Lecture
Process Modeling
Chapter 4: Process Modeling Languages

Dr. Jens Heidrich
Summer Term 2011

Goals of Chapter 4

- What languages to use for modeling software development processes?
- How to select the “right” modeling language?
- What are the advantages and disadvantage of process modeling languages?
Process Modeling Languages

- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPEM)
- Others
- Summary

**Overview**

- Process Modeling Languages (PMLs) came up in the mid-1980s (Osterweil’s “Software Processes are Software too”)

- Formalization of the software development process enables
  - Precise specification of software development processes
  - Tool-supported editing and maintenance of process models and generation of process descriptions
  - Automation of software development processes (e.g., for process engines or process-sensitive software development environments)
  - Standardization of software process descriptions
  - Common understanding of process models / process descriptions
  - Instrumentation of software development processes with measures
  - ...
Evaluation Schema for PMLs

- Natural Representation
  - Is the mapping of real-world objects on language concepts intuitive and easy for developers?
  - Is the schema defined by PML complete?
- Support of Quality Management
  - How are process and products attributes defined?
  - Can quality models be integrated?
- Model Adaptability
  - Which constructs for defining process variability exist?
  - Which constructs supporting process tailoring exist?
  - How are process models adapted?
- Formality
  - What is the formality level of the language?
  - How is the (enactment) semantic defined?
- Comprehensibility
  - Is the representation of the model suitable for the actor?
  - Is it easy to extract information from the process model?
- Execution & Enactment
  - Is the language interpretable?
  - Does operation system concepts exist in the language?
- Flexibility
  - Can process models be changed ‘on the fly’?
  - Can states be changed manually?
- Traceability
  - Is the context of a process easy to analyze?
  - Is it easy to navigate through the model?

Multi-View Process Modeling Language (MVP-L)

- 1st version (1989) at NASA Goddard Space Flight Center and University of Maryland by Dieter Rombach
- 2 subsequent versions at University of Kaiserslautern
- Supports process improvement by
  - Modeling
  - Planning
  - Enactment
  - Measurement and interpretation
  - Packaging
- Supports process modeling “in the large”
  - Clear interfaces
  - Level completeness
Process Modeling Languages
- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPEM)
- Others
- Summary

Language Constructs
- Models (Types) for description of
  - Artifacts, e.g., documents (product_model)
  - Activities, e.g., review of a document (process_model)
  - Resources, e.g., people, tools (resource_model)
  - Quality aspects, e.g., effort prediction model (quality_model)
- Attributes
  - Refer to each model type
  - Attributes correspond to measures
  - Attribute values correspond to measurement data
- Models (types) are adapted to actual contexts through the instantiation to objects
- Executable objects are assembled in project plans (project_plan)

Relationships between Objects
- Product flow
  - consume, produce, consume_produce
- Control flow
  - implicit via pre- and post-conditions and invariants (i.e., constraint-oriented description of control flows)
  - Constructs: entry_criteria, exit_criteria, and invariant
- Refinement / aggregation
- Information hiding
  - Models are based on
    - Model-Interface (visible for other models) and
    - Model-Body (only visible internally)
  - Models should be designed in such a way that potential changes are local
**MVP-L – Definition of a Project Plan**

**Definition:** A project plan in MVP-L is a 13-tuple including the following components:

1. **PC:** Set of all processes
2. **PD:** Set of all products
3. **RS:** Set of all resources
4. **ATT:** Set of all attributes
5. **BE:** Set of all Boolean expressions
6. **consume** \(PD \times PC\)
7. **produce** \(PC \times PD\)
8. **consume_produce** \(PD \times PC\)
9. **attribute** \((PC \cup PD \cup RS) \times ATT\)
10. **entry** \(PC \times BE\)
11. **exit** \(PC \times BE\)
12. **resource** \(PC \times RS\)
13. **S0** \(ATT \rightarrow Value\)

**Graphical Representation**

![Diagram of Process Modeling Languages](image)

- **Products**
- **Processes**
- **Resources**
- **Entry / Exit Criteria**
- **consume**
- **produce**
- **produce/consume**
Process Modeling Languages

- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPEM)
- Others

Summary

Extract of a MVP-L-Process Model

process_model E14_Write_Specifications () is
process_interface
<...>
effort : Process_effort := 0;
product_flow
consume
externRe: External_References_Container;
prorep: Project_Reports_Container;
worpap: Working_Papers_Container;
<...>
produce
consume Produce

mv: Mission_Volume;
fs: Functional_Spec;
<...>
entry_exit_criteria
local_entry_criteria
(externRe.status = 'complete' and worpap.status =
'complete' and <...>);
local_exit_criteria
(mv.status = 'complete' and fs.status = 'complete') and
<...>;
end process_interface

Product Interface and Criteria

process_interface
<...>
product_flow
consume

T2_SSM: T2_Software_Specification_Files;
T4_SCF: T4_Software_Certification_Files;
T7_PRS: T7_Pre_Release_Software;
T8_FRE: T8_Failure_Rep_ECN;
consume Produce
T1_PDF: T1_Project_Document_Files;
T3_SDF: T3_Software_Development_Files;
T5_PMF: T5_Project_Management_Files;
T6_QC: T6_Questions_Container;
context
entry_exit_criteria
local_entry_criteria
(T1_PDF.status = 'complete' and T2_SSM.status = nonexistent'
and T4_SCF.status = 'nonexistent' and T5_PMF.status =
'complete');
global_entry_criteria
local_exit_criteria
(T1_PDF.status = 'complete' and T2_SSM.status = 'complete'
and T4_SCF.status = 'complete') and global_exit_criteria
(T2_SSM.status = 'complete' and T3_SDF.status = 'complete'
and T4_SCF.status = 'complete' and T5_PMF.status =
'complete' and T7_PRS.status = 'complete');
global_exit_criteria
end process_interface
Semantics

- Basic concepts
  - All processes are independent
  - Each product / each process has a state (defined by the values of all its attributes)
  - Each project plan has a state (defined by the states of all the products’ processes)
  - State transitions are fired by external events:
    - Initiation of a process (start)
    - Completion of a process (complete)
    - “Enforced” state change (set)

- Instantiation of a project plan
  - An initial state is defined by
    - Binding formal interface parameters to actual objects
    - Specification of default parameters

- Execution (Process machine MVP-S)
  - State transitions are defined by state transition graphs in the attribute models
  - External events are ordered by the process machine

Execution Semantics: Transition Graph Product States

non_existent
producing process starts

producing process terminates

incomplete

complete

error is found in the document
Execution Semantics: Transition Graph Process States

Process models
- cd : Create Design ("cd" is instance of Create Design)
- cc : Create Code

Product models
- rs : Requirements Specification
- dd : Design Document
- c : Code

Strict sequential process
Control by pre- and post-conditions of the processes
Process in MVP-L: Waterfall

MVP-L – Execution Semantics: Transition Table for Waterfall

User Events:
MVP-L – Execution Semantics: Transition Table for Waterfall

<table>
<thead>
<tr>
<th>Status</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>rs complete</td>
<td>complete</td>
<td>complete</td>
<td>complete</td>
<td>complete</td>
</tr>
<tr>
<td>dd non existent</td>
<td>incomplete</td>
<td>complete</td>
<td>complete</td>
<td>complete</td>
<td>complete</td>
</tr>
<tr>
<td>c non existent</td>
<td>non existent</td>
<td>non existent</td>
<td>incomplete</td>
<td>complete</td>
<td>complete</td>
</tr>
<tr>
<td>Process</td>
<td>cd enabled</td>
<td>active</td>
<td>disabled</td>
<td>disabled</td>
<td>disabled</td>
</tr>
<tr>
<td>cc disabled</td>
<td>disabled</td>
<td>enabled</td>
<td>active</td>
<td>disabled</td>
<td>disabled</td>
</tr>
</tbody>
</table>

User Events:
- cd.start
- cd.complete
- cc.start
- cc.complete

Process in MVP-L: Waterfall with Iterations for Rework Activity

- Req.status = complete ∧ Design.status = (non existent ∨ incomplete)
- Design.status = complete ∧ Code.status = (non existent ∨ incomplete)
- Code.status = complete

Read "Design.status = (non existent ∨ incomplete)"
as "(Design.status = non existent) ∨ (Design.status = incomplete)"
Process in MVP-L: Parallelized Process Steps

