- UML 1.5
- UML 1.4 (action semantics)
- UML 1.3 (extensibility)

UML 1.1 (OMG Standard)
- Rumbaugh
- Booch
- Harel
- Jacobson

Foundations of OO (Nygaard, Goldberg, Meyer, Stroustrup, Harel, Wirfs-Brock, Reenskaug,...)

- 1967
- 1996
- 1997
- 1998
- 2001
- 3Q2004
UML 1.x: What Went Right

- Timeliness (meeting a real need)
- Emphasis on semantics as opposed to notation
  - model-based approach (versus view-based)
  - detailed semantic specifications
- Higher-level abstractions beyond most current OO programming language technology
  - state machines and activity diagrams
  - support for specifying inter-object behavior (interactions)
  - use cases
- Customizability (extensibility)
Traditional Approach to Views in Modeling

- Multiple, informally connected views
  - Combined in the final (integration) phase of design
UML Approach: Single Model

✿ Views are projections of a complete model
  - Continuous integration of views with dynamic detection of inconsistencies

Well-formedness rules defined by the UML metamodel

Mapping rules defined by the UML specification
The Model and the System

The model is a representation of the system. The model repository contains the model and its components. The model is modeled by the computer, which processes data and provides the "meaning" of the model (semantics).
The “4-Layer” Architecture

- **Real Objects**
  - Customer
    - id
  - CustomerOrder
    - item
    - quantity
  - <Ben&Jerry’s>
  - <sawdust> 01011</sawdust>
  - <2 tons> 01011</2 tons>
  - <lard> 01011</lard>
  - <5 tons> 01011</5 tons>

- **Model**
  - (M1) Model (model repository)
  - CustomerOrder
  - Class
  - Association

- **Meta-Model**
  - (M2 = UML, CWM)
  - Class (MOF)
  - «specifiedBy»
  - «modeledBy»

- **Meta-Meta-Model**
  - (M3 = MOF)
  - «specifiedBy»

**Real Objects** (computer memory, run-time environment)
Avoiding the PL/I syndrome ("language bloat")
- UML standard as a basis for a “family of languages”

Using built-in extensibility mechanisms: profiles, stereotypes, etc.
UML 1.x: What Went Wrong?

- Inadequate semantics definition
  - Vague or missing (e.g., inheritance, dynamic semantics)
  - Informal definition (not suitable for code generation or executable models)
- Does not fully exploit MDD potential of models
  - E.g., “C++ in pictures”
- Inadequate modeling capabilities
  - Business and similar processes modeling
  - Large-scale systems
  - Non-functional aspects (quality of service specifications)
- Too complex
  - Too many concepts
  - Overlapping concepts
- No diagram interchange capability
- Not fully aligned with MOF
  - Leads to model interchange problems (XMI)
Sources of Requirements

- **MDA (retrofit)**
  - Semantic precision
  - Consolidation of concepts
  - Full MOF-UML alignment

- **Practitioners**
  - Conceptual clarification
  - New features, new features, new features...

- **Language theoreticians**
  - My new features, my new features, my new features...
  - Why not replace it with my modeling language instead?

- **Dilemma: avoiding the “language bloat” syndrome**
Formal RFP Requirements

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

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

3) OCL – Constraint language
   - Full conceptual alignment with UML

4) Diagram interchange standard
   - For exchanging graphic information (model diagrams)
Infrastructure Requirements

- Precise MOF alignment
  - Fully shared “common core” metamodel

- Refine the semantic foundations of UML (the UML metamodel)
  - Improve precision
  - Harmonize conceptual foundations and eliminate semantic overlaps
  - Provide clearer and more complete definition of instance semantics (static and dynamic)

- Improve extension mechanisms
  - Profiles, stereotypes
  - Support “family of languages” concept
OCL Requirements

- Define an OCL metamodel and align it with the UML metamodel
  - OCL navigates through class and object diagrams ⇒ must share a common definition of Class, Association, Multiplicity, etc.

- New modeling features available to general UML users
  - Beyond constraints
  - General-purpose query language
Diagram Interchange Requirements

- Ability to exchange graphical information between tools
  - Currently only non-graphical information is preserved during model interchange
  - Diagrams and contents (size and relative position of diagram elements, etc.)
Superstructure Requirements (1 of 2)

- More direct support for architectural modeling
  - Based on existing architectural description languages (UML-RT, ACME, SDL, etc.)
  - Reusable interaction specifications (UML-RT protocols)
- Behavior harmonization
  - Generalized notion of behavior and causality
  - Support choice of formalisms for specifying behavior
- Hierarchical interactions modeling
- Better support for component-based development
- More sophisticated activity graph modeling
  - To better support business process modeling
Superstructure Requirements (2 of 2)

- **New statechart capabilities**
  - Better modularity
- **Clarification of semantics for key relationship types**
  - Association, generalization, realization, etc.
- **Remove unused and ill-defined modeling concepts**
- **Clearer mapping of notation to metamodel**
- **Backward compatibility**
  - Support 1.x style of usage
  - New features only if required
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
  - Foundations
  - Actions
  - Activities
  - Interactions
  - Structures
  - State machines
  - Profiles
  - Templates
  - Summary
Multiple competing submissions (5)

- Most involved consortia of companies representing both UML tool vendors and UML users
- One prominent (800-lb gorilla) submission team (“U2P”) with most of the major vendors (Rational, IBM, Telelogic, ...) and large user companies (Motorola, HP, Ericsson…)

Over time:

- Some submissions lapsed
- Some submissions were merged into the U2P
- Currently there is just one submission
Approach to Evolving UML 2.0

- Evolutionary rather than revolutionary
- 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 + a set of optional “sub-languages”
- Defined in three separate compliance levels

Multiple levels of compliance
UML-MOF Alignment

- Shared conceptual base
  - MOF: language for defining modeling languages
  - UML: general purpose modeling language
Infrastructure Library – Metamodel Structure

- Shared by MOF, UML, and other languages

Diagram:
- **Primitive Types**
  - Boolean, Integer, String, ...
  - Imported from Basic definition of Class concept
- **Abstractions**
  - Namespace, Classifier, Relationship, Generalization, ...
  - Imported from Constructs
- **Constructs**
  - Extended notion of Class, Association,
  - Imported from Profiles
- **Basic**
  - Basic definition of Class concept
  - Imported from Profiles
- **Profiles**
  - Imported from Constructs

Legend:
- «import»
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
  - Actions
  - Activities
  - Interactions
  - Structures
  - State machines
  - Profiles
  - Templates
  - Summary
Class Diagram Semantics

- Represent relationships between *instances* of classes

Formal (set theoretic) interpretation:

Set of all Companies

- C1
- C2
- C3
- ... Cn

Set of all Persons

- P1
- P2
- P3
- ... Pm

Link <C1, P2>

Set: employee(s) of C1
Generalization Semantics

- Subclasses = specialized subsets of parent
- Subclass inherits all features of the parent and may add its own
- Relationship between classes

```
<table>
<thead>
<tr>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>position : Point</td>
</tr>
</tbody>
</table>

Set of all instances of Shape

Ellipse  Polygon  Spline

s1  s2  s3  ...  sn  sp
```
Association Specialization

- Also used widely in the definition of the UML metamodel
  - Avoids covariance problems

Diagram:

- Customer
  - Corporate Customer
    - company
      - 1
        - {subsets owner}
  - Private Customer
    - owner
      - 1
        - {subsets owner}
  - {owner}
    - 0..1
  - {accounts}
    - *{subsets accounts}

- Account
  - Corporate Account
    - accounts
      - *{subsets accounts}
  - Private Account
    - accounts
      - 0..5
        - {subsets accounts}
Example: Classifier Definition

- Constructed from a basic set of primitive concepts

Diagram:
- Element
  - NamedElement
    - name : String
- Namespace
- Ownership
- Classifiers
  - Namespace
  - NamedElement
  - Classifier
  - Feature
Kinds of Classifiers

The following are kinds of Classifier in UML 2.0:
- Association (including AssociationClass)
- Class (including structured classes)
- Component
- Collaboration
- Interface
- Data Type
- Use Case
- Signal
- **Behavior !**
- etc.

Kinds of Behavior
- Activity
- Interaction
- State Machine
- Protocol State Machine
The re-factoring of the UML metamodel into fine-grained independent concepts

- Eliminates semantic overlap
- Provides a better foundation for a precise definition of concepts and their semantics
- Conducive to MDD
In some cases we would like to modify a definition of a class without having to define a subclass.

- To retain all the semantics (relationships, constraints, etc.) of the original.
Package Merge: Semantics

- Optional incremental definition of concepts

Diagram:

- Library
  - Customer
    - name: String
  - Account
    - 1
    - *
  - Extend
    - Customer
      - age: Integer

«merge»
Package Merge: Metamodel Usage

- Enables common definitions for shared concepts with the ability to extend them according to need
  - E.g. MOF and UML definitions of Class
UML 2.0: Run-Time Semantics

- Activities
- State Machines
- Action Semantics
- Interactions (Communications)

Basic Structural Concepts
**Basic Structural Elements**

- **Values**
  - Universal, unique, constant
  - E.g. Numbers, characters, object identifiers (“instance value”)

- **“Cells” (Slots/Variables)**
  - Container for values or objects
  - Can be created and destroyed dynamically
  - Constrained by a type
  - Have identity (independent of contents)

- **Objects (Instances)**
  - Containers of slots (corresponding to structural features)
  - Just a special kind of cell

- **Links**
  - Tuples of object identifiers
  - May have identity (i.e., some links are objects)
  - Can be created and destroyed dynamically
In UML, all behavior results from the actions of (active) objects.
How Things Happen in UML

- An action is executed by an object
  - May change the contents of one or more variables or slots
  - If it is a communication ("messaging") action, it may:
    - Invoke an operation on another object
    - Send a signal to another object
    - Either one will eventually cause the execution of a procedure on the target object…
    - …which will cause other actions to be executed, etc.

- Successor actions are executed
  - Determined either by control flow or data flow
Active Object Definition

- From the spec:

An active object is an object that, as a direct consequence of its creation, [eventually] commences to execute its classifier behavior [specification], and does not cease until either the complete behavior is executed or the object is terminated by some external object.

The points at which an active object responds to [messages received] from other objects is determined solely by the behavior specification of the active object...
Metamodel Structure

- Classes
  - Structure-Behavior Dependency
  - Shared Behavior Semantics
- Common Behaviors
  - Activities
  - Interactions
  - StateMachines
  - UseCases
  - Actions
  - Different Behavior Formalisms

- Shared Behavior Semantics
- Structure-Behavior Dependency

IBM Software Group | Rational software
The “classifier behavior” of a composite classifier is distinct from the behavior of its parts (i.e., it is NOT a resultant behavior)
UML 2.0: Kinds of Behavior

- Actions
  - Formally, defined in the context of an Activity
- Activities
- Interactions
- State machines
- Use cases
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
  - Activities
  - Interactions
  - Structures
  - State machines
- Profiles
- Templates
- Summary
Actions in UML

- **Action = fundamental unit of behavior**
  - for modeling fine-grained behavior
  - Level of traditional programming languages

- **UML defines:**
  - A set of action types
  - A semantics for those actions
    - i.e. what happens when the actions are executed
    - Pre- and post-condition specifications (using OCL)
  - No concrete syntax for individual kinds of actions (notation)
    - Flexibility: can be realized using different concrete languages

- **In UML 2, the metamodel of actions was consolidated**
  - Shared semantics between actions and activities (Basic Actions)
Shared Action/Activity Semantics

- Data/control flow foundations for maximal implementation flexibility
Categories of Actions

- Communication actions (send, call, receive, …)
- Primitive function action
- Object actions (create, destroy, reclassify, start, …)
- Structural feature actions (read, write, clear, …)
- Link actions (create, destroy, read, write, …)
- Variable actions (read, write, clear, …)
- Exception action (raise)
General Notation for Actions

- No specific symbols (some exceptions)

```c
portP->send (sig)
```

```
for(int i = 0; i < s)
ia[i] = i++;
```

```c
«postcondition»
{port.state > 1}
```

```
«precondition»
{port.state > 0}
```
Activities

- Significantly enriched in UML 2.0 (relative to UML 1.x activities)
  - More flexible semantics for greater modeling power (e.g., rich concurrency model based on Petri Nets)
  - Many new features

- Major influences for UML 2.0 activity semantics
  - Business Process Execution Language for Web Services (BPEL4WS) – a de facto standard supported by key industry players (Microsoft, IBM, etc.)
  - Functional modeling from the systems engineering community (INCOSE)
Activity Graph Example

Order Processing
Invoice : InvoiceKind

«precondition» Order entered
«postcondition» Order complete

Input pin

Order

Receive order
Fill order
Ship order
Close order
Cancel order

Send invoice
Invoice
Make payment
Accept payment

Order cancel request

Interruptible Region

parameter
contracts
“Unstructured” Activity Graphs

- Not possible in 1.x
  - But, business processes are not necessarily well structured
### Partitioning capabilities

<table>
<thead>
<tr>
<th>Company</th>
<th>Accounting</th>
<th>Seattle</th>
<th>Reno</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Receive order</td>
<td>Ship order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fill order</td>
<td>Close order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send invoice</td>
<td>Accept payment</td>
</tr>
<tr>
<td></td>
<td>Invoice</td>
<td>Make payment</td>
<td></td>
</tr>
</tbody>
</table>

**Seattle**
- Receive order
- Fill order
- Ship order
- Close order

**Reno**
- Send invoice
- Accept payment
- Make payment
- Invoice
Activities: Basic Notational Elements

- Control/Data Flow
- Activity or Action
- Object Node (may include state)
- Pin (Object)
- Choice
- (Simple) Join
- Control Fork
- Control Join
- Initial Node
- Activity Final
- Flow Final
Extended Concurrency Model

- Fully independent concurrent streams ("tokens")

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
  - identified by a `{stream}` adornment on a pin or object node
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
- Activities
  - Interactions
- Structures
- State machines
- Profiles
- Templates
- Summary
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 (based on ITU Standard Z.120 – MSC-2000)
  - communication diagram
  - interaction overview diagram
  - timing diagram
  - interaction table
Interaction Diagrams

Interaction Frame

Lifeline is one object or a part

Interaction Occurrence

Combined (in-line) Fragment

sd ATM-transaction

client:    atm:    dbase:

insertCard

ref

CheckPin

alt

[chk= OK]

ref

DoTransaction

error(badPIN)

[else]

sd CheckPin

client:    atm:    dbase:

askForPIN

data(PIN)

check(PIN)

result(chk)

result(chk)

Combined (in-line) Fragment
Structural Decomposition in Sequence Diagrams

Decomposed lifeline

Folding to the decomposed lifeline

Decomposition with global constructs corresponding to those on decomposed lifeline

Detailed context

Decomposition with global constructs corresponding to those on decomposed lifeline

Go Home Setup

Service User

Service Terminal

Authorization

Find Location

Set Home

Set Invocation Time

Set Transport Preferences

SB Go Home Setup

SB Authorization

Central

Authorizer

Time Keeper

Decomposition with global constructs corresponding to those on decomposed lifeline

Decomposition with global constructs corresponding to those on decomposed lifeline

Decomposition with global constructs corresponding to those on decomposed lifeline
sd GoHomeInvocation(Time invoc)

:ServiceUser

InvocationTime

:Clock

FindLocation

:ServiceBase

TransportSchedule

:ServiceTerminal

GetTransportSchedule

ScheduleIntervalElapsed

[Now>invoc]

[Now>interv+last]

[Now>invoc]

[Now>interv+last]

[Now>interv+last]

[Now>invoc]

[Now>invoc]

[Now>invoc]

[Now>invoc]

Guarding Data

Constraint

Choice

Operand

Separator

loop

loop

loop

loop

Operand

Separator

Guarding Data

Constraint
Combined Fragment Types (1 of 2)

- **Alternatives** *(alt)*
  - choice of behaviors – at most one will execute
  - depends on the value of the guard (“else” guard supported)

- **Option** *(opt)*
  - Special case of alternative

- **Break** *(break)*
  - Represents an alternative that is executed instead of the remainder of the fragment (like a break in a loop)

- **Parallel** *(par)*
  - Concurrent (interleaved) sub-scenarios

- **Negative** *(neg)*
  - Identifies sequences that must not occur
Critical Region (region)
- Traces cannot be interleaved with events on any of the participating lifelines

Assertion (assert)
- Only valid continuation

Loop (loop)
- Optional guard: [<min>, <max>, <Boolean-expression>]
- No guard means no specified limit
Like flow charts

- using activity graph notation for control constructs
Timing Diagrams

- Can be used to specify time-dependent interactions
  - Based on a simplified model of time (use standard “real-time” profile for more complex models of time)

**sd DriverProtocol**

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>d</strong> : Driver</td>
<td>Idle</td>
<td>Wait</td>
<td>Busy</td>
<td>Idle</td>
</tr>
</tbody>
</table>

**o : OutPin**

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0111</strong></td>
<td><strong>0011</strong></td>
<td><strong>0001</strong></td>
<td><strong>0111</strong></td>
<td></td>
</tr>
</tbody>
</table>
Timing Diagrams (cont.)

