UML 2.0

Components,
System Family Modeling,
Domain Specific Languages,
and MDA

Birger Møller-Pedersen
University of Oslo, Norway
UML standardization within OMG – for Ericsson

Requirements from Ericsson developers world-wide

better tools

Ericsson UML standardization team

contributing in cooperation with tool vendors

issuing requirements in cooperation with alllies

OMG
U2 Partners

- **Submitters**
  - Alcatel, CA, Ericsson, Fujitsu, HP, IBM, I-Logix, IONA, Kabira, Motorola, Oracle, Rational, SOFTEAM, Telelogic, Unisys

- **Supporters**
Background – modeling architecture of systems

- BBD: Blackboard Block Diagrams
- SDL (agents, gates, channels, ...)
- ROOM/UML-RT (capsules, ports, connectors, ...)
- ADLs (components, interfaces, connectors)

- for Ericsson, Motorola, Alcatel, Nokia (telecom, realtime)
  - SDL only one vendor
  - UML-RT (RoseRT) only one vendor

- UML2.0 combining features from these
ATM: Domain Model 1

From
Haugen, Ø., B. Møller-Pedersen, and T. Weigert,
Modeling Embedded Systems in UML 2.0, in
The Embedded Systems Handbook,
ATM: Domain Model II

ATM

CardReader
Keyboard
Screen
CashDispenser
Context model with UML1.x class diagrams

- with plain composition and **no** encapsulation
- with only **provided** interfaces on classes

![Diagram of a context model with UML1.x class diagrams showing relationships between User, ATM, Bank, CardReader, Keyboard, Screen, and CashDispenser. Connections are labeled User-ATM and ATM-Bank.]
UML2.0: Composite Class

- with parts, ports and connectors
Context Models in UML2.0 - 1

- composite structures also as part of Collaborations
Context Models in UML2.0 - II

- including multiplicities on parts
Structured Classes are like other Classes

- Structured Classes may have
  - attributes & operations, interfaces, …
- Internal structure is inherited, inherited parts may be redefined by extension
What about Components?

- Components have all the properties of structured classes

Note that these are just derived, that is they are also defined for classes.
Notation

- ... and for classes
Delegation connector

- ... and for classes
Assembly connectors

- ... and between class-defined parts
What is special for Components

- Realization by a number of classes
... and

- may be kind of ‘package’, i.e. it may have model elements that you would **not** have for classes

A component may have e.g. use cases, sequence diagrams, packages, dependencies, components
Deployment of components

- Artifacts,
- Nodes,
- Network of Nodes,
- ...
Artifacts
Nodes with deployed artifacts
Must be profiled for actual component models

Table 29 - Example Profile for Enterprise Java Beans

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Parent</th>
<th>Tags</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJBEntityBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EJBEntityBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBSessionBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EJBSessionBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBMessage DrivenBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EJBMessage DrivenBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBHome</td>
<td>Interface</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EJBHome»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBRemote</td>
<td>Interface</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EJBRemote»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Must be profiled for actual component models

Table 31 - Example Profile for .NET Components

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Parent</th>
<th>Tags</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetComponent</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NetComponent»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NETProperty</td>
<td>Property</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NETProperty»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NETAssembly</td>
<td>Package</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NETAssembly»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSI</td>
<td>Artifact</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«MSI»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLL</td>
<td>Artifact</td>
<td>«file»</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«DLL»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE</td>
<td>Artifact</td>
<td>«file»</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EXE»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Components = composite classes + deployment
- i.e. composite structures (parts, ports and connectors) not specific to components!
  - Some reasons why this is a god idea
    - Even analysis models may be structured
    - Models may be deployed (e.g. on real-time or embedded platforms) even without component technology
    - Example: System family modeling based upon plug-ins (but not using components)
  - In general: users of UML tend to
    - tailor UML to whatever they need, except MicroSoft: Domain Specific Languages
    - transform models to whatever platform they are using (the essence of MDA)
System Family Modeling (Product Line Modeling)

- The modeling of architecture is important
  - including the modeling of commonalities and variations
  - independent of whether components are used or not

- Approaches
  - Frameworks with hot-spots/plug-ins
  - Feature models: 'Stereotyping' variation model elements
  - Templates
Example – Access Control System

From
Haugen, Ø., B. Møller-Pedersen, and T. Weigert,
*Structural Modeling with UML 2.0*, in *UML for Real*,
L. Lavagno, G. Martin, and B. Selic, Editors. 2003,
Variations
Plug-in approach

- Directly supported by ports and connectors with well-defined interfaces
- Important that this works with classes and not just for components
Feature models

- One large system family model, with variations by stereotyping model elements
- If not modeling, this is called Generative Programming

System Family Model = Union of all potential System Models
Stereotyping classes

- Good news
  - UML2.0 allows more than one stereotype for each class
  - <<optional>> in addition to a real stereotype (e.g. <<embedded>>)

- Still does not help'
  - This tells which classes are defined, but it does not tell which classes from a given library are used in a given family or system
Even better news

- Usages of classes are highlighted in UML2.0

- ... and parts may of course be stereotyped
Templates

Variation by constrained template parameter: ACSSystem knows that aType is at least AccessPoint
Yet another approach:
Product Lines
call for
Software Factories,
which in turn call for
Domain Specific
Languages (DSL):

Forget UML,
make one language for
each domain,
for each product family
and/or for each platform
Recall this one?

- UML cannot be used because UML Components do not support .NET Component properties

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Parent</th>
<th>Tags</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetComponent</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NetComponent»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NETProperty</td>
<td>Property</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NETProperty»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NETAssembly</td>
<td>Package</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«NETAssembly»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSI</td>
<td>Artifact</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«MSI»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLL</td>
<td>Artifact</td>
<td>«file»</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«DLL»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE</td>
<td>Artifact</td>
<td>«file»</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>«EXE»</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Framework/plug-in approach

Feature Model approach

System Family Model = Union of all potential System Models

Feature selection

Transformation

System Model
The Domain-Specific-Language Approach

- no system family model
- new language for each family, many systems made in this language
What about MDA

- and Domain Specific Languages (DSL)?

MOF

UML  CWM  . . .

UML profiles . . .

SDL  DSL  LSD  LDS
MDA –

- The dominant approach
  - ‘press-the-button’ approach

  - executable UML, code generation
  - programming at another level?

  - Requires profiling, closing the many semantic variation points of UML
MDA: Model Driven Architecture

- From "MDA explained"
  - MDA is about transformations
  - from PIMs (Platform Independent Models)
  - to PSMs (Platform Specific Models)
Looks familiar!?

From our compiler course textbook
MDA -

- An alternative approach
  - the ‘multiple-models-are-better-than-one-model’ approach

- Property models (e.g. interactions, collaborations, use cases)
- Object models (class models, state machines)

- Comparing/checking object models against property models
MDA by means of profiling?

- Domain, company, department, project standard?

```
UML
/ | |
/  |  |
/   |   |
PIM  -> different languages

PSM
```

Profiles

- platform1 or domain 1
- platform2 or domain 2
- platform3 or domain 3
Domain Specific Languages <-> UML profiles

- **Domain Specific Languages**
  - Gives the exact right language concepts
  - Often defined from scratch, have to be maintained
  - Multi-domain models will have multiple DSLs, commonality between languages not solved

- **UML profiling**
  - Exploits the common (but sparse) semantics of UML
  - A model can use several profiles, in principle
  - Profiles tend to tweak UML, and in fact make new metamodels
  - Models with stereotypes often unreadable

- Developers will anyhow require real tool support comparable to tools they have for programming languages