UML 2: A Key MDA Technology

Bran Selic
IBM Distinguished Engineer
IBM Canada
bselic@ca.ibm.com
The Setting: Model-Driven Development (MDD)

UML 2 Highlights and Related Work

State of the Art in MDD
A Bit of Modern Software…

Can you see the architecture?
Can you see it now?
SC_MODULE(producer)
{
    sc_outmaster<int> out1;
    sc_in<bool> start; // kick-start
    void generate_data ()
    {
        for(int i =0; i <10; i++) {
            out1 =i ; // to invoke slave;
        }
    }
    SCCTOR(producer)
    {
        SC_METHOD(generate_data);
        sensitive << start;}};
SC_MODULE(consumer)
{
    sc_inslave<int> in1;
    int sum; // state variable
    void accumulate (){
        sum += in1;
        cout << "Sum = " << sum << endl;
    }
    SCCTOR(consumer)
    {
        SC_SLAVE(accumulate, in1);
        sum = 0; // initialize
    }
SC_MODULE(top) // container
{
    producer *A1;
    consumer *B1;
    sc_link_mp<int> link1;
    SCCTOR(top)
    {
        A1 = new producer("A1");
        A1.out1(link1);
        B1 = new consumer("B1");
        B1.in1(link1);}};
An approach to software development in which the focus and primary artifacts of development are models (vs programs). Based on two time-proven methods:

1. **ABSTRACTION**
   ```
   SC_MODULE(producer)
   {sc_inslave<int> in1;
    int sum; //
    void accumulate(){
      sum += in1;
      cout << "Sum = " << sum << endl;
    }
   }
   ```

2. **AUTOMATION**
   ```
   SC_MODULE(producer)
   {sc_inslave<int> in1;
    int sum; //
    void accumulate(){
      sum += in1;
      cout << "Sum = " << sum << endl;
    }
   }
   ```

Realm of modeling languages

Realm of tools
Model-Driven Architecture (MDA)

- An OMG initiative to support model-driven development through a series of open standards

(1) ABSTRACTION
(2) AUTOMATION

MDA™

(3) OPEN STANDARDS
- Modeling languages
- Interchange standards
- Model transformations
- Software processes
- etc.
UML: The Foundation of MDA

- **UML 2.1.1**
- **UML 2.0 (MDA)**
- **UML 1.5**
- **UML 1.4**
- **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
- 2003
- 2005
- 2006
Agenda

- The Setting: Model-Driven Development (MDD)
- UML 2 Highlights and Related Work
- State of the Art in MDD
What is UML For?

- For modeling *software systems* and their contexts
  - Using concepts from the world of object-oriented languages (class, operation, object, etc.)
- However, the general nature of these concepts makes UML suitable for extension to other and broader domains
  - E.g. systems engineering (SysML, UPDM)
UML 2 Highlights

Greatly increased level of precision to better support MDD
- More precise definition of concepts and their relationships
- Extended and refined definition of semantics

New language architecture
- Highly modularized structure for incremental adoption
- Greatly simplified compliance model for easier tools interchange

Improved support for modeling large-scale software systems
- Modeling of complex software structures (architectural description language)
- Modeling of complex end-to-end behavior
- Modeling of distributed, concurrent process flows (e.g., business processes, complex signal processing flows)

Greatly improved support for defining domain-specific languages (DSLs)

Consolidation and rationalization of existing concepts
UML 2 Semantics

- A layered and modularized semantics base

Behavioral Semantic Base
- Actions
- Object Behavior
- Object Interactions

Structural Semantic Base

Activities
State Machines
Interactions

Under further refinement by the upcoming *Executable UML Foundation* submission.
The New UML 2 Language Architecture

- A core language + a set of optional “language units”
  - Some language units have multiple increments

- Core UML
  - Classes, Basic behavior, Internal structure, Use cases...

- Level 1
  - UML Infrastructure

- Level 2
  - Structured Classes and Components
  - Activities
  - Interactions
  - Detailed Actions

- Level 3
  - State Machines
  - Flows

MOF Profiles OCL

Multiple levels of compliance
UML Compliance

- 4 levels of compliance (L0 – L3)
  - compliance(L_x) \implies compliance (L_{x-1})

- Dimensions of compliance:
  - Abstract syntax (UML metamodel, XMI interchange)
  - Concrete syntax
    - Optional Diagram Interchange compliance

- Forms of compliance
  - Abstract syntax
  - Concrete syntax
  - Abstract and concrete syntax
  - Abstract and concrete syntax with diagram interchange
Are Class Diagrams Sufficient for Modeling Structure?

