Selective Backtracking of Model Changes

Iris Groher and Alexander Egyed (Johannes Kepler University, Linz, Austria)

Why Backtracking?

- Exploring **design alternatives**
- Recovering from **dead ends**
- Rolling back **inconsistencies** (from soft to hard constraints)

Problem

1) traditional mechanisms (e.g. undo or version control) are **chronological** (also undoing unrelated, intermittent changes).
2) recovering a design change requires the recovering of all **logically related design changes**.

Approach

The designer selects model element versions for backtracking and our approach

1) automatically **identifies related design changes** that need backtracking also and
2) **recovers all of them** without having to undo other changes that happened since.

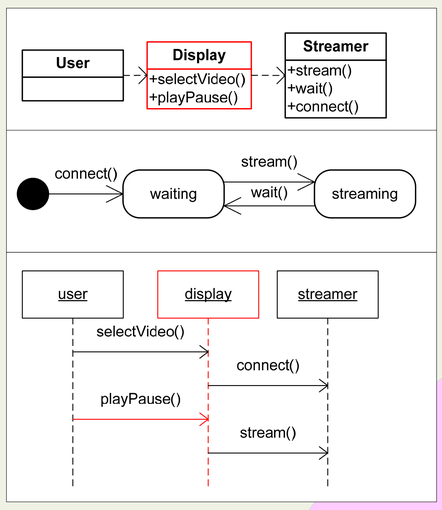
It works for both new and legacy designs.

Pre-Requisite

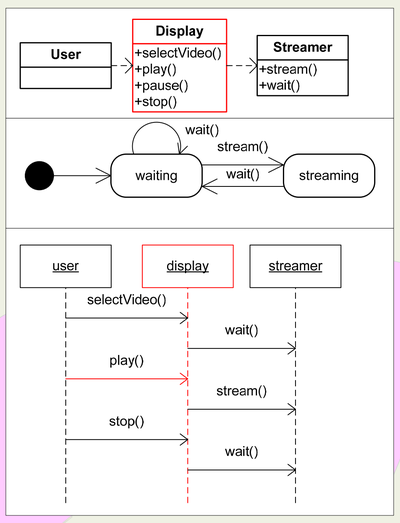
Change History:
We require a model and previously existing versions of model elements.

1) automatically recordable in new designs
2) automatically computable out of version-controlled models for legacy designs

Example: Video-on-Demand System



Version 1

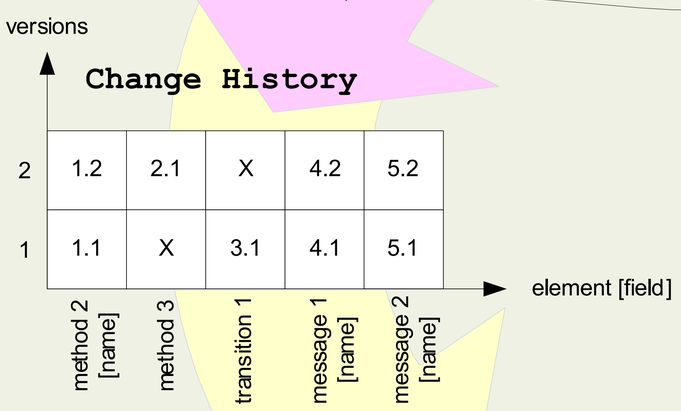


Version 2

Version 1 and Version 2 represent **design snapshots** each with 3 different views (structural, behavioral, scenario). Logical dependencies among model elements in different views are expressed by **model constraints**.

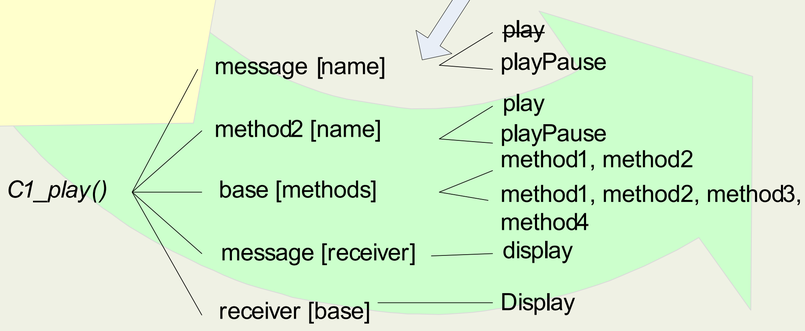
Constraint 1	**Name of message must be declared as method in receiver class** methods = message.receiver.base.methods return (methods -> name -> contains (message.name))
Constraint 2	**Calling direction of object must match association** in = object.base.incomingAssociations out = object.incomingMessages -> sender.base.outgoingAssociations return (in.intersectedWith(out) < > {})
Constraint 3	**Sequence of messages must correspond to events** start = state transitions that correspond to first message return (start -> exists (message sequence equal reachable sequence from start)

