ProSeminar
Traceability & Megamodellierung in MDE und Softwararchitektur
- Notizen -

Contents

1 Aufgabenstellung **3**
- 1.1 Materialien . . . 3
- 1.2 Fragen & Aufgaben . . . 3

2 Was ist Traceability in MDE und Softwararchitektur? **4**
- 2.1 Traceability . . . 4
  - 2.1.1 traces & traceability links . . . 4
- 2.2 traceability in Softwarearchitektur . . . 5
- 2.3 traceability in MDE . . . 6

3 Ziele von Traceability **8**
- 3.1 ... in Software-Engineering . . . 8
- 3.2 ... in Model-Driven-Engineering . . . 8
- 3.3 Ziel Kategorien . . . 9

4 Traceability Methoden, Techniken und Anwendungen **10**
- 4.1 Methoden . . . 10
  - 4.1.1 Traceability Schemas & Metamodelle . . . 10
  - 4.1.2 Traceability (link) Matrizen . . . 10
  - 4.1.3 Traceability Cross-Refrences . . . 10
  - 4.1.4 Traceability Graphen . . . 11
- 4.2 Techniken . . . 11
- 4.3 Anwendungen . . . 11

5 Ansätze in den gegeben Materialien **12**
- 5.1 Architecture frameworks (M. 3) . . . 12
- 5.2 Dynamic hierarchical mega models (M. 2) . . . 12
- 5.3 Linguistic architectures (M. 4) . . . 12
- 5.4 Megamodels at Runtime (M. 5) . . . 13
  - 5.4.1 Runtime Model . . . 13
  - 5.4.2 Runtime Model Kategorien . . . 13

6 Vergleich **14**
- 6.1 Ansätze nach Entwicklungsphasen . . . 14

7 Traceability in Megamodelierungsansätzen **15**

1 Aufgabenstellung

1.1 Materialien

1. S. Winkler and J. von Pilgrim. A survey of traceability in requirements engineering and model-driven development. Software and System Modeling, 9(4):529– 565, 2010.

2. A. Seibel, S. Neumann, and H. Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. Software & Systems Modeling, 9(4):493–528, 2010.

3. R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione. Realizing Architecture Frameworks Through Megamodelling Techniques. In Proc. of ASE'10, pages 305–308. ACM, 2010.

4. Ralf Lämmel and Andrei Varanovich. Interpretation of Linguistic Architectur. Unpublished, 2014

5. Thomas Vogel, Andreas Seibel, and Holger Giese. The Role of Models and Megamodels at Runtime.

6. Jean-Marie Favre, Ralf Lämmel, and Andrei Varanovich. Modeling the Linguistic Architecture of Software Products.

7. IEEE Std 610.12-1990.

1.2 Fragen & Aufgaben

1. Was ist *traceability* in MDE und Softwararchitektur? (Insbesondere Material 1)

2. Welche Methoden, Techniken und Anwendungen gibt es?

3. Wie kommt *traceability* in Megamodellierungsansätzen vor? (Materialien 2, 3 und 4)

4. Vergleiche die Ansätze in Materialien 2, 3 und 4.

2 Was ist Traceability in MDE und Softwararchitektur?

Materiaien

- **Material 1**

2.1 Traceability

- *Traceability* ≈ Verfolgbarkeit
- Keine einheitliche Definition, da der Begriff *traceability* in verschiedenen Forschungsgebieten Verwendung findet.
- Intuitives Verständnis (bzgl. Software):
  "... the ability to describe and follow the life of software artifacts ..." (M.1, S. 529)
  "... Die Fähigkeit das Leben eines Software-Artefakts zu beschreiben und zu verfolgen ..."
  (Kann jedoch aufgrund der Allgmeinheit auch auf andere Software-gestützte Entwicklungsprozesse neben der Software-Entwicklung angewandt werden, z.B. Projektmangement)
- Ursprüngliche (historisch erste) Verwendung im Anforderungsmanagement (RE = Requirements Engineering)
- *traceability* als Qualitätsmerkmal wegen der oft inkrementellen und iterativen Natur eines Software-Herstellungsprozesses:

  Vision → Anforderungen → Architektur/Modelle → Code & Tests
  parallel dazu: Dokumentation

  Elemente dieser Phasen sind untereinander Verbunden und begründen einander (**Verifizierung & Validierung**);z.B. geänderte oder neue Anforderungen führen zu Änderungen oder Erweiterungen in Folgephasen. (M. 1, S. 531)
- *traceability* im IEEE Standard Glossary of Software Engineering Terminologoy:
  1. *The degree to which a relationschip can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another.*
  2. *The degree to which each element in a software development product establishes its reason for existing.*

  (M. 1, S. 531)

2.1.1 traces & traceability links

- *trace* ≈ Spur

- *traceability* impliziert, dass Änderungen und Erweiterungen *traces* (Spuren) hinterlassen, z.B. `commit` in der Versionskontrolle. Änderungen zeigen allerdings auch eine zugrunde liegende Idee, die sich u.U. in informalen Gesprächen mit Stakeholdern begründet. Solche Gespäche können auch als Spuren betrachtet werden, die sich allerdings nur schwerlich dokumentieren lassen.

- *trace* im IEEE Standard Glossary of Software Engineering Terminologoy:

  1. *A record of the execution of a computer program, showing the sequence of instructions executed, the names and values of variables, or both. Types include execution trace, retrospective trace, subroutine trace, symbolic trace, variable trace.*
  2. *To produce a record as in (1).*
  3. *To establish a relationship between two or more products of the development process; for example, t o establish the relationship between a given requirement and the design element that implements that requirement.*

- Funktionale und nichtfunktionale Spuren nach Pinheiro (M. 1, S. 532):
  - **Funktionale Spuren:** Funktionale Spuren entstehen als Nebenprodukt einer Transformation von einem Artefakt zu einem anderen. Solche Transformationen werden entweder automatisch oder händisch durchgeführt, unterliegen jedoch einer genau definierten Regel.
  - **Nichtfunktionale Spuren:** Nichtfunktionale Spuren entstehen als Nebenprodukt meist kreativer Prozesse, z.B. formalisierung des Protokols eines Kundeninterviews. Solche Spuren umfassen *"... reason, context, descision and technical ..."* Aspekte. Im Protokolbeispiel begründen sie die Art der gewählten Formalisierung.

- *traceability links.* (M. 1, S. 532)
  Spuren können teils als Metadaten und teils als Beziehungen zwischen Stakeholdern und Artefakten oder Artefakten und anderen Artefakten vorkommen. Spuren in form von Beziehungen werden *traceability links* genannt; vgl. IEEE Glossary Eintrag Punkt 3.
  Achtung: *traceability links* sind entweder uni- oder bi-direktional. Dennoch lassen sie sich in alle Richtungen verfolgen.

2.2 traceability in Softwarearchitektur

- Im Fall von Anforderungsmanagement: Die Fähigkeit Spuren von und zu Anforderungen zu verfolgen. (M. 1, S. 532)

- Nach Gotel und Finkelstein: *"... the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use,*

and through periods of on-going refinement and iteration in any of these phases).” (M. 1, S. 532)

- *pre-RS & post-RS traceability* nach Gotel und Finkelstein (M. 1, S. 532)
  RS = Requirement Specification
  - *pre-RS traceability.* Betrifft Spuren die während der Anforderungserhebung entstehen. Häufig Spuren die informale Kommunikation betreffen.
  - *post-RS traceability.* Betrifft Spuren die während der Implementierung der Anforderungen in Architektur und Code entstehen.

- *backward & forward traceability* nach ANSI/IEEE Std 830-1984 (M. 1, S. 532)
  Triviale bi-direktionale Artefakt-Quellen Beziehung.

- *horizontal & vertical traceability* nach Ramesh und Edwards (M. 1, S. 533)
  Unterscheidet Spuren nach Entwicklungsphase und Abstraktionsebne, wobei *horizontal traceability* Spuren **innerhalb** einer Phase oder Abstraktionsebene beinhaltet, *vertical traceability* Spuren zwischen Phasen oder Abstraktioneben.
  Achtung: Es ist unklar ob sich Spuren wirklich so unterscheiden lassen, oder ob sich die horizontalen und vertikalen Dimensionen auf die iterativen und inkrementellen Phasen abbilden lassen.

2.3 traceability in MDE

- *traceability* in MDE ≈ *traceability* in Softwarearchitektur/-engineering
  mit dem Unterschied, dass MDE sehr automatisiert ist. (M. 1, S. 533)

- *”the ’MDD way’ ”*: definiere Modelle und Metamodelle, die kontextgebunden *traceability* für den gegebenen Kontext beschreiben (M. 1, S. 533)

- *traceability* nach OMG (M. 1, S. 533f): *”A trace [...] records a link between a group of objects from the input models and a group of objects in the output models. This link is associated with an element from the model transformation specification that relates the groups concerned.”*
  Achtung: Betrachtet *traceability* und *traces* nur als Nebenprodukte von (automatischen) Modelltransformationen.

- *traceabiltiy* nach Paige et al (M. 1, S. 534): *”[...] the ability to chronologically interrelate uniquely identifiable entities in a way that matters. [...] [It] refers to the capability for tracing artifacts along a set of chained [manual or automated] operations.”*
  Achtung: Schließt schließt zwar manuelle Modelltransformationen ein, , betrachtet allerdings nur sequenzielle Transformationen.

- *traceability* nach Aizenbud-Reshef et al (M. 1, S. 534): *” any relationship that exists between artifacts involved in the software-engineering life cycle. This definition includes, but is not limited to the following:*

- *Explicit links or mappings that are generated as a result of transformations, both forward (e.g., code generation) and backward (e.g., reverse engineer- ing)*
- *Links that are computed based on existing informa- tion (e.g., code dependency analysis)*
- *Statistically inferred links, which are links that are computed based on history provided by change management systems on items that were changed together as a result of one change request.*

”

Erlaubt manuelle und automatische Transformationen, *traces* werden allerdings nicht auf Nebenprodukte von Modelltransformationen beschränkt.

- *pre-, intra- & post-model traceability* (M. 1, S. 534):
  - **pre**-model *traceybility*: Verfolgbarkeit zwischen ”frühen” Artefakten (Vision, Notizen, etc.) und dem ersten Modell
  - **intra**-model *traceybility*: Verfolgbarkeit zwischen Ergebnissen von Modelltransformationen in der Entwicklungsphase.
  - **post**-model *traceybility*: Verfolgbarkeit zwischen dem finalen Modell und anderen Nichtmodell-Artefakten.
- *explicit & implicit traceability links* (M. 1, S. 534):
  - nach Paige et al:
    explicit *traceability links* sind syntaktischer Teil des Modells,
    implicit *traceability links* entstehen durch Anwendung von Modell-Management-Operationen
  - nach Philippow und Riebisch:
    explicit *traceability links* sind (wie bei Paige et al) syntaktischer Teil des Modells,
    implicit *traceability links* sind gleiche (gleich benamte) Verbindungen in verschiedenen Modellen

