SUPPORTING THE MDA APPROACH WITH UML PROFILES

Philippe DESFRAY
www.softeam.fr
www.objecteering.com
Benefits of an MDA based approach

MDA necessitates the formalization of knowledge involved in software development, thus leading to:

- A better control of an organization’s know how (architecture, methodology, etc.)
- Ability to apply proven practices in a regular and systematic manner.
- Thesaurus of software development knowledge and practices

_This is the key to software quality and development productivity. MDA contains the basis for development automation, and continuous process improvement._
Example of a profile based MDA configuration
Typical organization for an MDA based approach

- **UML Modeler**
  - UML
  - (packaged profiles)
  - Designer
    - Use a customized Case tool adapted to your domain

- **UML Profile Builder**
  - Software Process
  - Components
  - Real Time
  - XML
  - C++
  - EJB
  - Java
  - Domain Expert
    - Design and implement UML expertise for any kind of domain

- Development team
- "Know how" team

UML WorkShop 2001 - SOFTEAM
MDA – Pragmatic issues
Formalizing the « know how »

- MDA requires a high degree of formalization of the different models, architectures, and methodologies
- In addition, mappings have to be specified in detail
- This involves highly skilled people, and implies a great risk to development.
- It necessitates investments in time and money before starting development, and getting ROI

Rule 1 Use as much as possible standards : standard models, profiles. UML should be the basis
MDA – Pragmatic issues
« Evangelizing » the MDA based approach

• Managers and developers need to be convinced very fast
• ROI has to be proven
• « Practicability » issues have to be worked out in detail:
  – How comfortable is it to use the toolset in every development phase
  – Cultural aspects, such as use of well known approach, automation of well approved and shared patterns
  – MDA approaches can very easily become obscure and counter productive to developers and managers

**Rule 2**: Have a pragmatic iterative approach of implementing MDA, based on well identified ROI, well accepted approaches, already existing solutions and standards

**Rule 3**: Having an efficient and flexible toolset is crucially important
Use UML and UML profiles

- Developing new metamodels for new models is highly risky for any organization
- Using UML profiles guarantees that the adapted models will still be legal UML models
- Standard solutions exist for some PSM’s or PIMS (examples: CORBA, EJB)
- Some very important targets have not yet standards, but on the shelf solutions (RDB, C++, etc.)
UML for a specific purpose
= a specific profile

- Profiles exist for technical targets (C++, RDB, Java, CORBA, XML, etc.)
- Profiles exist for specific disciplines (Analysis, design, etc.)
- Profiles exist for requirement analysis
- Profiles exist for tests (test for Java, test for EJB, test for C++, etc.)
- There are even profiles for metrics, quality checking, and profiles for managing configuration management rules.
- From early requirement down to final tests, UML profiles can drive each specific discipline involved.
Model transformation techniques

- Model transformation is central to MDA
- Model transformation should be specified and implemented in synch with profiles: Profiles should have a language for specifying transformations
- Example: The Objecteering/UML profile builder environment has a dedicated language called « J » - Java like syntax, navigation close to OCL, structured by profiles, profile & metaclass polymorphism
- An RFP for a transformation language is underway in the OMG …
Types of Model transformation

• Model->code transformations
• Model->Model transient transformations
• Internal model persistent transformations
• Transformation between two different models
Model -> code transformation

- The model is close to a technical target (PSM), and the target has frequently a syntax based interface.
- A UML profile maps the Model’s notions to target specific features by specializing or filtering out model elements.
- There is frequently no metamodel for the target (C++ metamodel?)
- Transformation = set of rules expressing how each model construct is translated into a specific syntactic construct.

- In addition, model checking rules, wizards specific to the target are important for driving the development.
Model->Model transient transformation

- No human interaction needed
- Well known, well accepted and relatively « low level » patterns
- For PSM’s mainly

**Example**: transforming an association into accessor operations
Internal model transformation

• Enhancement of the same model
• Generally PIM to PIM, or PSM to PSM
• Uncompleted transformation, necessitating human intervention
• Can be expressed by metamodel mapping techniques, or using transformation rules
• Transformation transaction is an important notion
Internal Model transformations: Example

```mermaid
classDiagram
    Project "create()" -\|-- delProject() -\|-- rollbackPhase() -\|-- deliverProject() -\|-- changeState() -\|-- initStates() -\|-- changeState() -\|-- getCurrentState() -\|-- setCurrentState() -\|-- notifyChangeState() -\|-- addProjectManagementListener() -\|-- removeProjectManagementListener()
    ManhoursSpent "total: integer" -\|-- create() -\|-- increment() -\|-- decrement()
    +ProjectManagementListeners
    ProjectManagementListener
    ProjectManagementEvent
    ProjectState "currentState" *\|-- ProjectManagementListener
    ProjectState "currentState" --> ProjectManagementEvent
    Delegation
```

UML WorkShop 2001 - SOFTEAM
Transformation between different models

- New independent model elements are created from old one
- Frequently for PIM to PSM’s transformations
- Traceability management is crucial

### Logical model

<table>
<thead>
<tr>
<th>A</th>
<th>leA</th>
<th>leB</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ {persistence(persistent)} ⊟</td>
<td>0..1</td>
<td>0..1</td>
</tr>
<tr>
<td>pkA : integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Physical model

<table>
<thead>
<tr>
<th>&lt;&lt;table&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌊ ≤ ⌋</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ {persistence(persistent)} ⊟</td>
<td></td>
</tr>
<tr>
<td>pkB : integer</td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
</tr>
<tr>
<td>leA</td>
<td>leB</td>
</tr>
<tr>
<td>0..1</td>
<td>0..1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ pkA : integer ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ leA of B_FK ⊟</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;&lt;table&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌊ ≤ ⌋</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ pkB : integer ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
</tr>
<tr>
<td>leB of A_FK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ pkA : integer ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ leA of B_FK ⊟</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;&lt;table&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌊ ≤ ⌋</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊟ pkB : integer ⊟</td>
<td></td>
</tr>
<tr>
<td>⊟ {primaryKey(1)} ⊟</td>
<td></td>
</tr>
<tr>
<td>leB of A_FK</td>
<td></td>
</tr>
</tbody>
</table>
Traceability – a key issue to model transformations and MDA

- Should be automatically carried out, or the entire MDA approach falls apart
- Necessitates an identification mechanism for Model elements (not Names!)
- Intrinsic model traceability (should be supported by all respectable case tools)
- Model/Model traceability: example between a PIM and a PSM
- Model/implementation or external element traceability
Industrializing UML profiles: MDA components

UML profiles packaged in the “MDA component”, including J code

Resources (help, icons, external executables, …)

MDA components should be extensible

External interface (for the UML modeler, for the end user, …)
Conclusion

• PIM’s and PSM should be based on standard models (UML and standard profiles)
• Profiles, supported by an appropriate toolset are an important technology for tooling UML based MDA
• A transformation language linked to the profile technology is necessary
• Traceability management is key
• On the shelf, parameterizable MDA components are necessary