

Tarski: A Platform for Automated Analysis of Dynamically Configurable Traceability Semantics

Ferhat Erata^{1,2} Moharram Challenger^{1,4} Bedir Tekinerdogan¹
Anne Monceaux³ Eray Tuzun⁵ Geylani Kardas⁴

¹Information Technology Group, Wageningen University, The Netherlands

²UNIT Information Technologies R&D Ltd., Izmir, Turkey

³System Engineering Platforms, AIRBUS Group Innovations, Toulouse, France

⁴International Computer Institute, Ege University, Izmir, Turkey

⁵Academy Directorate, HAVELSAN Inc., Ankara, Turkey

3rd Workshop on Dependability at Izmir Institute of Technology

Acknowledgements

- Scientific and Technological Research Council of Turkey (TUBITAK), Technology and Innovation Funding Programs Directorate (TEYDEB) under project# 9140014, 9150181
- Minister for the Economy, Industry and Digital Affairs of France, Directorate-General for Enterprise (DGE) under contract# 14293020
- European Cooperation in Science and Technology (COST) Action IC1404 "Multi-Paradigm Modelling for Cyber-Physical Systems"



EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Exploitations

ITEA-ModelWriter: Synchronized Document Engineering Platform
<https://itea3.org/project/modelwriter.html>

ITEA-ASSUME: Affordable Safe & Secure Mobility Evolution
<https://itea3.org/project/assume.html>



Source codes, datasets and screencasts are available at:
<https://github.com/ModelWriter/WP3>

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

What is Traceability?

Traceability can be defined as the degree to which a relationship can be established among work products (aka. artefacts) of the development process.

What is case-based or project-based traceability configuration?

Rigorously specification the semantics of traceability elements.

Why is Reasoning about Traceability important?

Richer and precise automated traceability analysis.

Compliance and Certification in automotive and aviation industries.

Challenges of Traceability in Industry

Semantically meaningful traceability

- traceability relations should have a rich semantic (meaning) instead of being simple bi-directional referential relation

Configuration of traceability (possibly dynamically)

- Traceability Semantics is often statically defined.

Challenges of Traceability in Industry

Semantically meaningful traceability

- traceability relations should have a rich semantic (meaning) instead of being simple bi-directional referential relation

Configuration of traceability (possibly dynamically)

- Traceability Semantics is often statically defined.
- The semantics cannot be easily adapted for the needs of different projects.

Challenges of Traceability in Industry

Semantically meaningful traceability

- traceability relations should have a rich semantic (meaning) instead of being simple bi-directional referential relation

Configuration of traceability (possibly dynamically)

- Traceability Semantics is often statically defined.
- The semantics cannot be easily adapted for the needs of different projects.
- Different traceable elements and the relation types exist in industrial settings,

Challenges of Traceability in Industry

Semantically meaningful traceability

- traceability relations should have a rich semantic (meaning) instead of being simple bi-directional referential relation

Configuration of traceability (possibly dynamically)

- Traceability Semantics is often statically defined.
- The semantics cannot be easily adapted for the needs of different projects.
- Different traceable elements and the relation types exist in industrial settings,
- Likewise, different traceability analysis scenarios exists.
Several industries demands formal proofs of Traceability.

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

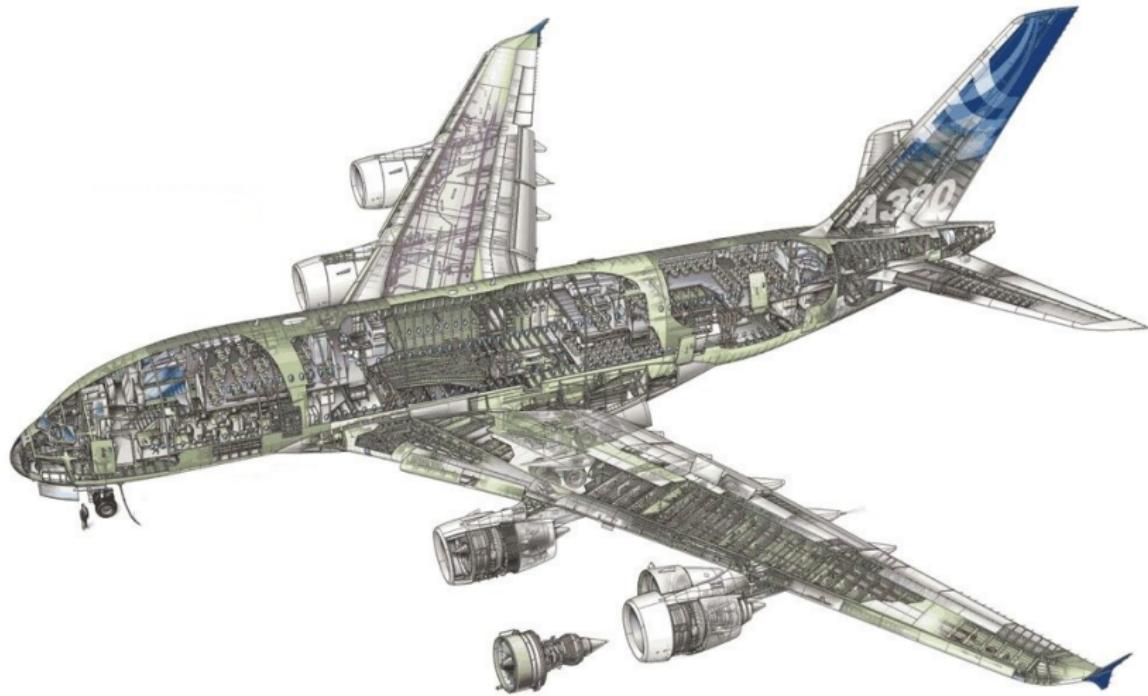
3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

Airbus Group Innovations

System Installation Design Principles



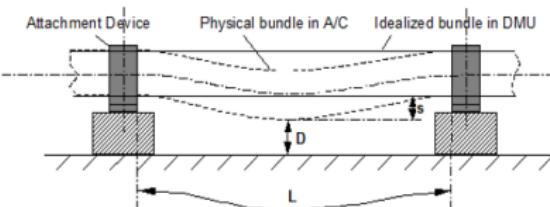
Airbus Group Innovations

System Installation Design Principles



SIDP92A001V-A-784

For installation of optical and electrical harnesses additional clearance for sagging (s) shall be provided as detailed below:



s ... Sagging of bundle (real behavior of physical bundle in A/C due to gravity, ageing, etc.)

D... Required Distance

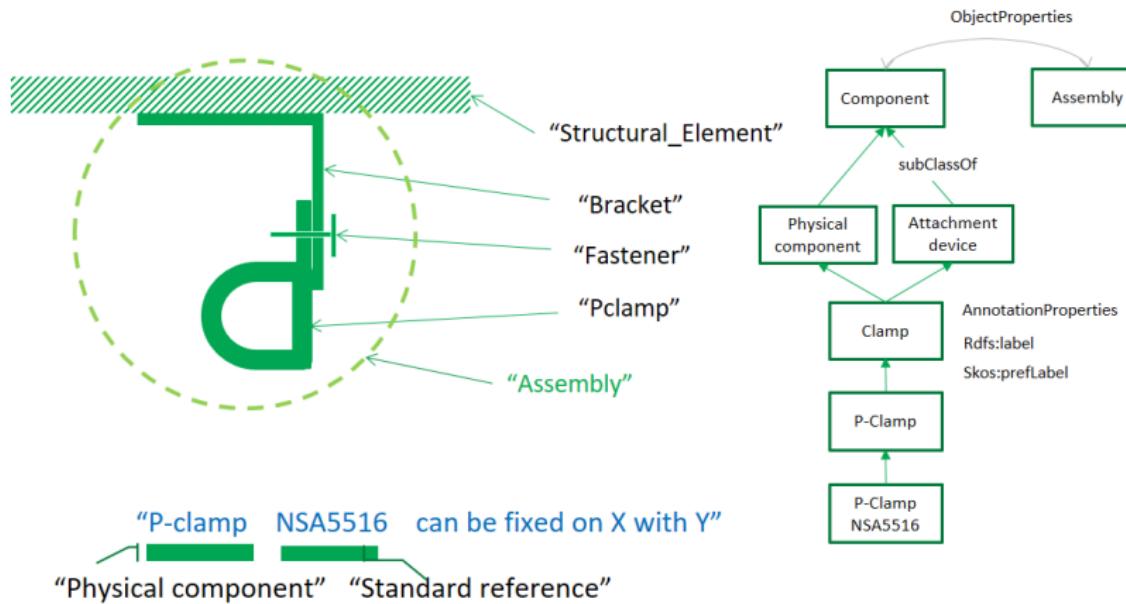
L... Actual length of a bundle segment between two Attachment Points (as designed in DMU)

Figure 6: Sagging of bundles between attachment points

Note: Unless the bundle has a straight routing, L is bigger than the pitch between the Attachment Points.

Airbus Group Innovations

System Installation Design Principles



Havelsan Aerospace Electronics Industry

Application Lifecycle Management

DO-178C

Software Considerations in Airborne Systems and Equipment Certification

Traceability

DO-178 requires a documented connection (called a **trace**) between the certification artifacts. For example, a **Low Level Requirement (LLR)** traces up to a **High Level Requirement (HLR)**. A **traceability analysis** is then used to ensure that each *requirement* is **fulfilled** by the **source code**, that each *requirement* is **tested**, that each line of **source code** has a purpose (is connected to a requirement), and so forth. Traceability ensures the system is complete.