- Not always!
  - Because they abstract out certain specifics, class diagrams are not suitable for performance analysis
- Sometimes it is necessary to model structure at the instance level

Same class diagram describes both systems!
A UML collaboration describes a set of communicating “roles”

- Roles represent instances but abstract away specifics of instances
  \[ \Rightarrow \text{generic instance diagrams} = \text{patterns} \]
- Connectors represent communication links

**Diagram:**

- **MiniHamlet**
  - **Ophelia:** YoungWoman
  - **Claudius:** OlderMan
  - **Hamlet:** YoungMan
  - **Ghost**

- **Connectors**
  - **Constrained role**
  - **Unconstrained role**
Structured Objects: External View

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

  - e.g., Database Admin port
  - e.g., Database Object
  - e.g., Database User ports
Structured classes may contain an internal collaboration structure that represents its implementation.
Architecture Refinement Through Inheritance

- Using standard inheritance mechanism (design by difference)
Categories of Actions

- For modeling fine-grained behavior
  - UML 2 unified the action and activity modeling paradigms

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)
UML 2: Action and Activity Basics

- Support for multiple computational paradigms

Diagram:
- Activity (context)
- Action 1
- Action 2
- Action 3
- Input Pin (typed)
- Output Pin (typed)
- Control Flow
- Data Flow
- Variable A
Major influences for UML 2 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)

Significantly enriched in UML 2

- More flexible semantics for greater modeling power
- Many new features
Activities

- For modeling procedural behavior
  - Similar to flowcharts, but...
  - ...includes support for concurrency modeling (parallel behaviors)
  - Based on the Petri Net formalism

- “Higher-level” actions
  - Shares same conceptual model as actions:
    - Control and data flows
    - Potential concurrent execution
    - Pins (parameters)

- Can be nested hierarchically to an arbitrary depth
  - Like procedures in traditional programming languages
Activity Example

Order Processing

- «precondition» Order entered
- «postcondition» Order complete

Order cancel request

Contracts

Interruptible Region

Input parameter

Order

Receive order

Fill order

Ship order

Cancel order

Close order

Send invoice

Invoice

Make payment

Accept payment

Order entered

Order complete

Order cancel request

Contracts

Interruptible Region

Send invoice

Invoice

Make payment

Accept payment

Order entered

Order complete

Order cancel request

Contracts

Interruptible Region
UML 2 Interactions Modeling

sd ATM-transaction

client: atm: dbase:

insertCard

ref

CheckPin

[chk= OK]

alt

DoTransaction

error(badPIN)

[else]

Interaction Occurrence

sd CheckPin

client: atm: dbase:

askForPIN

data(PIN)

check(PIN)

result(chk)

Combined (in-line) Fragment

result(chk)
OMG’s UML as a Platform for DSMLs

- Designed as a “family of modeling languages”
  - Contains a set of **semantic variation points** (SVPs) where the full semantics are either unspecified or ambiguous
  - SVP examples:
    - Precise type compatibility rules
    - Communications properties of communication links (delivery semantics, reliability, etc.)
    - Multi-tasking scheduling policies
  - Enables domain-specific customization
Success Criteria for a Computer Language

- **Technical validity**: absence of major design flaws and constraints (ease of writing correct programs)
- **Expressiveness**: ability to succinctly specify the necessary domain concepts
- **Simplicity**: absence of complexity (eases learning)
- **Efficiency**: potential to minimize space and performance overheads
- **Familiarity**: proximity to widely-available skills sets
- **Interoperability**: language compatible with other technologies
- **Support**: availability of the infrastructure required for effective exploitation
  - Availability of *effective* tools (editors, compilers, debuggers, static and dynamic analyzers, build tools, version control tools, merge/diff tools, etc.)
  - Availability of *program libraries*
  - Availability of *skilled practitioners*
  - Availability of *textbooks* and *training courses*
  - Institutions for evolution and maintenance
Example: Adding a Semaphore Concept to UML

«metaclass»
UML::Class

«stereotype»
Semaphore

limit : Integer
getSema : Operation
relSema : Operation

“Extension”
(NB: filled arrowhead)

Iconic Representation

Constraints

limit <= MAXlimit
**Profile:**

- A special kind of package containing stereotypes and model libraries that, in conjunction with the UML metamodel, define a group of domain-specific concepts and relationships.
- The profile mechanism is also available in MOF where it can be used for other MOF-based languages.