sd Reader

```
Reading
Idle
Uninitialized
```

r : Reader

```
State
{d..d+0.5}
{t1..t1+0.1}
```

Event Occurrence

```
Observation
```

Constraint

```
```

Event Occurrence
Sidebar: Information Flows

- For specifying kinds of information that are exchanged between elements of the system – *at a very abstract level*
  - Do not specify details of the information (e.g., type)
  - Do not specify how the information is relayed
  - Do not specify the relative ordering of information flows
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
- Activities
- Interactions
- **Structures**
  - State machines
  - Profiles
  - Templates
  - Summary
Aren’t Class Diagrams Sufficient?

- **No!**
  - Because they abstract out certain specifics, class diagrams are not suitable for performance analysis
- **Need to model structure at the instance level**

Same class diagram describes both systems!
In UML 2.0 a collaboration is a **purely structural concept**
- (But, can include one or more associated interactions)
- More general than an instance model
Roles and Instances

Specific object instances playing specific the roles in a collaboration

c1/client :Cl

a/arbiter :Arb

d1/primary :Dbase

d2/backup1 :Dbase

d2/backup2 :Dbase
Alternative Notation

Diagram:

- c1:Cl (client)
- d1:Dbase (primary)
- d2:Dbase (backup1)
- a:Arb (arbiter)
- collaboration occurrence

Relationships:
- rs1:RedundantSystem
- client → rs1:RedundantSystem
- arbiter → rs1:RedundantSystem
- primary → rs1:RedundantSystem
- backup1 → rs1:RedundantSystem
- backup2 → rs1:RedundantSystem

Components:
- c1:Cl
- d1:Dbase
- d2:Dbase
- a:Arb
Collaboration Protocols

- For dynamic relationships between interfaces

**FaxProtocol**

- **FaxSender**
  - AckCall(par)
  - AckData( )
  - AckHangup( )

- **FaxReceiver**
  - Call()
  - CallParms( )
  - Data()
  - Hangup( )

**FaxSender**

- **sd** StartCall

**FaxReceiver**

- **sd** EndCall

**FaxSender**

- AckCall (parms)
- CallParms(par)

**FaxReceiver**

- Data
- AckData

**loop**
Structured Classes

- Classes with
  - Internal (collaboration) structure
  - Ports (optional)
- Primarily intended for architectural modeling
- Heritage: architectural description languages (ADLs)
  - ACME: Garlan et al.
  - SDL (ITU-T standard Z.100)
Structured Objects: Ports

- Multiple points of interaction
  - Each dedicated to a particular purpose

  e.g., Database Admin port

  e.g., Database Object

  e.g., Database User ports
New Feature: Ports

- Used to distinguish between multiple collaborators
  - Based on port through which interaction is occurring
- Fully isolate an object’s internals from its environment

```c
void E () {
    q.setA(d)
    ...
}
```
A port can support multiple interface specifications
- Provided interfaces (what the object can do)
- Required interfaces (what the object needs to do its job)
Ports: Alternative Notation

- Shorthand “lollipop” notation with 1.x backward compatibility
Assembling Structured Objects

- Ports can be joined by connectors
- These connections can be constrained to a protocol
  - Static checks for dynamic type violations are possible
  - Eliminates “integration” (architectural) errors
Structured classes may have an internal structure of (structured class) parts and connectors.

Delegation connector

sender:Fax

remote

Part

receiver:Fax

FaxCall
Using standard inheritance mechanism (design by difference)
A kind of structured class whose specification

- May be realized by one or more implementation classes
- May include any other kind of packageable element (e.g., various kinds of classifiers, constraints, packages, etc.)
A system stereotype of Component («subsystem») such that it may have explicit and distinct specification («specification») and realization («realization») elements

- Ambiguity of being a subclass of Classifier and Package has been removed (was intended to be mutually exclusive kind of inheritance)

- Component (specifications) can contain any packageable element and, hence, act like packages
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
- Activities
- Interactions
- Structures
- **State machines**
- Profiles
- Templates
- Summary
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

**ReadAmountSM**

- **ENTRY point**
- **EXIT point**
- **Submachine definition**
- **aborted**
- **again**
- **ok**
- **amount**
- **otherAmount**
- **selectAmount**
- **EnterAmount**
- **abort**
Modular Submachines: Usage