Traceability Analysis Activities defined in DO-178

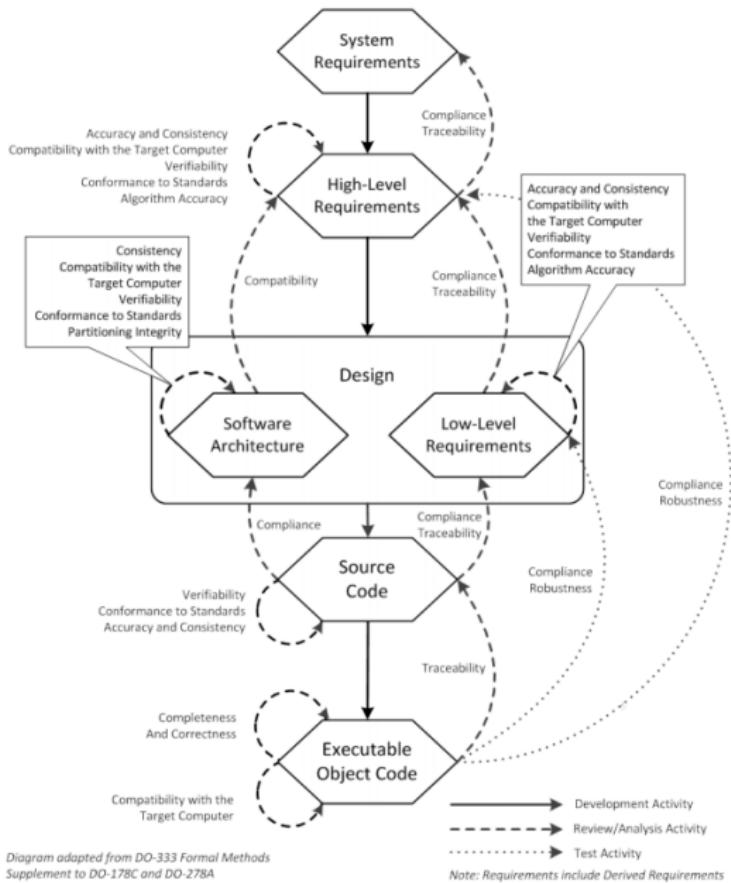


Diagram adapted from DO-333 Formal Methods
Supplement to DO-178C and DO-278A

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

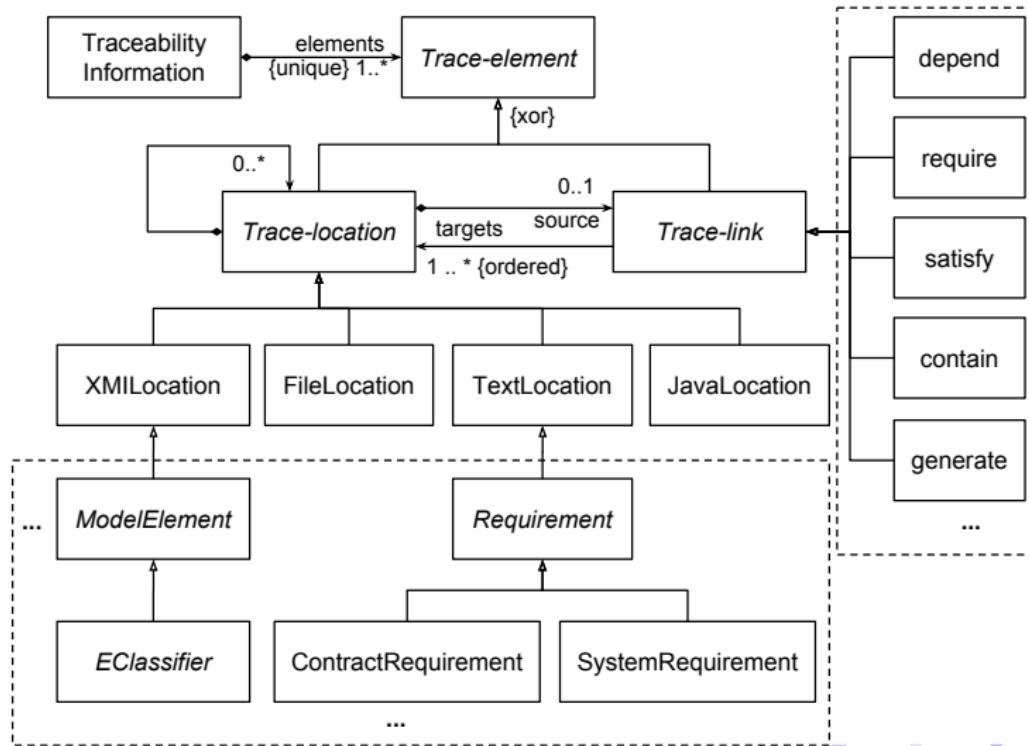
- **Traceability Domain Model**
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

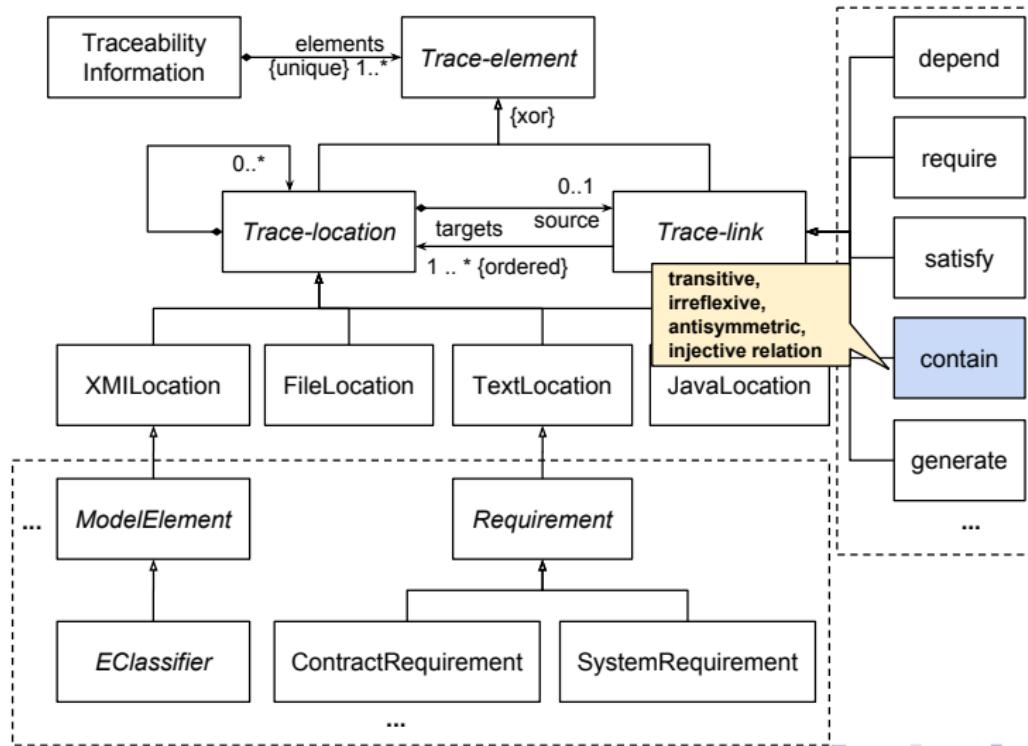
- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

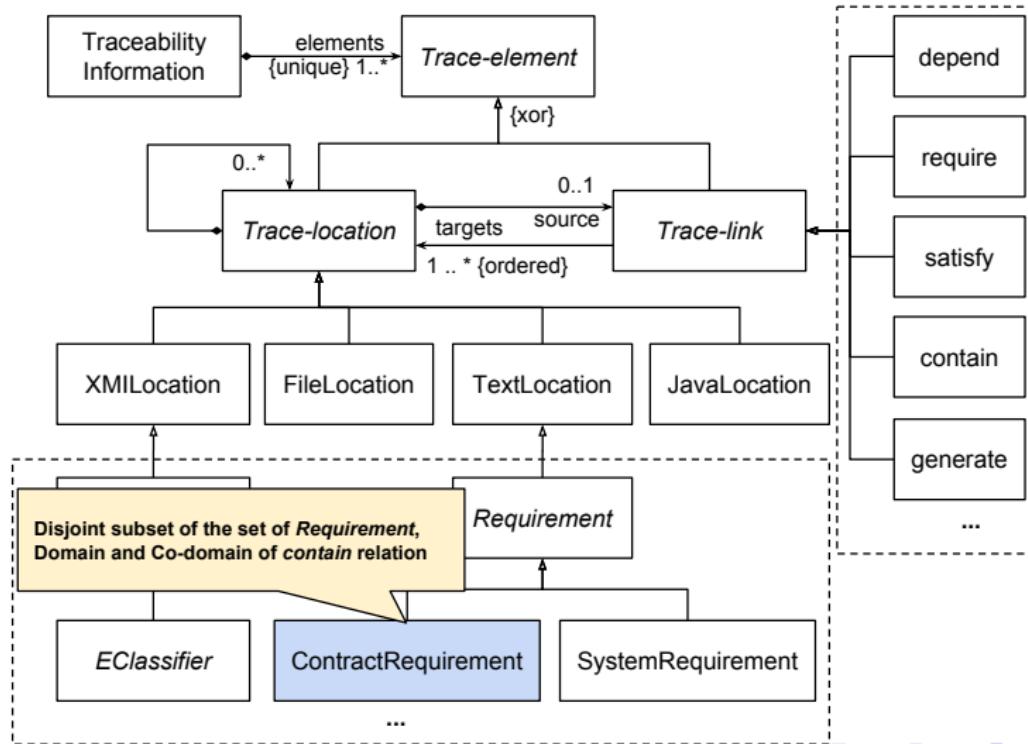
A conceptual model for traceability and its extension



Semantics of *contain* relation (represents decomposition)



Semantics of *ContractRequirement*



Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

- Traceability Domain Model
- **First-order Relational Model and Logic**
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

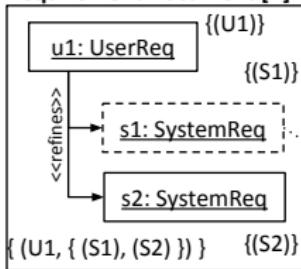
- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

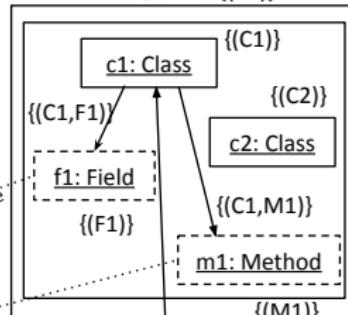
Fragments of a traceability instance

$$X = \{(U_1), (S_1), (S_2), (U_1, S_1), (U_1, S_2)\} \quad Z = \{(P_1), (C_1), (C_2), (F_1), (M_1), (C_1, F_1), (C_1, M_1)\} +$$

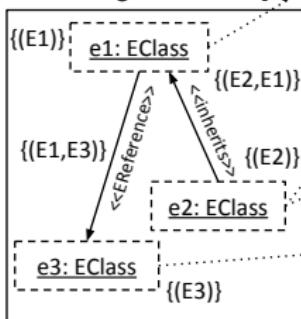
Requirement Document [X]



W

Java Package [Z] $\{(P_1)\}$ 

ECore Design Document [Y]



$$Y = \{(E_1), (E_2), (E_3), (E_2, E_1), (E_1, E_3)\}$$

First-order relational model of the traceability instance

The *universe* of traceability of the current state

$$D_T : \{S_1, E_1, E_2, E_3, F_1, M_1, M_2, P_2\}$$

The *type signature*

$$\Sigma_T : \{R_{EJ} \sqsubseteq E \rightarrow C \sqcup M \sqcup F, R_{JRE} \sqsubseteq F \rightarrow S \rightarrow E\}$$

The *relational model* under the signature Σ_T

$$M_t : \{S = \{\langle S_1 \rangle\}, E = \{\langle E_1 \rangle, \langle E_2 \rangle, \langle E_3 \rangle\}, J = \{\langle F_1 \rangle, \langle M_1 \rangle, \langle M_2 \rangle, \langle P_2 \rangle\}, R_{EJ} = \{\langle E_2, M_1 \rangle, \langle E_2, M_2 \rangle, \langle E_3, P_2 \rangle\}, R_{JRE} = \{\langle F_1, S_1, E_1 \rangle\}\}$$

First-order Relational Logic (FOL + Relational Calculus)

. Relational Join and \sim Transpose

The *dot join* and *transpose* operators ensure a uniform way of navigation between *trace-locations* through *trace-links* in constraints.

$*^$ (Reflexive) Transitive Closure

Transitive Closure allows the encoding of common reachability constraints that otherwise could not be expressed in FOL, such as preventing cyclic dependencies between *trace-locations*.

Domain and Range Restrictions

The restriction operators are used to filter relations to a given domain or range.



First-order Relational Logic (FOL + Relational Calculus)

. Relational Join and \sim Transpose

$$\begin{aligned}E.R_{EJ} &= \{\langle E_1 \rangle, \langle E_2 \rangle, \langle E_3 \rangle\}. \{\langle E_2, M_1 \rangle, \langle E_2, M_2 \rangle, \langle E_3, P_2 \rangle\} \\&= \{\langle M_1 \rangle, \langle M_2 \rangle, \langle P_2 \rangle\}\end{aligned}$$

$$\begin{aligned}J. \sim R_{EJ} &= \{\langle F_1 \rangle, \langle M_1 \rangle, \langle M_2 \rangle, \langle P_2 \rangle\}. \{\langle M_1, E_2 \rangle, \langle M_2, E_2 \rangle, \langle P_2, E_3 \rangle\} \\&= \{\langle E_2 \rangle, \langle E_3 \rangle\}\end{aligned}$$

* \wedge (Reflexive) Transitive Closure

$$\wedge \{\langle M_1, E_1 \rangle, \langle E_1, C_1 \rangle\} = \{\langle M_1, E_1 \rangle, \langle E_1, C_1 \rangle, \langle M_1, C_1 \rangle\}$$

Domain and Range Restrictions

$$P <: R_{JE} = \{\langle P_2 \rangle\} <: \{\langle M_1, E_2 \rangle, \langle M_2, E_2 \rangle, \langle P_2, E_3 \rangle\} = \{\langle P_2, E_3 \rangle\}$$

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

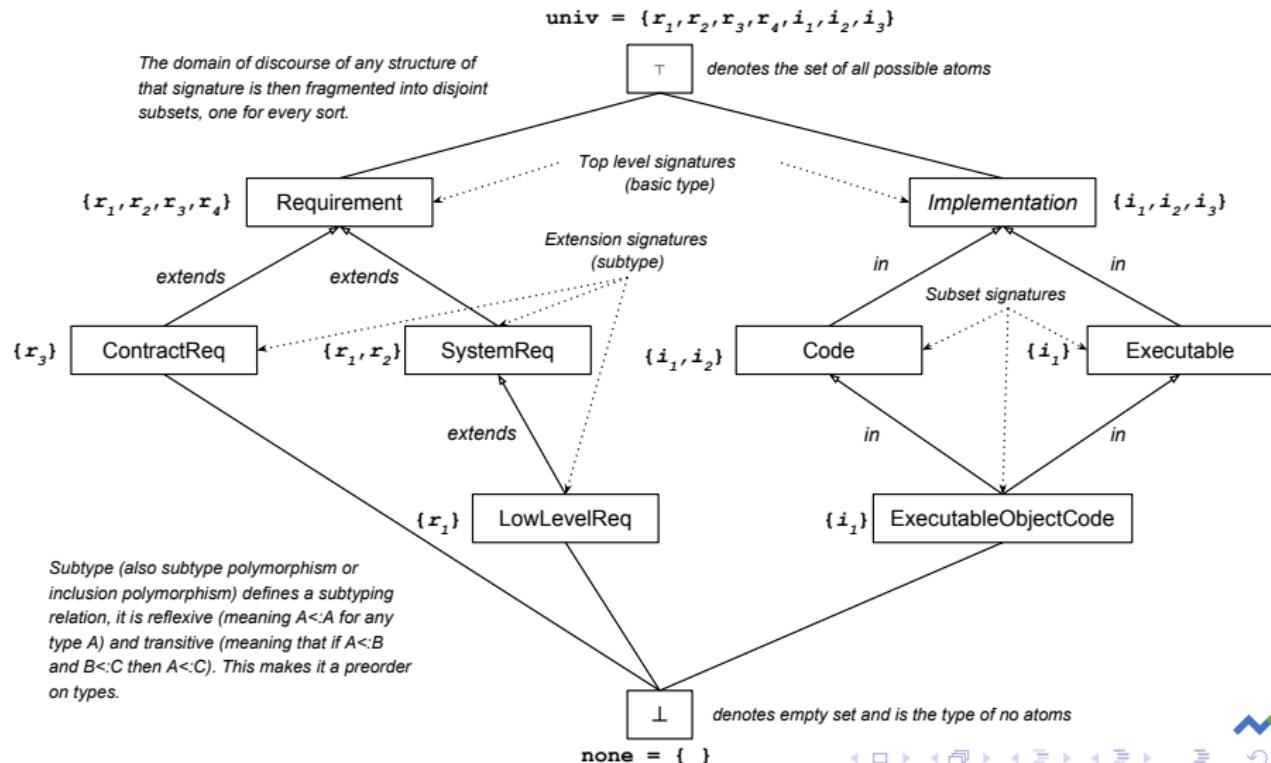
- Traceability Domain Model
- First-order Relational Model and Logic
- **Type Annotation and Trace-Relations**
- Formal Semantics and Automated Analysis