3 Ziele von Traceability

3.1 ... in Software-Engineering

- *Prioritizing requirements* (Anforderungen priorisieren)
- *Estimating change impact* (Schätzung der Änderungsauswirkung)
- *Proving system adequateness* (Adequatheit des Systems beweisen)
- *Validating artifacts* (Artefaktvalidierung)
- *Testing the system* (Testen des Systems)
- *Supporting special audits* (Unterstützung für spezielle Prüfungen)
  vgl.: *Understanding the system*
- *Improving changeability* (Verbesserung der Änderbarkeit / Systemflexibilisierung)
- *Extracting metrics* (Metriken extrahieren)
- *Monitoring progress* (Fortschrittsbeobachtung)
- *Assessing the development process* (Beurteilung des Entwicklungsprozesses)
- *Understanding the system* (Das System verstehen)
  WICHTIG!: sihe M.4 und M.6
- *Tracking rationale* (Dokumentation der Änderungsargumentationen)
- *Establishing accountability* (Festlegung der Verantwortlichkeiten)
- *Documenting re-engineering* (Dokumentation von Umstrukturierungen)
- *Finding reusable elements* (Finden wiederverwendbarer Elemente)
- *Extracting best practices* (Extrahieren sog. Best Practices)

3.2 ... in Model-Driven-Engineering

- *Supporting design decisions* (Unterstützung von Designentscheidungen)
- *Proving adequateness/Validation* (Beweis der Adequatheit/validität)
- *Understanding and managing artifacts* (Artefakte verstehen und verwalten)
- *Deriving usable visualizations* (Ableitung brauchbarer Visualisierungen)
- *Change impact analysis* (Änderungsauswirkungsanalyse)
- *Synchronizing models* (Modelsynchronisation)
- *Driving product-line development* (Führung von Produktlinienentwicklung)

3.3 Ziel Kategorien

- **Systemverständniss**
  - –
- **Unterstützung und Automatisierung der Systemverwaltung**

4 Traceability Methoden, Techniken und Anwendungen

4.1 Methoden

4.1.1 Traceability Schemas & Metamodelle

- Regeln zur Aufnahme von *traceability* Informationen
- Schemas vs. Metamodelle:
  - **Schemas:** haupsächlich informale Regeln zur manuellen Aufnahme von *traceability* Information in SE und RE
  - **Metamodelle:** Schemas formalisiert als Metamodellen in MDE

Traceability link types. asdf

4.1.2 Traceability (link) Matrizen

	A1	A2	A3	A4
A1		×		
A2			×	
A3	×			
A4				×

- 2D Visualisierung von *traceability links* zwischen Artefakten
- Unhandlich für große Anzahlen von Artefakten
- Ursprünglich nur Darstellung der Existenz (×) von *traceability links*. Kann aber mit anderen Formen erweitert werden (z.B. $\triangleleft, \triangleright, \bullet$, etc.)

4.1.3 Traceability Cross-Refrences

A1	Lorem ipsum ...	$\triangleleft$ A2 $\triangleright$ A3 ...
A2	Lorem ipsum ...	$\triangleleft$ A3 $\triangleright$ A3 ...
A3	Lorem ipsum ...	$\triangleleft$ A1 $\triangleright$ A2 ...

($\triangleleft \approx$ in, $\triangleright \approx$ out)

- tabellarische Darstellung von *traceability links* als Artefakt-Metadaten

4.1.4 Traceability Graphen

- intuitive graphische Notation
  - Knoten $\approx$ Artefakte
  - Kanten $\approx$ *traceability links*
- üblicherweise definiert in Metamodellen

4.2 Techniken

4.3 Anwendungen

5 Ansätze in den gegeben Materialien

Materiaien

- **Material 2**
- **Material 3**
- **Material 4**

5.1 Architecture frameworks (M. 3)

- Erstellen von *Architecture frameworks* als wiederverwendbare Megamodelle mittels *MEGAF* bestehend aus:
  - *Viewpoints*,
  - *System Concerns*,
  - *Model Kinds* (Metamodelle),
  - *Stakeholders*,
  - und *Correspondence Rules*
- *Architecture frameworks* beschreiben wie eine konkrete Architektur zur modelieren ist
- **Kein expliziter Ansatz zu *Traceability***
- Mittels *Model Kinds* und *correspondence Rulse* können konkrete *Traceability*-Modelle wiederverwendbar spezifiziert werden

