Part IV. Megamodels in a Software Factory
40. Requirements and Test Megamodels

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de/teaching/most
Version 15-0.16, 23.01.16

1) Traceability and Megamodels
2) Requirements Management and Tracing in a Megamodel
3) Tracing Requirements and Testing
4) Tracing Goals and Requirements with ODRE
Literature

  - http://www.easst.org/eceasst/
References

  ▪ http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-155839
Q12: A Software Factory's Heart: the Multi-TS Megamodel

Software Factory

Multi-TS Megamodell

Technical Space

Method Engineering
Mega- and Macromodels
Model Management
Mapping, Transf., Composition
Model Analysis
Querying, Interpretation
Metapyramid (Metahierarchy) for Token Modeling

Technical Space Bridges

Pattern Languages

Technical Space

Method Engineering
Mega- and Macromodels
Model Management
Mapping, Transf., Composition
Model Analysis
Querying, Interpretation
Metapyramid (Metahierarchy) for Token Modeling
A **software factory** schema essentially defines a recipe for building members of a software product family.

Jack Greenfield
40.1 Traceability between Models
Why Traceability in a Megamodel?

System Comprehension:
- To improve orientation by navigating via trace links along model transformation chains

► Change Impact Analysis:
- to analyze the impact of a model change on other models
- to analyze the impact of a model change on existing *generated* or *transformed* output
- To enable to do model synchronization (hot updating dependent parts)

► Orphan Analysis: finding orphaned elements in models

Validation and Verification:
► System Validation: Connecting the requirements with the customer's goals and problems (see ZOPP method)
► (Test) Coverage analysis: to determine whether all requirements were covered by test cases in the development life cycle
► Debugging: To locate bugs when tracing code back to requirements
  - To locate bugs during the development of transformation programs
Traceability Metamodel: CRUD Types of Trace Links between Model Elements of Different Models

Derivation
- Classification of source-target relationships [CH06]
- CRUD Actions

Source-Target Relations

New Target Model
- Create Link

Existing Target Model
- Update Transformation
  - Destructive
  - Extension-Only

- In-Place Transformation
  - Destructive
  - Extension-Only
Extensible Traceability Metamodel acc. to Grammel

- New facets for new trace link types can be created

Diagram:
- Tracemodel
  - Model (to be traced)
    - Links 1..* 1..* 0..*
    - Source
    - Target
  - TraceLinkFacet
    - Configuration
      - Granularity
      - Scope
    - TraceLink
      - MonotonicLink
        - CreateLink
        - RetrieveLink
        - ChangesLink
        - UpdateLink
        - DeleteLink
        - ContainmentLink
Examples for TraceLinkFacet

- Facets factorize inheritance hierarchies; new facets extend inheritance hierarchies
Adding a Trace Link Generator to Tools

- TraceLinkGenerators can be connected in two ways, following a generic traceability interface:

  - **Black-box connector**: Transformation engine need not know but is extended invasively or by AOP.
  - **Invasive connector**: Transformation engine must know and call the generator.

![Diagram of TraceLinkGenerators connection](image)
Traceability in Megamodels

- Piecemeal growth of megamodels in the software process:
  - Start with requirements, then add more stuff and models

- Add links
  - **Create links** are drawn between model element MA from model A and model element MB whenever MB is generated or added because of MA
  - **Retrieve links** are drawn when MB is extracted from a model A and added to another model B
  - **Containment links** are drawn, when in a new model B the model element MA is contained in another model element MB'
  - **Delete links** are drawn if in model B the model element MB should be deleted
  - **Update links** are drawn if MA has changed and MB should be changed too
In link-tree models, a skeleton tree exists, in which every model element has a unique tree node number (hierarchical number).

Trace links can be added with tree node number and stored externally of the model in the megamodel.

In link-treeware, megamodels maintain *tracelink models* linking and tracing all models and their elements by referencing the hierarchical numbers of all nodes.
Q12: The ReDeCT Problem and its Macromodel

- The inter-model mappings between the Requirements, Design model, Code, Test cases are traceability links stemming for example from:
  - Lifted results of deep model analysis (reachability analysis)
  - Generated trace links from added trace link generators
- A ReDeCT macromodel has maintained intermodel mappings between all 4 models
40.2. Megamodels for Test and Requirements Management
Tool References


- Dominic Tavassoli, IBM Software. Requirements Definition and Management - Ten steps to better requirements management. June 2009


- Teach videos of Axiom
  - http://www.iconcur-software.com/resources.html
  - Video on linking matrix (traceability matrix) http://iconcur-software.com/tutorials/matrix.htm
Introduction to Requirements Management (RM)

- RM bridges the needs of the customer to testing, design, coding, and documentation
- RM continuously manages requirements in the entire software life cycle
- RM relies on inter-model mappings between requirements, test cases, design, and code
Tools in an Integrated Development Environment (IDE)

- Requirements Tool
- Coding Tool
- Testing Tool

- Model mappings
- Model slicing
- Model composition

- Reachability analysis (traceability)
- Attribute analysis

- Reasoning engine
- Relational engine
- GRS engine
- TRS engine
- XML engine

- Requirements Repository
- Test Case Repository
- Metamodel Repository (M2)
- Design Repository (PIM, Arch)
- Implementation Repository (PSI, Code)
Deficiencies of Current RE Methods

- Relationships among requirements are inadequately captured
  - Causal relationship between consistency, completeness and correctness [Zowghi2002]
  - Completeness and consistency are not verified
- Requirement problems (e.g. conflicts, incompleteness) are detected too late or not all
- Relationships between requirements and dependent artifacts are insufficiently managed (test, documentation, design, code)
- Desirable:
  - Models for RE need richer and higher-level abstractions (goals, problems, needs) to validate that they are fulfilled [Mylopoulos1999]
    - Metamodels can be used to define these concepts
    - Ontologies deliver reasoning services
  - Model mappings (direct and indirect) between the artifacts (design, code) and the goals, problems, needs of the customer
    - Based on the model mappings, the requirements are consistently managed with design, code, and documentation
40.2.2 Metamodel-Based Requirements Management
Requirements Tools on the Requirement Database
Metamodelling of Requirements

- Metamodelling is very helpful in RM
  - Requirements are domain-specific, i.e., need domain models
  - The granularity of requirements is very different, and need to be balanced
    - → metamodelling helps to type the requirements
  - Requirements can be treated as models, and **model mappings** can map them to design, implementation, and test models (**traceability**, **Verfolgbarkeit**)

- Many requirement tools are metamodelling-controlled
  - typing requirements
  - linking them
40.2.3 Requisite Pro
RequisitePro (IBM)

- **Metamodel-driven Repository of requirements (requirements database)**
  - Metamodel for requirements *(requirement types)* in metalanguage ERD
    - Attributes: Status, Priority, Difficulty, Stability, Costs
    - Dependencies and traces of requirements
    - Hierarchical requirements
    - Views on requirements
    - Query facility; configuration management
    - Integration into processes and IDE, e.g., Rational Unified Process with Rational Rose UML, ClearCase and MS Project.

- **Traceability Matrix** allows for linking requirements with test cases (direct inter-model mapping)

- Create **software requirements specifications (SRS)** with template documents:
  - Support of different types of SRS (system product, software, service).
Metaclass RequirementType (Ex.)

<<metaclass>>
RequirementType

ReqTag tag;
String name;
Enum status = {proposed, approved, incorporated};
Person[] authors;
Date date;
Version version;
Person responsible;
Text rationale;
Text estimated_cost;
Enum difficulty;
Enum stability;
RiskFactor risk;

RiskFactor
Money damage;
Propability probability;

ReqTag
String prefix = {SR, FEAT, ..};
Int number;

Performance
Time deadline;
RequisitePro – Main Windows

Selection of different requirements types and views

Description of Requirement PR3
FURPS Classification of Requirements

FURPS delivers RequirementTypes for RequisitePro [Wikipedia] [Grady/Caswell] in Hewlett-Packard

- **Functionality** - Feature set, Capabilities, Generality
  - Semi-functionality: Security

- **Qualities:**
  - **Usability** - Human factors, Aesthetics, Consistency, Documentation
  - **Reliability** - Frequency/severity of failure, Recoverability, Predictability, Accuracy, Mean time to failure
  - **Performance** - Speed, Efficiency, Resource consumption, Throughput, Response time
  - **Supportability** - Testability, Extensibility, Adaptability, Maintainability, Compatibility, Configurability, Serviceability, Installability, Localizability, Portability
FURPS+ (FURPS-DIIP)

- **Design Requirement**: a constraint on the design of a system
  - Architecture Requirement: a constraint on the architecture
- **Implementation Requirement**: a constraint on the code of the system
- **Interface Requirement**: a constraint on the external interfaces of the system (the “context model”)
- **Physical Requirement**: a constraint on the hardware environment
Attribute Matrix of Requisite Pro

The attribute matrix is a hierarchical table (relation) of requirement objects and their attributes

- Super and subrequirements
- Priority and Status, and other attributes
Formalizing Requirement Texts

- If requirements are entered in free text (in Word processor), they can be formalized by text mining with:
  - Verb-noun-analysis
  - Keyword identification: MUST, MAY, SHALL, SHOULD, WILL, CUSTOMER
  - Markup information, such as section headers, emphasizing, etc.
  - Concept recognition by looking up nouns in domain models (glossaries, taxonomies, ontologies)

- Requirements can also be recognized from tables in Word documents [RPro]
The Traceability Matrix connects and relates requirements by direct traces and indirect traces over trace_to and trace_from relationships

- The trace relationship is a model mapping within the requirements model
- External projects can be imported, and traces to their public requirements can be defined

Direct traces are entered

- into a form
- into the corresponding bitfield of the traceability matrix

If somebody changes the requirements later, the trace links become suspect and should be checked
Metamodel of Requirements Managements in RequisitePro (Metalanguage ERD)
## Other Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Website</th>
</tr>
</thead>
</table>
70.3 Traceability in Practical RM Tools
Direct Traceability

- With a **direct model mapping**, a requirements model can be linked
  - to a test case specification
  - to a documentation
  - to an architectural specification
  - via the architectural specification, to the classes and procedures in the code
Innovator can be employed simultaneously for requirements, design and implementation models

How to relate these models?
Example: imbus TestBench

http://www.imbus.de/produkte/imbus-testbench/hauptfunktionen/
Requirements get “red-yellow-green” Test Status Attribute

Anforderungsverwaltung von Car Konfigurator (Version 2.1, Abnahmetest)

- Name: Händler gewährt Rabatt
- ID: WHY162
- Version: 1.1
- Eigentümer:
- Status: Review Complete
- Priorität: Essential
- Test-Status: Getestet PASS
Direct Model Mappings between Requirements and Test Tools

- Most often, these tools are in Link-treeware (hierarchical requirements, hierarchical test cases and test suites)
- The trace models can be stored externally in the megamodel
  - Every trace link refers to link-tree node numbers in the requirements and test specifications
40.4 Traceability to Goals in Goal Models with Ontology-Driven Requirements Engineering (ODRE)

Uwe Aßmann¹, Katja Siegemund¹, Edward J. Thomas², Jeff Pan², Yuting Zhao²

¹ Technische Universität Dresden, Germany
² University of Aberdeen, UK

SWESE Oct 24, 2011
Why Ontology-Driven Requirements Engineering (ODRE)?

- Objective: Trace goals from a goal model to requirements to designs and domain models

- Use graph-logic isomorphism to store requirements and their requirement types in logic, more precisely, in an OWL ontology
  - Provide a metamodel (T-Box of requirements ontology) with a huge set of relevant metadata and requirement relationships

- Use reasoning services to
  - provide meaningful checks for completeness and consistency, e.g., as queries to the A-Box with SparQL
  - Make specific suggestions to repair inconsistencies and incompleteness

- Ontology consists of T- and A-Box
  - TBox (Terminological Box) provides metadata
  - ABox (Axiom Box, Fact Base) provides requirements, goals, relationships,...
Lamsweerde defines **goals** as "declarative statements of intent to be achieved by the system under consideration" [Lamsweerde2000]

Benefits of explicit specification of goals in GORE:

- Goals drive the identification of requirements
- Goals provide a criterion for sufficient completeness of a requirement specification
  - Specification of pertinent requirements
  - Relationships between goals and requirements can help to choose the best one
- Concrete requirements may change over time whereas goals pertain stable
Goal-Oriented Requirements Engineering (GORE) – TBox of GORE Ontology
Goal-Oriented RE (Motivation Example)

Win the game

Goal

...
Goal-Oriented RE (Motivation Example)

<Goal>
Win the game

<Objective>
Goal

<Objective>
...

<Obstacle>
Fouls

<Obstacle>
aggressive Fans

<Scenario>
1st. Half time offensive play
Goal-Oriented RE (Motivation Example)

- **Objective**
  - Goal
  - Win the game

- **Obstacle**
  - Foul
  - Aggressive fans

- **FR**
  - Early attack

- **NFR**
  - Fast and good backing
  - Good concentration

- **Scenario**
  - 1st. Half time
  - Offensive play

- **Use-Case**
  - Nowotny backs Schweinsteiger

- **Misuse-Case**
  - Red card for a player
Goal-Oriented RE (Motivation Example)

- **Goal:** Win the game

- **Objective:**
  - Goal
  - ... (omitted)

- **Scenario:**
  - 1st. Half time offensive play

- **Use-Case:**
  - Nowotny backs Schweinsteiger

- **Misuse-Case:**
  - Red card for a player

- **Obstacle:**
  - Fouls
  - Aggressive Fans

- **FR:**
  - Early attack

- **NFR:**
  - Fast and good backing
  - Good concentration

- **Constraint:**
  - max. play time

- **Metric:**
  - Keeps 90% of the goals
  - Attack until 10th. minute
Goal-Oriented RE (Motivation Example)

- **Goal**: Win the game
- **Objective**: Goal
  - **Scenario**: 1st. Half time offensive play
  - **Use-Case**: Nowotny backs Schweinsteiger
- **Obstacle**: Foul
- **Obstacle**: Aggressive Fans
- **FR**: Early attack
- **NFR**: Fast and good backing
- **NFR**: Good concentration
- **Decision**: Neuer as goalkeeper
- **Misuse-Case**: Red card for a player
- **Constraint**: Max. play time
- **Metric**: Keeps 90% of the goals
- **Metric**: Attack until 10th. minute
Goal-Oriented RE (Motivation Example)

<Goal>
Win the game

<Objective>
Goal

<Objective>
...