Sample Constraints

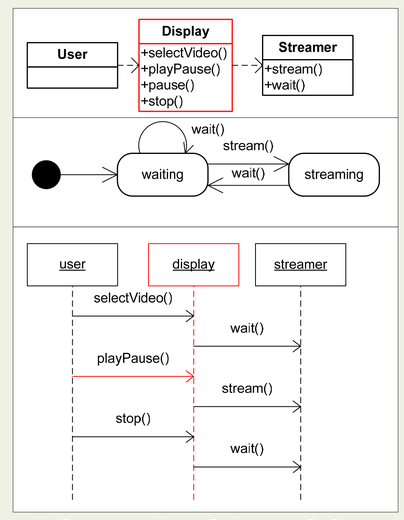


User Input:

backtrack the name of the play() message to playPause().



Constraint 1 (play message) is affected
All versions of model elements that affect the truth value of constraint 1 are considered.



After Backtracking

Selective Backtracking in Principle

The designer initiates the backtracking by selecting the versions of model elements:

Our approach automatically **creates**, **deletes**, or **modifies** the model element(s) selected for backtracking.

	element did not exist	element existed
element still exists	delete	modify
element was deleted	-	create + modify

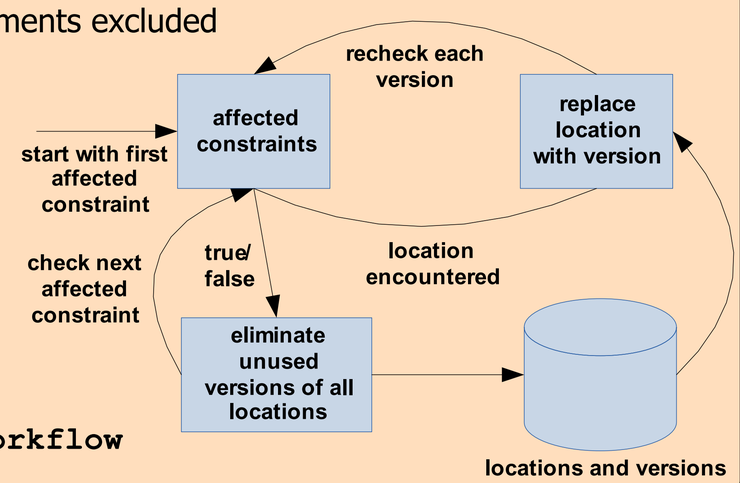
Constraints		After		
		Consistent	Inconsistent	Disposed
Before	Consistent	no problem	problem	no problem
	Inconsistent	no problem	no problem	no problem
	Disposed	no problem	problem	no problem

We compute the **constraints affected by the changes caused during backtracking**. The backtracking is complete if the changes to not cause new inconsistencies.

Our approach investigates every inconsistency caused during backtracking. The goal is to eliminate them by exploring additional model element versions to backtrack.