- Req.status = complete ∧ Design.status = non existent
- Design.status = complete
- Design.status = (complete ∨ incomplete) ∧ Code.status = non existent
- Code.status = complete

Process in MVP-L: Integrated Quality Model

- Req.status = complete ∧ Design.status = non existent
- Create Design.effort <= 7500 ph
- Design.status = complete ∧ Code.status = non existent
- Code.status = complete ∧ Create Code.effort <= 2500 ph

Effort Distribution
Goal: at most 10000 person-hours

75% Create Design
35% Create Code
Refinements in MVP-L

- Interfaces of objects are mapped onto each other
- Computation rules for the determination of values for abstract objects (processes, products, resources)
  - “bottom-up”
  - arithmetic and logic expressions
- Alternative refinements can be specified (operator "|")
- Multiple instantiations (operator "**")

MVP-L: Extract Process Model (Refinement)

```plaintext
process_body
refinement
imports
product_model High_level_design_document, Low_level_design_document,
One_step_design_document;
process_model High_level_design, Low_level_design, One_step_design;
objects
hl_des_doc: High_level_design_document('non_existent');
ll_des_doc: Low_level_design_document('non_existent');
os_des_doc: One_step_design_document('non_existent');
hld: High_level_design(0, max_effort_0 / 2);
lld: Low_level_design(0, max_effort_0 / 2);
osd: One_step_design(0, max_effort_0);
object_relations
((hld & lld) | osd);
interface_refinement
des_doc = ((hl_des_doc & ll_des_doc) | os_des_doc);
interface_relations
hld(hl_des_doc => des_doc.hl_des_doc, req_doc => req_doc);
lld(ll_des_doc => des_doc.ll_des_doc, hl_des_doc =>
des_doc.hl_des_doc);
osd(os_des_doc => des_doc.os_des_doc, req_doc => req_doc);
attribute_mappings
effort := hld.effort + lld.effort;
max_effort := hld.max_effort + lld.max_effort;
end process_body
```
Refinement Structures

Problem Description

Work on System Requirements
- Work on User Requirements
- Work on Acceptance TCs
- Work on Dev. Requirements
- Work on Traceability Matrix for Requirements
- Work on System TCs

\[ \text{pb.status} = \text{complete} \land \text{psd.status} = (\text{non existent} \lor \text{incomplete}) \]
\[ \text{psd.status} = \text{complete} \land \text{time} \leq \text{target} \land \text{effort} \leq \text{target} \land \text{reliability} \geq \text{target} \]

Industrial Applications

- Projects
  - Process improvement (e.g., telecommunications industry)
  - Improvement of project manuals (e.g., TRW)
  - Review of standards (e.g., V-Modell 97)
- Experiences
  - Graphical representation is necessary
  - Questions of generic processes still not clear
  - Multiple-instance problem difficult to model
  - Modeling of typical control flows would be helpful for understanding
Discussion (1/2)

- Documentation of existing processes
  (process identification and process modeling)
  - Uniform external representation
  - Supports the view-concept
  - Methods for descriptive modeling

- Integrating aspects of processes into project plans
  - Traceability between project aims/characteristics and elements of the project plan

- Reuse of knowledge gained in processes
  - Modularization of knowledge upon models and views
  - Support for packaging / storage / search and changeability

- Analysis of project plans
  - Existence of static algorithms of analysis
  - Dynamical algorithms of analysis

Discussion (2/2)

- Guidance for project teams
  - Execution machine
  - Feedback based on the state of the project
    (synchronous/asynchronous)

- Documentation of project execution
  - Simulator for SW projects
  - Use of project traces for examinations of causes for defects

- Support for changes during a project run
  - Change of goals (e.g. expenditure for the whole project)
  - Refinement of processes
  - Replacing/adding of processes
Evaluation of MVP-L

<table>
<thead>
<tr>
<th>MVP-L</th>
<th>Natural Representation</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Support of Quality Management</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Model Adaptability</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Formality</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Understandability</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Execution &amp; Enactment</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Flexibility</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Traceability</td>
<td>O</td>
</tr>
</tbody>
</table>