3 Demonstration

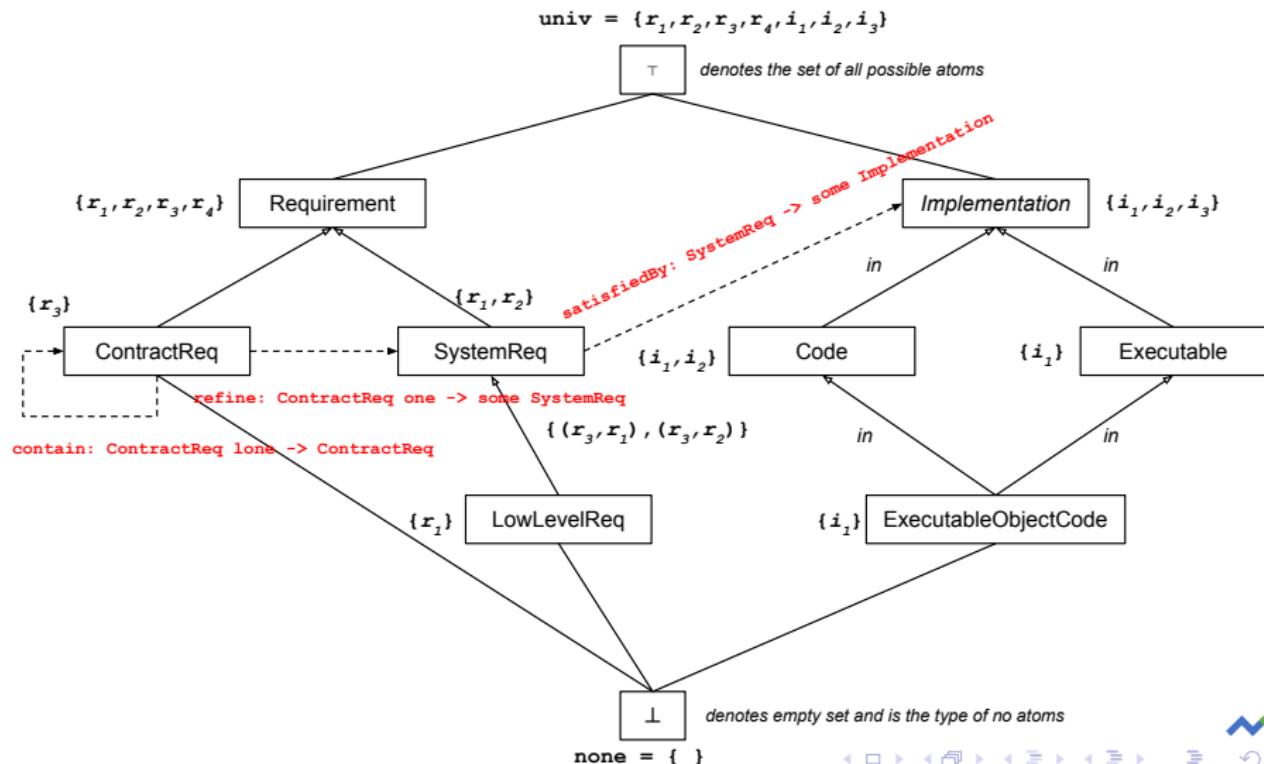
- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

Basic Type and SubType



Relation Types



Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- **Formal Semantics and Automated Analysis**

3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

Formal Specification of an example configuration

```
1 abstract sig Artefact { depends: set Artefact}
2
3 -- Locate@File
4 one sig Specification extends Artefact {
5     contract: some ContractRequirement}
6
7 -- Locate@Text
8 sig ContractRequirement extends Artefact {
9     system: set SystemRequirement,
10    contains: set ContractRequirement}
11
12 -- Locate@ReqIF
13 sig SystemRequirement extends Artefact {
14     satisfiedBy: set Implementation,
15     requires: set SystemRequirement,
16     refines: set SystemRequirement}
```

```
17 abstract sig Implementation extends Artefact {  
18     fulfills: lone ContractRequirement}  
19  
20 -- Locate@Java  
21 sig Code, Component extends Implementation {}  
22  
23 -- Locate@EMF  
24 sig Model extends Implementation {  
25     transforms, conforms: set Model,  
26     generates: set (Code ∪ Component)}  
27  
28 -- Semantics@SystemRequirement.satisfiedBy  
29 fact {∀ i: Implementation | some i.^satisfiedBy}
```

Automated analysis functions over Traceability Model

Consistency Checking

The system checks whether the user model satisfies the specification or not.

Reasoning about Trace-relations

If the model is a partial (incomplete), the platform tries to complete the model with respect to the semantics declared in the specification inferring new trace-relations on the model.

Trace-elements Discovery

If a de-synchronization occurs on one or more ends of a *trace-link* probably caused by a change such as deletion of a trace-location, we try to repair the broken link based on the specified semantics.



Reasoning about Trace-relations

```
30 -- Reason@ContractRequirement.system
31 fact { $\forall$  s: SystemRequirement, s': s.*~refines |
32     s'.~system = s.~system}
33
34 -- Reason@SystemRequirement.requires
35 fact {  $\forall$  s, s': SystemRequirement |
36     s' in s.refines  $\implies$  s in s'.requires }
37
38 -- Reason@Implementation.fulfills
39 fact { $\forall$  i: Implementation, s: i.~satisfiedBy
40     | i.fulfills = s.~system }
```

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

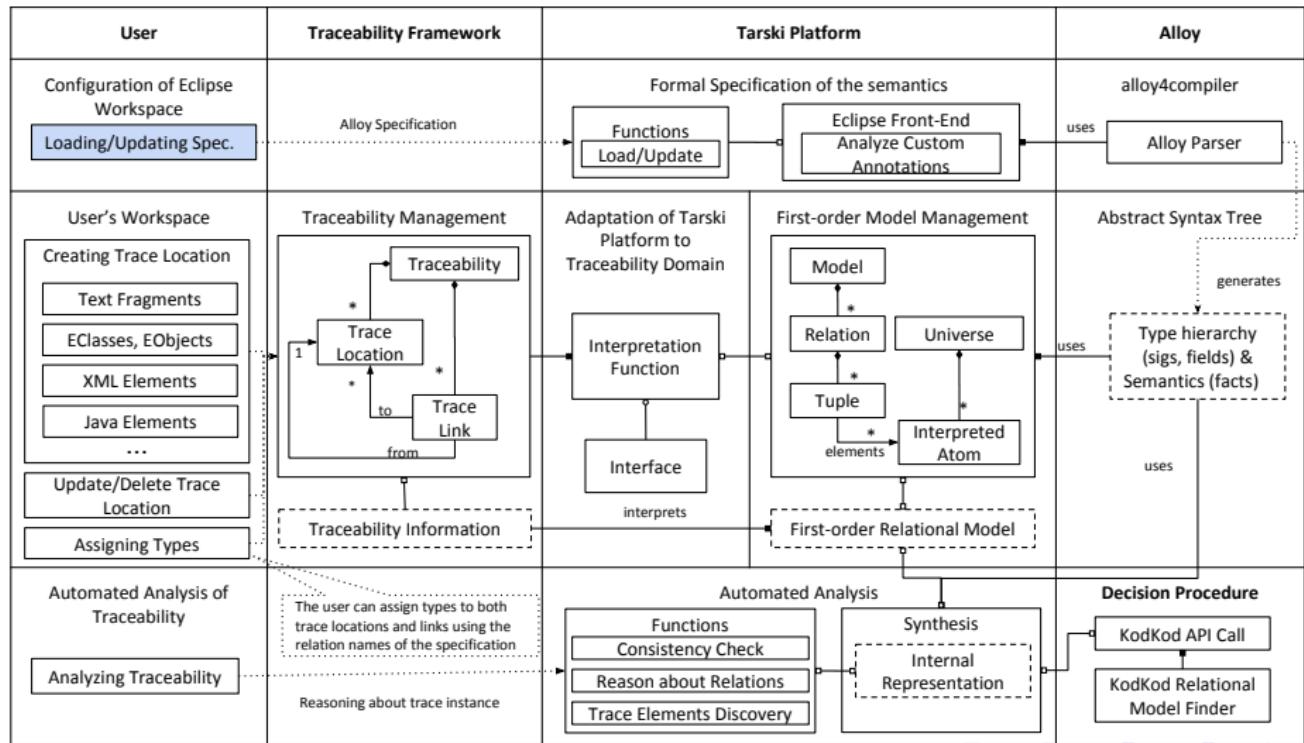
- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

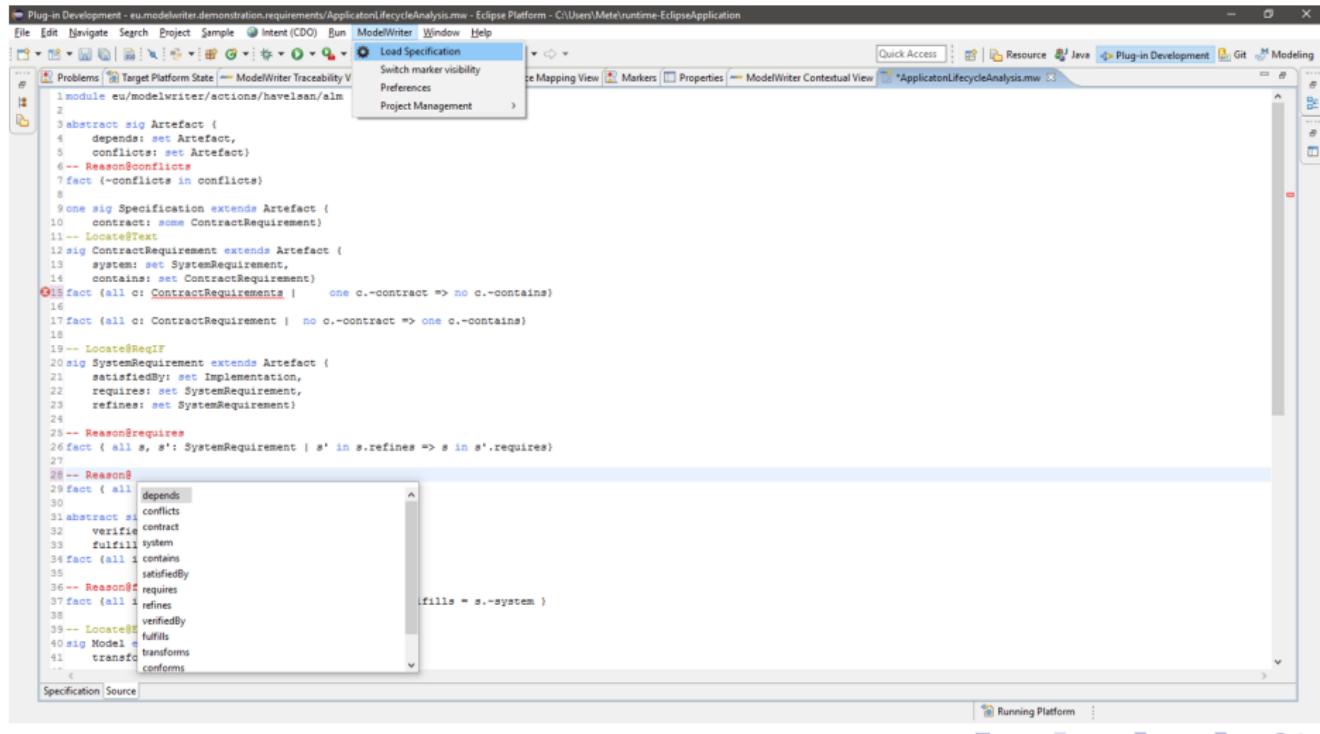
- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

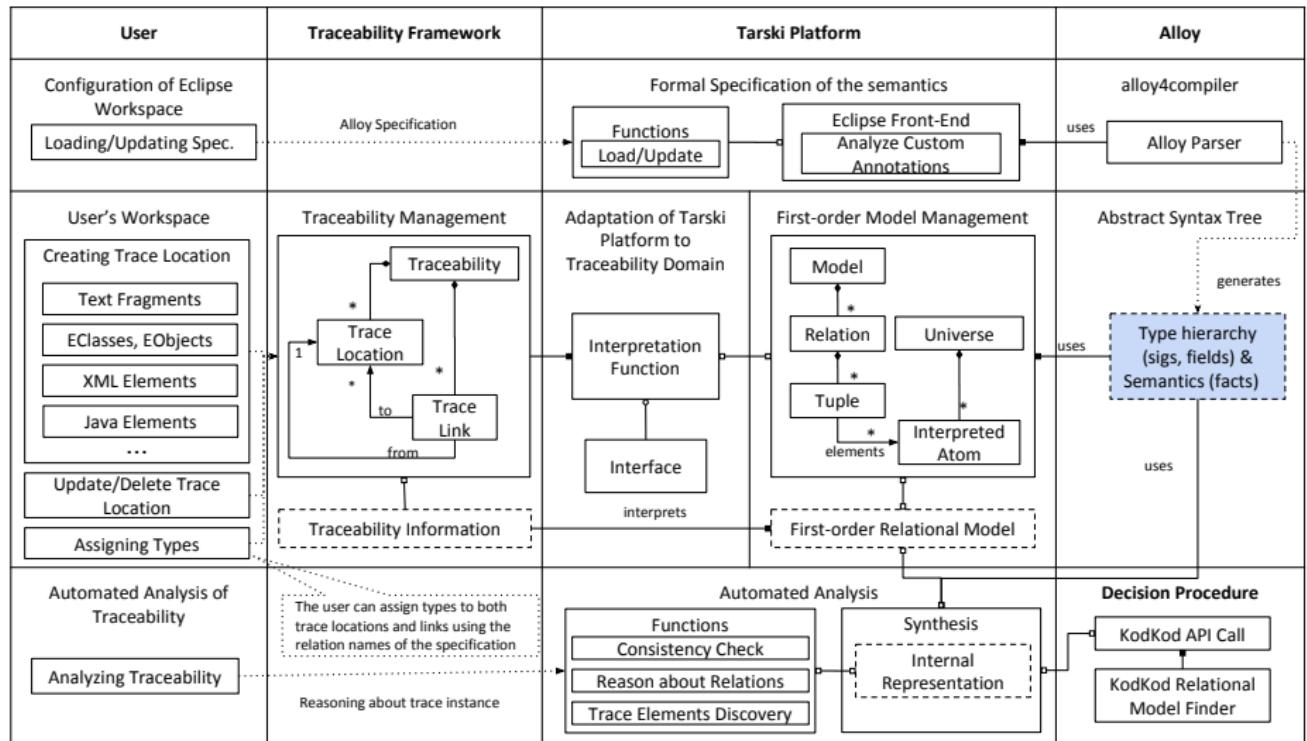
Configuration of User's Workspace



Configuration of User's Workspace



Type Hierarchy from the Specification



Type Hierarchy from the Specification

Plugin-in Development - Eclipse Platform - C:/Users/Mete/runtime-EclipseApplication-havelsan

File Edit Navigate Search Project Sample Intent (CDO) Run Tarski Window Help

ApplicationLifecycleAnalysis.mw

```

1 module eu.modelwriter.actions.havelsan.alm
2
3 abstract sig Artefact {
4   depends: set Artefact,
5   conflicts: set Artefact
6 -- Reason@conflicts
7 fact {-conflicts in conflicts}
8
9 one sig Specification extends
10 contract: lone ContractRequirement
11 -- Locate@ReqIF
12 -- Discover@ContractRequirements
13 sig ContractRequirement extends
14 system: set SystemRequirement
15 contains: set ContractRequirement
16
17 -- Semantics@ContractRequirements
18 fact {all o: ContractRequirement
19 fact {all o: ContractRequirement
20
21 -- Locate@ReqIF
22 sig SystemRequirement extends
23 satisfiedBy: set Implementation
24 requires: set SystemRequirement
25 refines: set SystemRequirement
26
27 -- Reason@System
28 fact {all s: SystemRequirement
29
30 -- Reason@requires
31 fact {all s, s': SystemRequirement
32
33 abstract sig Implementation {
34 verifiedBy: set Verification
35 fulfills: lone ContractRequirement
36 fact {all i: Implementation
37
38 -- Reason@fulfills
39 fact {all i: Implementation, s: i.-satisfiedBy | i.fulfills = s.-system }
40
41 -- Trace@BMPF
42

```

Quick Access Resource Java Plug-in Development Git Modeling

Preferences

Sets and Relations

Sets

- universe
 - Artefact (abs)
 - Specification
 - ContractRequirement
 - SystemRequirement
 - Implementation (abs)
 - Model
 - Code
 - Component
 - Verification (abs)
 - Simulation
 - Analysis
 - Test

Relations

- depends : Artefact -> set of Artefact
- conflicts : Artefact -> set of Artefact
- contract : Specification -> some of ContractRequirement
- system : ContractRequirement -> set of SystemRequirement
- contains : ContractRequirement -> set of SystemRequirement
- satisfiedBy : SystemRequirement -> set of Implementation
- requires : SystemRequirement -> set of SystemRequirement
- refines : SystemRequirement -> set of SystemRequirement
- verifiedBy : Implementation -> set of Verification
- fulfills : Implementation -> lone of ContractRequirement
- transforms : Model -> set of Model
- conforms : Model -> set of Model
- generates : Model -> set of Code, Component

Specification C:\Users\Mete\git\Demonstrations\eu.modelwriter.demonstration.requirements\ApplicationLifecycleAnalysis.mw

Restore Defaults Apply OK Cancel

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

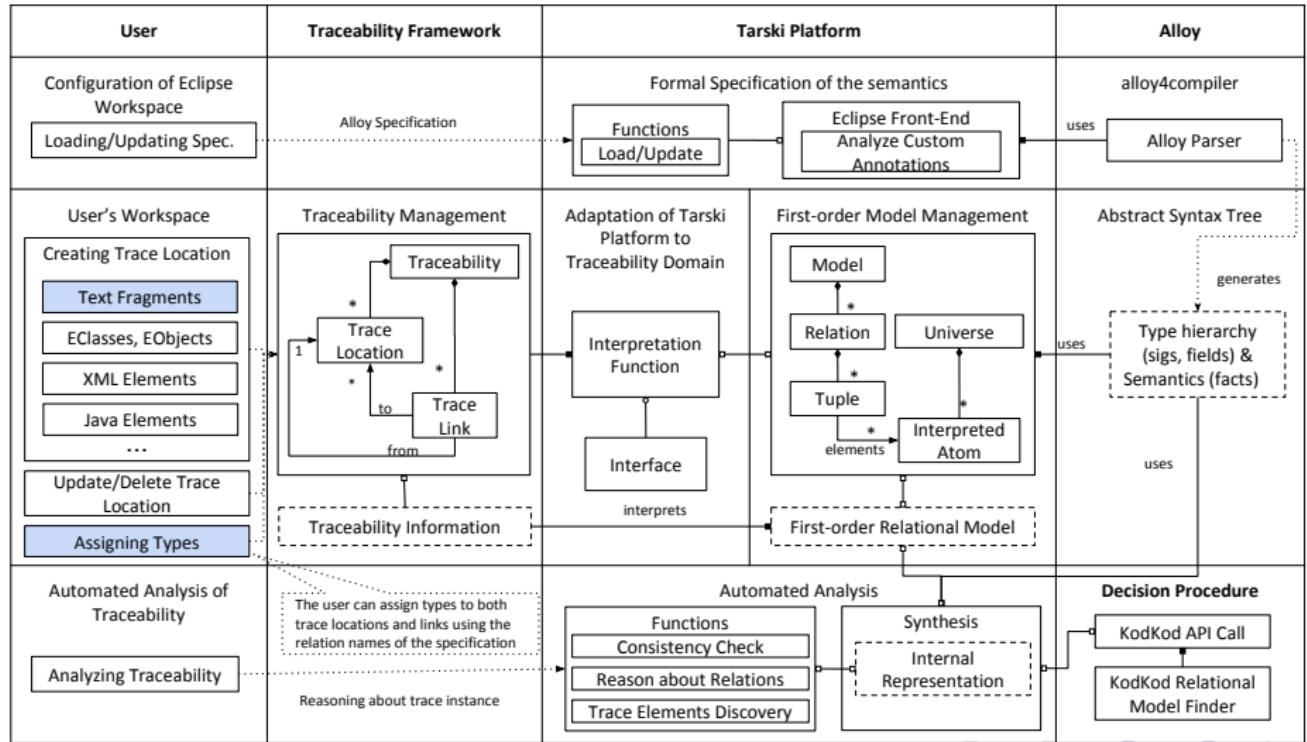
- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