5.2 Dynamic hierarchical mega models (M. 2)

-

5.3 Linguistic architectures (M. 4)

- MDE-Ansatz: Automatisierte *Traceability*-Analyse.
- Materialspezifischer Ansatz: *"Ersetze UML durch DSL"*
- (Mega-)Modelle werden einer DSL (*MegaL*) beschrieben.
  - Megamodelle beschreiben hier den **Ist**-Zustand eines Systems (im Gegensatz zur üblichen MDE-Praxis, den Zustand eines zukünftigen Systems zu beschreiben)
  - Metamodelle/Schemas werden häufig durch (Programmier-)Sprachen ersetzt.
  - Einheiten in *MegaL* werden durch URIs auf physisch existierende Artefakte abgebildet. (URIs bilden bis zur Inhaltsebene von Dateien ab, z.B. kann eine URI auf eine Metode innerhalb einer Klasse innerhalb einer Datei zeigen.)

  - Relationen in *MegaL* werden durch designierte Programme (*"tools"*) evaluiert.

- *Traceability Links* entstehen als Nebenprodukt von Relationsevaluierungen.
- *Traceability Links* als 1-zu-1-Abbildung einer Regel-URI auf eine Ressource-URI (wobei die angegbene Regel ebenfalls als physikalische Ressource existiert)
- Bensoders interessant zur automatischen Spurgewinnung in Generierungs- und Kompilierungsprozessen, die auf diskreten semantischen Regeln basieren und gewisse Annahmen als erfüllt angesehen werden können.
  Beispiel: Eine Java `.class`-Datei ist das Produkt der Kompilierung der korrespondierenden `.java`-Datei. Nach erfolgreicher Kompilierung kann angenommen werden, dass die `.class`- und die `.java`-Datei Element der (*elementOf*-Relation) Java-Sprache sind. Gleichzeitig kann ebenfalls angenomen werden, dass die `.class`-Datei konform zur (*conformsTo*-Relation) `.java`-Datei ist, dass die `.java`-Datei konform zur Java-Grammatik ist und dass die enthaltene Klasse und ihre Methoden konform zu den entsprechenden Grammatikregeln sind.
  Anmerkung: Obiges Beispiel ist trivial und beschreibt lediglich den Java- Kompilierungsprozess, der den meisten Softwareentwicklern vertraut sein sollte. Ist man allerdings in der Lage kontextspezifischere Regeln zu formulieren und zu beschreiben wie diese auf die Java-Regeln abgebildet werden, ist die Informationsgewinnung ungleich höher einzuschätzen.

5.4 Megamodels at Runtime (M. 5)

5.4.1 Runtime Model

- Runtime ≈ Execution-time
- Phase im Software Lebenszyklus
- Instanzen von Metamodellen (diese sind wiederum Instanzen von Meta-Metamodellen, etc.)

5.4.2 Runtime Model Kategorien

- *Implementation Models* [Impl.Models]
- *Configuration & Architectural Models* [Conf+Arch.Models]
- *Context & Resource Models* [Cxt+Rsrc.Models]
- *Configuration Space & Variability Models* [ConfSpace+Var Models]
- *Rules, Strategies, Constraints, Requirements and Goals* [RSCRGs]

6 Vergleich

6.1 Ansätze nach Entwicklungsphasen

Pre-Model	Intra-Model	Post-Model
Architecture frameworks **Material 3** Beschreibt einen Ansatz wie *Traceability* innerhalb eines wiederverwendbaren Architecture Frameworks definiert werden kann. AF beschreiben wie *Traceability* zu implementieren ist.	*Dynamic hierarchical mega models* **Material 2** Beschreibt einen Ansatz wie *Traceability* während eines Entwicklungsprozesses konsistent und korrekt erhalten bleibt.	*Linguistic/Semantic architectures* **Material 4** Beschreibt einen Ansatz wie *Traceability* während, aber auch nach eine Entwicklungsprozess erhalten/rekonstruiert werden kann.

7 Traceability in Megamodelierungsansätzen