**Profiles can be used for two different purposes:**

- To define a domain-specific modeling language.
- To define a domain-specific viewpoint.
Current Catalog of OMG Profiles

- UML Profile for CORBA
- UML Profile for CORBA Component Model (CCM)
- UML Profile for Enterprise Application Integration (EAI)
- UML Profile for Enterprise Distributed Object Computing (EDOC)
- UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms
- UML Profile for Schedulability, Performance, and Time
- UML Profile for System on a Chip (SoC)
- UML Profile for Systems Engineering (SysML)
- UML Testing Profile
- New profiles
  - UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
  - UML profile for DoDAF/MoDAF (UPDM)
  - UML profile for Modeling Services (UPMS)
MARTE Usage Example

- Seamless inter-working of specialized tools based on shared standards

UML Modeling Tool

Model Analysis Tool

Quality of Service Specifications

Automated model conversion

Inverse automated model conversion
- The Setting: Model-Driven Development (MDD)
- UML 2 Highlights and Related Work
- State of the Art in MDD
The MDD Maturity Model

- **Levels of Abstraction Automation**
  - Code only
  - Visualization
  - Round Trip Engineering
  - Model-centric
  - Model only

- **Code only**
  - "What's a Model?"

- **Visualization**
  - "The code is the model"

- **Round Trip Engineering**
  - "Manage code and model"

- **Model-centric**
  - "The model is the code"

- **Model only**
  - "Let's talk models"

**Levels of Abstraction**
- M (Model)
- C (Code)

**Automation**
- Visualize
- Synchronize
- Generate

**Time**

IBM Software Group | Rational software
Model-driven development at the “Model” level
- Higher levels of abstraction and
- Higher levels of automation (advanced tools with full or partial automatic code generation)

Major improvements in:
- Product reliability
- Developer productivity

MDD tools have reached the critical maturity threshold:
- Performance (tools and code)
- Interoperability
- Capability
- Scalability
Automatic Code Generation: Full Generation

- **State of the art:**
  - All development done via the model (i.e., no modifications of generated code)
  - Size: Systems equivalent to ~ 10 MLoC
  - Scalability: teams involving hundreds of developers
  - Performance: within ±5-15% of equivalent manually coded system
Major Telecom Equipment Manufacturer

- Adopted MDD Tooling
  - Rose RealTime, Test RealTime, RUP
- Example 1: Radio Base Station
  - 2 Million lines of C++ code (87% generated by tools)
  - 100 developers
- Example 2: Gateway
  - 300,000 lines of C++ code (83% generated by tools)
  - 30 developers
- Example 3: Network Controller
  - 4.5 Million lines of C++ code (80% generated by tools)
  - 400 developers
Major Telecom Equipment Manufacturer

- Adopted MDD Tooling
  - Rose RealTime, Test RealTime, RUP

- Example 1: Radio Base Station
  - 2 Million lines of C++ code (87% generated)
  - 100 developers

- Example 2: Gateway
  - 300,000 lines of C++ code (83% generated)
  - 30 developers

Benefits
- 80% fewer bugs
- 30% productivity increase
- 30% reduction in required documentation

"I can't see how we could have done it in the given timeframe without Rational Rose RealTime"  Project Manager, Telecommunications Equipment Manufacturer

"We cut production time from 16 to 12 months. The quality was dramatically improved."  Corporate Director, Software Technology Development
The MDD Maturity Model vs State of the Practice

State of the Practice

- Code Visualization
  - M
  - C
  - "What's a Model?"
  - "The code is the model"

Round Trip Engineering

- M
- C
- "Manage code and model"
- "The model is the code"

Model-centric

- M
- C
- "The model is the code"

State of the Art

Levels of Abstraction

Automation

- Manage code and model

"Let's talk models"

"What's a Model?"

"The code is the model"

"Manage code and model"

"The model is the code"
Conclusions

- Model-Driven Development (MDD) has a *proven* potential to make a significant difference in productivity and quality of software development
  - The OMG is supporting MDD through its MDA standardization initiative
- UML 2 is a key member of the MDA standards family
  - As a core modeling language positioned for MDD
  - As the foundation for a set of standard domain-specific modeling languages
- The first generation of MDD-based products and supporting tools and methods are available
  - It has proven the effectiveness of this highly automated approach to software development
UML References

- General modeling specs:

- UML 2 specs:

- Books: