Model-centric Tools
Reminder: Model-Centric Tool

- Requirements Analysis
- Documentation
- Diagram Manipulation
- Project Management
- Version Tracking
- "Groupware"
- Code Editing
- Packaging
- Quality Control
Model-centric Tools

Requirements management tools

- use of generic tools for requirements capturing and analysis
- requirements management tools

Diagrammatic tools

- creation and representation of high level models through diagrams
- transitions based on a software model

Also covered here: project management tools

- project planning and controlling
- tools for project status visualization and overview
- ..
Requirements Engineering (1)

*Generic* tools:
- text processing (descriptions, contracts, glossaries)
- graphics (see, for example, use case diagrams)

Problems with text and graphic tools:
- relating analysis documents, e.g.,
  - use cases diagrams visualizing textual descriptions and
  - glossaries explaining entities found on diagrams
- supporting traceability
- representing non-functional requirements for implementation decisions (interfaces, queries, ...)
- specifying test cases from requirement descriptions
- capturing relationships between phases
Requirements Engineering (2)

*Specific* requirements engineering tools

- Emphasis on documents
  - managing requirements through a set of individual documents
  - example: IBM Rational RequisitePro

- Emphasis on relationships
  - defining relationships between analysis documents
  - example: Telelogic DOOR
IBM Rational RequisitePro

A requirements management tool for the communication of project goals and collaborative development in project teams.

Properties:

- integration with Microsoft Word
- database with Word document synchronization
- definable requirement, attribute, document types; queries and filters.
- notification about changes of requirements
- traceability views for parent/child relationships
- integration with IBM software development tools
Example

reflect the evolving business needs. A suspect link (red slashed arrow in Figure 9) indicates that use case UC1.2 may need to be revisited due to a change in business need BUS1.4. Querying on suspect links answers the last question presented at the beginning of the article: Are my use cases staying in touch with the evolving business needs they are supposed to solve?

Figure 9: Traceability relationships between use cases and business needs
Telelogic DOOR

Captures, traces, and manages information to support compliance with specified requirements

Properties:
- insertion of data from various sources into documents
- multi-user environment (editable documents, sections)
- traceability over user-defined relationships, e.g.,
  - requirements to design
  - design to code
  - requirements for test
  - requirements for tasks
- generation of link reports across as many levels as required; link display in the same view
- customizable views
Telelogic DOOR (2)
Diagrammatic Tools

Based on a visual language; most often UML

Levels of support

- free drawing
- shape-based drawing
- shape-based drawing with (semantic) rules
- modeling tools

- manipulation of *one* underlying software model through *different* graphical representations
- thus, diagrams are interrelated
- model amendments (e.g., application of patterns)
- code generation and reengineering
Basic Functionality of Diagramatic Tools

Create and develop diagrams ...

- drawing capabilities of traditional drawing tools combined with
- diagram manipulation techniques that facilitate creation of diagrams in your language (e.g. UML)

... which visualize results of different phases

- requirements analysis
- design phase

(Non-) Interoperability

- Created diagrams need to be in a format that is understood by other tools (XMI)
- This is not a “nice-to-have” feature but essential to enable transitions between phases.
Model Verification

Modeling tools usually offer limited support for model checking

- mainly covering language syntax
- semantics are checked to a limited extent
- based on, e.g., UML meta model

Reasons:

- Languages like the UML have weak semantics.
- The intention of models cannot be checked automatically.
Model - Code Synchronization

code only, no model  code visualization  model centric  roundtrip engineering  model only, no code

Model  Model  Model  Model

Code  Code  Code  Code

wild hacking  “the code is the model”  model becomes the code  “synchronized” coexistence of code and model  specification
Transitions: Diagram Relationships

At the end of each development phase, you transform your artifacts to suit the next phase.

This can be done:

- manually
- semi-automatically

Model-centric tools usually offer some support for this.

Keep in mind that a full transformation is not possible unless you specify all implementational details in the model.
Transition: Semi-automatic Code Generation

Example of a semi-automatic transition: Many UML tools can generate skeleton code based on your diagrams

Generation of partial code based on model:

- simplest case: class diagram + annotations
- *Together*: diagrams in lock-step with code (missing semantics of code are expressed as special comments)
- perspective: diagrams + constraints $\Rightarrow$ MDA

Observation: impedance mismatch between the artifacts of the two phases

- this makes roundtripping difficult
Model-Driven Architecture

Models have well-defined notation and semantics
- cornerstone to understanding large systems.
System building is organized around the models
- series of transformations between models,
- organized into an architectural framework of layers and transformations.

Formal metamodels
- facilitates meaningful integration and transformation among models,
- are the basis for automation through tools.
Acceptance will require cross-vendor openness
- Exchanging models between tools is vital.
Model-Driven Architecture: Methodology

Three related software models:

- **Computation-Independent Model (CIM)**: domain-specific that is expressed in the vocabulary of the domain practitioner.
- **Platform-Independent Model (PIM)**: Often targets a platform-neutral virtual machine (composed of specific objects and services).
- **Platform-Specific Model (PSM)**: Specifies how the PIM will be mapped to the target platform.