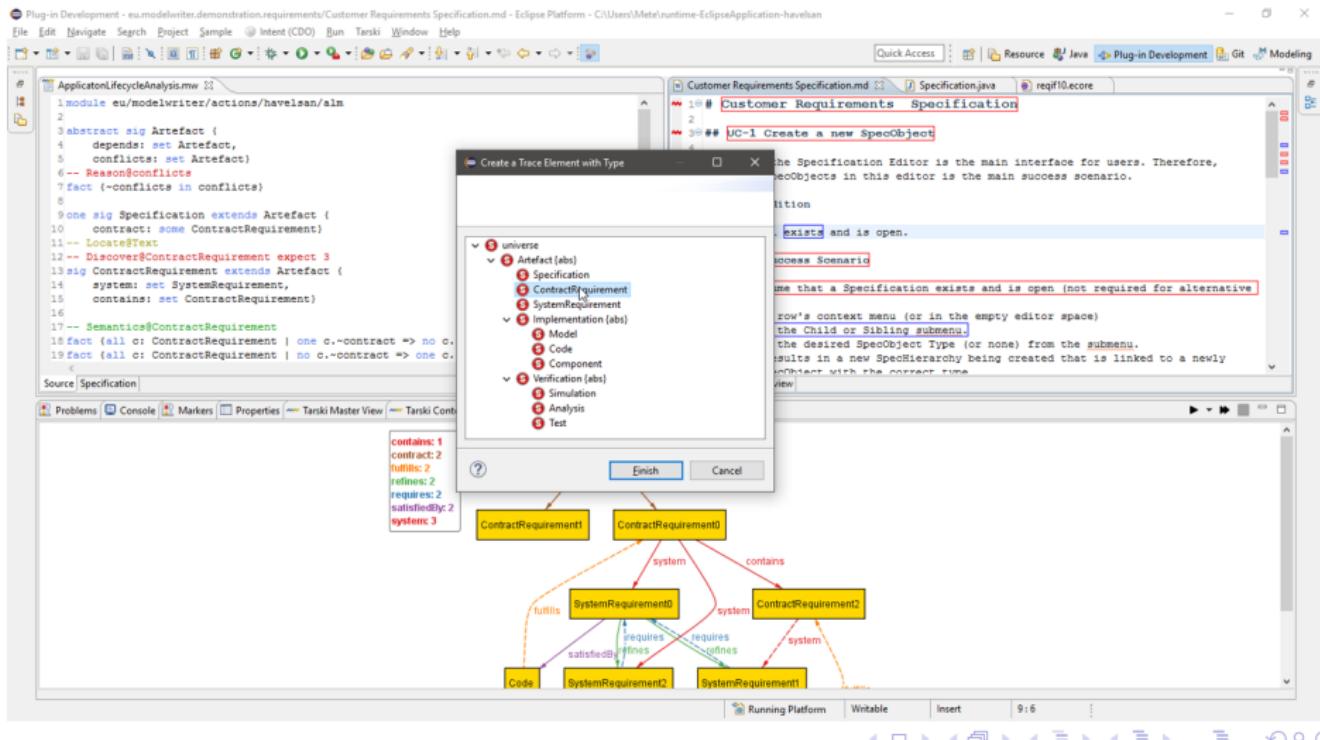
- Formal Specification of Traceability Semantics
- **Traceability Management**
- First-order Model Management
- Automated Analysis of Traceability

4 Conclusion and Future Work

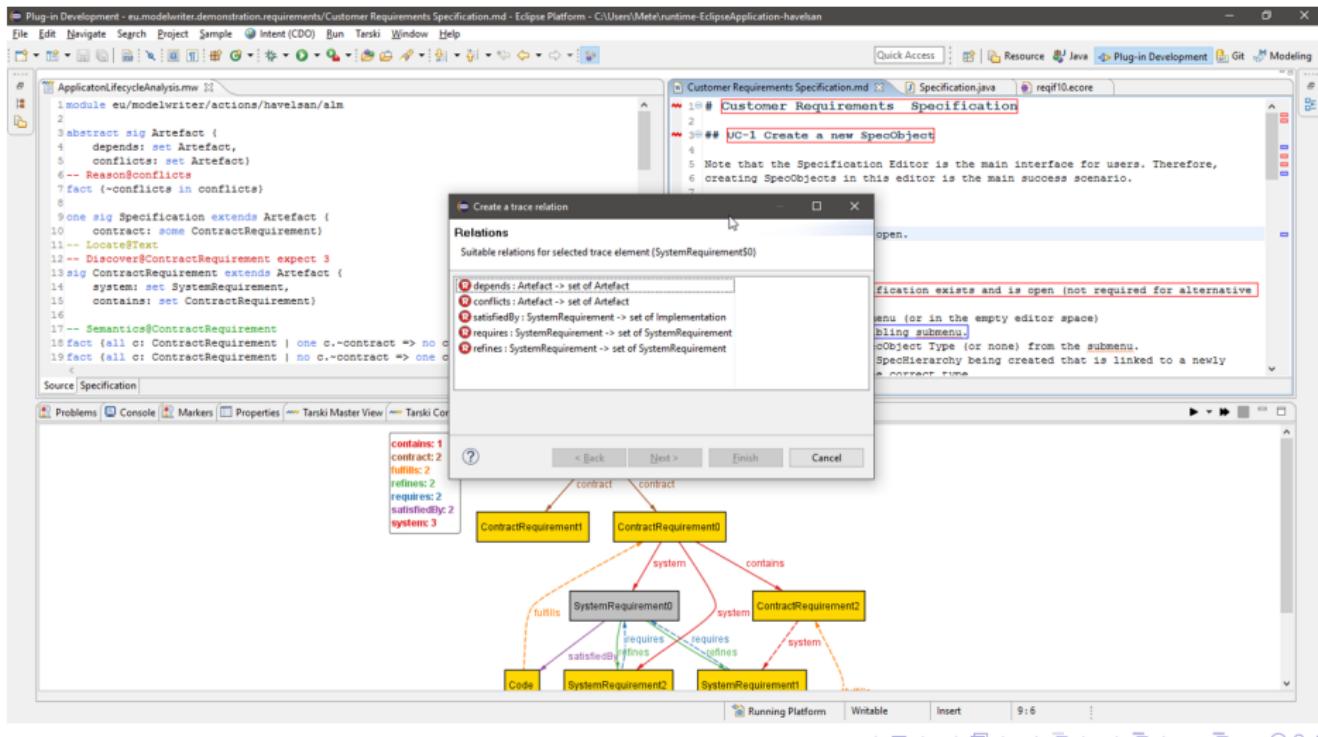
Creating *Trace-locations* and Assigning Types



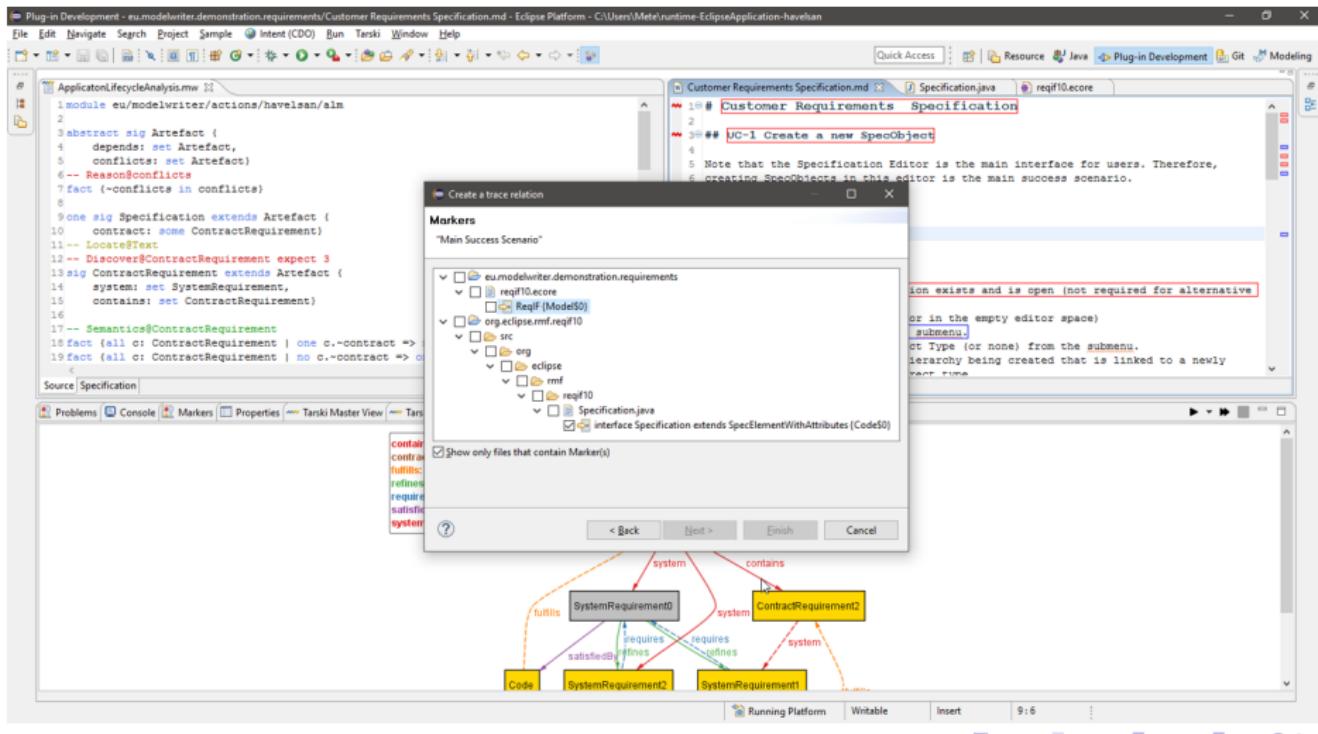
Assigning a *Sub Type* to a *Trace-location*



