

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"



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)

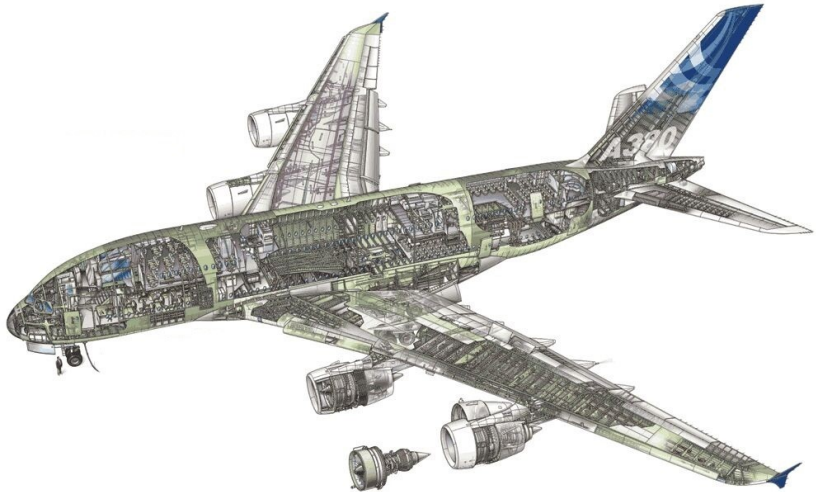
- Traceability Semantics is often statically defined.
- The semantics cannot be easily adapted for the needs of different projects.
- Different traceable elements and the relation types exist in industrial settings,
- Likewise, different traceability analysis scenarios exists. Several industries demands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

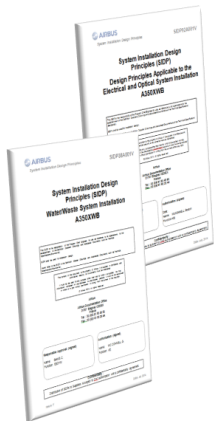
Airbus Group Innovations

System Installation Design Principles



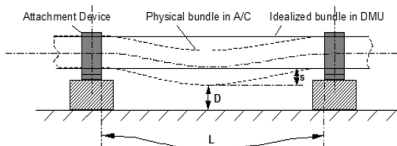
Airbus Group Innovations

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:



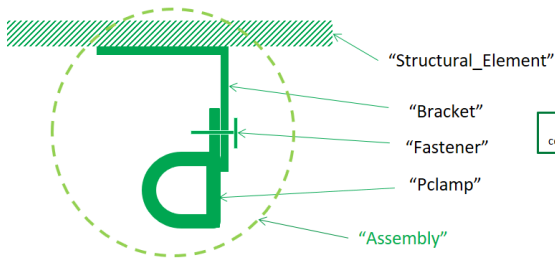
- s... Sagging of bundle (real behavior of physical bundle in A/C due to gravity, ageing, etc.)
- D... Required Distance
- L... Actual length of a bundle segment between two Attachment Points (as designed in DMU)

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.

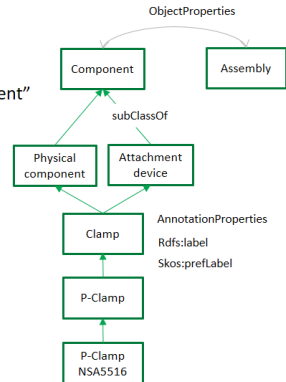
Airbus Group Innovations

System Installation Design Principles



"P-clamp NSA5516 can be fixed on X with Y"

"Physical component" "Standard reference"



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

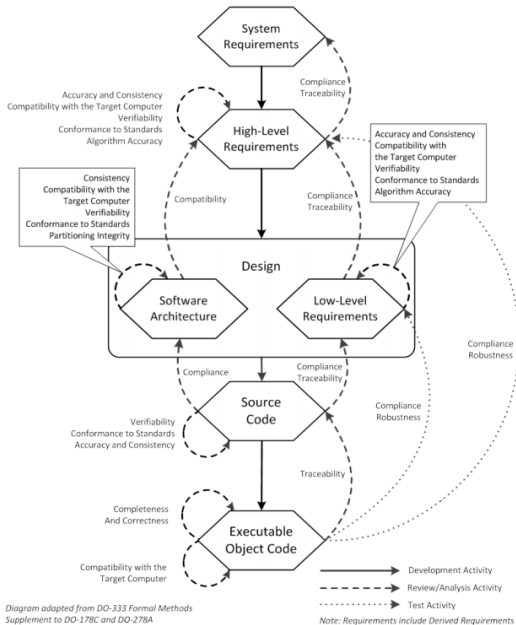


Diagram adapted from DO-333 Formal Methods Supplement to DO-178C and DO-278A

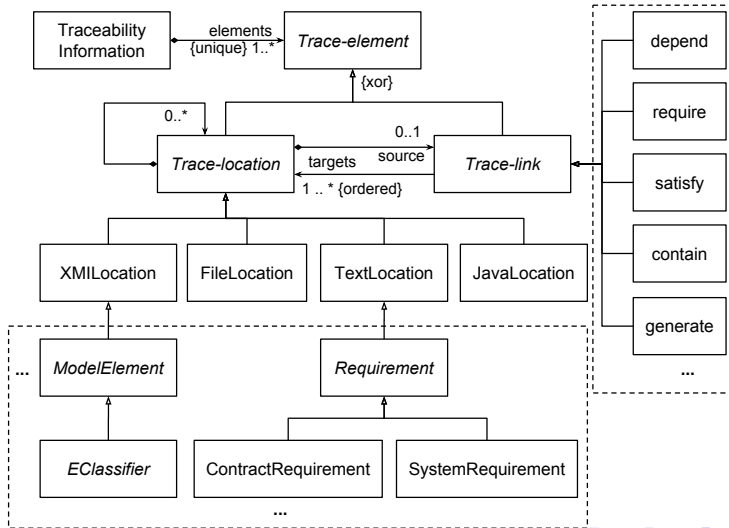
Note: Requirements include Derived Requirements



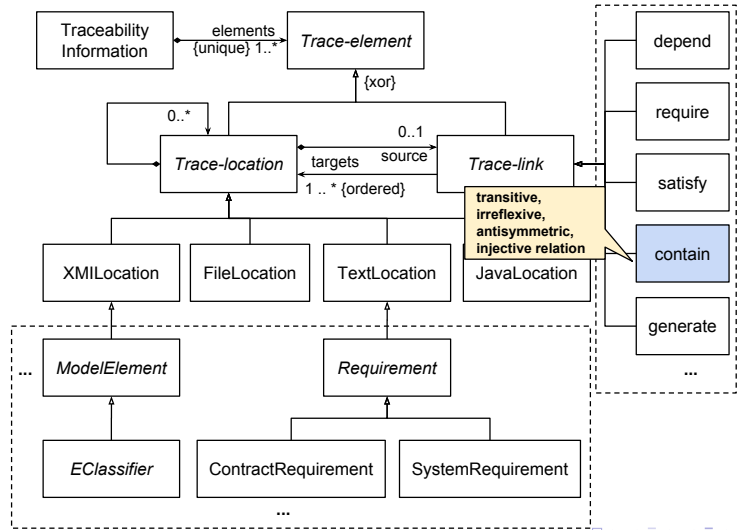
Outline

- 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

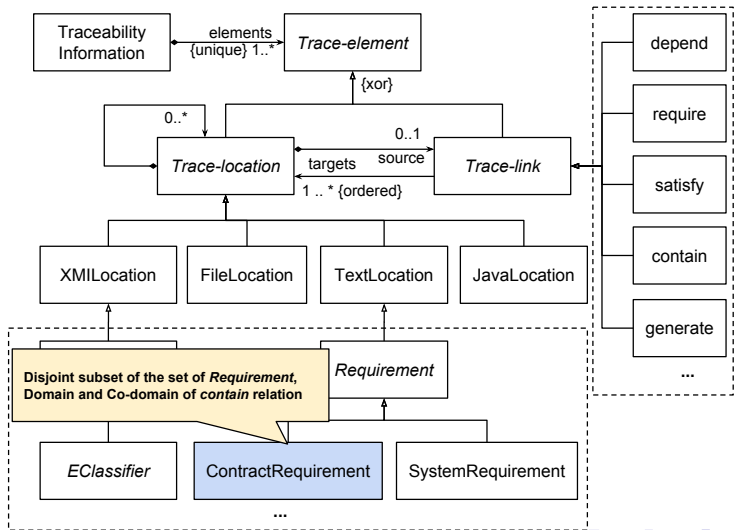
A conceptual model for traceability and its extension



Semantics of *contain* relation (represents decomposition)



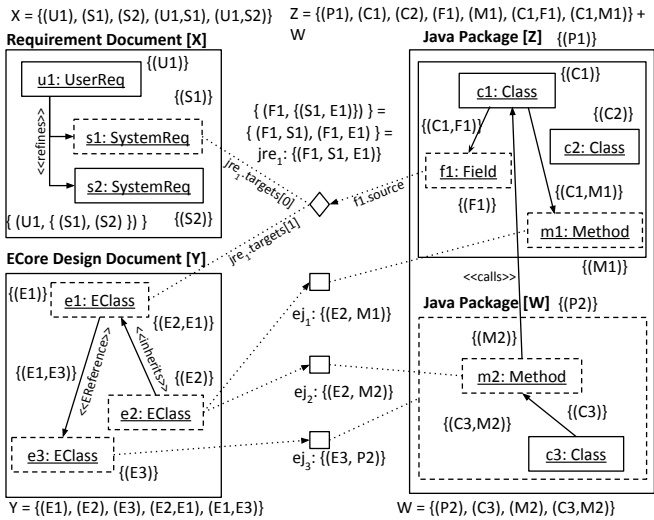
Semantics of *ContractRequirement*



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



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

$$E.R_{EJ} = \{\langle E_1 \rangle, \langle E_2 \rangle, \langle E_3 \rangle\} \cdot \{\langle E_2, M_1 \rangle, \langle E_2, M_2 \rangle, \langle E_3, P_2 \rangle\}$$

$$= \{\langle M_1 \rangle, \langle M_2 \rangle, \langle P_2 \rangle\}$$

$$J. \sim R_{EJ} = \{\langle F_1 \rangle, \langle M_1 \rangle, \langle M_2 \rangle, \langle P_2 \rangle\} \cdot \{\langle M_1, E_2 \rangle, \langle M_2, E_2 \rangle, \langle P_2, E_3 \rangle\}$$

$$= \{\langle E_2 \rangle, \langle E_3 \rangle\}$$

* $\hat{\ }$ (Reflexive) Transitive Closure

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

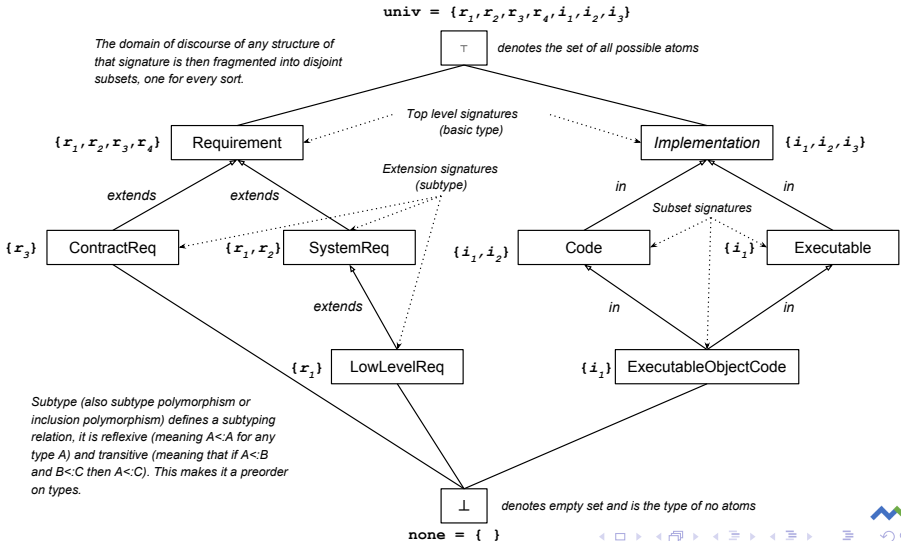
Domain and Range Restrictions

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

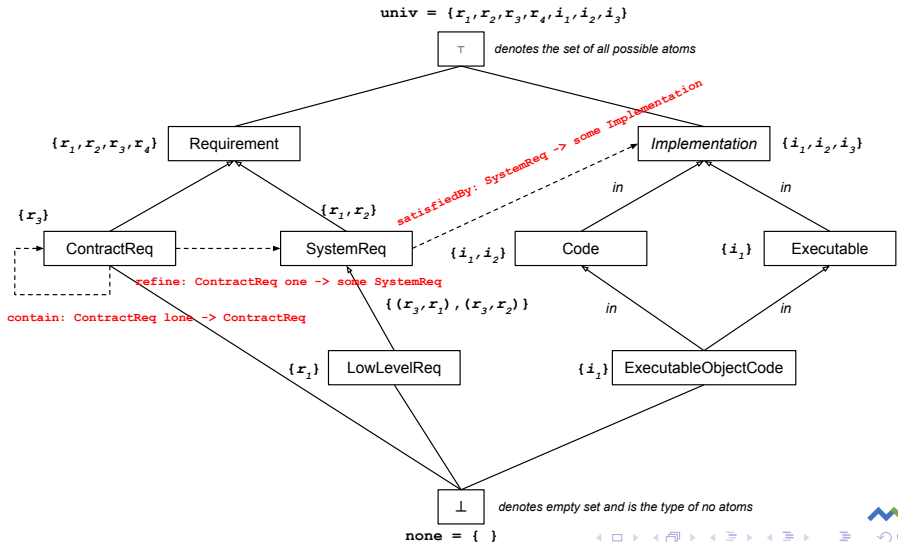
Outline

- 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 U Component)}
27
28 -- Semantics@SystemRequirement.satisfiedBy
29 fact { $\forall$  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