Software & Systems Process Engineering Meta-Model (SPEM)

- Original abbreviation for: Software Process Engineering Model
- Released 2005 by the Object Management Group (OMG)
  - Based on UML notation
  - Uses parts of UML meta-model
  - Defined as a subset of UML
- Version 2.0 from 2008:
  Software & Systems Process Engineering Meta-Model
- Meta-model for description of software development processes (or a family of processes) and their components
- SPEM 2.0
  - Clear separation of concerns
  - Re-usability
    - Method, process plug-in
    - Process pattern application
    - Process components
The SPEM 2.0 Meta-Model

- **Core**: Common classes and abstractions that build the base in all other meta-model packages
- **Process Structure**: Base for all process models (static structure)
  - Breakdown of nested Activities
  - Lists to performing Role classes as well as input and output Work Product classes
  - Provides mechanisms for process reuse (such as process patterns)
- **Process Behavior**: Defines connections to behavioral models (dynamic structure)
- **Managed Content**: Manages the textual content of process descriptions (linking to the structure)
- **Method Content**: Documented knowledge of software development methods, techniques, and best practices (independent from process)
- **Process With Methods**: Integrates processes with method contents
- **Method Plugin**: Manages reusable and configurable libraries of method content and processes

---

SPEM 2.0's Conceptual Usage Framework

- Standardize representation and manage libraries of reusable Method Content
- Develop and manage Processes for performing projects
  - Process assets patterns
  - Standard or reference processes
  - Enactable project plan templates
- Configure a cohesive process framework customized for my project needs
- Create project plan templates for Enactment of process in the context of my project
Method Content versus Process

- Method Content
  - Work Definition
  - Role Definition
  - Task Definition
  - Category

- Process
  - Task Use
  - Role Use
  - Work Product Use
  - Activity
  - Process

Self-contained description of a method/technique that can be used in different processes

Description of a temporal sequence of methods

[From SPEM 2.0 Report]

Two Processes with Different Lifecycle Models

- GMS Method Incremental Process
- RUP Iterative Process

One data structure that enables the application of a number of development lifecycle types

[From SPEM 2.0 Report]
Example for a Method Plug-in

A Process Pattern

[From SPEM 2.0 Report]
Process Modeling
Languages
- Motivation
- Multi-View Process
  Modeling Language
  (MVP-L)
- Software Process
  Engineering Meta-
  Model (SPEM)
- Others
- Summary

Process Components Connected via Work Product Ports

[From SPEM 2.0 Report]

UML 2 Diagrams using SPEM 2.0 Profile

[From SPEM 2.0 Report]
Key Elements of the Process Structure Package

Key Elements of the Process Behavior Package
Process Modeling Languages
- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPEM)
- Others
- Summary

Process Enactment with Project Planning Systems
- SPEM 2.0 supports process enactment with planning and resource management tools
  - IBM Rational Portfolio Manager
  - Microsoft Project
- Processes are mapped to project plans by instantiating the process model
  - A planner has to substitute the types with concrete instances
  - Work products for work product type declarations
  - Resources for roles
  - If a task has multiple instances, it is simply replicated for the project plan
- SPEM 2.0 models are separated from behavior models
  - Enactment machines for behavior modeling approaches may be used
  - Activities may, e.g., be linked to BPMN diagrams

Mapping between Activity Diagram, Process, and Project Plan

[SPEM 2.0 Report]
### Evaluation of SPEM 2.0

<table>
<thead>
<tr>
<th>SPEM 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Representation</td>
</tr>
<tr>
<td>Support of Quality Management</td>
</tr>
<tr>
<td>Model Adaptability</td>
</tr>
<tr>
<td>Formality</td>
</tr>
<tr>
<td>Understandability</td>
</tr>
<tr>
<td>Execution &amp; Enactment</td>
</tr>
<tr>
<td>Flexibility</td>
</tr>
<tr>
<td>Traceability</td>
</tr>
</tbody>
</table>

### Little-JIL

- **Little-JIL**
  - Agent coordination language
  - Process divided into small units of work: steps
  - Steps can be assigned to agents
- **JIL vs. Little-JIL**
  - JIL: complex process language
    - Text-based
    - Based on ADA
    - Many complex functions
  - Little-JIL: coordination language
    - Subset of JIL
    - Only essential functions
    - Graphical representation for every function
Agents and Steps

- **Agent**
  - Autonomous entity
  - Can be human or automated
  - Each agent has one or more agendas with work (i.e., steps) assigned to it
  - When work is done, the agent has to report success or failure

- **Step**
  - Basic building block
  - Represents a unit of work
  - Can be decomposed into sub-steps
  - Every Little-JIL program is represented by its root step, which is decomposed to describe the process

States of a Step

- **Posted**
  - Step is posted to the agenda of the agent
- **Started**
  - Start is indicated by the agent
- **Completed**
  - Completion is indicated by the agent
- **Terminated**
  - Step failed to complete (e.g., after an exception was thrown)
- **Retracted**
  - Step failed to complete (e.g., after an exception was thrown)
- **Opted-Out**
  - Step is removed from agenda without being started (can be reposted to another agent)
Graphical Representation of a Step

- Step is annotated with badges
  - Every badge provides additional information or specifies the control flow
- Sub-steps are drawn below parent step
  - Connected with arcs to sequencing badge of parent step

**Step Badges**

- Sequencing badge
  - None:
    - Step is not decomposed
  - Sequential:
    - Sub-steps are executed from left to right, each one starting after previous sub-step is completed
  - Parallel:
    - Sub-steps are executed concurrently
  - Choice:
    - Only one of the sub-steps is executed
  - Try:
    - Try sub-steps from left to right until one sub-step succeeds

[From JIL 1.5 Report]
Step Badges

- **Interface badge**
  - Resource declaration
    - Defines resources used by the step
    - Products produced by other steps and agents can also be needed resources
  - Parameter declaration
    - Defines parameters used by the step
    - Types: In, Out, In/Out, Locals
    - For object exchange between parent step and sub-steps
  - Channel declaration
    - Specifies all channels the step can write into or read from
    - Communication between two arbitrary steps
  - Exception declaration
    - Specifies all exceptions that can be thrown by the step
    - Exceptions can be thrown to show that a process did not complete correctly
  - Message declaration
    - Defines all messages that can be sent by the step
    - Messages can be sent during execution of the step (automatically or by the agent)

- **Pre-/Post-requisite badge**
  - Define pre- and post-conditions for the execution of the step
  - If the step is started / terminated and the condition fails, an exception is thrown

- **Reactions badge**
  - Every reaction defines a step for a certain message
  - If the step is in state 'started' and the message is sent by some other step, the reaction step is executed immediately (parallel to other active steps)

- **Handler badge**
  - Every exception handler defines the types of exception he handles and the steps executed when the exception occurs
  - If an exception occurs and no handler is defined for this exception, the step is terminated and the exception is thrown to the parent step
Example Little-JIL: Simple Process

Example Little-JIL: Process with Exceptions and Messages

[From JIL 1.5 Report]

[From "Specifying Process Coordination Using Little-JIL", Wise et al.]
**Evaluation of Little-JIL**

- **Natural Representation**: +
- **Support of Quality Management**: -
- **Model Adaptability**: O
- **Formality**: +
- **Understandability**: +
- **Execution & Enactment**: O
- **Flexibility**: O
- **Traceability**: O

---

**Appl/A**

- Ada Process Programming Language with Aspen
- Arcadia project
  (first large DoD-sponsored project that aimed at the automation of development activities)
- Sutton, Heimbigner und Osterweil
  University of Colorado, Boulder
  Development during the period 1988-1990
Characteristics of Appl/A

- Processes
  - Process structure is flexible and dynamic
  - High degree of concurrency
  - Reactive control (event-based control)

- Product characteristics
  - Provides concepts for consistency and persistency

- Product flow
  - Allows for temporary inconsistencies

- Technical extensions to Ada
  - Relations (persistent data storage), trigger, predicates (constraints), transactions (composite statements)

- No domain-specific extensions

- Translation of the process program to Ada and execution using Ada runtime system ("APT" compiler)