<Scenario>
1st. Half time offensive play

<Use-Case>
Nowotny backs Schweinsteiger

<FR>
Early attack

<NFR>
Fast and good backing

<NFR>
Good concentration

<Obstacle>
Fouls

<Obstacle>
aggressive Fans

<Decision>
Neuer as goalkeeper

<Risk>
Early exhaustion

<Misuse-Case>
Red card for a player

<Constraint>
max. play time

<Metric>
Keeps 90% of the goals

<Metric>
Attack until 10th. minute
Architecture for ODRE Tool

1. Requirement Engineering background knowledge
2. Requirement Specifications of a project
3. Completeness checking rules
4. Consistency checking rules
5. RE Meta Model
6. Requirement ontology
7. TBox
8. ABox
9. Complete queries
10. Consistent queries
Reasoning for RE – Completeness Check

Example of Completeness Rule:

“Every Functional Requirement (FR) must define whether it is mandatory or optional.”

The GORE ontology of Lambsweerde needs about 50 completeness rules
- Implemented as SPARQL queries on the A-Box
- The requirements model is deemed incomplete if a specific rule fails
- Reasoning Strategy: Closed World Reasoning (for negation as failure)
  - supported by SPARQL 1.1 and TrOWL reasoner
Reasoning for RE – Completeness Check (Example)

“Every Functional Requirement (FR) must define whether it is mandatory or optional.”

- SPARQL rule:

```sparql
IF FR is NOT mandatory AND NOT optional THEN
  Print error: "You did not specify whether the following FRs are mandatory or optional: [FR_n]."
  "Please specify whether these FRs are mandatory or optional."
```

© Prof. U. Aßmann
Reasoning for RE – Completeness Check (Example)

Extract of individuals and relationships of the A-Box from the SPARQL analysis:

- isRelatedTo(Goal2; UseCase7)
- NonFunctionalRequirement (NonFunctionalRequirement1)
- IsOptional(NonFunctionalRequirement1; true)
- FunctionalRequirement(FunctionalRequirement1)

Error.
You did not specify whether the following FR are mandatory or optional:
- FunctionalRequirement1. Please specify this attribute for the FR:
- FunctionalRequirement1. Every FR must specify AT LEAST ONE requirement relationship.
Reasoning for RE – Consistency Check

▷ GORE needs 6 consistency rules among requirement artefacts (valid relations between requirement artefacts)
  - Based on a chosen subset of requirement artefacts
  - Consistency rules are encoded as DL axioms in the A-Box

▷ Instance specific error messages resulting from validation displayed by Guidance Engine
Reasoning for RE – Consistency Check (Example)

- Extract of individuals and relationships of the A-Box from the SPARQL analysis:

  isExclusionOf (FunctionalRequirement5; FunctionalRequirement7)
  ChosenRequirement(FunctionalRequirement5)
  ChosenRequirement(FunctionalRequirement7)

Error.
The following requirements exclude others: FunctionalRequirement5.
Please choose one of the following options:

Suggestion.
Exclude the following requirements from the chosen requirement set: FunctionalRequirement5. OR
Find alternatives for: FunctionalRequirement5 or
Revise the requirement relationships of (FunctionalRequirement5, FunctionalRequirement7).
Reasoning for RE – Verification Methods (Example)

Consistency check of requirement selection (6 rules)

Excluding requirements must not be included in one set.

IF excluding requirements are included in one set
THEN print error: "The following requirements exclude Others: [R_n]."
"Please choose one of the following options:
Exclude the following requirements: [R_n],
Find alternatives for [R_n] or
Revise the requirement relationships of [[R x, R y],...]."
Status of ODRE

- All Requirement artefacts and meaningful relationships can be captured within an Ontology Metamodel
- ODRE Approach detects **inconsistent** and **incomplete** requirements
- Standard tooling (reasoners) are useful
  - Specification of requirements uses OWA
  - Verification needs CWA
- First evaluation proves applicability for medium requirement specifications
  - Problem: available requirement specifications do not provide sufficient information (much less than could be captured by ODRE)
  - Primary evaluation within MOST Project
    - Capture all requirement artefacts
    - Detect all inconsistencies and incomplete metadata
- PhD Thesis of Katja Siegemund (2014)
The End