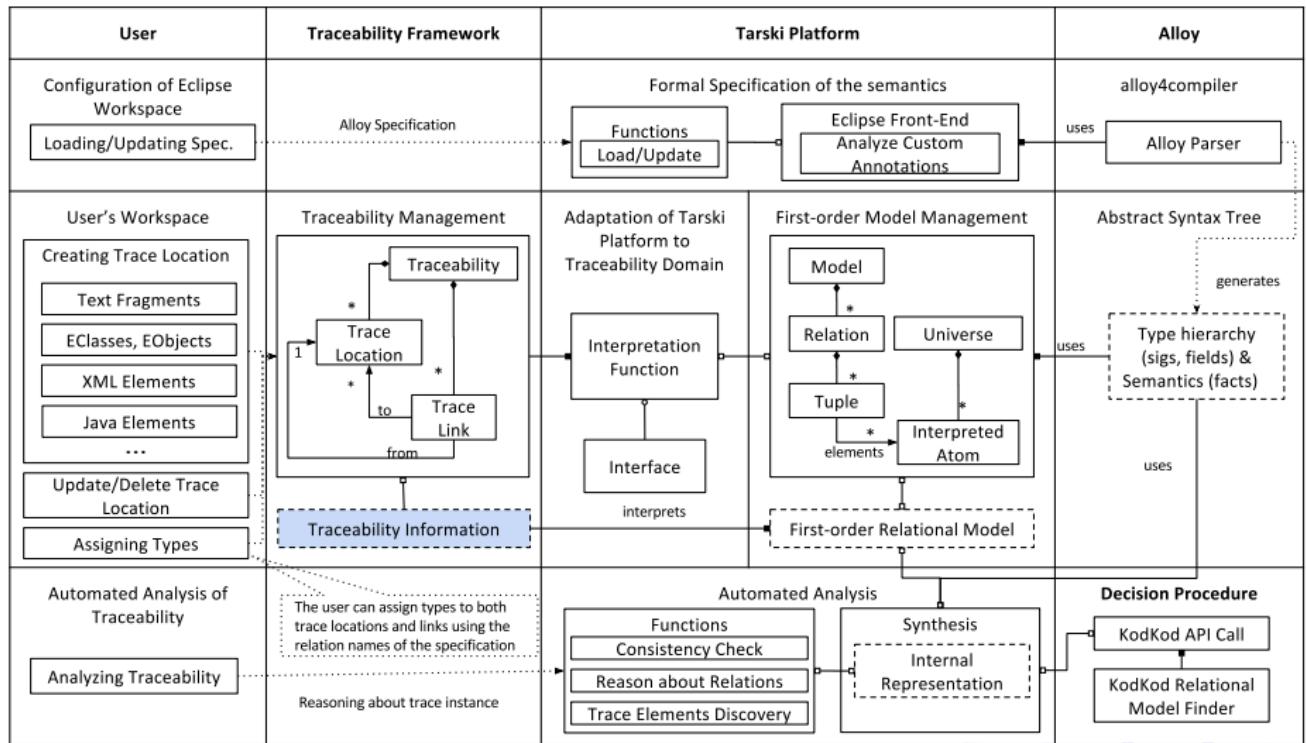
Assigning a binary *Field Type* to a *Trace-link*



Selecting a *Trace-Location* from the co-domain of the *type*



Traceability Information



Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

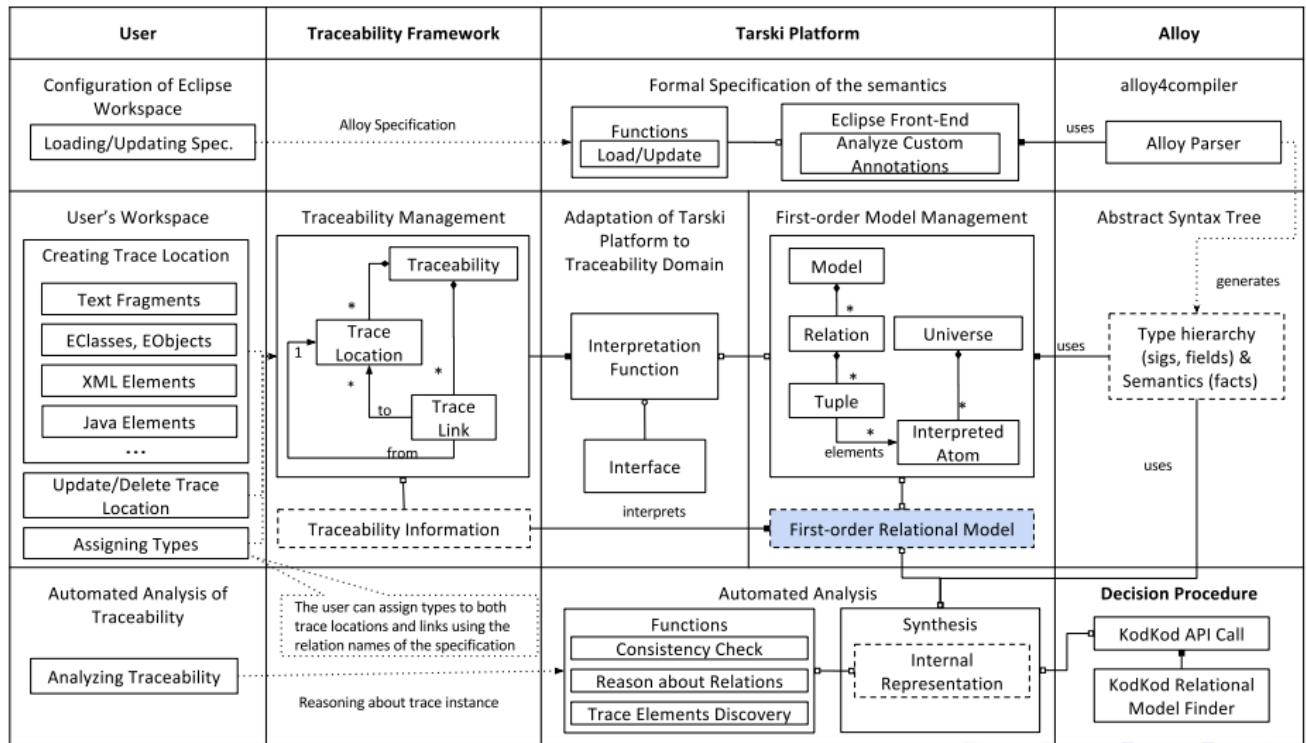
- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- **First-order Model Management**
- Automated Analysis of Traceability

4 Conclusion and Future Work

First-order Relational Model



Dynamic Configuration & Model Management

The screenshot shows two main windows. On the left is a code editor titled "ApplicationLifecycleAnalysis.mw" containing Tarski model code. On the right is the "Tarski Traceability View" showing a traceability matrix with various entities like Specification, ContractRequirement, SystemRequirement, Code, and Model.