Example: Appl/A - Relation

```ada
with Compile;
with Code_Types;

Relation Source_Compile is
  type src_compilations_tuple is tuple
    name : in name_type;
    src : in Code_Types.source_code;
    obj : out Code_Types.object_code;
    msgs : out messages;
  end tuple;

entries
  insert (name : name_type; src: source_code);
  delete (name : name_type; src : source_code; obj :
      object_code; msgs: messages);
  update (name : name_type; ...)
dependencies
  determine obj, msgs by compile (src, obj, msgs);
End Source_Compile;
```
Example: Appl/A - Constraint

```plaintext
-- a predicate on a property of an attribute
predicate Source_Length_Within_Limits is
    begin return
every t in Source_Repository satisfies
    Length (t.src) <= Max_Length;
end every;
End Source_Length_Within_Limits
```

Example: Appl/A

```plaintext
package Change_Engineering_Tasks is
    subtype engineering task_enum is change_task
    range
        Modify_Design, Review_Design, Modify_Code, Modify_Test_Plans, Modify_Unit_Test, Test_Unit;
    task type Modify_Design is
        entry start_up(design_end_id, manager_id : in emp_id_type; requirement_id : in req_id_type);
        entry completed;
        entry deactivate;
        end Modify_Design;
    type modify_design_a is access modify_design;
```
**Example: Appl/A**

```plaintext
procedure Reschedule_Tasks (c_task: in task_enum; mgr_id: emp_id_type) is
  -- Reschedule the given task and any dependent tasks
  Begin
    send_msg(mgr_id, "'Reschedule change process tasks' & 'beginning with task ' &
              task_enum'image(c_task));
    serial read Change_Process_Team,
               Task_Assignments, Task_Status,
               Task_Start_Schedule, Task_Order;
    begin
      mgr_assign_tasks(m_id);
      mgr_schedule_tasks(m_id);
      update_project_plans(m_id, subproj_id);
      notify_affected_personnel;
    end serial;
  End Reschedule_Tasks;
```

**Evaluation of Appl/A**

- **Natural Representation**: +
- **Support of Quality Management**: -
- **Model Adaptability**: Ø
- **Formality**: Ø
- **Understandability**: -
- **Execution & Enactment**: +
- **Flexibility**: -
- **Traceability**: -
**Funsoft Nets**

Workflow modeling for the development environment LDE (Lion Entwicklungsumgebung)

Gruhn and Deiters
University of Dortmund
End of 1980s

Research prototype based on Petri nets

Industrial evaluation

Also applied as workflow management system

---

**Workflow Modeling with Funsoft-Nets**

- Specification of process order
- Description of input- and output behavior of processes
- Structuring of complex work flows into manageable parts

Activity (Process) manual / automatic
[Agency]

Connection
[Edge]

Document storage
[Channel]

Document / Object
[Token]
Predicates

- Activity can be performed only when
  - There exists a document in every document storage and
  - A requirement (predicate) is fulfilled
- Predicates can specify the dependencies between values and
  chronological dependencies

Example of a Funsoft-Net

Process Modeling Languages
- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPCM)
- Others
- Summary

Example of a Funsoft-Net

Process Modeling Languages
- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPCM)
- Others
- Summary
Experiences with Funsoft-Nets

- Approach is applicable for real workflow applications
- Observing the net behavior via simulation improves the understanding
- Stochastic methods can be used for determining process parameters (e.g., optimal team size)
- High number of exceptions in real processes
- Changing the nets is a complex task

Evaluation of Funsoft Nets

<table>
<thead>
<tr>
<th>Funsoft Nets</th>
<th>Natural Representation</th>
<th>Support of Quality Management</th>
<th>Model Adaptability</th>
<th>Formality</th>
<th>Understandability</th>
<th>Execution &amp; Enactment</th>
<th>Flexibility</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O</td>
<td>–</td>
<td>O</td>
<td>+</td>
<td>O</td>
<td>+</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

© Heidrich, Münch, Verlage 5/30/2011 Slide 71
Marvel Strategy Language (MSL)

- Process programming language for the process-sensitive software development environment Marvel, implemented in C, lex and yacc
- Gail Kaiser, Naser Barghouti, George T. Heineman
  Columbia University
  Beginning of 1990s - 1994
- Further development into a distributed multi-user system (Successor “Oz”)
- Central concepts
  - Central repository (“engineering database”)
  - Rules

Support with the help of Marvel

- Developers have a “working environment”
  - Entities (i.e., products)
    - Defined by a data description language (structure is similar to the structure of variables in C)
  - Tools
    - Encapsulation of tools by envelopes → no changes when integrating new tools
  - Rules
    - Pre-condition
    - Activity
    - Post-condition
Example of a Rule in MSL

```
edit(\:DOCFILE):
    # if the file has been reserved,
    # you can go ahead and edit it
    {\:reservation_status = Checked_out}

    [ EDITOR edit ?f ]

    (and (?f.reformat_doc = Yes)
        {\:timestamp = CurrentTime});
```

Pre-condition

Activity

Post-condition

Analysis of Forward and Backward Chaining

Which rules must be performed in order to fulfill the precondition?

Which rules can be performed after completion of the currently processed rule?
**Evaluation of MSL**

Process Modeling Languages
- Motivation
- Multi-View Process Modeling Language (MVP-L)
- Software Process Engineering Meta-Model (SPEM)

Others
- Summary

<table>
<thead>
<tr>
<th>Natural Representation</th>
<th>Support of Quality Management</th>
<th>Model Adaptability</th>
<th>Formality</th>
<th>Understandability</th>
<th>Execution &amp; Enactment</th>
<th>Flexibility</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>〇</td>
<td>+</td>
<td>〇</td>
<td>+</td>
<td>+</td>
<td>〇</td>
</tr>
</tbody>
</table>

**Statemate**

- Tool for specification of reactive systems based on Harel’s work (e.g., State Charts)
- Uses as process modeling environment
  - Three perspectives on a process
    - activity charts (What is done?)
    - state charts (When and how is it done?)
    - module charts (Where and by whom is it done?)
  - Allows for checking completeness and consistency
  - Allows for simulation
    - qualitative, quantitative
    - “what if” games
Characteristics of Statemate

- Applicable for specification, design, and analysis
- Formal graphic language
- Abstraction mechanisms
- Automatic analysis and simulation
  - Consistency between views
  - Simulation of costs, project duration, and errors possible
- Suitable for
  - Process improvement
  - Process management

Statemate-Example: Activity Chart
Experiences with Statemate

- Relatively easy modeling of process designs
- The three suggested perspectives offer an adequate description of processes
- Discrete event simulation is sufficient for technical processes

Evaluation of Statemate

<table>
<thead>
<tr>
<th>Statemate</th>
<th>Natural Representation +</th>
<th>Support of Quality Management -</th>
<th>Model Adaptability -</th>
<th>Formality +</th>
<th>Understandability +</th>
<th>Execution &amp; Enactment +</th>
<th>Flexibility -</th>
<th>Traceability +</th>
</tr>
</thead>
</table>

Others

Summary
IDEF0

Integration Definition for Function Modeling (IDEF0)

Federal Information Processing Standards Publication 183

Initial document:

Function modeling
- Systems
- Company (Process modeling)
- Based on SADT (Structured Analysis and Design Technique)

Background of IDEF0

Project
- USAF ICAM (Integrated Computer Aided Manufacturing)
- 1975 - 1980
- IDEF = ICAM DEFinition Language

Three languages
- IDEF0 (What do I do?)
  "function model"
- IDEF 1-1x (What do I need to know to do what I do?)
  "information model"
- IDEF2 (When do I need to know what I need to know to do what I do?)
  "dynamics model"
Characteristics of IDEF0

- Simplicity of process documentation
- Advantages
  - Comprehensive graphical representation
  - Simple notation
  - Supports human communication
  - Widely applied in practice
    - Prescriptive models (especially organization-specific reference models)
  - Tool support

Syntactical Elements of IDEF0

- Boxes - Function
  - Function name (Verb)
  - Number

- Arrows - Data / Objects
  - Straight-line arrow segment
  - Curved-arrow segment (corners are rounded with 90 degree arcs)
  - Forking arrows
  - Rejoining arrows
Syntax Rules of IDEF0

- **Boxes**
  - Boxes shall be sufficient in size to insert box name
  - Boxes shall be rectangular in shape, with square corners
  - Boxes shall be drawn with solid lines