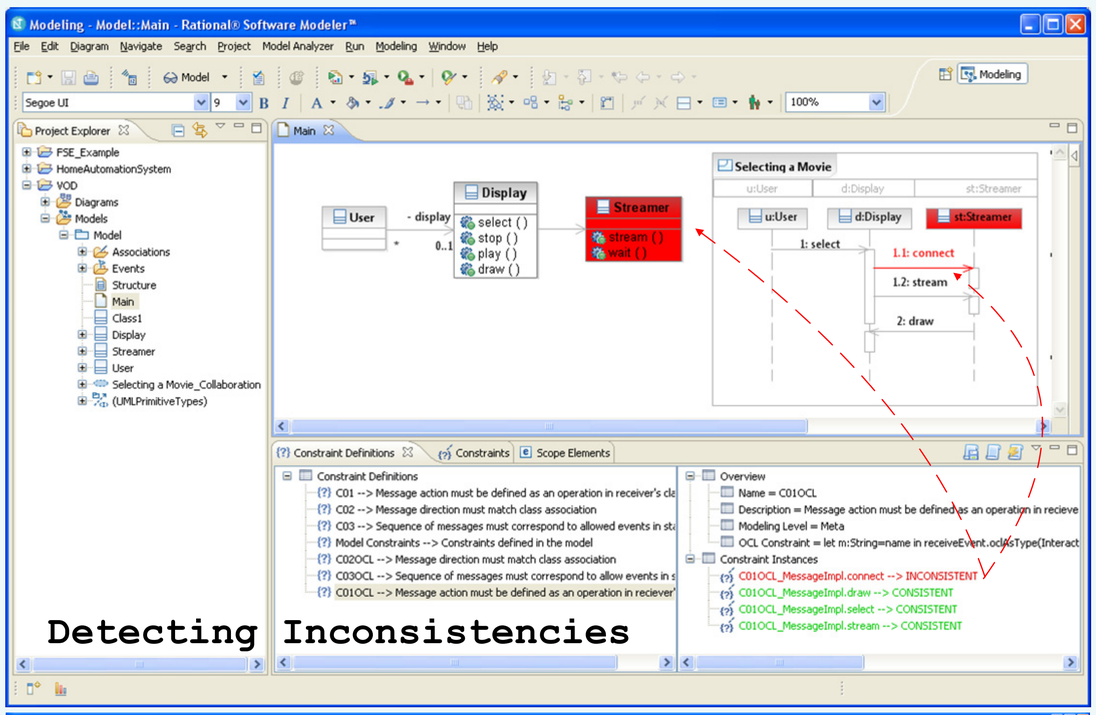
=> explore cross-product of all their versions to find the ones that fix the inconsistency

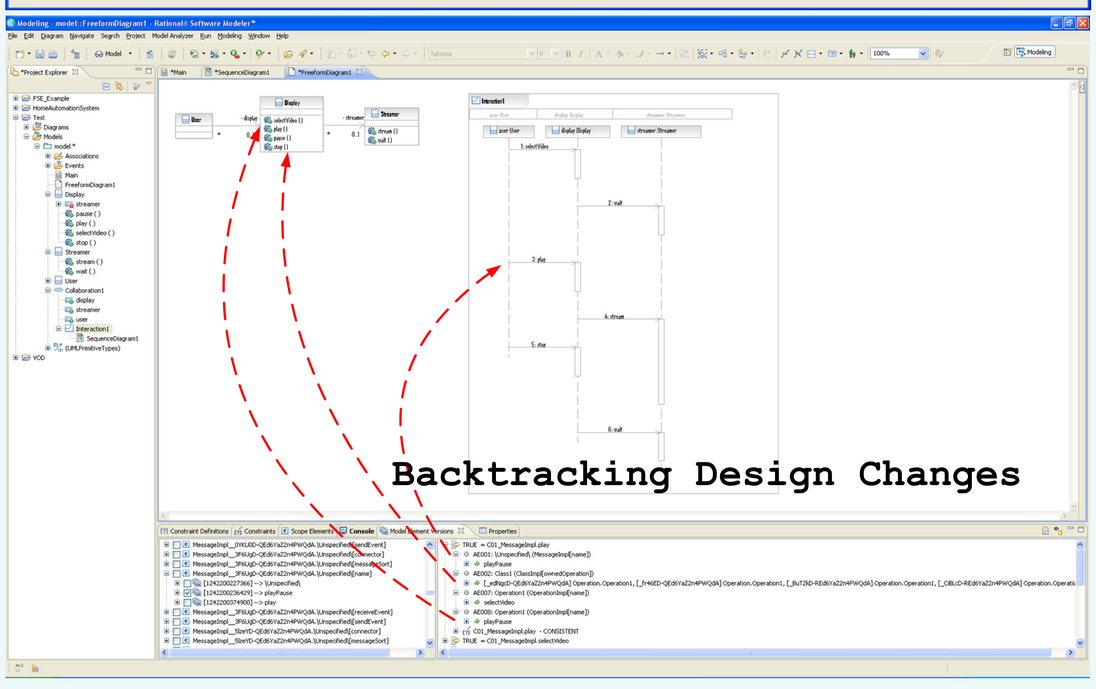
- User selected elements excluded
- Unchangeable elements excluded



Workflow

Model Analyzer Tool





Contact

Institute for Systems Engineering and Automation (SEA)
Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz, Austria
http://www.sea.uni-linz.ac.at/ or Iris.Groher@jku.at

JOHANNES KEPLER UNIVERSITY LINZ
Research and teaching network