![Diagram of MDA process](image-url)
MDA Example: CIM

Excerpt from a domain specific model:

- There are lectures.
- Professors teach lectures.
- Professors have names.

Comment: UML is probably not used by domain experts.
MDA Example: CIM (2)

For an example we use the **Concept-oriented Content Management** platform developed at STS

- based on the concept of assets
- further details skipped, this is only an example

Sample CIM in Asset Definition Language (ADL):

```cim
University

class Professor {
  concept
    characteristic name
    relationship lectures :Lecture*
}

class Lecture
```
MDA Example: PIM

Platform independent model:

- the names of professors are implemented by string objects,
- the names are unique,
- lectures have a title given by a string object,
- lectures are taught by exactly one professor, and
- there should be a navigational path from a lecture to the teaching professor.
MDA Example: PIM (2)

In ADL:

```adl
class Professor {
    concept
        characteristic name : String
        relationship lectures : Lecture*
            ; relationship "lectures" can be inherited from cim
    constraint uniqueName {
        let p : Professor, p = self or p.name # name
    }
}

class Lecture {
    concept
        characteristic title : String
        relationship teacher : Professor
    constraint uniqueTeacher {
        let p1 : Professor, let p2 : Professor,
        p1 = p2 or
        not ({self} <= p1.lectures and {self} <= p2.lectures)
    }
```
MDA Example: PSM

The platform independent model can then be transformed (automatically) into a model tailored to a specific platform.

To Java in ADL:

```java
psm[Java] University
from Java import java.lang.String
from Java-API[university] import
    university.LectureIterator,
    university.AbstractProfessor

class Professor {
    concept
    characteristic name : java.lang.String
    relationship lectures : LectureIterator
}
class Lecture {
    concept
    characteristic title : java.lang.String
    relationship teacher : AbstractProfessor

```
AbstractMutableProfessor

AbstractLecture

MutableLecture

NewLecture

platform dependent model (assets):

class Professor {
  concept relationship l :Lecture*
}
class Lecture {
  concept relationship t :Professor
}
MDA Example: PSM (3)

There can be more than one PSM, depending on the target system.
For SQL in ADL:

```python
psm[SQL] University
from SQL import TABLE,VARCHAR
class Professor {
    concept
        characteristic name :VARCHAR(100)
        relationship lectures :TABLE(…)
    constraint uniqueNames PRIMARY KEY name
}
class Lecture {
    concept
        characteristic title :VARCHAR(200)
        relationship teacher :VARCHAR(100)
    constraint fk_teacher FOREIGN KEY Professor(name)
}
```
Importance of Context

Business System Developer

PIM

PSM

Business Model
Analysis Model
EJB Design Model
App Server Model
Operating System Model
Microcode Model
Hardware Model
Chip Model
Silicon Block Model
Shipping Model
Excavator Model
Mining Model

Middleware Developer
Mapping

A “Mapping” is a set of (automated) rules & techniques

Models must be “marked up” in some way to help direct mapping

Add code – needs round-trip support

Code

PSM→Code mapping

PSM→PSM mapping

PIM→PSM mapping

Platform Specific Design Model

PIM→PIM mapping

Platform Independent Analysis Model

Computation Independent Business Model

CIM→PIM mapping
AndroMDA view

AndroMDA is an extensible generator framework. Takes UML from tools as input (XMI). Generates deployable components based on mapping definitions. You can also write your own mappings (cartridges) to support your own architecture or framework.
Platform/Runtime bindings

Architecture concept/UML-Profile

- Presentation
- ActivityController
- ValueObject
- ProcessObject
- EntityObject

Platform binding/Generat

- Struts-ActionForm
- Struts-Action
- Java-Structure
- BusinessInterface
- EJB-EntityBean

Runtime/Platform

- Struts-Config
- BusinessDelegate
- EJB-SessionBean
- CMP 2.0 Mapping

- JSP
- Struts
- BaseClass
- JBOSS
- HSQL DB
- Tomcat
- JBOSS
Diagram Generation from Code

Code visualization, e.g.
- class diagrams from data models
- sequence diagrams from code

Needed for reengineering
- missing design documents are recreated ("invented")
- e.g., to work with foreign code

Sometimes also used to implement roundtrip engineering
- if different tools are used for design and coding
- if one tool is used which is not capable of handling roundtrips
- if design documentation is unavailable.
Diagram Generation from Code (2)

Common problems:

- Generated diagrams reflect physical code. Diagram generation does not recreate the exact same (conceptual) diagrams you created during design.
  - decisions made during transition from design to implementation are not reversible
  - patterns cannot be recognized in general