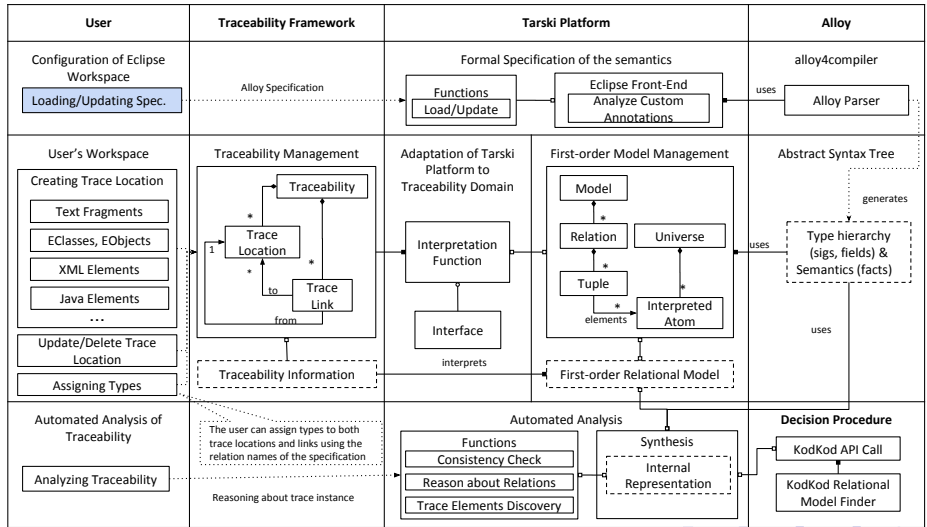
```
30 -- Reason@ContractRequirement.system
31 fact { $\forall$  s: SystemRequirement, s': s.*~refines |
32     s'.~system = s.~system}
33
34 -- Reason@SystemRequirement.requires
35 fact {  $\forall$  s, s': SystemRequirement |
36     s' in s.refines  $\implies$  s in s'.requires }
37
38 -- Reason@Implementation.fulfills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



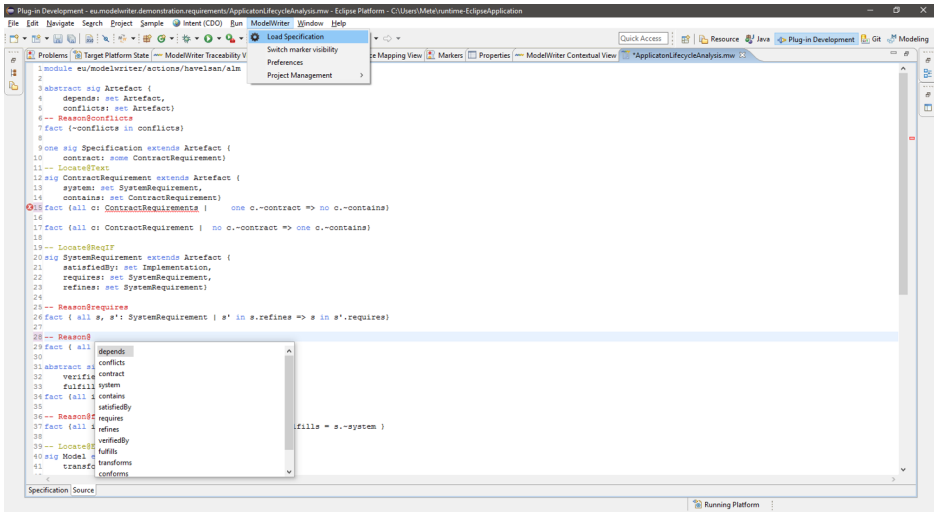
Outline

- 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

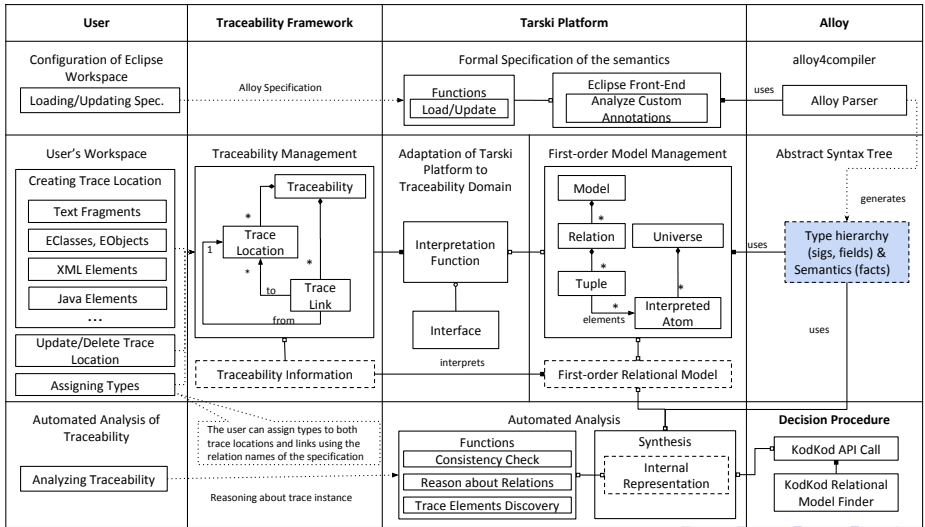
Configuration of User's Workspace



Configuration of User's Workspace



Type Hierarchy from the Specification



Type Hierarchy from the Specification

The screenshot shows the Eclipse IDE environment. On the left, a code editor displays a Tarski specification for `ApplicationLifecycleAnalysis.mw`. The code defines an `Artefact` abstract type and several concrete types like `ContractRequirement`, `SystemRequirement`, and `Implementation`, along with their relationships and constraints.

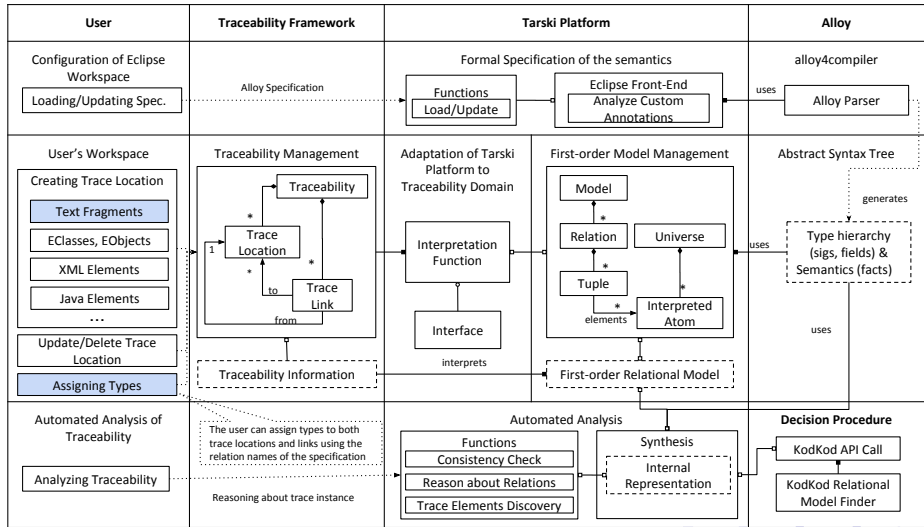
In the center, the 'Sets and Relations' dialog box is open. It shows a tree view of sets: `universe` (containing `Artefact (abs)`, `Specification`, `ContractRequirement`, `SystemRequirement`, `Implementation (abs)`, `Model`, `Code`, `Component`), and `Verification (abs)` (containing `Simulation`, `Analysis`, `Test`). The 'Relations' list on the right contains 12 entries, such as `depends: Artefact -> set of Artefact` and `fulfills: Implementation -> lone of ContractRequirement`.

On the right, a 'Tarski Traceability View' displays a graph of relationships between elements. Nodes include `ContractRequirement0`, `ContractRequirement2`, `SystemRequirement0`, `SystemRequirement1`, `SystemRequirement2`, and `Model`. Edges represent relationships like `contains`, `requires`, `fulfills`, and `satisfies`.

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*

Plug-in Development - eu.modelwriter.demonstration.requirements/Custom Requirements Specification.md - Eclipse Platform - C:\Users\Mete\nruntime-EclipseApplication\havelan

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

Quick Access Resource Java Plug-in Development Git Modeling

