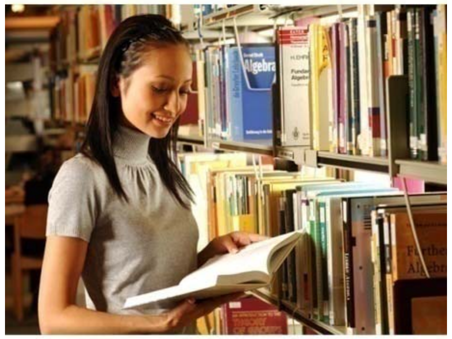
JOHANNES KEPLER UNIVERSITY LINZ | JKU



Fine-Tuning Transformation: Change Propagation

Alexander Egyed

Johannes Kepler University (JKU), Linz, Austria

http://www.sea.jku.at

Who am I?

Current Affiliations:

- Professor at **Johannes Kepler University**, 2008
- Head of **Institute for Systems Engineering and Automation** (~12 Staff Members)
- Research Fellow at **IBM**, 2010

Doctorate Degree:

- University of Southern California, USA 2000 (Dr. Boehm)

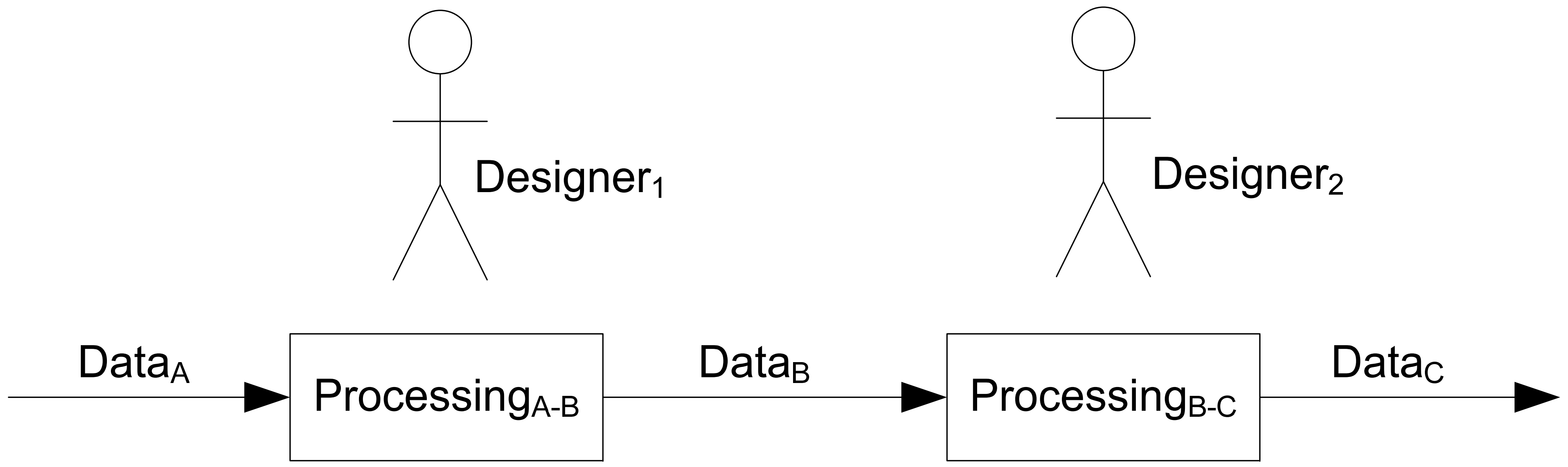
Past Affiliations:

- Research Fellow at University College London, UK 2007
- Research Scientist at Teknowledge Corporation, USA 2000

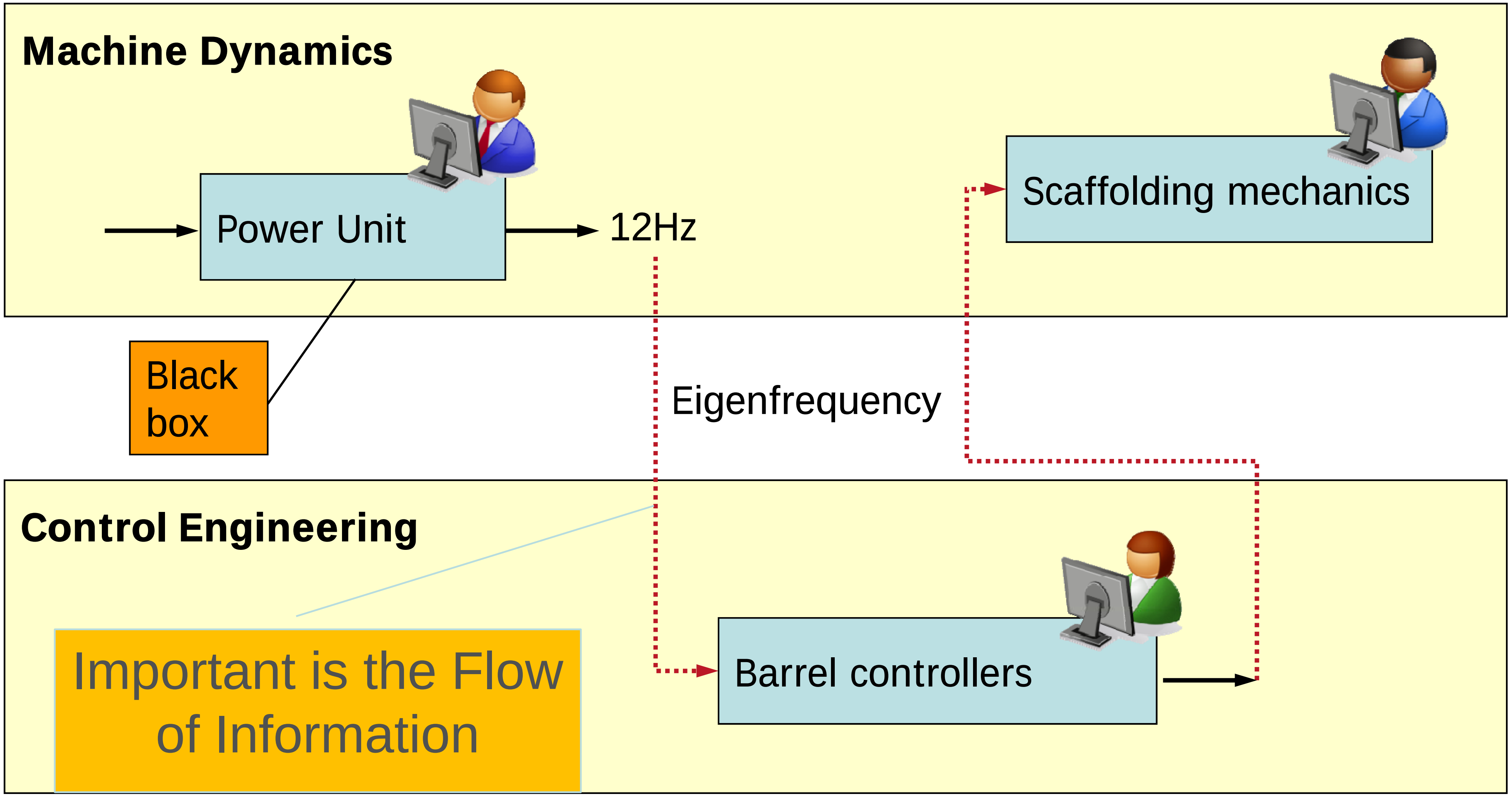
A Bright Future for Transformation

- It is my believe that the future of software modeling hinges on the ability to provide change propagation

Traditional Engineering View of Modeling and Collaboration

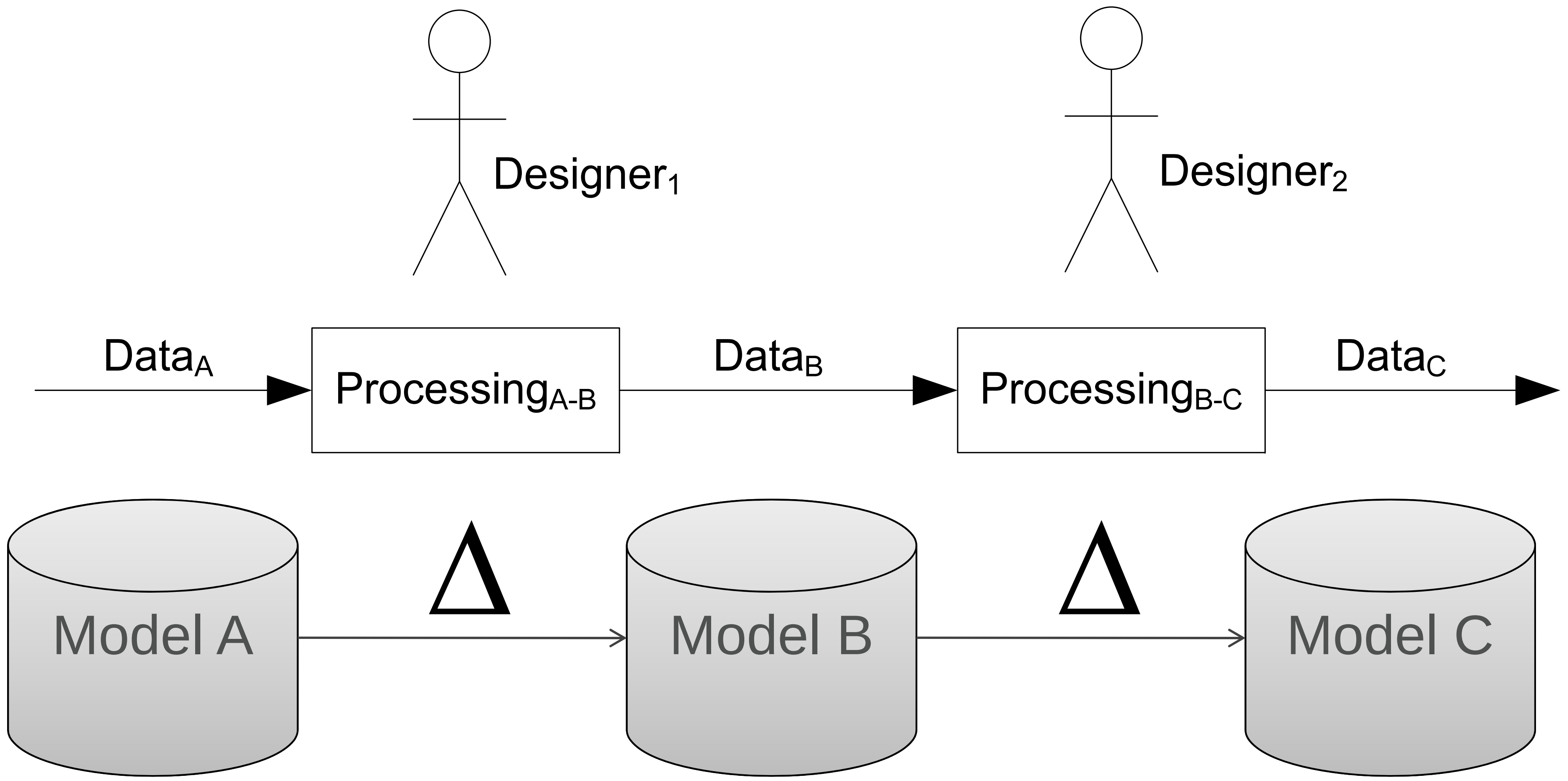


Inter-Disciplinary Collaboration



Important is the Flow of Information

Engineers Collaborating



Models and Data

- Models serve as vehicles for moving data
  - From discipline to discipline
  - From designer to designer
  - From tool to tool
- Data inside models are introduced at some point and consumed later
    - **Not documentation but communication**
    - Sometime in a different syntax or semantics
- The idea: enter a (modeling) fact once only and propagate it to where it is needed

Transformation in the Large

(focus on models)

90-2000 Transformation is about Tool Integration

- There was a “feeling” in the 90s/early 2000
  - The tools are great but they are not connected
  - It is not easy to move information between them
- Goal: if we could just connect these tools then many engineering problems would be eased

Why Transformation in the Large is problematic

(the devil is in the details)

3 Diagrams = 3 Models

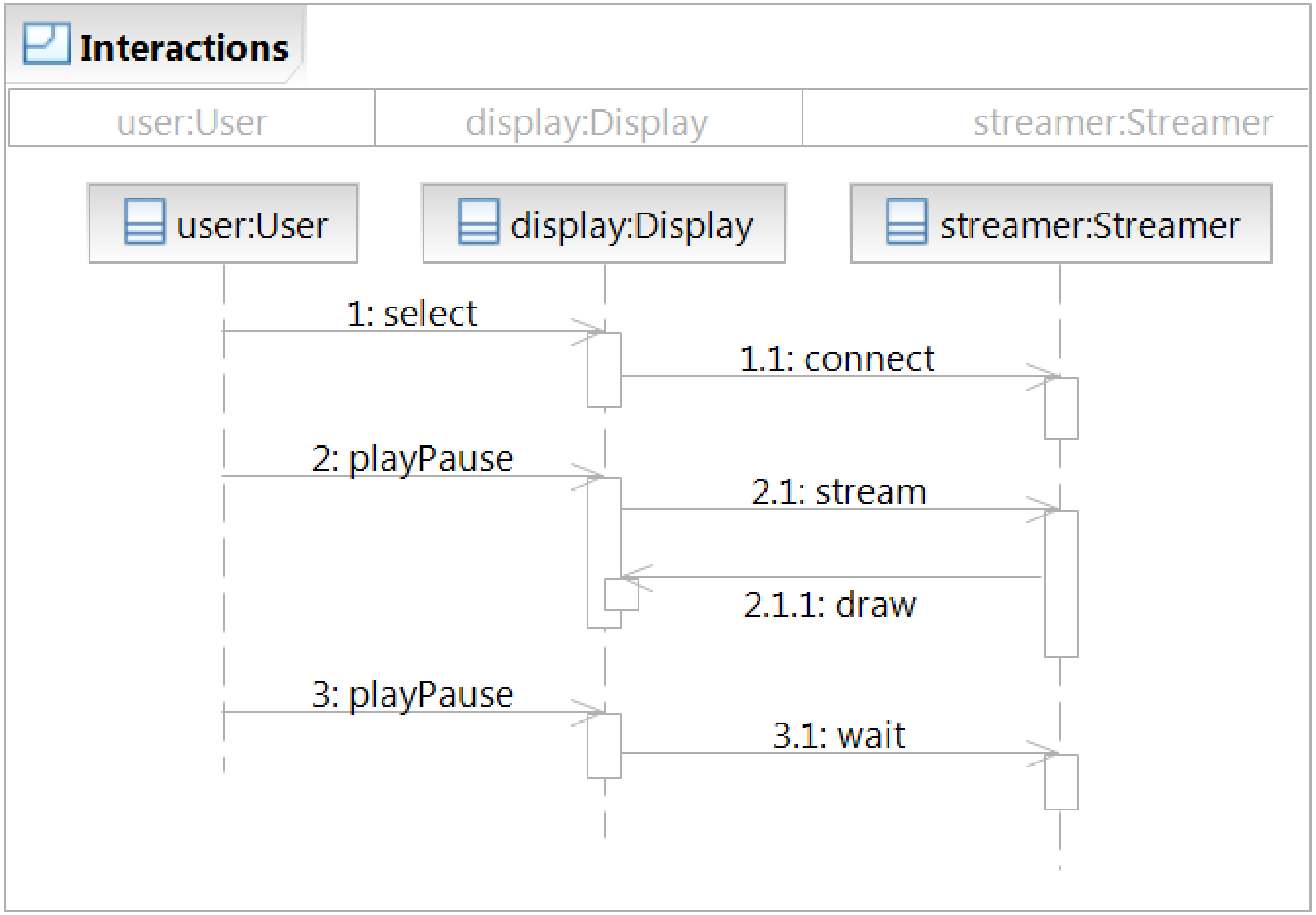
Class diagram

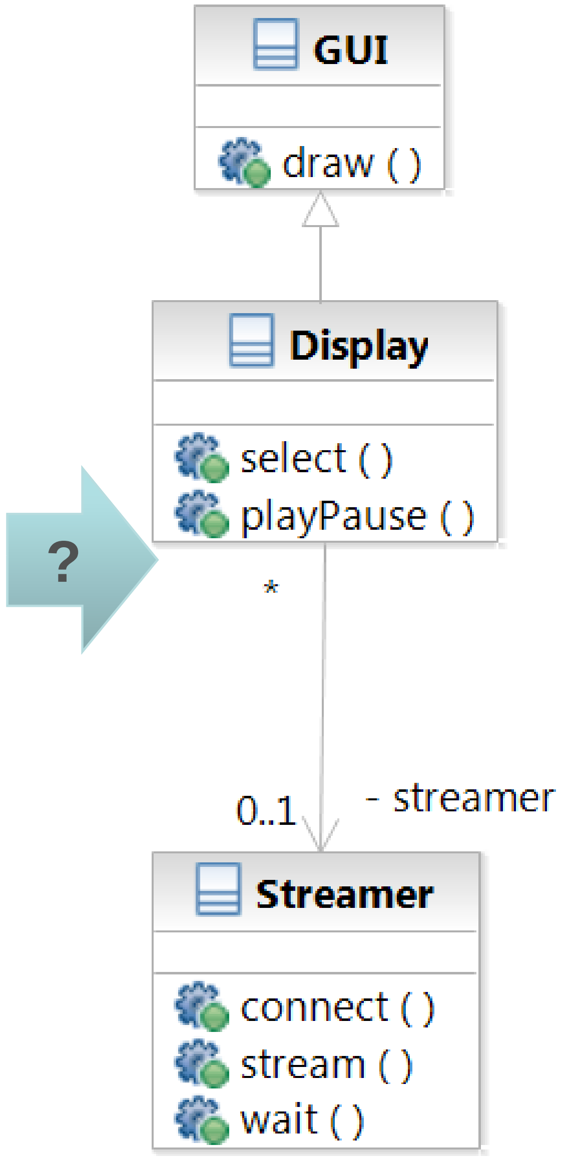
?

Statechart diagram

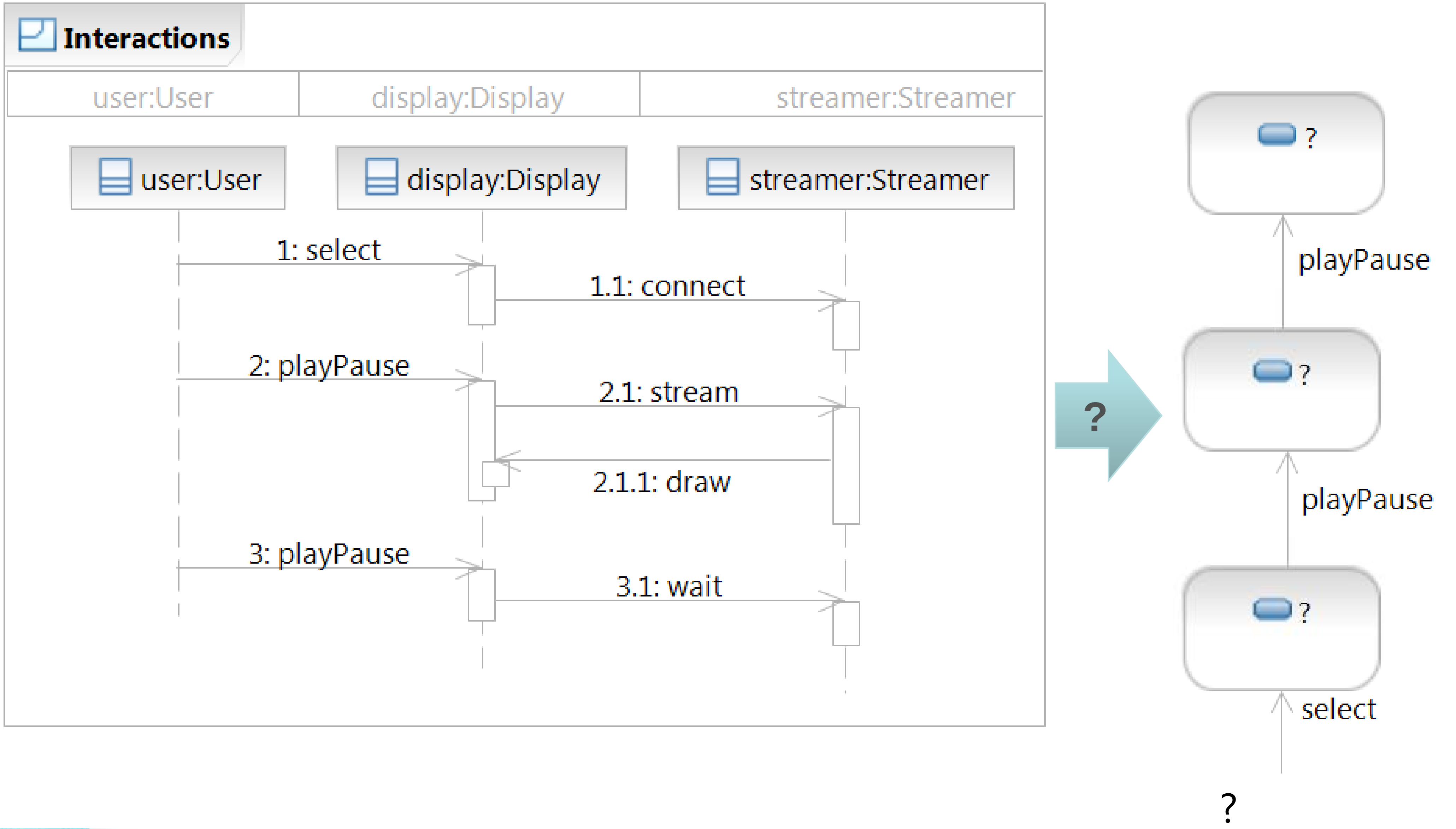
Sequence diagram

Transform a Sequence Diagram into a Class Diagram





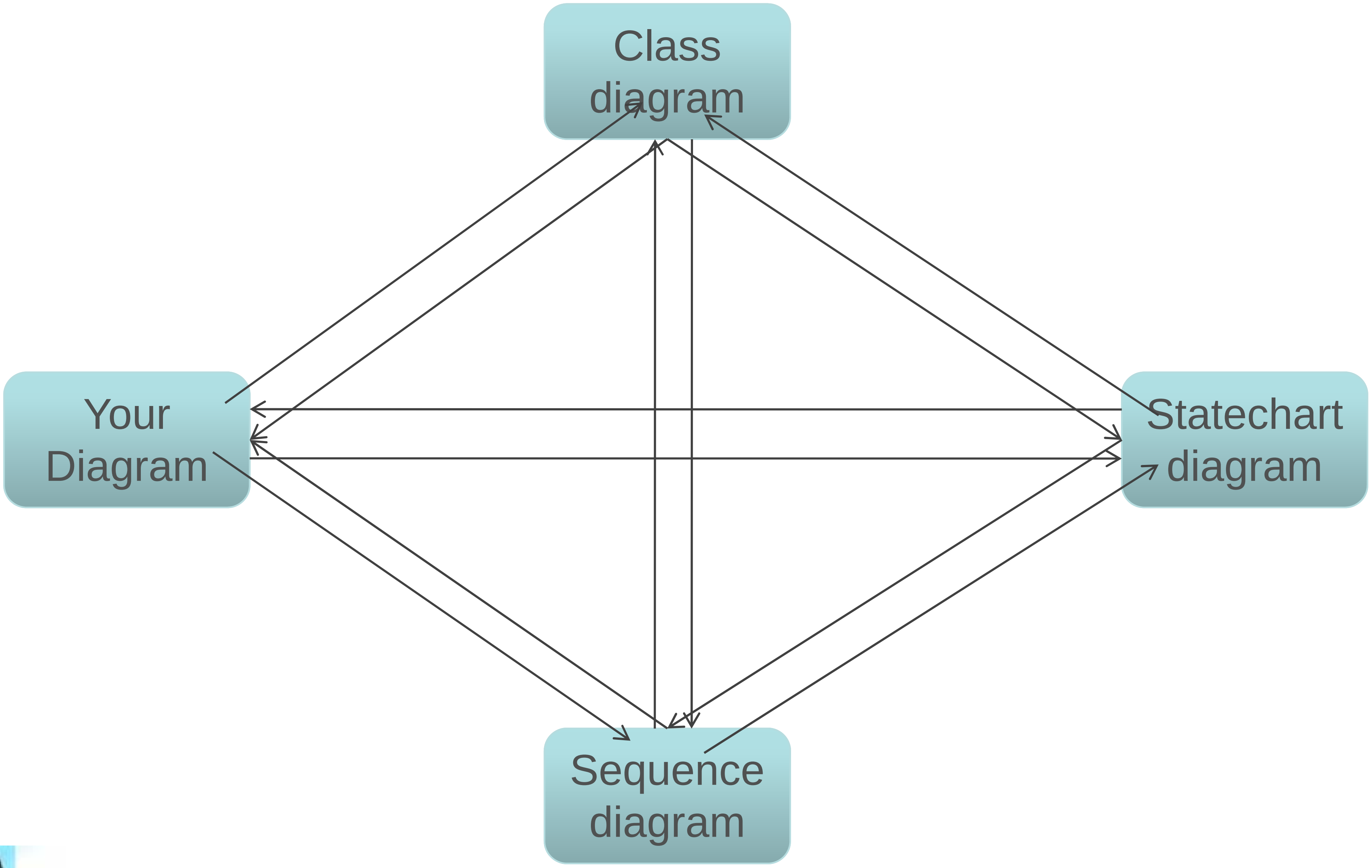
Transform a Sequence Diagram into a Statechart Diagram



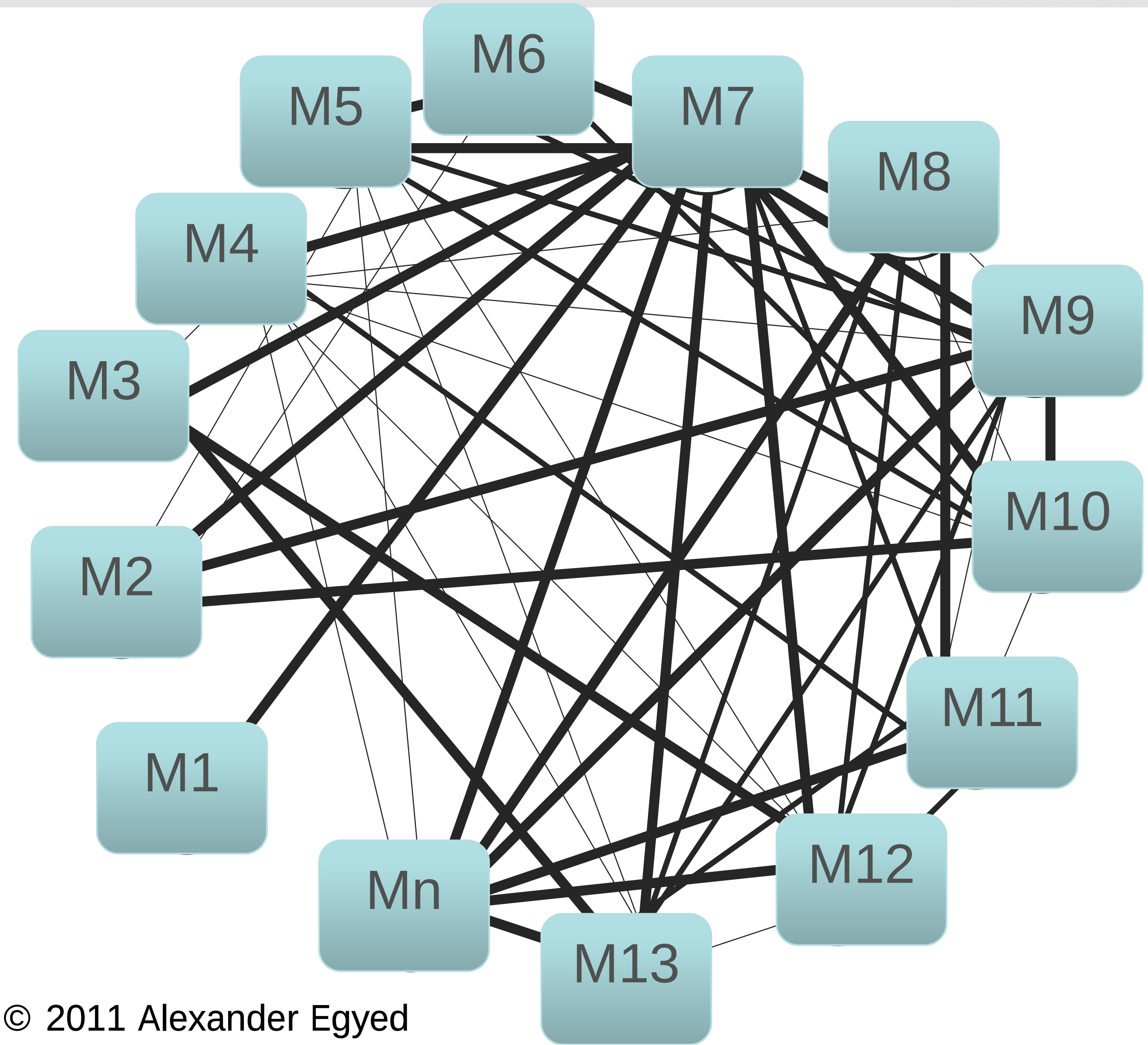
Problems

- **Many assumptions**
- **Many uncertainties**
- Need Bi-Directional Transformation
  - Ability to transform in one direction does not imply ability to transform into the other
- Scalability
  - Transform every model to every other model: n^2

Bi-Directional Transformation



Some Transformations more Comprehensive than Others



Summery

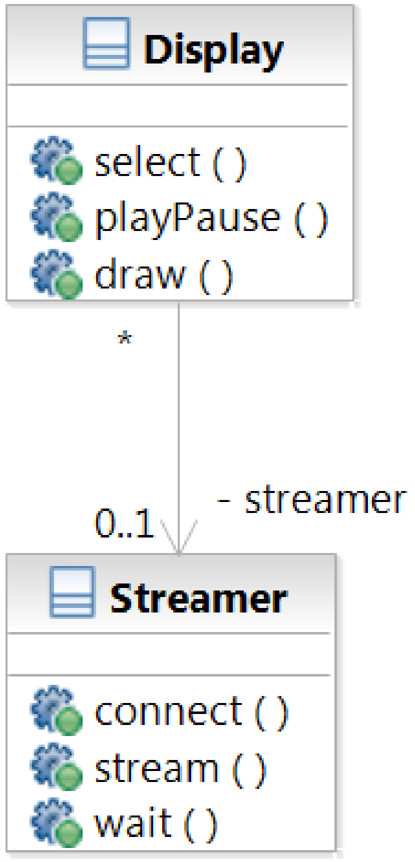
- Clearly, Transformation-In-The-Large is useful for larger tasks
  - Initial (batch) transformation

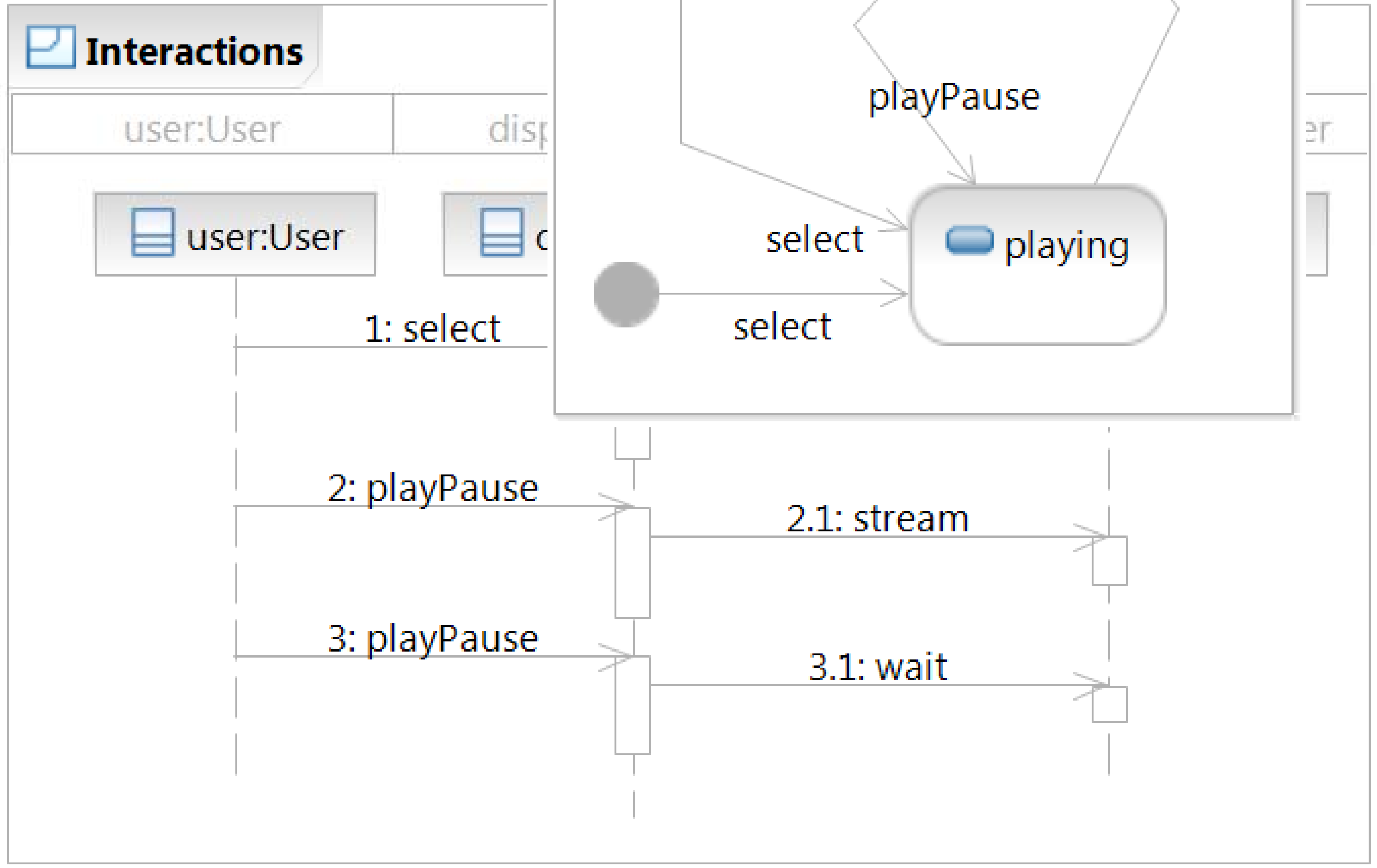
- But what about transforming changes?
  - Change can happen anytime, anywhere

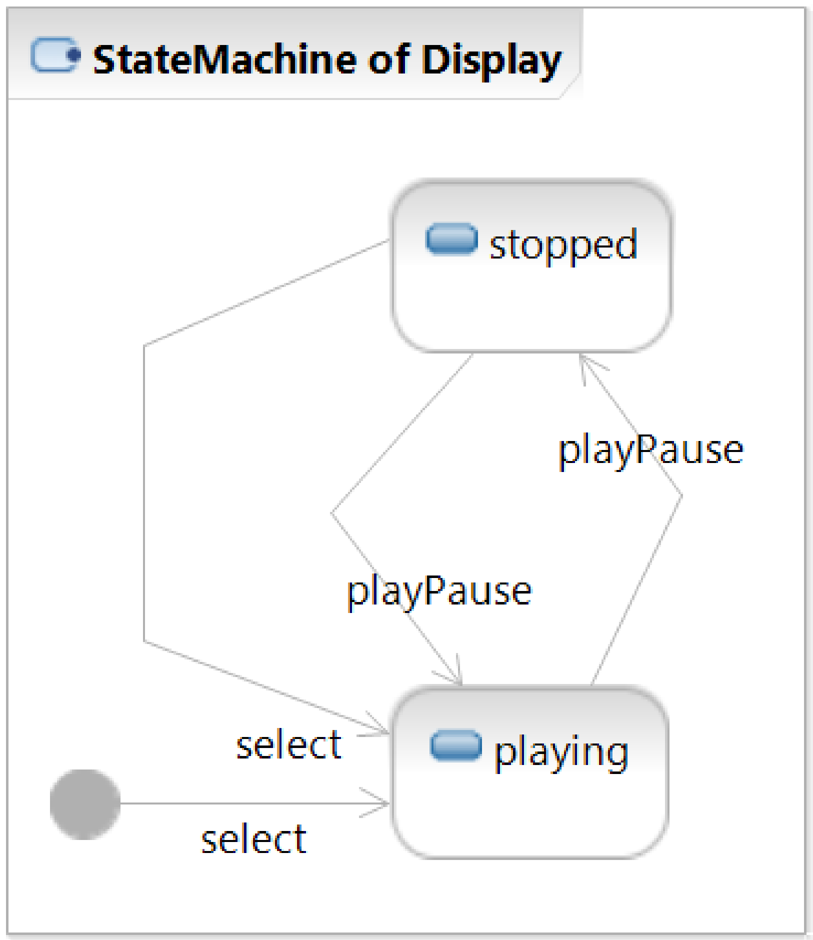
A Motivating Illustration for Change Propagation

(transforming changes, not models)

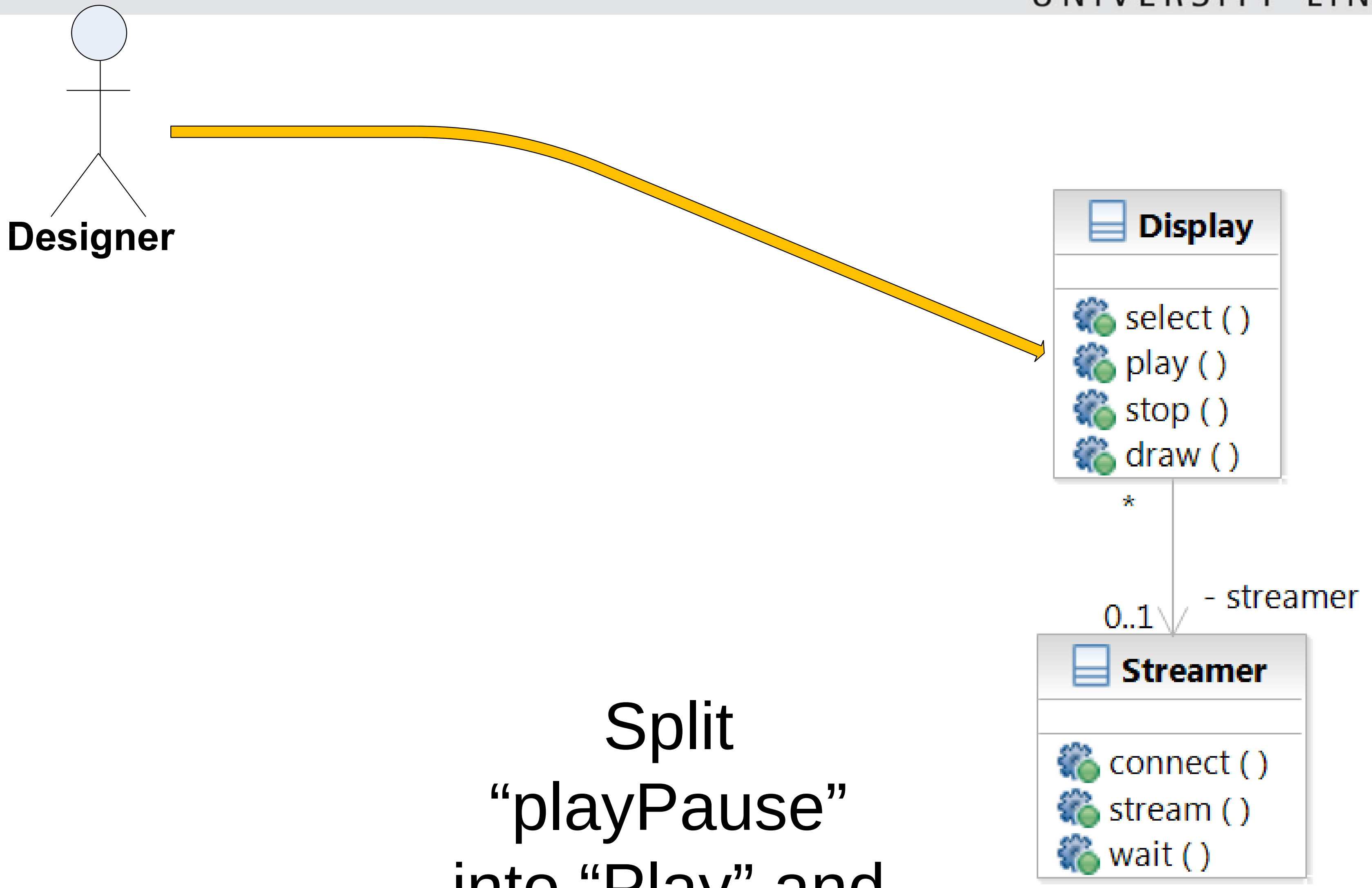
Modeling Languages are Diverse





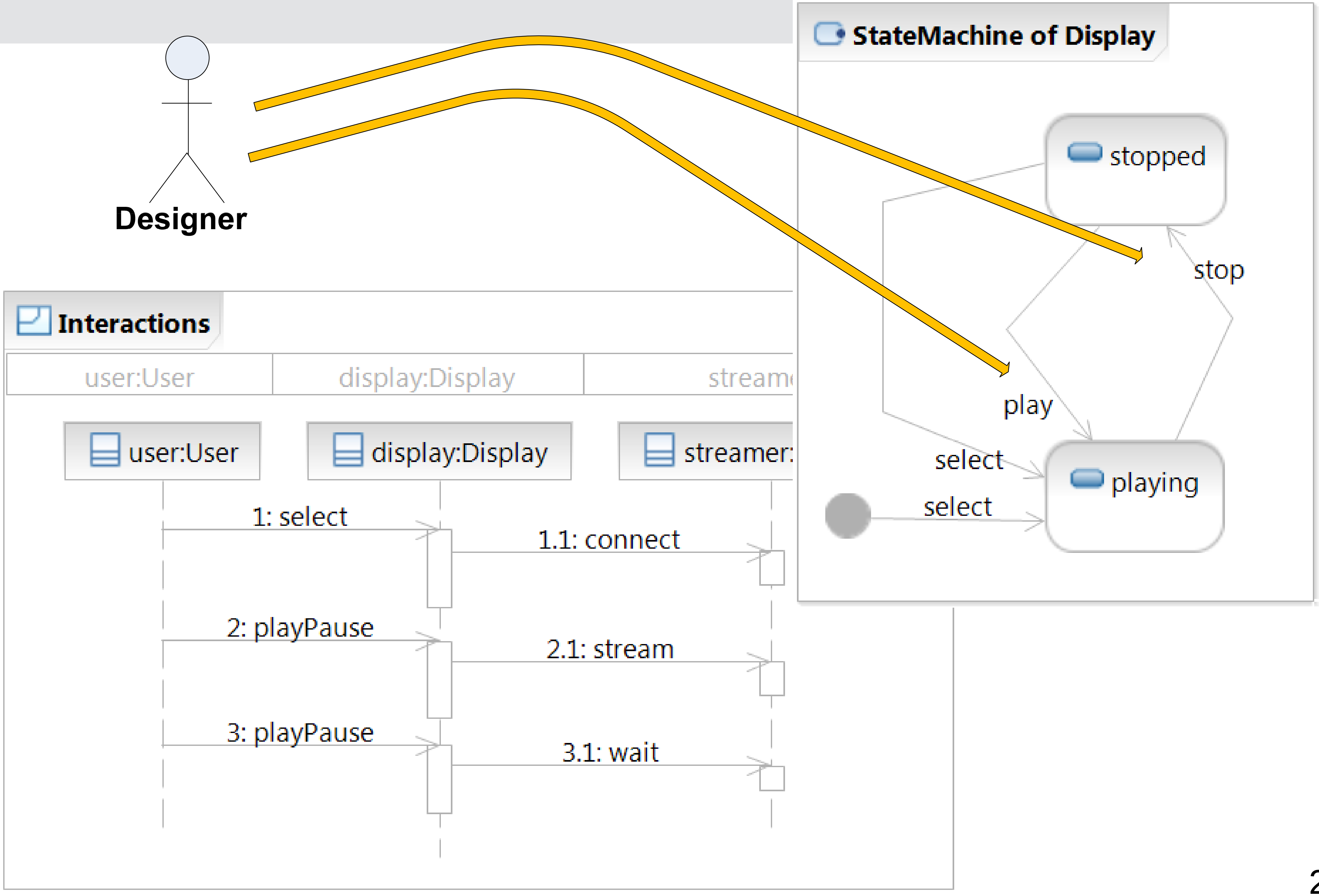


Designer Changes

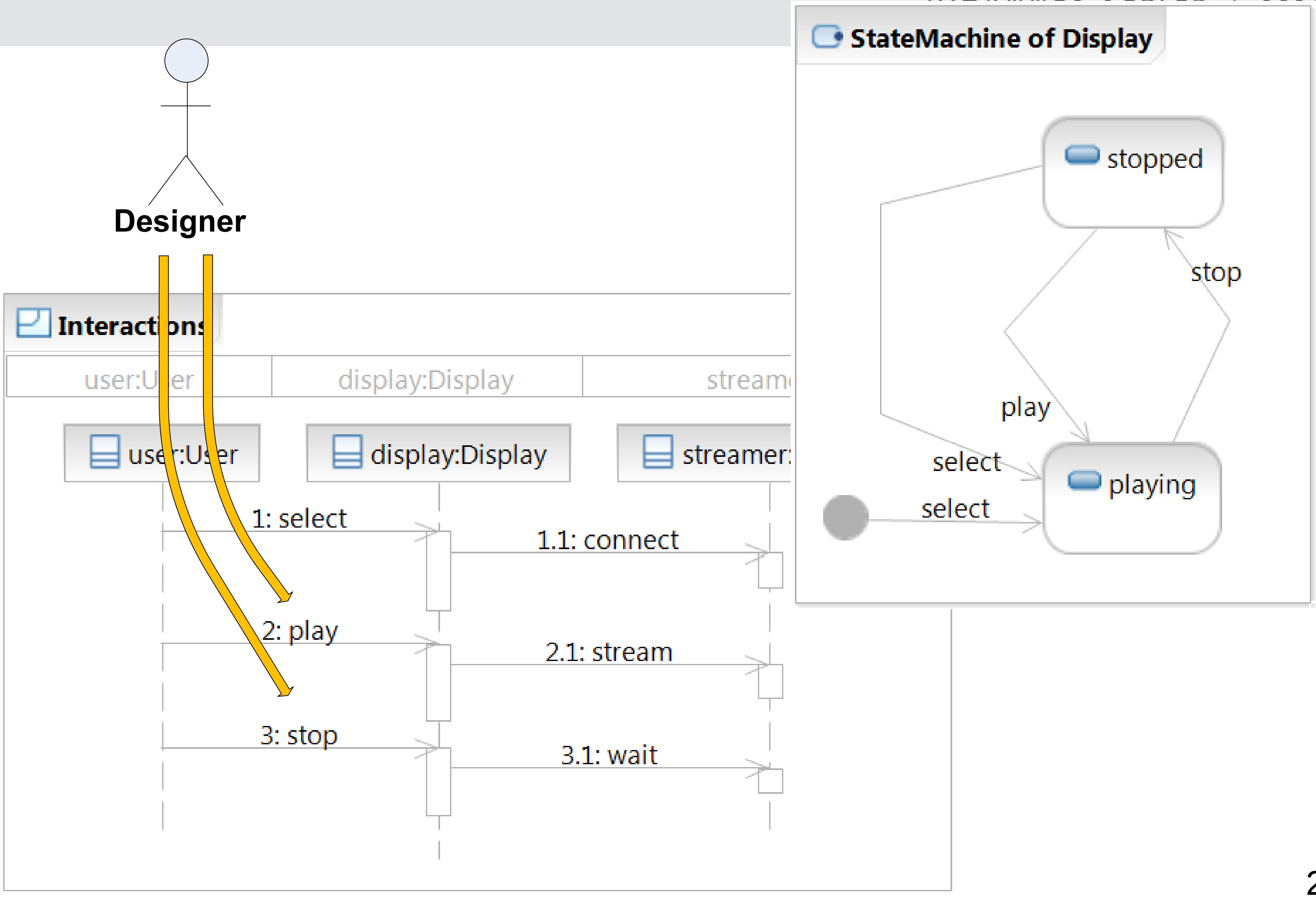


Split "playPause" into "Play" and "Stop"

Change Propagates

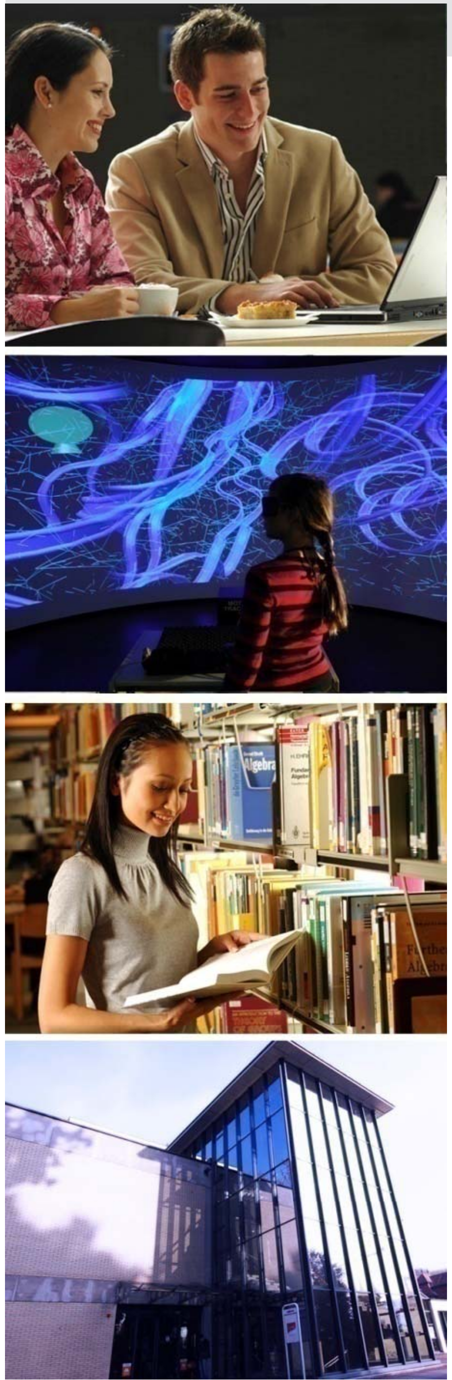


Change Propagates



Transformation in the Small

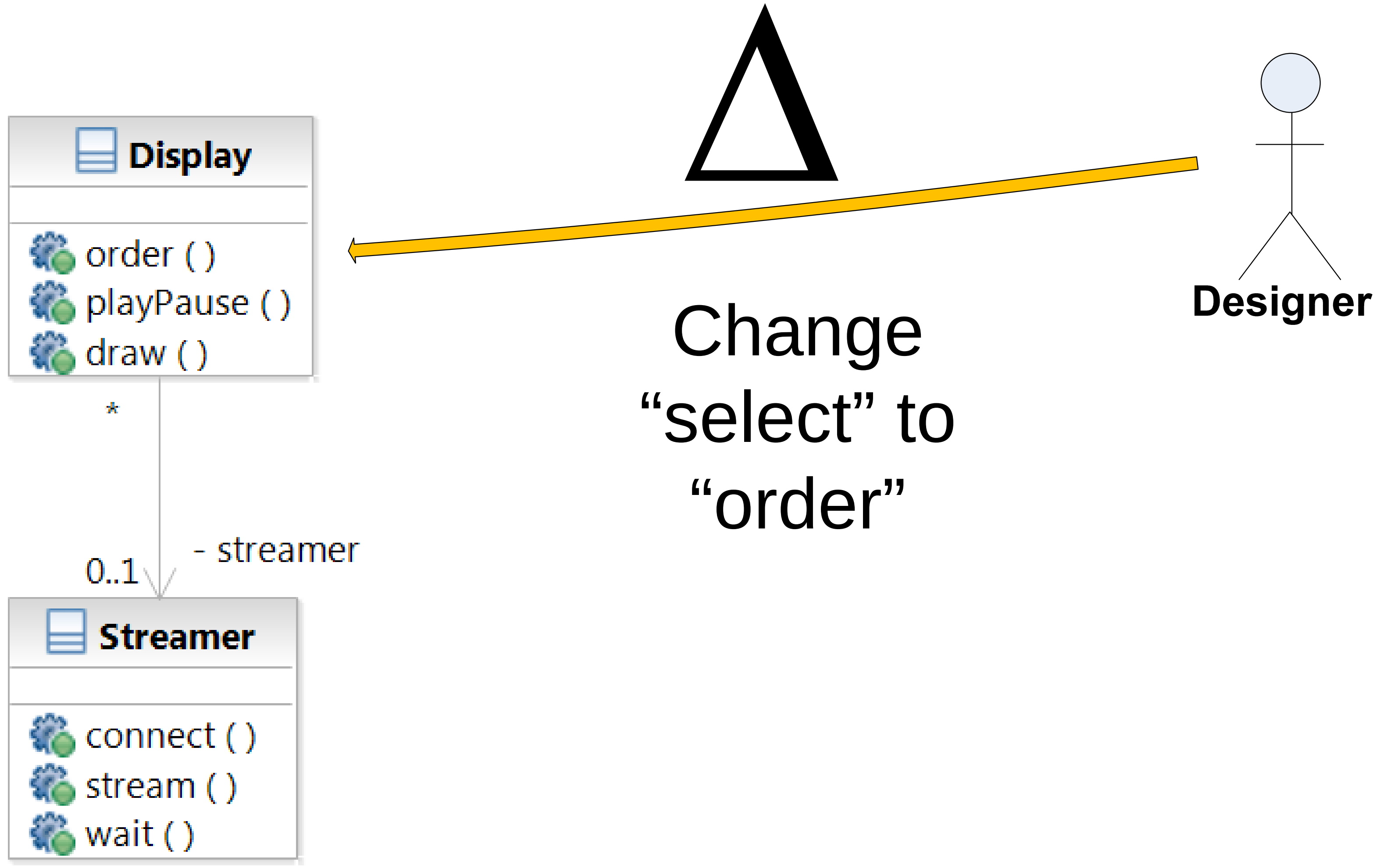
(transforming changes, not models)



Tool

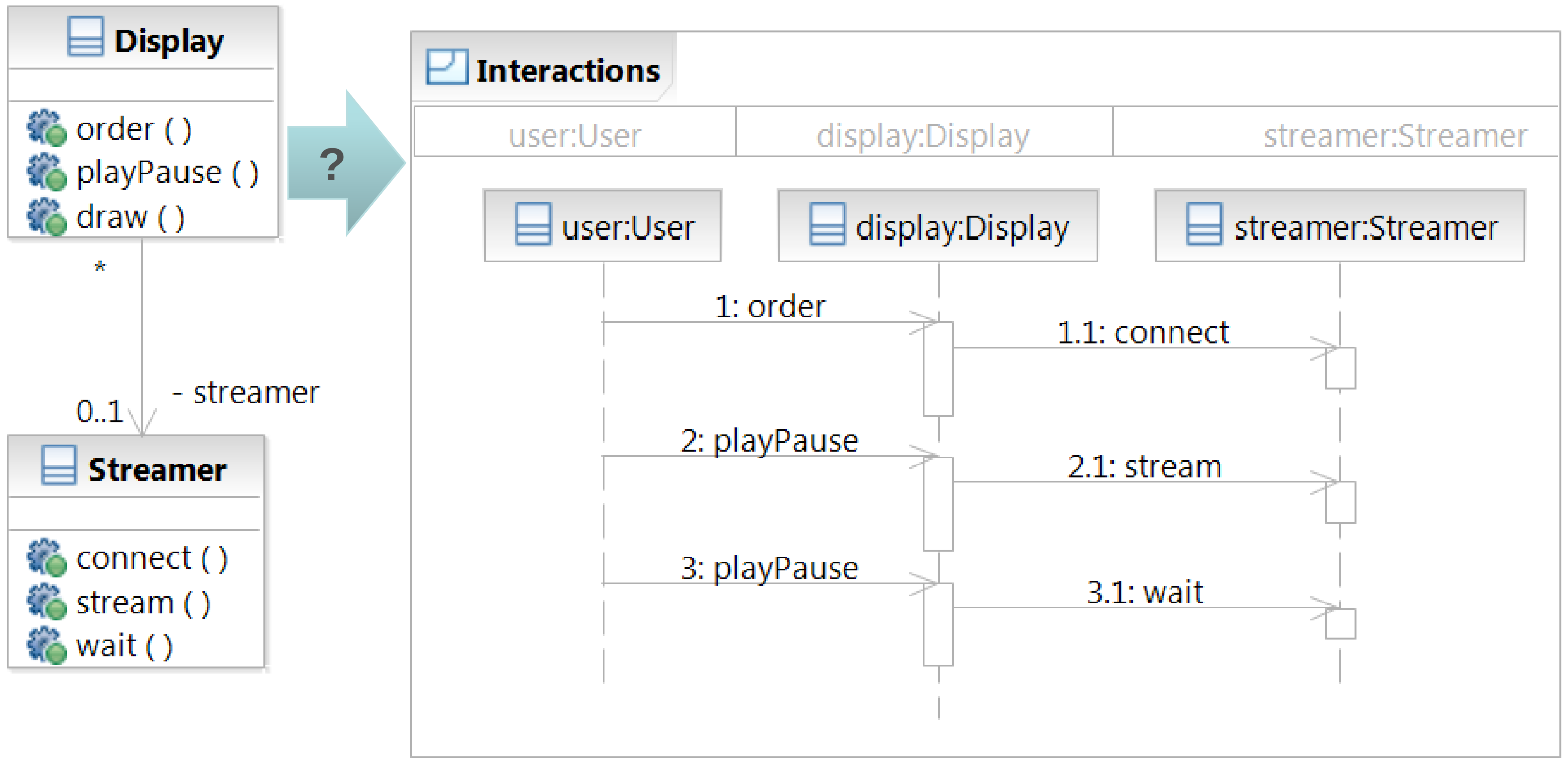
Change “select” method to “order”

Modeling Languages are Diverse



Change “select” to “order”

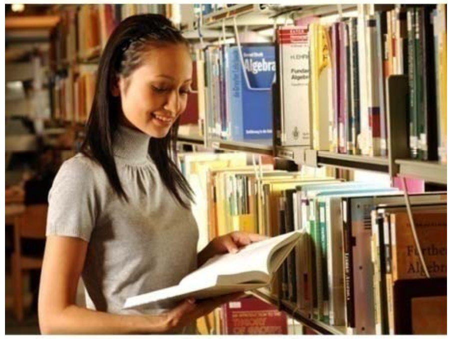
Propagate Change from Class Diagram to Sequence Diagram



Method Name Change Propagation

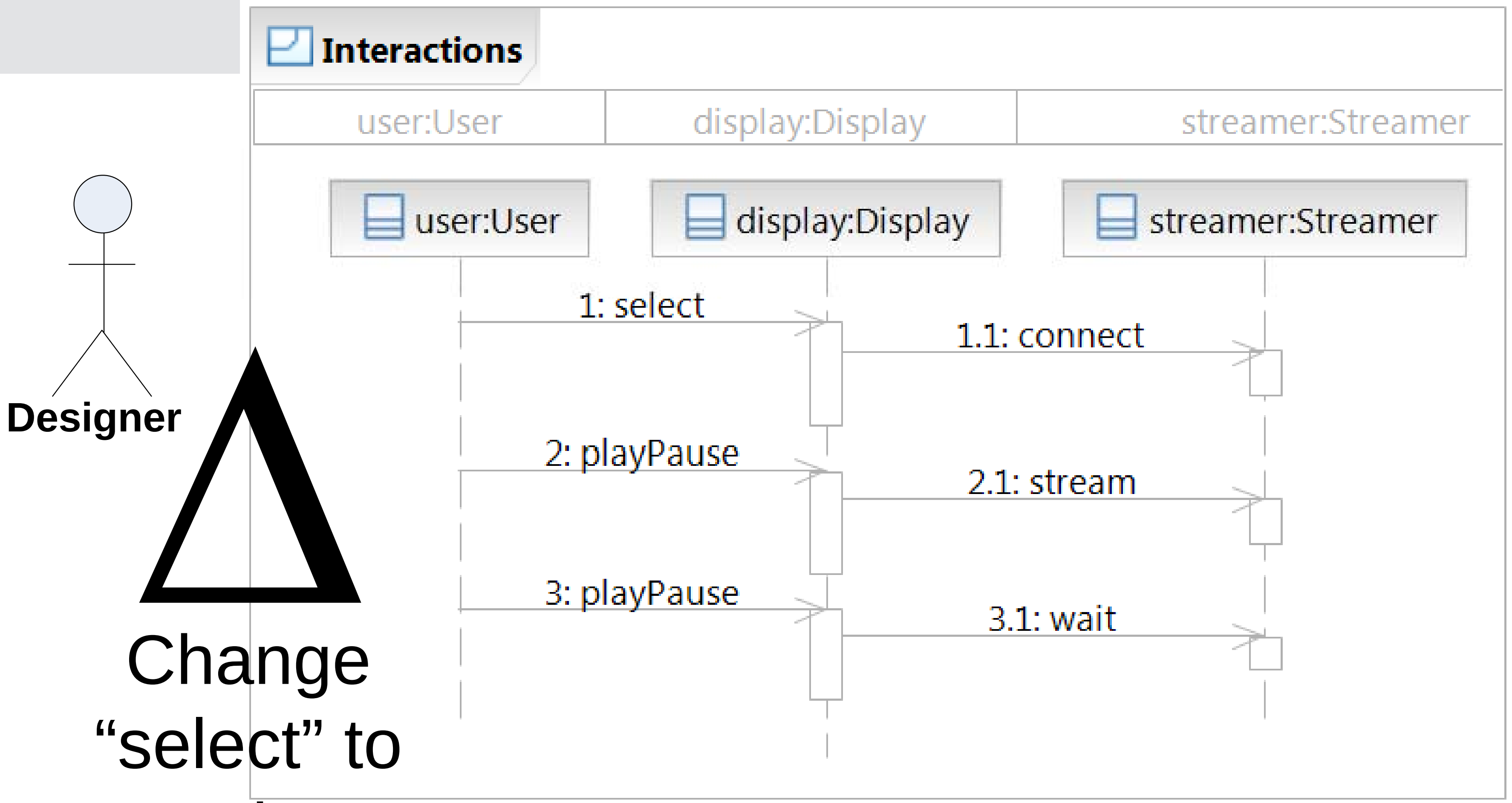
```
Context[method name change]
For all sequence diagrams that include
  instances of method owner
  – Rename incoming messages where
    message name = old method name
```

- But as we know, changes can happen anytime and anywhere



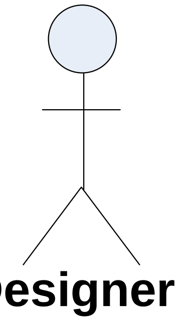
Tool

Change “select” message to “order”



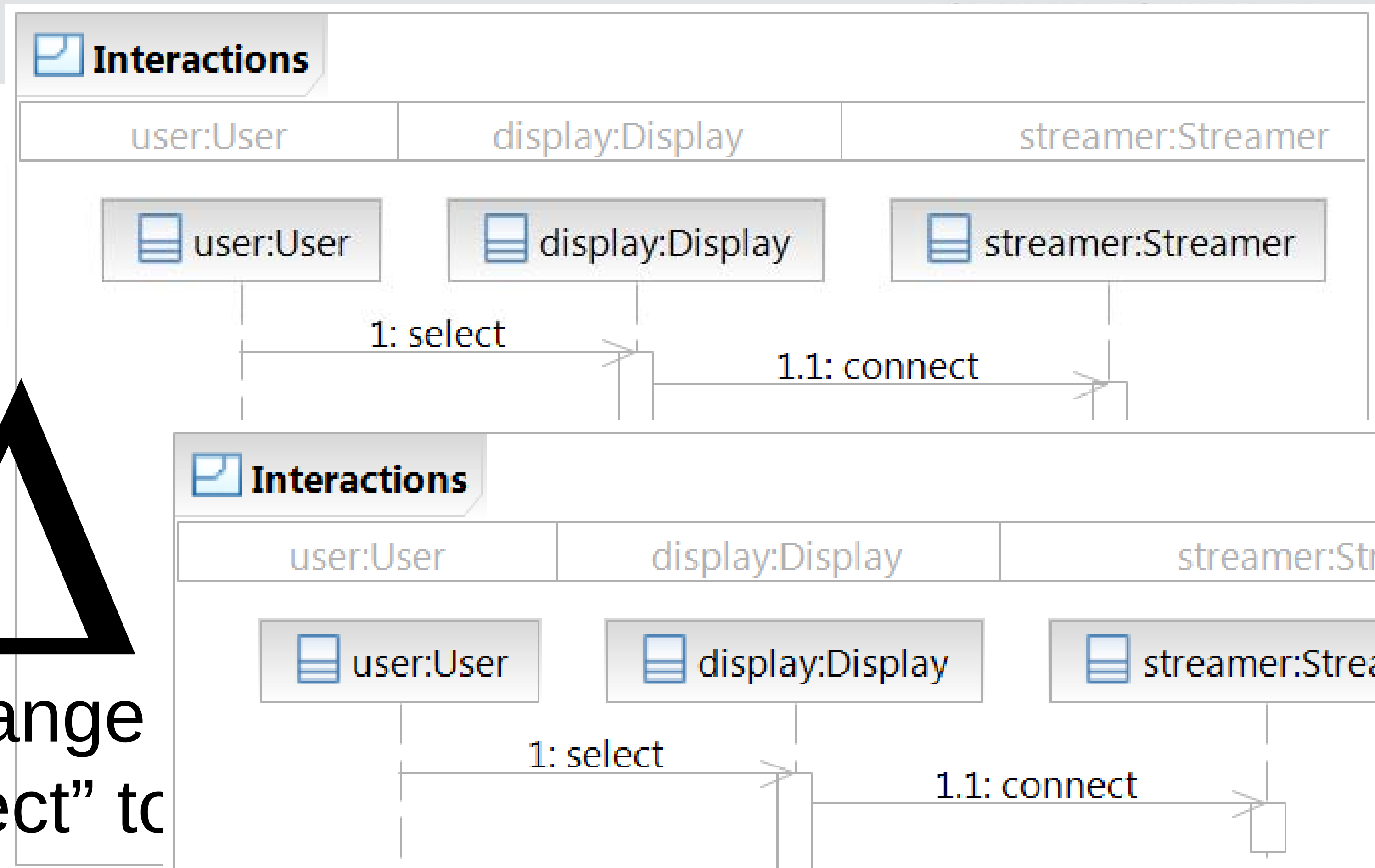
Δ

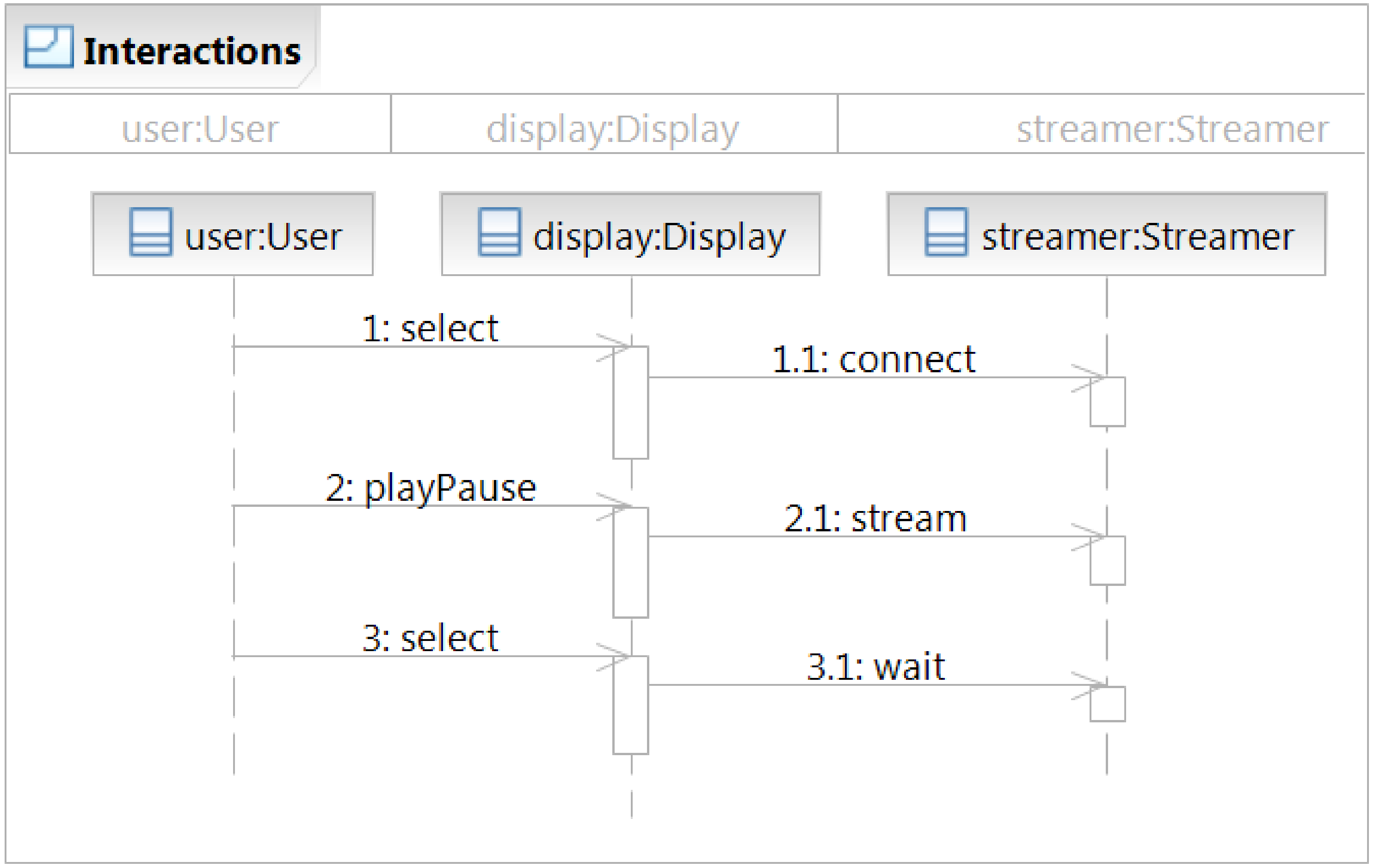
Change "select" to "order"



Δ

Change
“select” tc
“order”



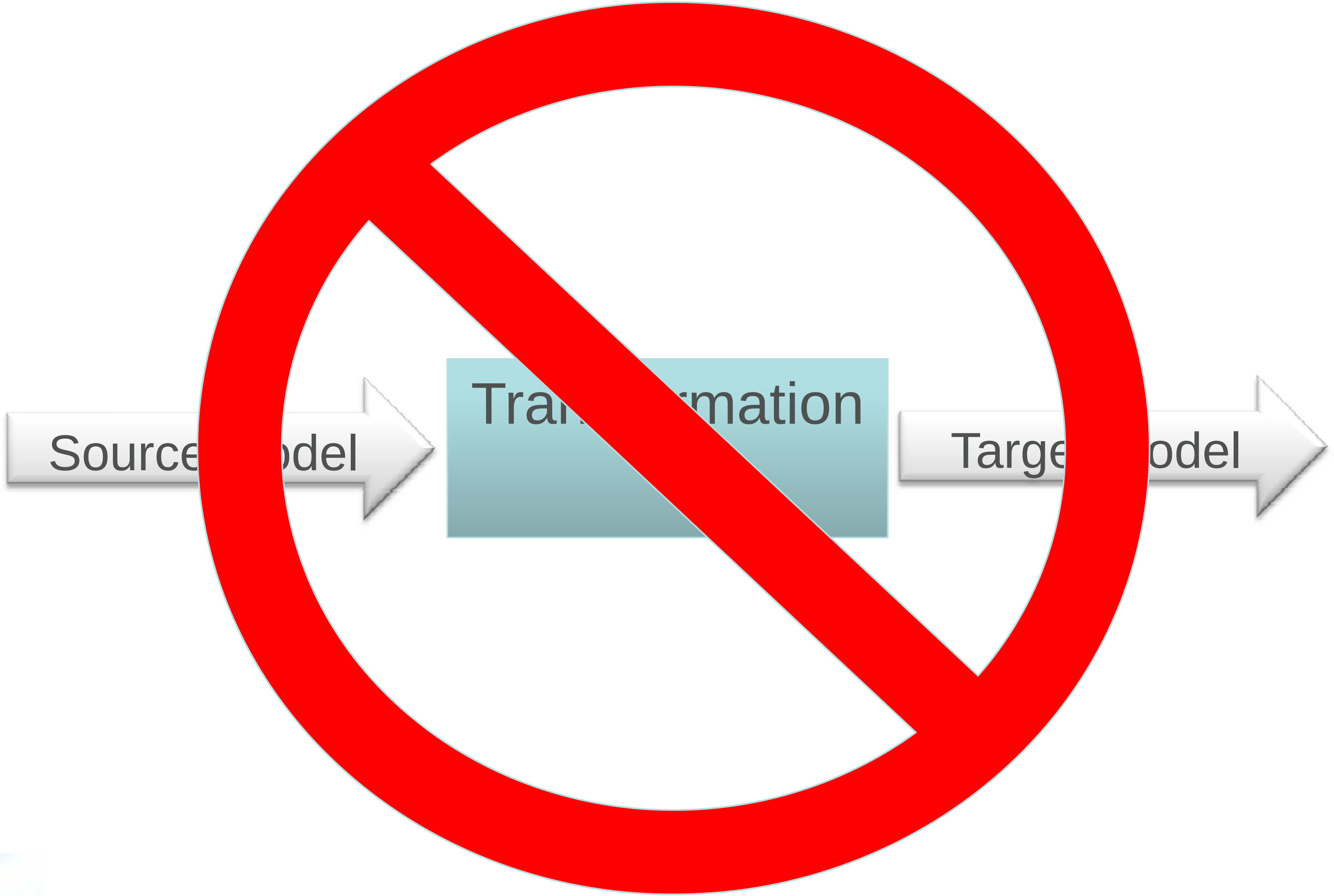


Message Name Change Propagation

JOHANNES KEPLER UNIVERSITY LINZ | JKU

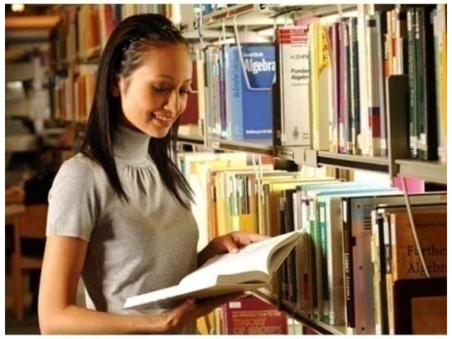
```
Context[message name change]                    source
Change all messages with same name and
  receiver?
  No: does method with new name exist?
        Yes: Done
        No: create method                       target
  Yes: does method with new name exist?
        Yes: Done
        No: rename method                       target
```

Change Propagation is No Classical Transformation



Another Interesting Observation

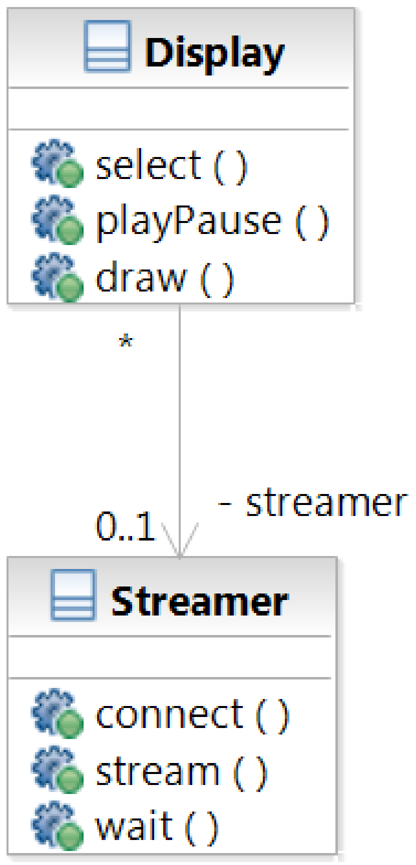
- On model level it was easier to transform sequence diagram to class diagram
- On model change level it was easier to transform method name change (class diagram) to message name change (sequence diagram)
- But n^2 problem still exists
  - Even made it worse: 3 diagram types, dozens of model element change types

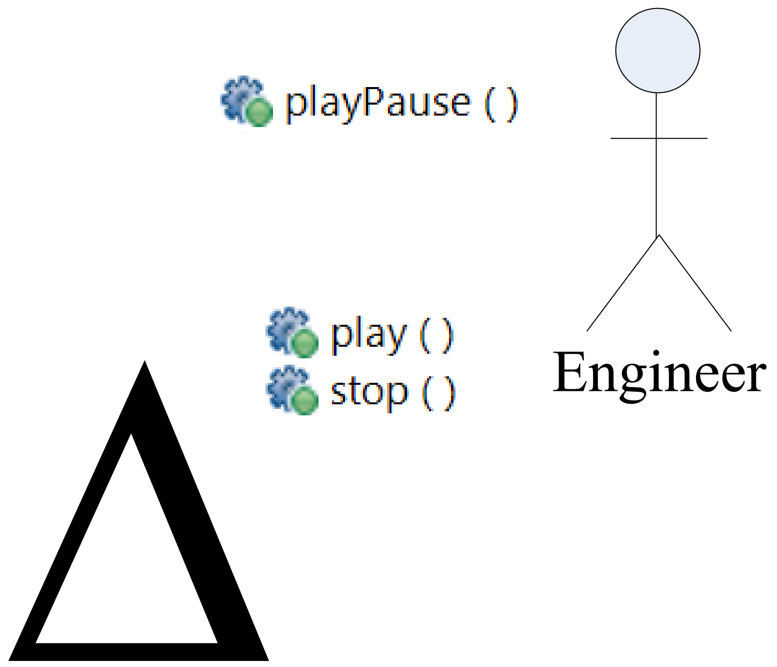


Tool

Show splitting of “playPause” to “play” and “stop”

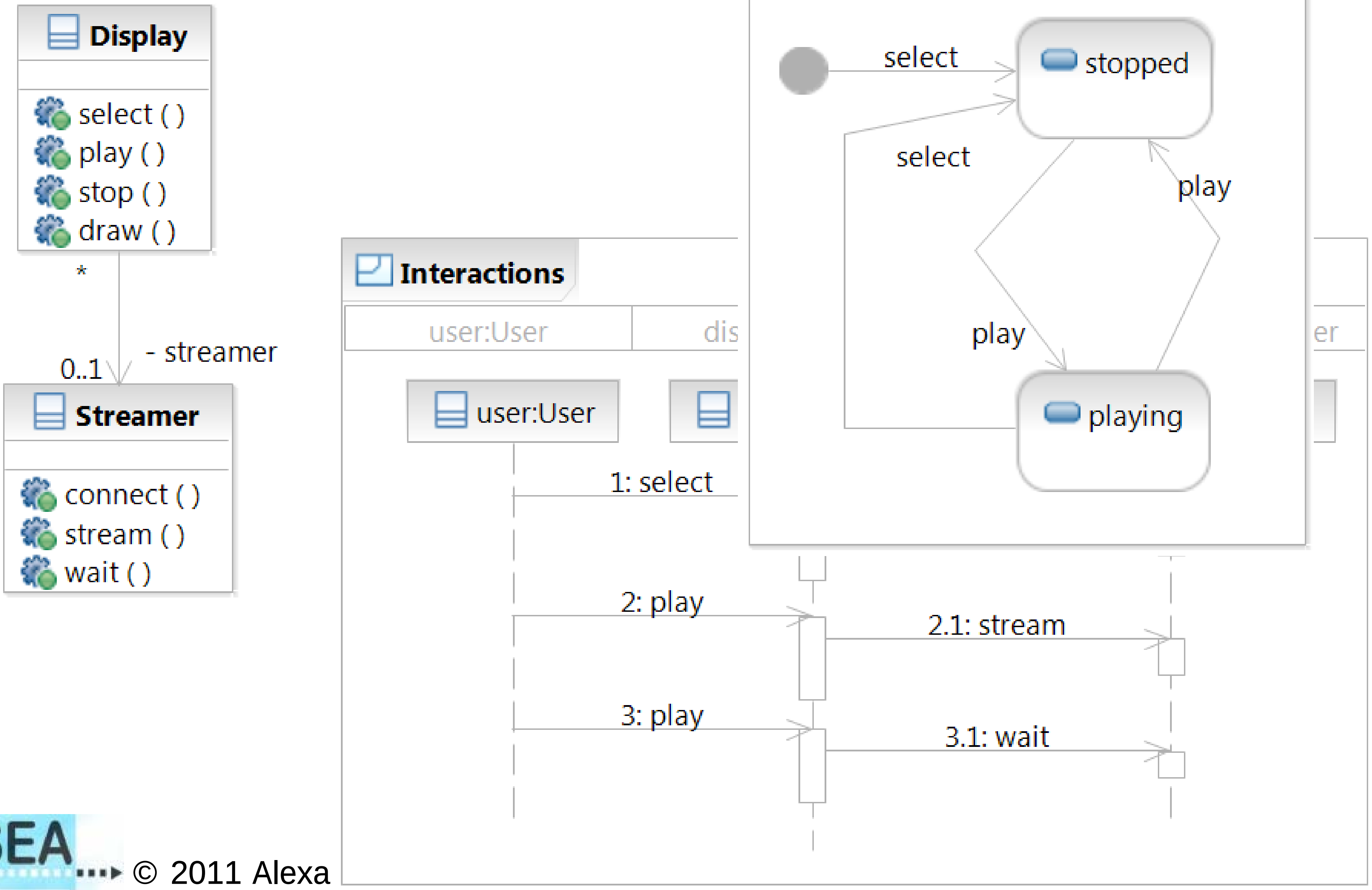
A slightly more complex name change

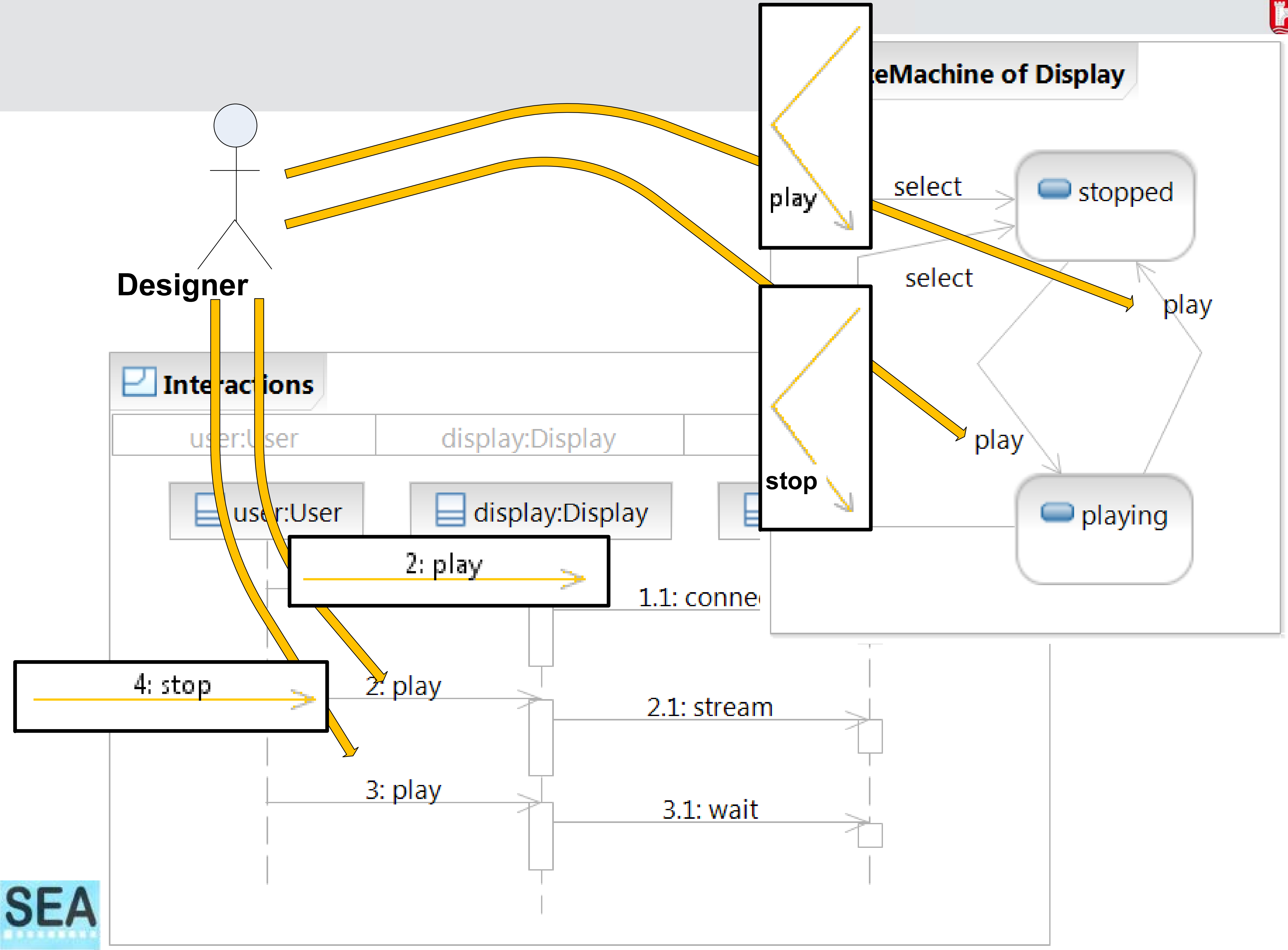




Split “playPause” into “Play” and “Stop”

A slightly more complex name change





Designer
Interactions
user:User
display:Display
user:User
display:Display
2: play
1.1: conne
4: stop
2: play
2.1: stream
3: play
3.1: wait
teMachine of Display
play
stop
select
select
stopped
play
play
playing

Problems

- **Split method**
  - Splitting of “playPause” to “play” and “stop” does not really work
  - Perhaps need a special “split” refactoring and transformation rules that react to it
- **Also need Separate Transformation for Message Name Change to Statechart**
  - But class, statechart and sequence diagrams do not just “live” next to each other. They interact

Transformation through Constraint Satisfaction

Many Consistency Rules

JOHANNES KEPLER UNIVERSITY LINZ | JKU

Rule 1: Name of message must match an operation in receiver's class

Context Message:
```
    self.receiveEvent.oclAsType(InteractionFragment).covered->
        forAll(represents.type.oclAsType(Class).ownedOperation->
        exists(name=self.name))
```

Rule 2: Sequence of object messages must correspond to events

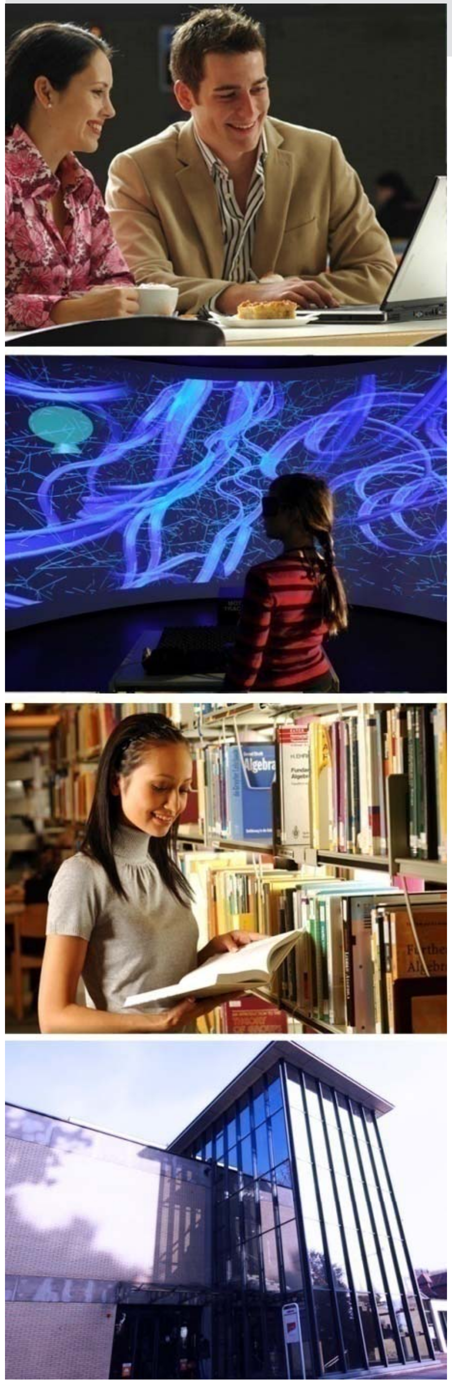
Rule 3: Calling direction of association must match calling direction of messages

…

Rule 100+

Plus other Constraints: Requirements, Domain (e.g., money or card)

JOHANNES KEPLER UNIVERSITY LINZ | JKU

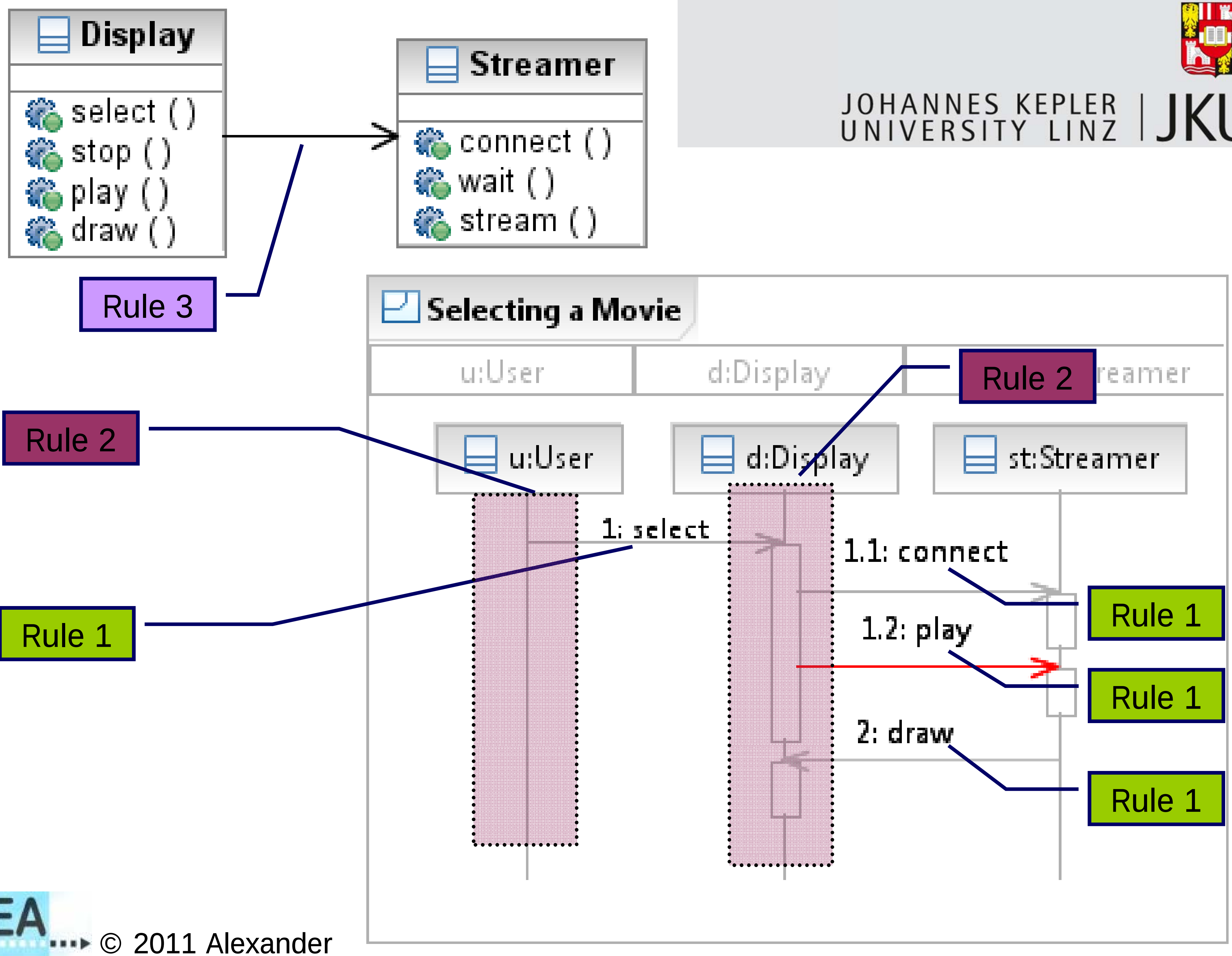


Tool

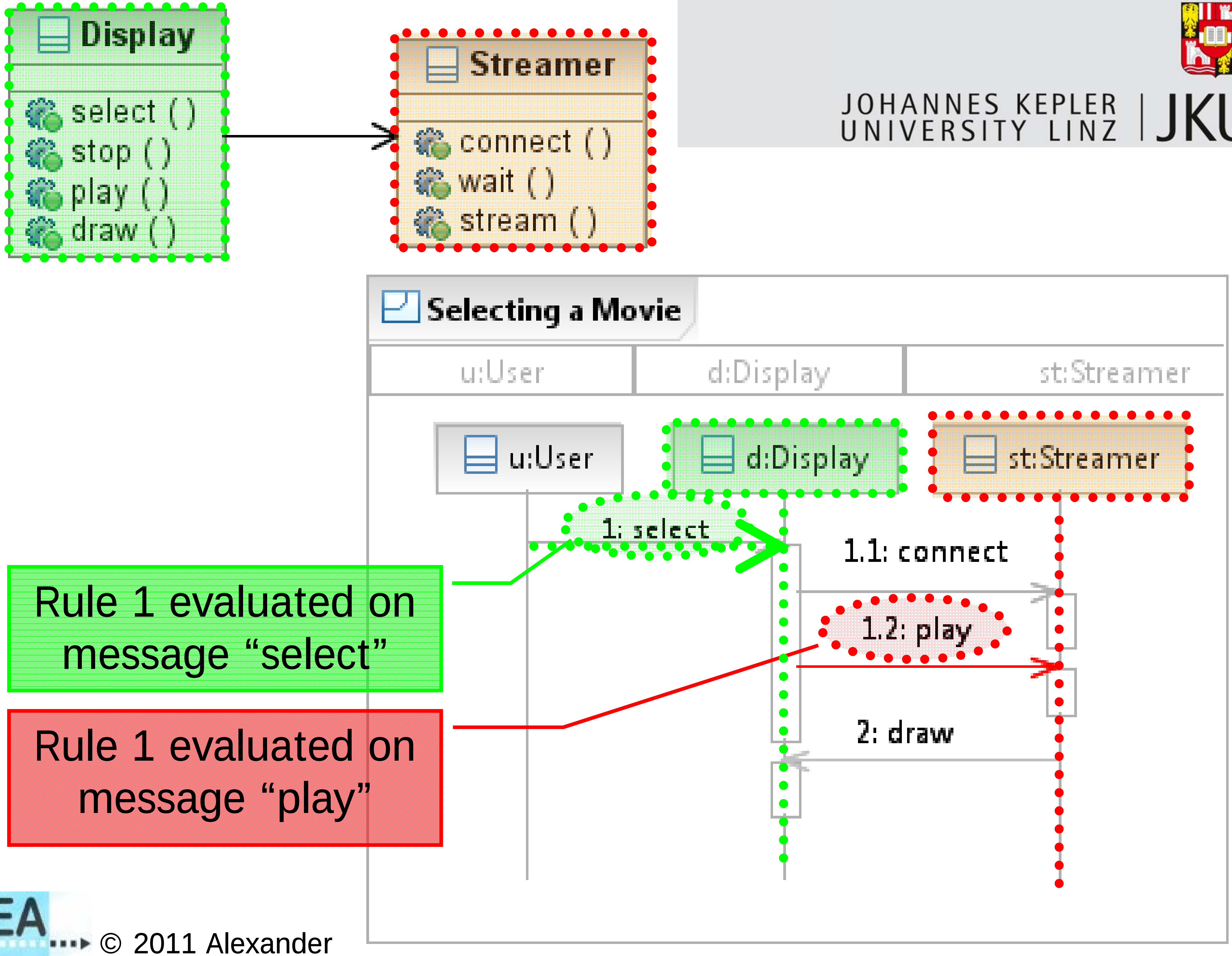
Rename playPause message to Play (disable IBM mapping of both messages to methods first). Detect inconsistencies instantly

Two Advances

1) We treat every evaluation of a consistency rule as a first class citizen – by maintaining change impact scopes for them individually and triggering individual re-evaluations
2) We use model profiling to observe the “behavior” of consistency rules during their evaluation to automatically compute change impact scopes



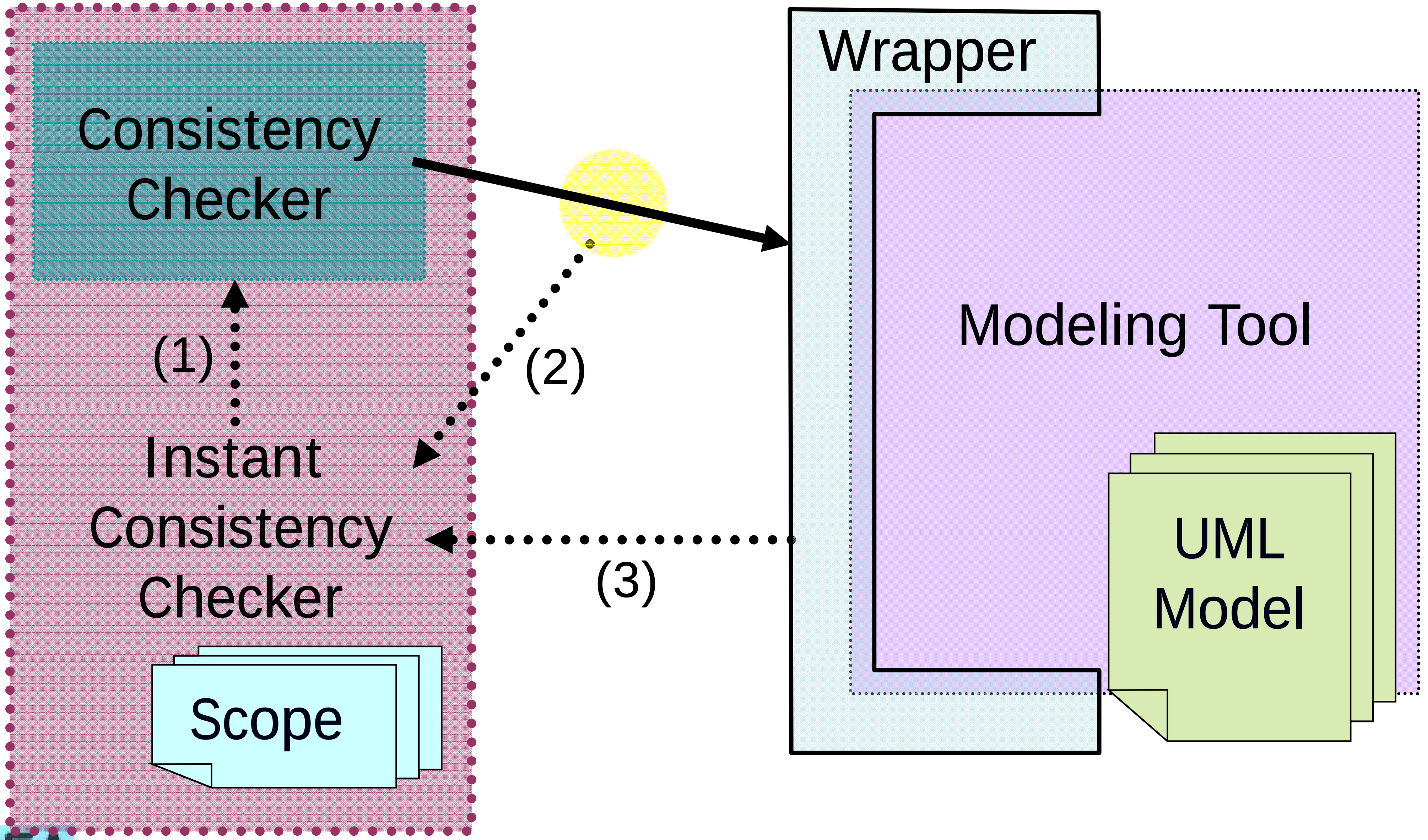
Display
select ()
stop ()
play ()
draw ()
Streamer
connect ()
wait ()
stream ()
Rule 3
Selecting a Movie
u:User
d:Display
reamer
Rule 2
Rule 2
u:User
d:Display
st:Streamer
1: select
1.1: connect
1.2: play
2: draw
Rule 1
Rule 1
Rule 1
Rule 1



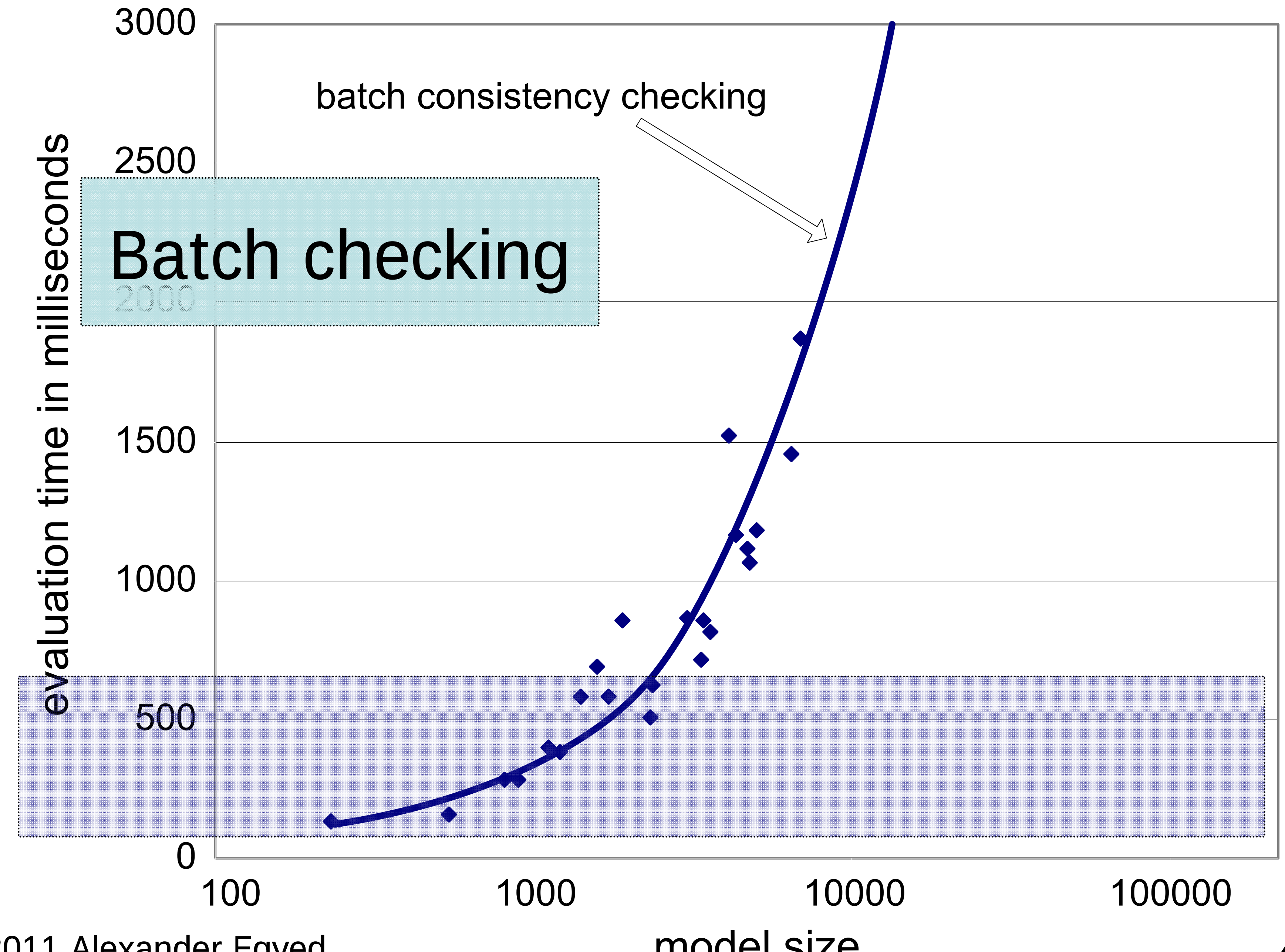
Rule 1 evaluated on message “select”

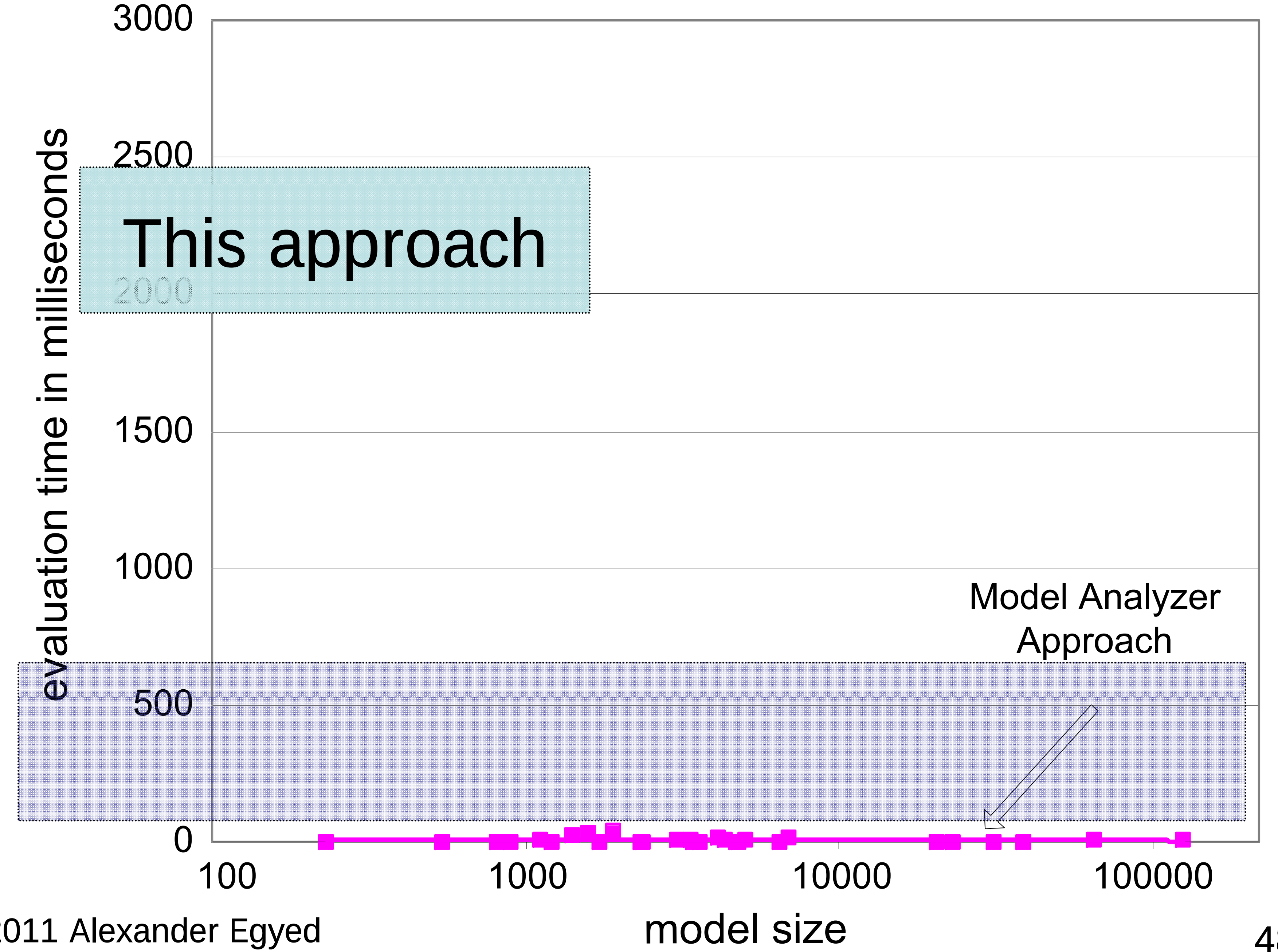
Rule 1 evaluated on message “play”

Model Analyzer Approach



Batch checking



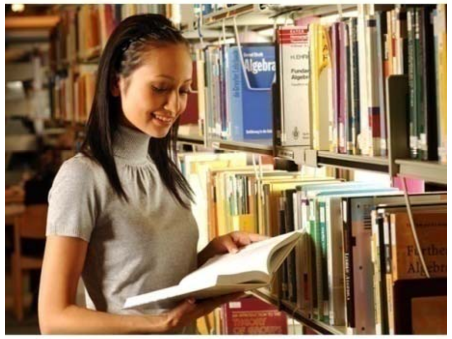

3000
2500
2000
1500
1000
500
0
evaluation time in milliseconds
This approach
Model Analyzer Approach
100
1000
10000
100000
model size

Implications

- We can quickly evaluate model changes

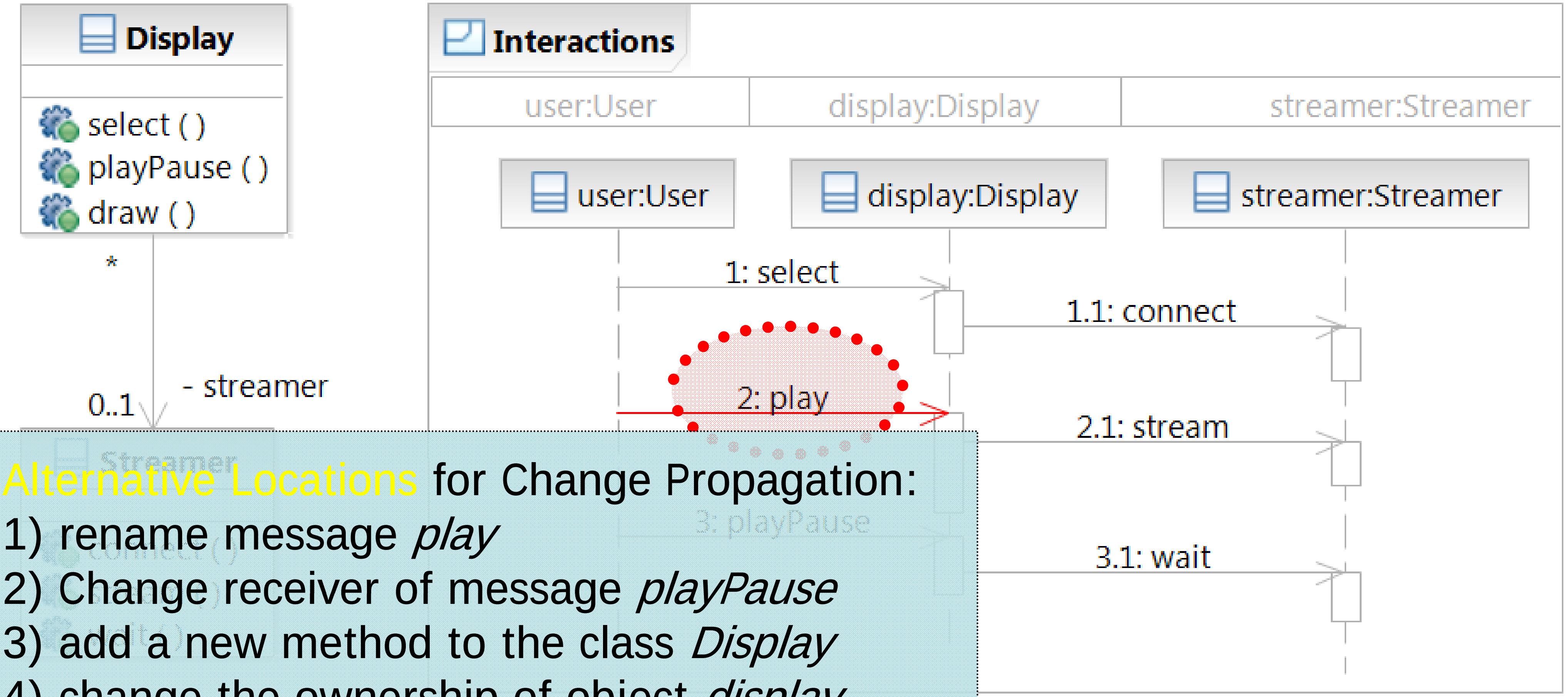
(also model changes done through transformations)

- And we can identify which model elements to change to resolve inconsistency (the first step of change propagation)



Tool

Enumerate change propagation alternatives of renamed “playpause” to “play” message



Alternative Locations for Change Propagation:

1) rename message *play*
2) Change receiver of message *playPause*
3) add a new method to the class *Display*
4) change the ownership of object *display*
6) rename method *select*
7) rename method *playPause*
8) rename method *draw*
9) delete message *play* (makes rule obsolete)

Transformation Alternatives

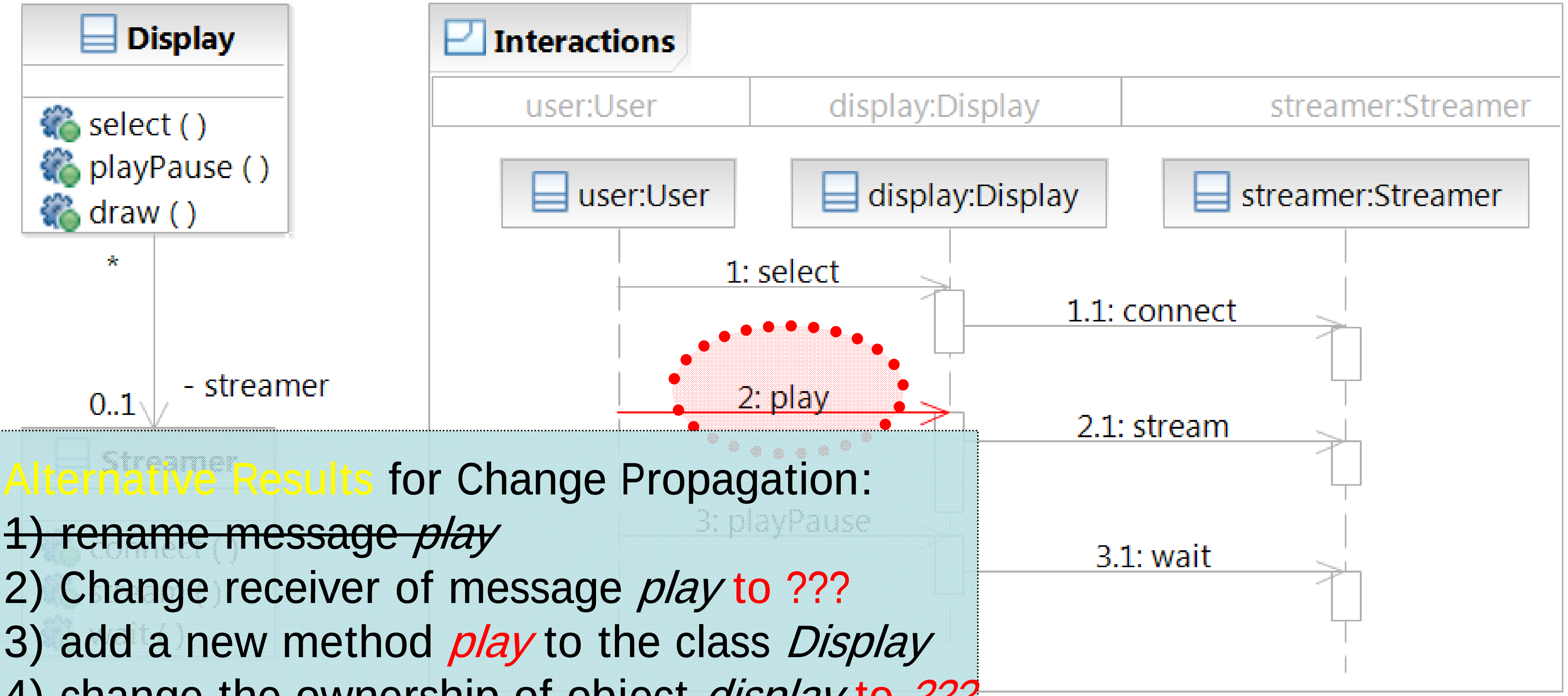
Change Propagation

  through alternative transformations

leads to

  alternative transformation results

The method/message name transformations discussed earlier were just two alternatives. It is not even clear whether they are even the most likely ones.



Alternative Results for Change Propagation:

~~1) rename message *play*~~

2) Change receiver of message *play* to ???

3) add a new method *play* to the class *Display*

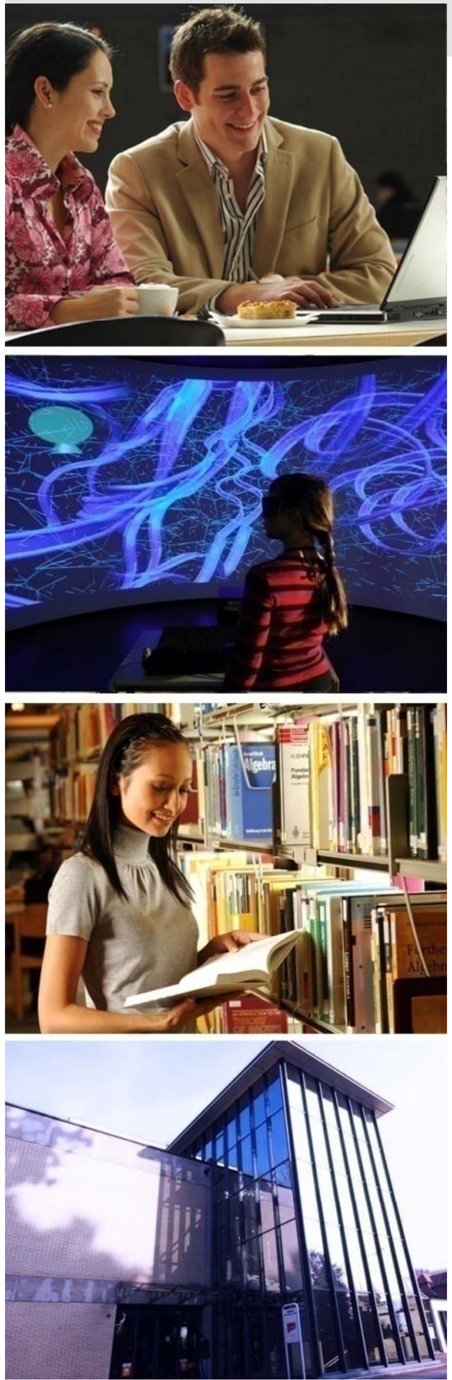
4) change the ownership of object *display* to *???*

6) rename method *select* to *play*

7) rename method *playPause* to *play*

8) rename method *draw* to *play*

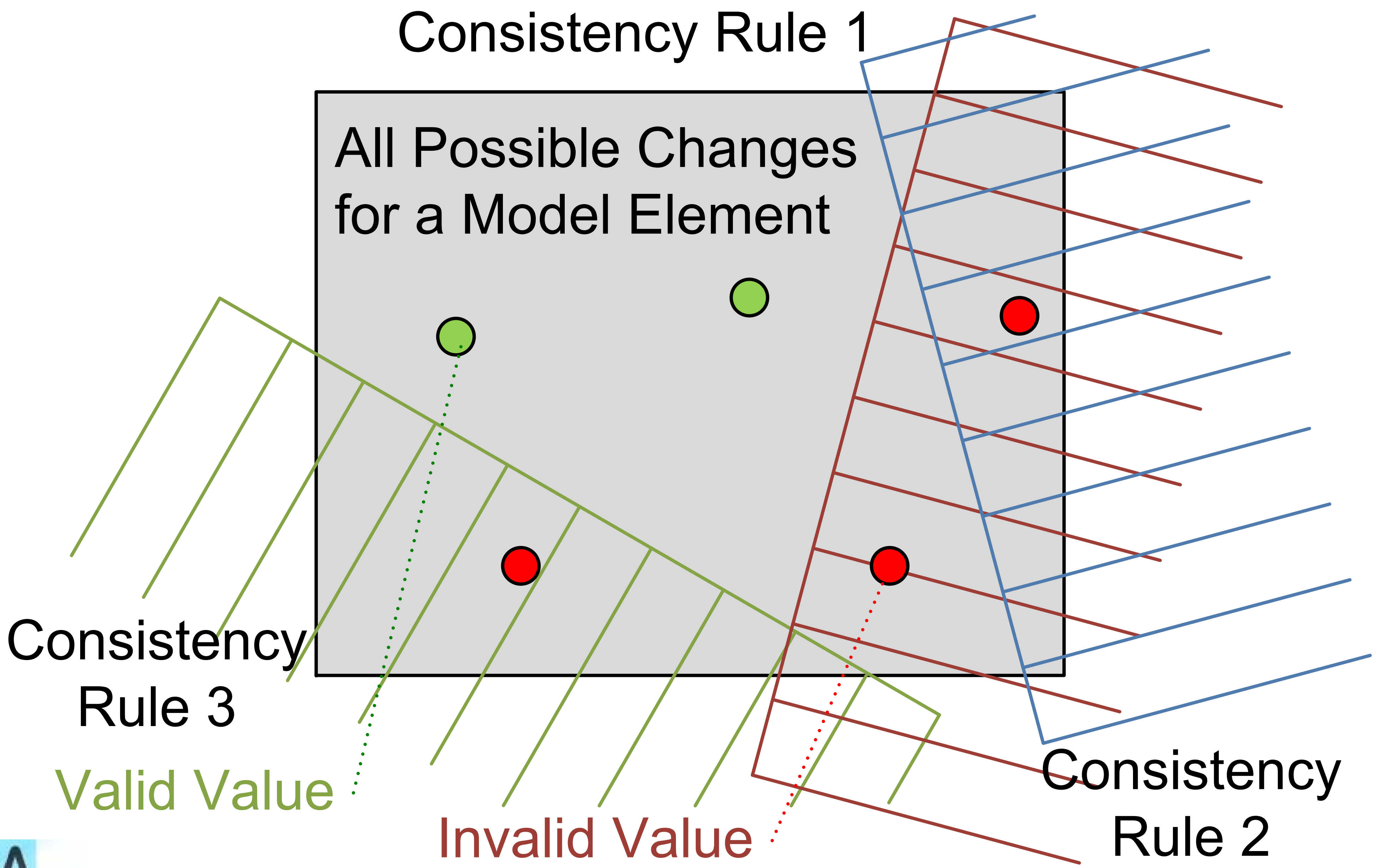
~~9) delete message *play* (makes rule obsolete)~~

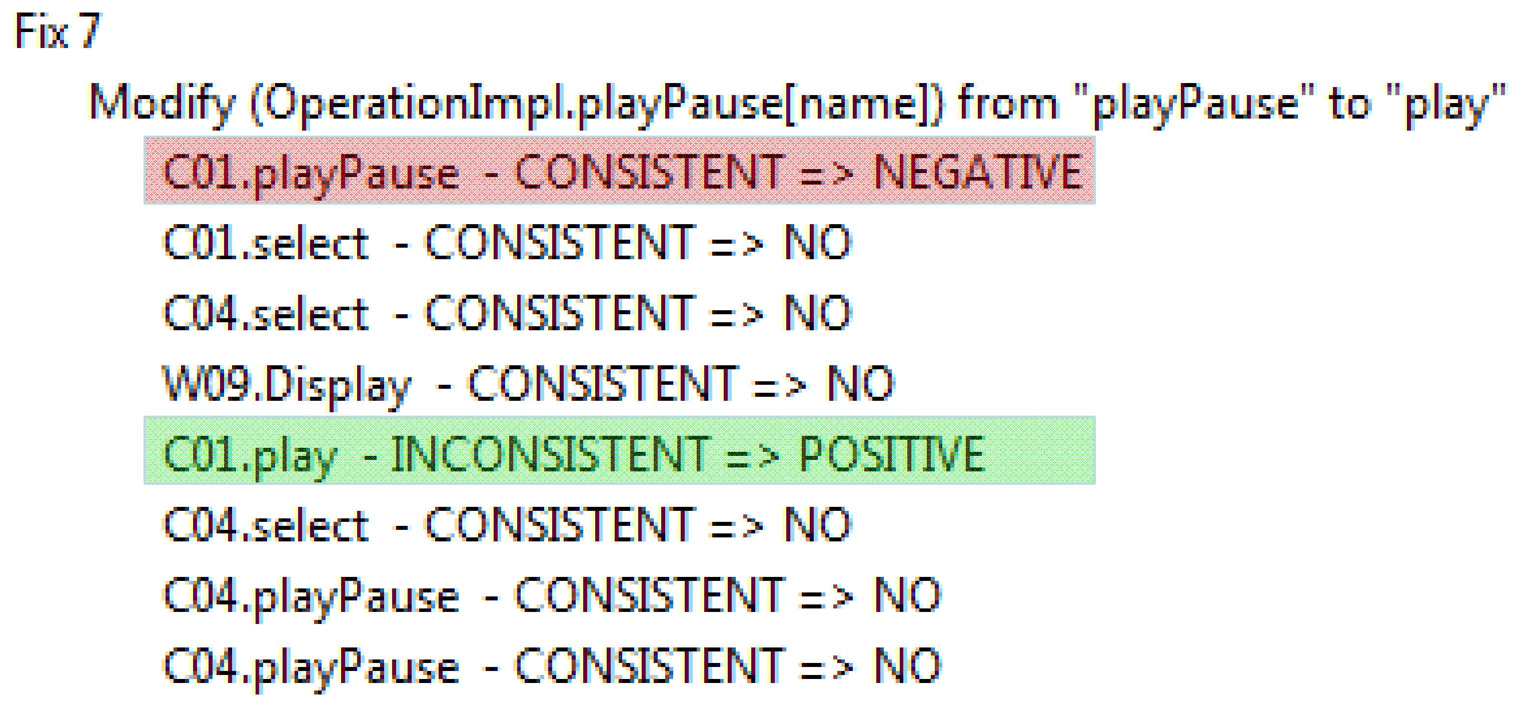


Tool

Execute change propagation that renames ‘playPause’ method to ‘play’ with follow-on inconsistencies

Eliminate False Choices





Fix 7

Modify (OperationImpl.playPause[name]) from "playPause" to "play"

C01.playPause - CONSISTENT => NEGATIVE

C01.select - CONSISTENT => NO

C04.select - CONSISTENT => NO

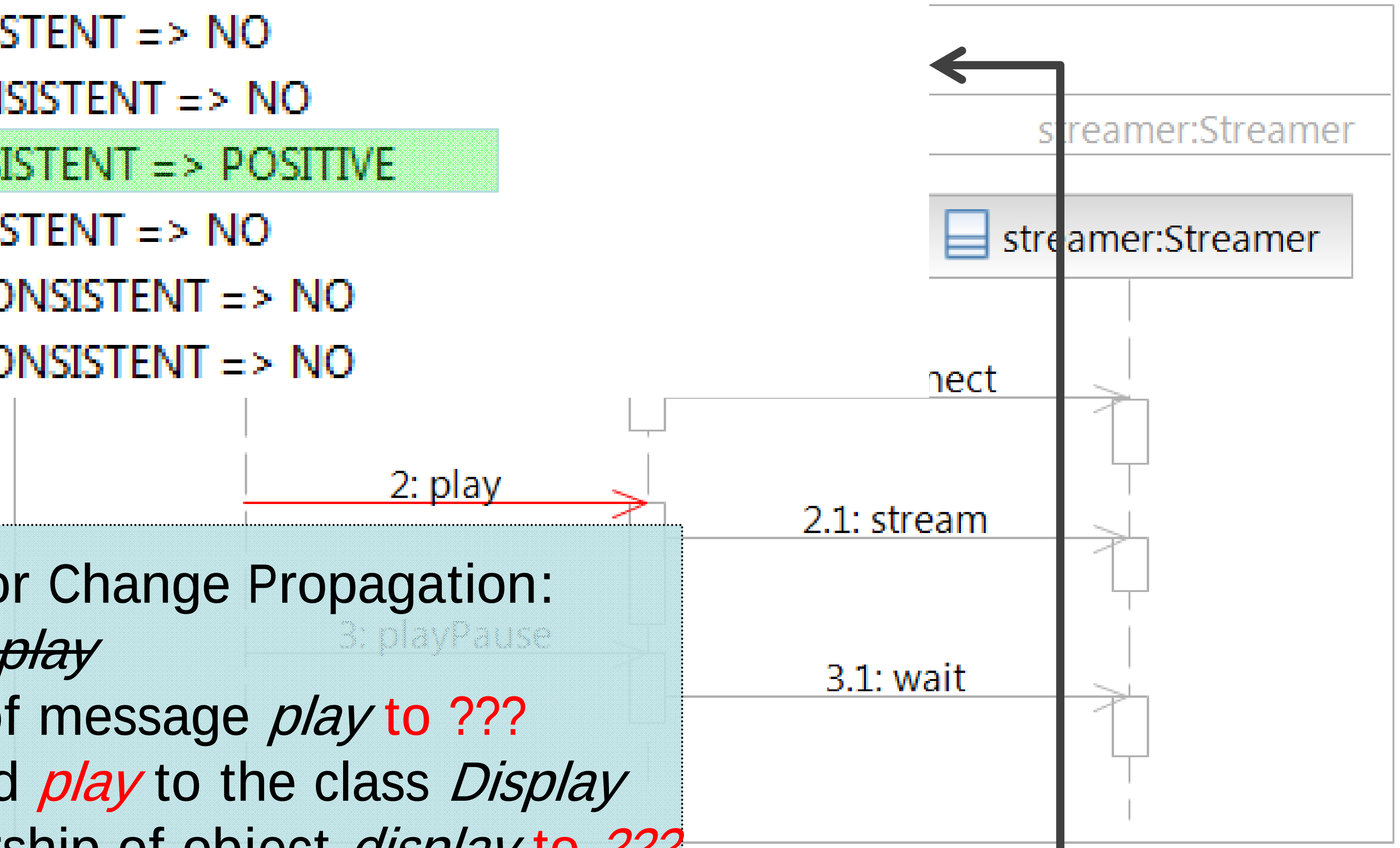
W09.Display - CONSISTENT => NO

C01.play - INCONSISTENT => POSITIVE

C04.select - CONSISTENT => NO

C04.playPause - CONSISTENT => NO

C04.playPause - CONSISTENT => NO



Alternative Results for Change Propagation:

~~1) rename message *play*~~

2) Change receiver of message *play* to ???

