Model transformations, traceability and provenance

Perdita Stevens

School of Informatics
University of Edinburgh
My cat is model-driven

Model transformations are a key element of model driven development (MDx)
but not yet as well understood as they need to be, especially fundamentally.

Provenance seems also to be fashionable...

Traceability is very old-hat.

What are the connections between them all? And are they useful?
For purposes of this talk:
A model is any [formal] artefact relating to a software system.
However, most commonly: a model is a UML model of a system.
That is, a collection of connected *model elements*, which conforms to a set of rules embodied in a *metamodel*.
(The model elements are often represented in diagrams, but that isn’t important here.)
Fundamental idea of MDD

Let models drive development in the sense that a set of related models pervades development (the code is a model) with human ingenuity used to solve relevant problems at each stage but mundane, repetitive aspects of transformations and consistency-maintenance delegated to tools.

Preliminary example: round-trip engineering between UML model and code.
Model transformations

Classically,

\[ T : M \rightarrow N \]

tells us how to get \( n \in N \) from \( m \in M \).

If we now modify \( m \) to \( m' \), we can reapply \( T \) to get a new \( n' \).

But what if we modify \( n \)?

This is where we need \textit{bidirectional} model transformations.
More general view of model transformations

Assume we have two sets of models $M$ and $N$ with some consistency relation $C$ and assume “no-information” models $\epsilon_M, \epsilon_N$

A model transformation is some algorithmic way to restore consistency, i.e. given $(m, n) \notin C$, replace with $(m', n') \in C$.

Unidirectional $M \rightarrow N$: always $m = m'$

Bidirectional: can modify both models, maybe only one at a time.
A basic formal framework

Let $M$ and $N$ be sets of models to be related. A transformation between $M$ and $N$ consists of:

- a relation $C \subseteq M \times N$ ("is consistent with")
- a function $\to C : M \times N \to N$ ("propagate forwards")
- a function $\leftarrow C : M \times N \to M$ ("propagate backwards")

There are obvious engineering advantages to being able to specify all three with one text, but for what follows we don’t need to stipulate that.

Let’s make explicit our implicit assertions, and then explore what further conditions we may require.
Implicit assertions

1. Transformations are functions, i.e., deterministic – because it will not be acceptable for tools to interpret the same transformation text on the same models differently.

2. Transformations may need to look at both models, not just one – otherwise we will be limited to bijective transformations.
Traceability (says Wikipedia)

In software development, the term traceability (or requirements traceability) refers to the ability to link requirements back to stakeholders’ rationales and forward to corresponding design artifacts, code, and test cases. Traceability supports numerous software engineering activities such as change impact analysis, compliance verification of code, regression test selection, and requirements validation. It is usually accomplished in the form of a matrix created for the verification and validation of the project. Unfortunately the practice of constructing and maintaining a requirements trace matrix [RTM] can be very arduous and over time the traces tend to erode into an inaccurate state. Alternate automated approaches for generating traces using information retrieval methods have been developed.
### Matrix approach

<table>
<thead>
<tr>
<th>contributes to</th>
<th>n1</th>
<th>n2</th>
<th>n3</th>
<th>n4</th>
<th>n5</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>m4</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Requirements traceability matrices: user requirements, system requirements, architectural design, tests (...)
- Decision structure matrices: design decisions
- Design structure matrices: components

Complications: tree structure.

Mature tool support (e.g. Telelogic DOORS)
Things to note about traceability matrices

1. They record in detail *what* is related – *how* is for humans
2. They are (almost) only useful if very sparse
3. The choice of decomposition is crucial
Provenance (says Wikipedia)

Provenance is the origin or source from which something comes,

[...]

Scientific research is generally held to be of good provenance when it is documented in detail sufficient to allow reproducibility

Provenance is a fundamental principle of archives, referring to the individual, group, or organization that created or received the items in a collection. According to archival theory and the principle of provenance, records of different provenance should be separated.
So what is provenance?

Exact version of transformation (or relevant part?)
– as emphasised in MDD?

Exact version of input model(s) (or relevant part?)
– as emphasised in traceability?

i.e. The process, or the arguments to the process?

Clearly both are needed, but focus differs.

Cheney: “an accurate explanation of a part of the output in terms of parts of the input” incorporates both (?)
Does traceability fit Cheney’s framework?

It depends what we mean by “explanation”.

You can’t completely explain a test by giving the requirement(s) it tests.

Does the software engineering process create a function

\[ f : \text{Requirements} \rightarrow \text{Tests} \]?

In your dreams.
Does MDD fit Cheney’s framework?

Well, *the transformation* explains the output in terms of the input (and the old version of the output)

**Parts?** Depends on the technology.

Approaches based on graph transformations, and QVT-R, make use of “trace” or “correspondence” information, i.e. they link particular model elements in the two models.

But this is not precisely what Cheney’s defn wants, because this is not full information.

E.g., the link may only exist because of the presence *or even absence* of non-linked model elements.
Bidirectional provenance

What would this mean?

Take a step backwards...
Why do we want provenance information, anyway?

- To influence trust
- To apportion blame
- To allow correction? What kind of correction?