```

ApplicationLifecycleAnalysis.mw
1 module eu.modelwriter/actions/havelan/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: some ContractRequirement
11  -- Locate@Text
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.}
19  fact {all c: ContractRequirement | no c.-contract => one c.}
20 }
    
```

Customer Requirements Specification.md

```

1 # Customer Requirements Specification
2
3 ## UC-1 Create a new SpecObject
    
```

Specification.java

```

req10.ecore
1 The Specification Editor is the main interface for users. Therefore,
2 SpecObjects in this editor is the main success scenario.
3
4 Location
5   exist and is open.
6
7 Success Scenario
8   one that a Specification exists and is open (not required for alternative
9
10 Row's context menu (or in the empty editor space)
11   the Child or Sibling submenu
12   the desired SpecObject Type (or none) from the submenu.
13   results in a new SpecHierarchy being created that is linked to a newly
14   SpecObject with the correct type
    
```

Create a Trace Element with Type

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

contains: 1
 contract: 2
 fulfills: 2
 refines: 2
 requires: 2
 satisfiable: 2
 system: 3

Running Platform Writable Insert 9:6

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

The screenshot shows the Eclipse IDE with the following components:

- Editor 1 (Left):** Contains the source code for `Application/ifecycleAnalysis.mw`. It defines an `Artefact` abstract type and a `Specification` class that extends it. The `Specification` class includes methods for `locateText`, `discoverContractRequirement`, and `semanticsContractRequirement`. It also contains several `fact` statements defining relationships between `ContractRequirement` and `SystemRequirement`.
- Editor 2 (Right):** Shows the `Customer Requirements Specification` with a user story: "UC-1 Create a new SpecObject". Below it, a note states: "Note that the Specification Editor is the main interface for users. Therefore, creating SpecObjects in this editor is the main success scenario." A red box highlights the text: "Specification exists and is open (not required for alternative)".
- Dialog Box (Center):** Titled "Create a trace relation", it lists "Relations" for the selected trace element `SystemRequirement$0`. The relations are:
 - depends: Artefact -> set of Artefact
 - conflicts: Artefact -> set of Artefact
 - satisfiedBy: SystemRequirement -> set of Implementation
 - requires: SystemRequirement -> set of SystemRequirement
 - refines: SystemRequirement -> set of SystemRequirement
- Diagram (Bottom):** A traceability graph showing nodes for `ContractRequirement1`, `ContractRequirement2`, `SystemRequirement2`, `SystemRequirement1`, and `Code`. Edges represent relationships: `contract` (between `ContractRequirement1` and `ContractRequirement2`), `system` (between `ContractRequirement2` and `SystemRequirement2`), `contains` (between `SystemRequirement2` and `SystemRequirement1`), `fulfills` (between `Code` and `SystemRequirement2`), `satisfies` (between `Code` and `SystemRequirement1`), `requires` (between `SystemRequirement2` and `SystemRequirement1`), and `refines` (between `SystemRequirement2` and `SystemRequirement1`).

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

The screenshot shows the Eclipse IDE interface. The main editor displays a file named 'Customer Requirements Specification.md' with the following content:

```

1 # Customer Requirements Specification
2
3 ## UC-1 Create a new SpecObject
4
5 Note that the Specification Editor is the main interface for users. Therefore,
6 creating SpecObjects in this editor is the main success scenario.