- **Arrows**
  - Arrows that bend shall be curved using only 90 degree arcs
  - Arrows shall be drawn in solid-line segments
  - Arrows shall be drawn vertically or horizontally, not diagonally
  - Arrow ends shall touch the outer perimeter of the function box and shall not cross over into the box
  - Arrows shall attach at box sides, not at corners

Connecting Arrows, Boxes and Names

Each side of a box has a specific meaning:

- a) Input: left side
- b) Control: upper side
- c) Output: right side
- d) Mechanisms: upwards, lower side
- e) Usage of a mechanism: downwards, lower side

A “squiggle” ( ) is used for connection of names to arrows when clear positioning is not possible.
Diagrams create a decomposition hierarchy!

### Evaluation of IDEF0

**Natural Representation**

**Support of Quality Management**

**Model Adaptability**

**Formality**

**Understandability**

**Execution & Enactment**

**Flexibility**

**Traceability**
ETVX

- Development by IBM [Radice] beginning of 1980s
- Notation for description of activities
  - Entry criteria – must be satisfied before a set of tasks can be performed
  - Tasks – set of tasks to be performed
  - Verification & validation - The means to determine that the tasks are completed properly
  - eXit criteria - criteria for task completion

Example ETVX: Size Estimation

- Top-level description

<table>
<thead>
<tr>
<th>Entry</th>
<th>Task</th>
<th>Validation / Verification</th>
<th>eXit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product structure is as detailed as technically possible</td>
<td>Precisely define standard of measurement</td>
<td>Compare to historical data of similar type / complexity</td>
<td>Estimate has been verified against historical data</td>
</tr>
<tr>
<td></td>
<td>Estimate size of each element</td>
<td>Sum size of each element</td>
<td>Calculations have been checked</td>
</tr>
<tr>
<td></td>
<td>Apply contingency factors</td>
<td></td>
<td>Estimate has been added to historical data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detailed size estimate produced</td>
</tr>
</tbody>
</table>
### Estimation ETVX

- **Advantages**
  - Intuitive understanding of tasks
  - Detailed enough for the implementation of processes for many purposes
  - Can be used for delegation of tasks (prescriptive models)

- **Disadvantages**
  - Becomes complex very fast
  - General flow difficult to understand

### Evaluation of ETVX

<table>
<thead>
<tr>
<th>ETVX</th>
<th>Natural Representation</th>
<th>Support of Quality Management</th>
<th>Model Adaptability</th>
<th>Formality</th>
<th>Understandability</th>
<th>Execution &amp; Enactment</th>
<th>Flexibility</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Natural Representation**:
  - +

- **Support of Quality Management**:
  - -

- **Model Adaptability**:
  - -

- **Formality**:
  - O

- **Understandability**:
  - +

- **Execution & Enactment**:
  - -

- **Flexibility**:
  - -

- **Traceability**:
  - O
Further Notations

- Software Process Modeling Notations
  - UML (e.g., Activity Charts)
  - Tool-specific notations

- Business Process Modeling Languages
  - ARIS: Notation for Business Processes, uses Event-driven Process Chains
  - BPML (Business Process Modeling Language)

Summary of Process Modeling Languages

<table>
<thead>
<tr>
<th>Process Modeling Languages</th>
<th>MVP-L</th>
<th>SPEM 2.0</th>
<th>APDL</th>
<th>Funsoft Nets</th>
<th>MSL</th>
<th>IDEF0</th>
<th>ETLX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Representation</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Support of Quality Management</td>
<td>+</td>
<td>O</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Model Adaptability</td>
<td>-</td>
<td>+</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Formality</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Understandability</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Execution &amp; Enactment</td>
<td>+</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Flexibility</td>
<td>+</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Traceability</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

© Heidrich, Münch, Verlage 5/30/2011 Slide 96
Future Topics

- Definition of a comprehensive and consistent terminology
- Relationships to languages in workflow management systems
- Concepts for process and product variability
- Concepts for process instantiation for project planning
- Concepts for re-planning during process enactment

Summary

- Several process modeling languages exist for prescriptive and descriptive modeling
- Different criteria have been introduced to assess process notations
- There is no universal process modeling language
- The suitability of a language essentially depends on the application purpose
- Concepts of abstraction must be available for descriptive modeling