Code Editor (ApplicationLifecycleAnalysis.mw):

```

1 module eu.modelwriter.actions.havelsan.alm
2
3 abstract sig Artefact {
4   depends: set Artefact,
5   conflicts: set Artefact)
6 -- Reason@conflicts
7 fact {conflicts in conflicts}
8
9 one sig Specification extends Artefact {
10   contract: lone ContractRequirement)
11 -- Locate@ReqIF
12 -- Discover@ContractRequirement expect 3
13 sig ContractRequirement extends Artefact {
14   system: set SystemRequirement,
15   contains: set ContractRequirement)
16
17 -- Semantics@ContractRequirement
18 fact {all c: ContractRequirement | one c.~contract => no c.~contains }
19 fact {all c: ContractRequirement | no c.~contract => one c.~contains }
20
21 -- Locate@ReqIF
22 sig SystemRequirement extends Artefact {
23   satisfiedBy: set Implementation,
24   requires: set SystemRequirement,
25   refines: set SystemRequirement)
26
27 -- Reason@system
28 fact {all s: SystemRequirement | one s.~system}
29
30 -- Reason@requires
31 fact {all s, s': SystemRequirement | s' in s.refines => s in s'.requires }
32
33 abstract sig Implementation extends Artefact {
34   verifiedBy: set Verification,
35   fulfills: lone ContractRequirement)
36 fact {all i: Implementation | some i.~satisfiedBy}
37
38 -- Reason@fulfills
39 fact {all i: Implementation, s: i.~satisfiedBy | i.fulfills = s.~system }
40
41 -- Trace@BMPF
42

```

Tarski Traceability View:

The traceability view displays a network of entities and their relationships:

- Specification** is connected to **ContractRequirement** via **contract**.
- ContractRequirement** is connected to **SystemRequirement** via **system**.
- SystemRequirement** is connected to **Code** via **fulfills**.
- SystemRequirement** is connected to **SystemRequirement2** via **refines**.
- SystemRequirement** is connected to **SystemRequirement1** via **requires**.
- SystemRequirement** is connected to **Model** via **satisfies**.
- Model** is connected back to **SystemRequirement** via **fulfills**.

A context menu is open over the **Management** node, showing options like Refresh, Zoom In, Zoom Out, Zoom to Fit, and Export to PNG or PDF.

Outline

1 Introduction

- Motivation
- Industrial Use Cases

2 Approach

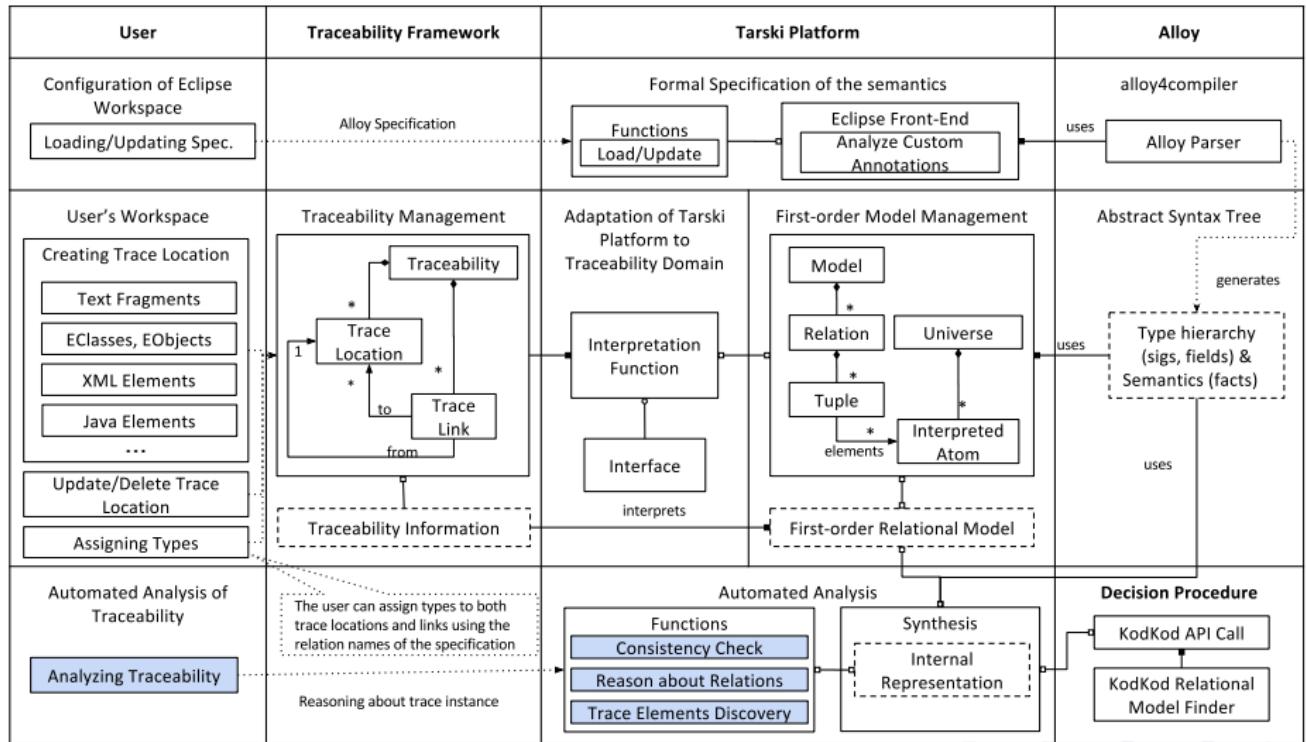
- Traceability Domain Model
- First-order Relational Model and Logic
- Type Annotation and Trace-Relations
- Formal Semantics and Automated Analysis

3 Demonstration

- Formal Specification of Traceability Semantics
- Traceability Management
- First-order Model Management
- Automated Analysis of Traceability**

4 Conclusion and Future Work

Reasoning about Trace-instance



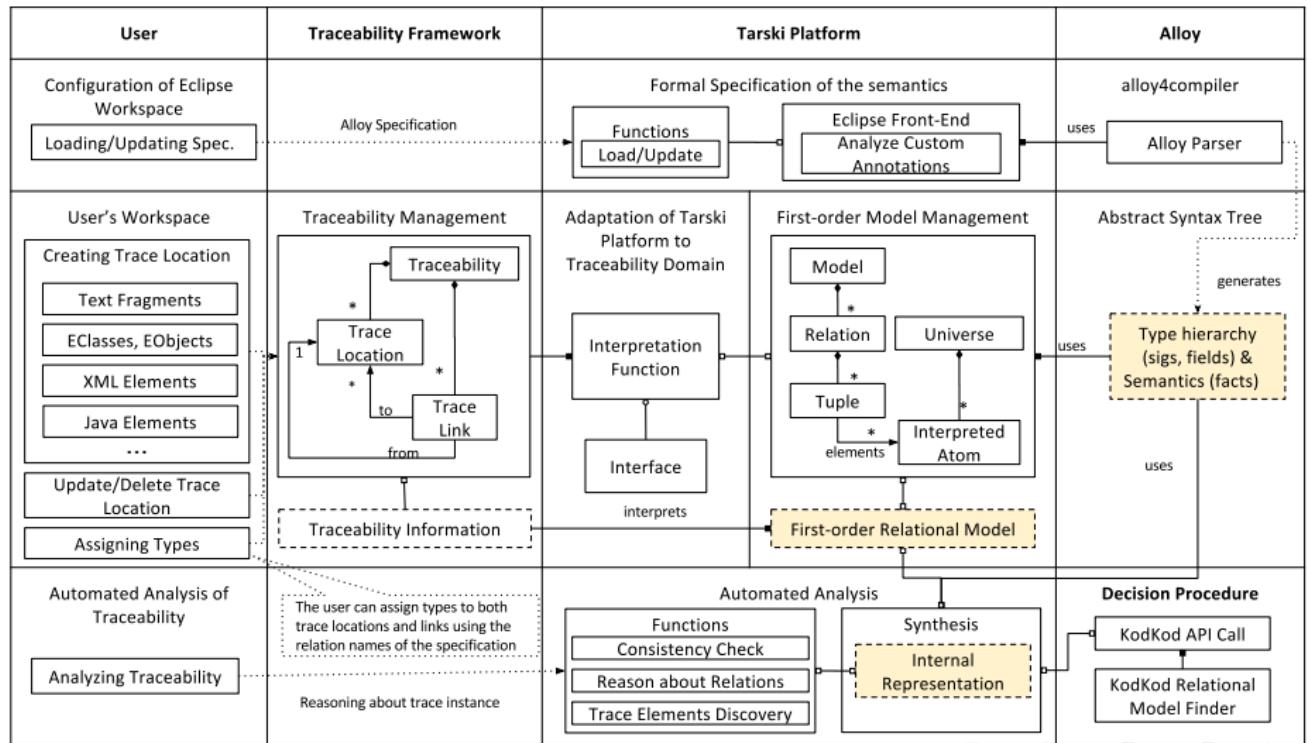
Automated Analysis of Traceability

The screenshot shows the Eclipse Platform interface with several open windows:

- Left Window:** ApplicationLifecycleAnalysis.mw (Specification Editor). It displays a UML-like class diagram with nodes like Specification, ContractRequirement, SystemRequirement, and Code, connected by various relationships.
- Middle Window:** Customer Requirements Specification.md (Specification Editor). It contains a text-based specification document with numbered steps and annotations. Step 14 is highlighted with a red border.
- Bottom Window:** Tarski Traceability View (Diagram Editor). It shows a detailed traceability graph with nodes and relationships between them. A context menu is open over the graph, with the "Reason on" option selected.

The overall interface is a composite of multiple Eclipse plug-in windows, illustrating the integration of formal specification and automated traceability analysis.

Synthesis of Internal Representation



- Should we consider also the **temporal behavior** of the traceability? Interesting analysis scenarios exist in industry
- We are not supporting **ordered sets** of Alloy which usually help model the dynamic behaviour.
- **First-order theory of relations** might be a candidate for traceability in Multi-paradigm Modeling for Cyber-physical Systems. Preliminary results show that the approach works on the synchronization of design rules with design/installation of physical components.
- However, DPLL(T) solvers do not currently exist for this fragment of the theory.
- Alloy Language is too expressive for the domain of traceability. We're working on the formalization of a **First-order theory for traceability** and the development of a **domain-specific language** for traceability.

Modeling and Reasoning Approaches