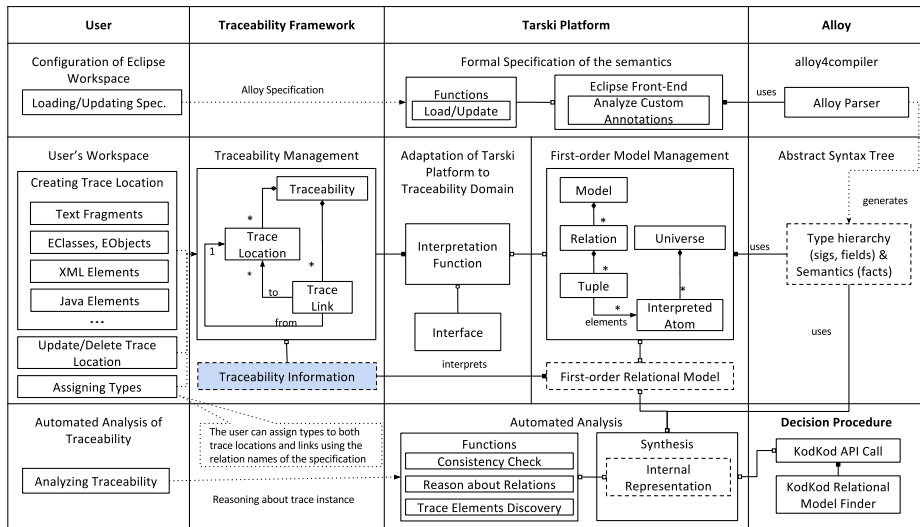
```

The 'Markers' dialog is open, showing a tree view of the project structure. The selected item is 'reqif10.ecore'. The dialog also shows a list of files containing markers, including 'Specification.java' and 'interface Specification extends SpecElementWithAttributes (Code50)'. The 'Show only files that contain Marker(s)' checkbox is checked.

Below the dialog, a diagram shows relationships between 'Code', 'SystemRequirement2', and 'SystemRequirement1' elements. The relationships are labeled as follows:

- Code fulfills SystemRequirement2
- Code fulfills SystemRequirement1
- SystemRequirement2 requires SystemRequirement1
- SystemRequirement2 refines SystemRequirement1
- SystemRequirement1 requires SystemRequirement2
- SystemRequirement1 refines SystemRequirement2
- SystemRequirement2 system SystemRequirement1
- SystemRequirement1 system SystemRequirement2
- SystemRequirement2 contains SystemRequirement1
- SystemRequirement1 contains SystemRequirement2

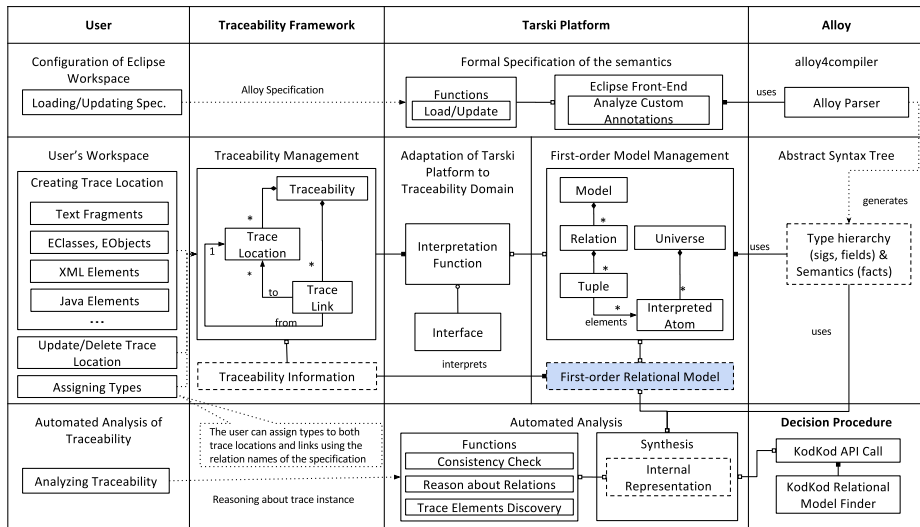
Traceability Information



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

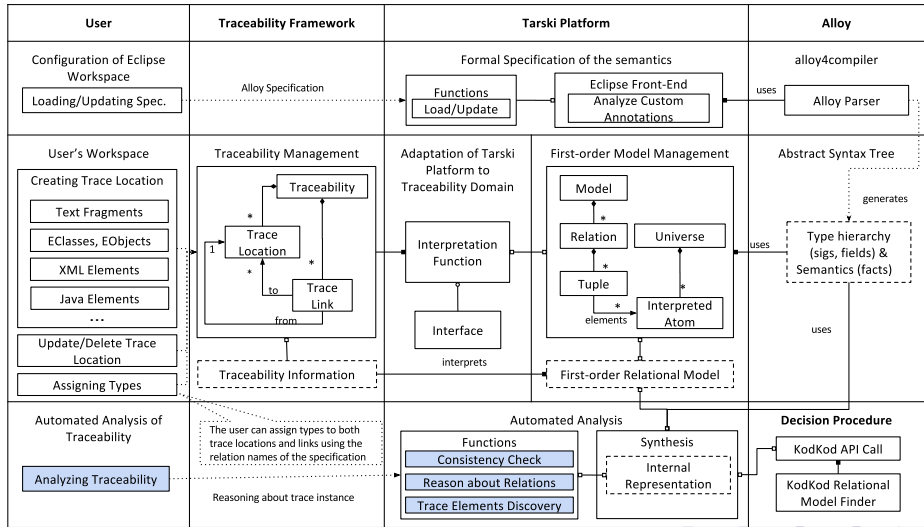
The screenshot displays the Eclipse IDE interface. On the left, the 'ApplicationLifecycleAnalysis.mw' file is open, showing a UML-like modeler code with classes like `Artefact`, `Specification`, `ContractRequirement`, `SystemRequirement`, and `Implementation`. The code includes dependencies, conflicts, and semantic rules.

On the right, the 'Tarski Traceability View' is active, showing a traceability graph. The graph consists of nodes: `Specification`, `ContractRequirement`, `ContractRequirement`, `SystemRequirement0`, `Code`, `SystemRequirement2`, `SystemRequirement1`, and `Model`. Edges represent relationships such as `contract`, `system`, `fulfills`, `satisfiedBy`, `requires`, `refines`, and `defines`. A context menu is open over the `SystemRequirement1` node, showing options like `Refresh`, `Zoom In`, `Zoom Out`, `Zoom to Fit`, `Export to PNG or PDF`, `Change Type`, `Delete Atom`, and `Map Atom`. A legend box in the top-left of the view lists: `contains: 1`, `contract: 2`, `fulfills: 2`, `refines: 2`, `requires: 2`, `satisfiedBy: 2`, and `system: 3`.

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 displays the Eclipse IDE interface with three main components:

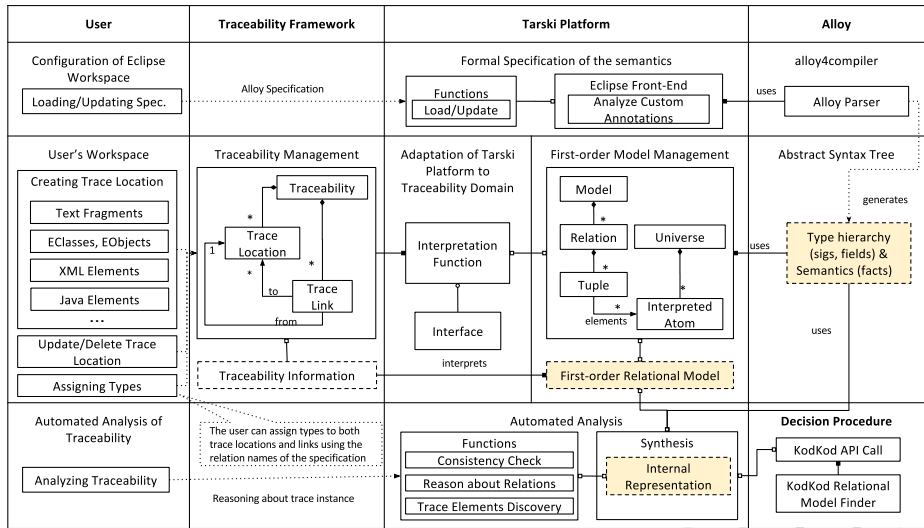
- Left Panel (Code):** Shows the source code for `ApplicationLifecycleAnalysis.mw`. It defines an `Artifact` abstract class and a `Specification` class that extends it. The `Specification` class includes methods for setting contract requirements, locating text, and discovering contract requirements.
- Right Panel (Requirements):** Shows the `Customer Requirements Specification.md` file. It contains a user story `UC-1 Create a new SpecObject` with a precondition and a main success scenario. The scenario steps are: 1. We assume that a Specification exists and is open (not required for alternative scenario); 2. Open a row's content menu (or in the empty editor space); 3. Select the Child or Sibling submenu.
- Bottom Panel (Tarski Traceability View):** Displays a traceability graph. The graph shows relationships between various elements:
 - Specification** (yellow box) is connected to **ContractRequirement1** and **ContractRequirement0** via `contract` relationships.
 - ContractRequirement0** is connected to **SystemRequirement0** and **ContractRequirement2** via `system` relationships.
 - SystemRequirement0** is connected to **SystemRequirement2** and **SystemRequirement1** via `requires` relationships.
 - SystemRequirement2** is connected to **SystemRequirement1** via `requires` relationships.
 - Code** (yellow box) fulfills **SystemRequirement2** and **SystemRequirement1**.
 - ContractRequirement2** fulfills **SystemRequirement1**.
 - SystemRequirement1** fulfills **ContractRequirement2**.

A context menu is open over the graph, showing options: Management, Analysis (selected), Refresh, Zoom In, Zoom Out, Zoom to Fit, and Export to PNG or PDF. The `Analysis` menu is further expanded to show: Check Consistency, Reason on Selection (highlighted), Discover Atoms, and Clear All Reasoned Tuples.

On the left side of the graph, a summary box indicates:

- contains: 1
- contract: 2
- fulfills: 2
- refines: 2
- requires: 2
- satisfiedBy: 2
- system: 3

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 shows that the approach works on the synchronization of design rules with design/installation of physical components.
- However, DPLL(T) solvers does not currently exists 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