ATM

- **VerifyCard**
  - acceptCard
- **readAmount : readAmountSM**
  - outOfService
  - **OutOfService**
  - abort
  - again
  - **rejectTransaction**
  - **verifyTransaction**
  - **releaseCard**
  - **ReleaseCard**

Invoked submachine

Usage of entry point

Usage of exit point
Specialization

- Redefinition as part of standard class specialization

ATM
- acceptCard()
- outOfService()
- amount()

FlexibleATM
- otherAmount()
- rejectTransaction()

Behaviour

Statemachine

<<Redefine>>
Example: State Machine Redefinition

- State machine of ATM to be redefined
Protocol State Machines

- Impose sequencing constraints on interfaces
  - (should not be confused with multi-party protocols)

Equivalent to pre and post conditions added to the related operations:

**takeOff()**

**Pre**
- in state "Ready"
- cleared for take off

**Post**
- landing gear is retracted
- in state "Flying"
### Notational Enhancements

#### Alternative transition notation

- **Idle**
  - eventA
    - [ID<=10]
      - MinorReq=Id;
        - Minor(Id)
      - MajorReq=Id;
        - Major(Id)
    - [ID>10]
  - Busy

#### State lists

- **VerifyCard, ReleaseCard** -> **Logged**
  - logCard

- **VerifyCard** -> **Logged**
  - logCard

- **ReleaseCard** -> **Logged**
  - logCard
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
- Activities
- Interactions
- Structures
- State machines
- Profiles
- Templates
- Summary
Specializing UML

- **Lightweight extensions**
  - Extend semantics of existing UML concepts by specialization
  - Conform to standard UML (tool compatibility)
  - Profiles, stereotypes

- **Heavyweight (MOF) extensions**
  - Add new non-conformant concepts or
  - Incompatible change to existing UML semantics/concepts
Stereotyping versus Inheritance

- For semantics not expressible through standard UML mechanisms
- Stereotypes can be standardized (application independent)

Instance of the UML Class concept

```
Integer
```

```
«clock» Stereotype of Class with added semantics: an active counter whose value changes synchronously with the progress of physical time
```

```
MyClockClass
{resolution = 500 ns}
```

```
SetTime()
```

Stereotype-specific attribute
Profiles: Metamodel

- Semantically equivalent to 1.x from a user’s perspective
  - But, new notation introduced
  - Extension concept: a form of “specialization” of metaclasses
Profiles: Notation

- E.g., specializing the standard Component concept

```
«profile» SimpleEJB

«metaclass» Component

«stereotype» Bean

Extension

«stereotype» Entity

«stereotype» Session
```
Contents

- A critique of UML 1.x
- Requirements for UML 2.0
- UML 2.0 architecture
- Foundations
- Actions
- Activities
- Interactions
- Structures
- State machines
- Profiles
- **Templates**
- Summary
More precise model than UML 1.x
Limited to Classifiers, Packages, and Operations

**Templates**

- Template parameter
- Template signature
- Template binding
- "Bound" class

**NumericArray**
- `arrayElement : T [k]`
  - `T > Number, k : IntegerExpression`

**IntegerArray**
- `arrayElement : Integer [10]`
  - "bind" `<T -> Integer, k -> 10>"
Collaboration Templates

- Useful for capturing design patterns

**ObserverPattern**

```
subject : sType
observer : oType
```

Collaboration template

`<<oType-->>DevicePoller >Device>>`

DeviceObserver

ObserverPattern `<oType->DevicePoller, sType>Device>`
Package Templates

- Based on simple string substitution

CustomerAccountTemplate

$\text{<customer>}$

owner

1..*

$\text{<kind>Account}$

0..*

SavingsBank

Person

owner

1..*

PersonalAcct

0..*

Name Expression

<bind> $\text{<customer->Person, kind -> Personal}$
Summary: UML 2.0

- First major revision of UML
- Original standard had to be adjusted to deal with
  - MDD requirements (precision, code generation, executability)
- UML 2.0 characterized by
  - Small number of new features + consolidation of existing ones
  - Scaleable to large software systems (architectural modeling capabilities)
  - Modular structure for easier adoption (core + optional specialized sub-languages)
  - Increased semantic precision and conceptual clarity
  - Suitable foundation for MDA (executable models, full code generation)
QUESTIONS?

(bselic@ca.ibm.com)