- Semantic mismatch between modeling and programming languages, e.g.,
  - no distinction between general associations and aggregations in Java
  - distinction of aggregation and composition is buried in code (remember: the distinction is a lifetime issue!)
Reminder: Project Management Tools

- Time, Budget, ... Resource Management
- Version Tracking
- Requirements Analysis
- “Groupware”
- Documentation
- Diagram Manipulation
- Quality Control
- Code Editing
- Packaging
Project Management Tools

A Software Manager’s Responsibilities:

☐ Proposal writing
☐ Project budgeting
☐ Project planning and scheduling
☐ Project monitoring and reviewing
☐ Personnel selection and evaluation
☐ Report writing and presentation

(Semi-) Formal Means:

☐ Gantt diagrams (see MS Project)
☐ Network analysis (e.g., PERT chart)
Activity Organization

**Milestone:** End-point of activity.

**Deliverable:** milestone delivered to customer.

Coding 80 % complete.

This is **not** a milestone because it cannot be decided whether it is reached.
Task Duration and Dependencies

<table>
<thead>
<tr>
<th>Task/Activity</th>
<th>Duration (days)</th>
<th>depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>15</td>
<td>T1, T2</td>
</tr>
<tr>
<td>T4</td>
<td>5</td>
<td>T2</td>
</tr>
<tr>
<td>T5</td>
<td>15</td>
<td>T3, T4</td>
</tr>
<tr>
<td>T6</td>
<td>20</td>
<td>T1, T2</td>
</tr>
</tbody>
</table>

Typical duration of tasks/activities: 1-10 weeks

- finer grained (days): increases time needed for reviewing and re-scheduling
- coarser grained (months): decreases control and delay detection
Activity Network / PERT Chart

1. Activity contributes to milestone.
2. Milestone with delivery date.
3. Milestone is built from several activities.
4. Task/activity with duration.
5. Activity contributes to several milestones.
6. Milestone is used in activity.

**Critical path:** any delay here causes delay of the project (no time buffer).
PERT Chart

Sophisticated variant of activity networks
Distinguishes pessimistic, likely, and optimistic durations or dates
Leading to many potential critical paths
Critical path analysis only with tool-support

(optimistic, likely, pessimistic)
Activity Bar Chart / Gantt chart

planned task duration

possible delays without affecting the project schedule
Staff Allocation

Include time for
- holiday,
- training courses, and
- other projects.

Be prepared to handle delays:
- people may be unavailable and
- specialists cannot be replaced on short notice.
A Family Routine

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Predecessor</th>
<th>Resource Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>alarm goes off</td>
<td>5 mins</td>
<td>dad</td>
</tr>
<tr>
<td>2</td>
<td>wake family</td>
<td>5 mins</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>mum shower</td>
<td>40 mins</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>son shower</td>
<td>30 mins</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>walk dog</td>
<td>10 mins</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>father shower</td>
<td>15 mins</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>prepare breakfast</td>
<td>15 mins</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>eat breakfast</td>
<td>15 mins</td>
<td>8,4,6</td>
</tr>
<tr>
<td>9</td>
<td>load car</td>
<td>5 mins</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>drive to game</td>
<td>25 mins</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>arrive at game</td>
<td>0 mins</td>
<td>11</td>
</tr>
</tbody>
</table>
Q1. What tasks are on the critical path?
Q2. What is the *minimum* time it would take for the family to reach the footy game after getting the alarm goes off?
Q3. How much more time could dad walk the dog before eating breakfast got delayed? (Note: Mum insists the entire family eats together)
Q4. What is this amount of time called?
Q5. If mum skipped her 40 minutes shower, how much earlier would they get to the game?
What is the critical path?

Path 1 = 5+5+40+15+15+5+25 = 110 min
Path 2 = 5+5+30+15+5+25 = 85
Path 3 = 5+5+10+15+15+5+25 = 80

The critical path is the longest path: path 1
Q2

What is the minimum time it would take for the family to reach the footy game after the alarm starts ringing?

The duration of the critical tasks... 110 minutes
How much more time could dad walk the dog before eating breakfast got delayed?

30 minutes...

Shower + Prep Brekky = 55 min vs Walk Dog + Dad Shower = 25 min ... 30 min diff

What is this amount of time called?

Slack time (or float)
Q5

If mum skipped her 40 minute shower, how much earlier would they get to the game?

When the critical path is reduced by 40 minutes, it stops being the critical path. Path 2, at 85 min, becomes the critical path. Since it is 25 min shorter than the original 110 minute critical path, there is a 25 minute saving.
References

Software - UML Tools:
- Together
- Rational Rose
- Gentleware Poseidon / ArgoUML
- Omundo
- Microsoft Visio
- Magicdraw UML

Books
- Three Amigos; three books
- Fowler: UML distilled

Articles
References (2)

Software - MDA Tools:
- ArchStyler
- IBM XDE
- OptimalJ
- AndroMDA

Articles MDA:
- Introduction to MDA
- OMG's MDA site