3) add a new method *play* to the class *Display*

4) change the ownership of object *display* to *???*

6) rename method *select* to *play*

7) rename method *playPause* to *play*

8) rename method *draw* to *play*

~~9) delete message *play* (makes rule obsolete)~~

Multiple Change Propagations

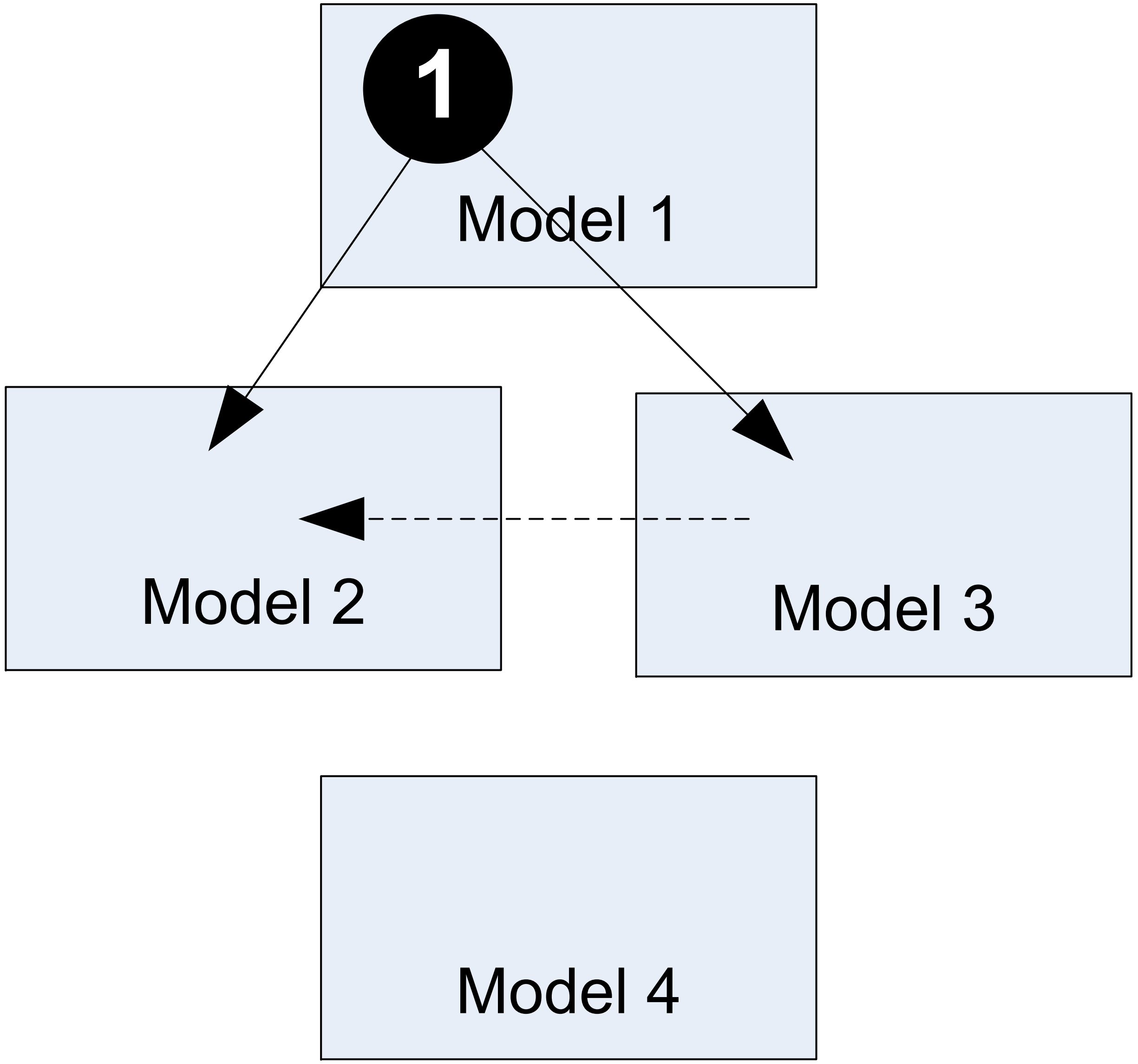
If after change propagation no inconsistency is caused

- Then propagation is complete
- Else further propagation is needed

Transformation Split and Merge

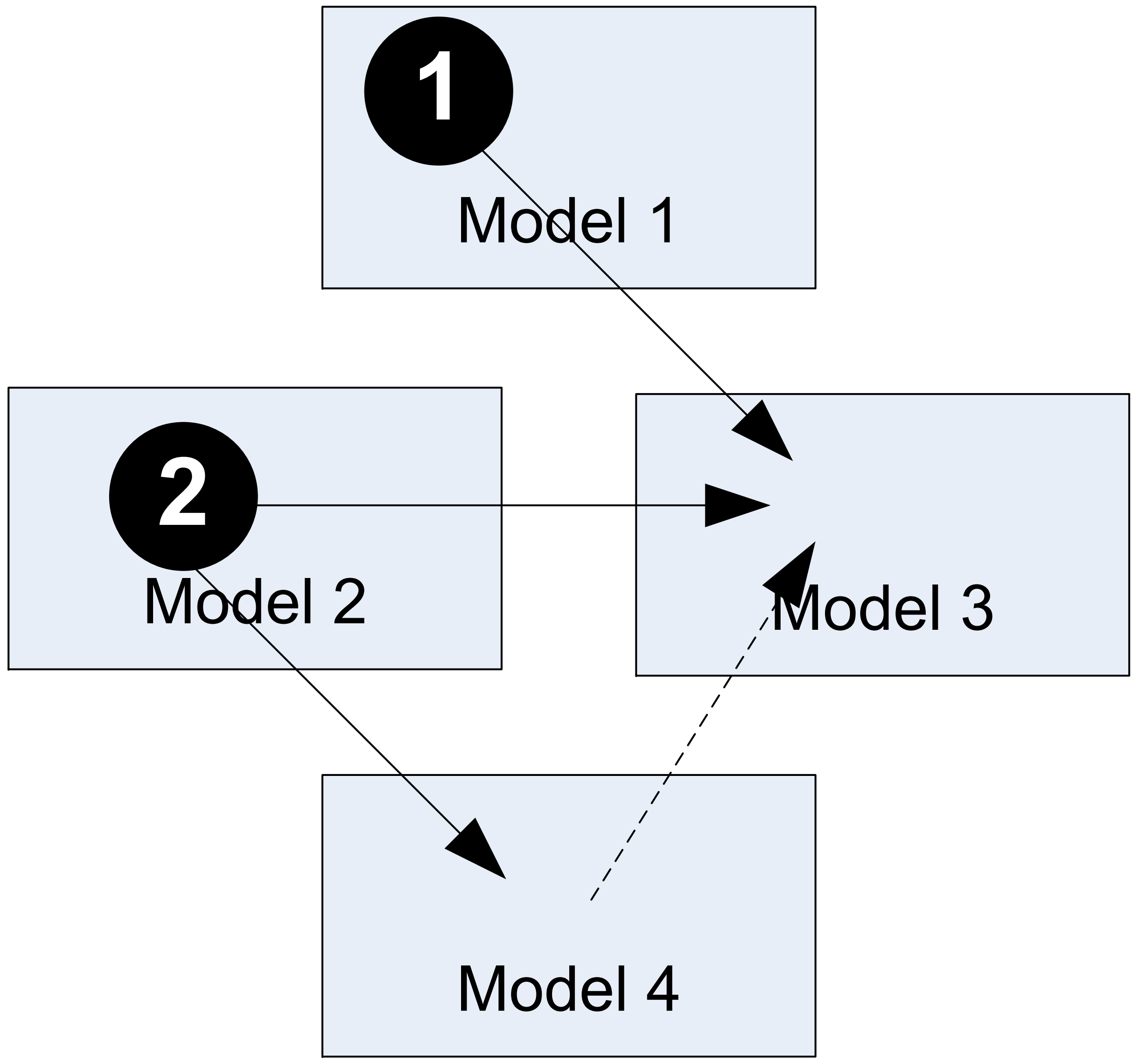
(serial and parallel transformation)

Change Propagation Unrolling



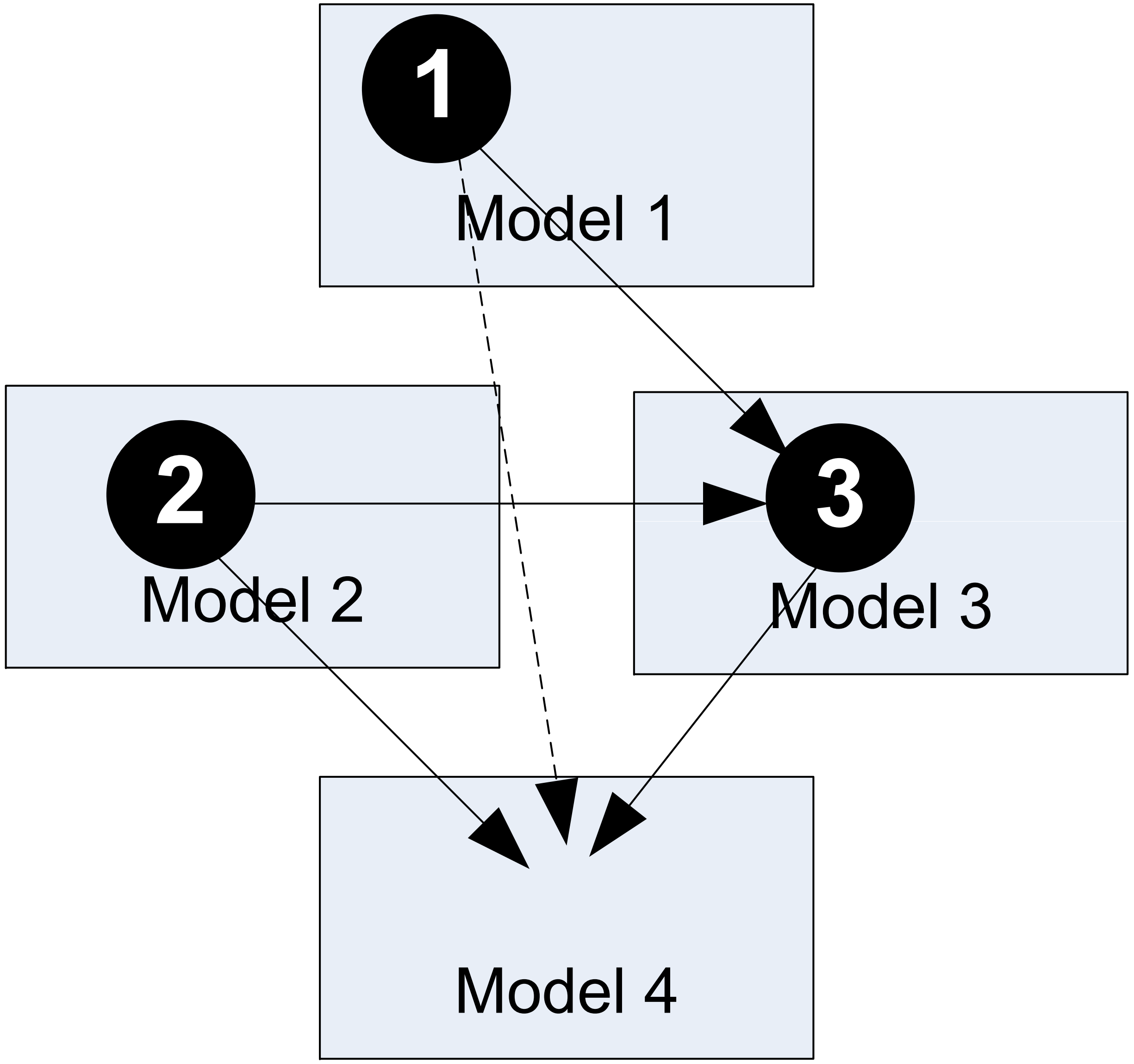
(a)

Change Propagation Unrolling



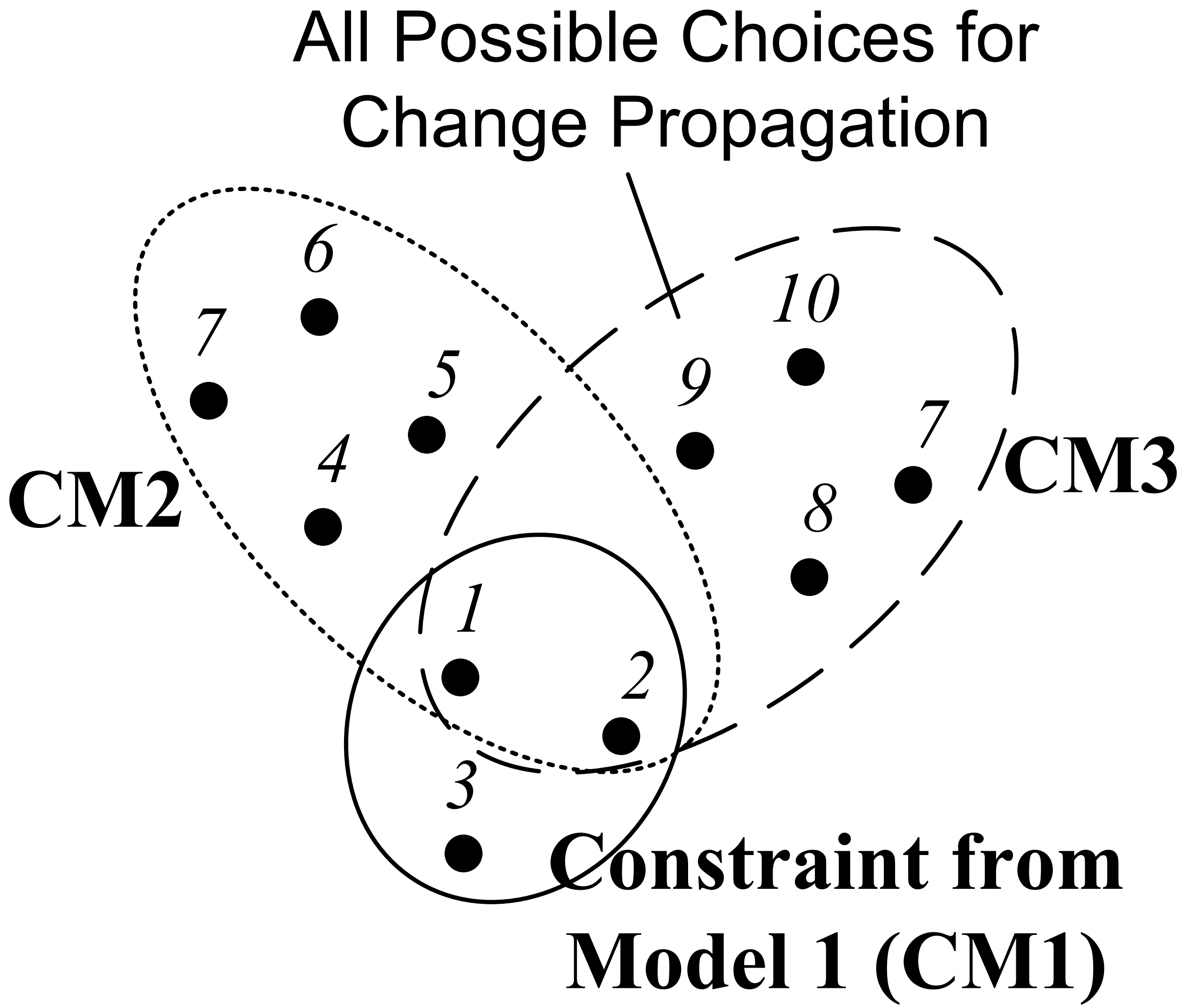
(b)

Change Propagation Unrolling



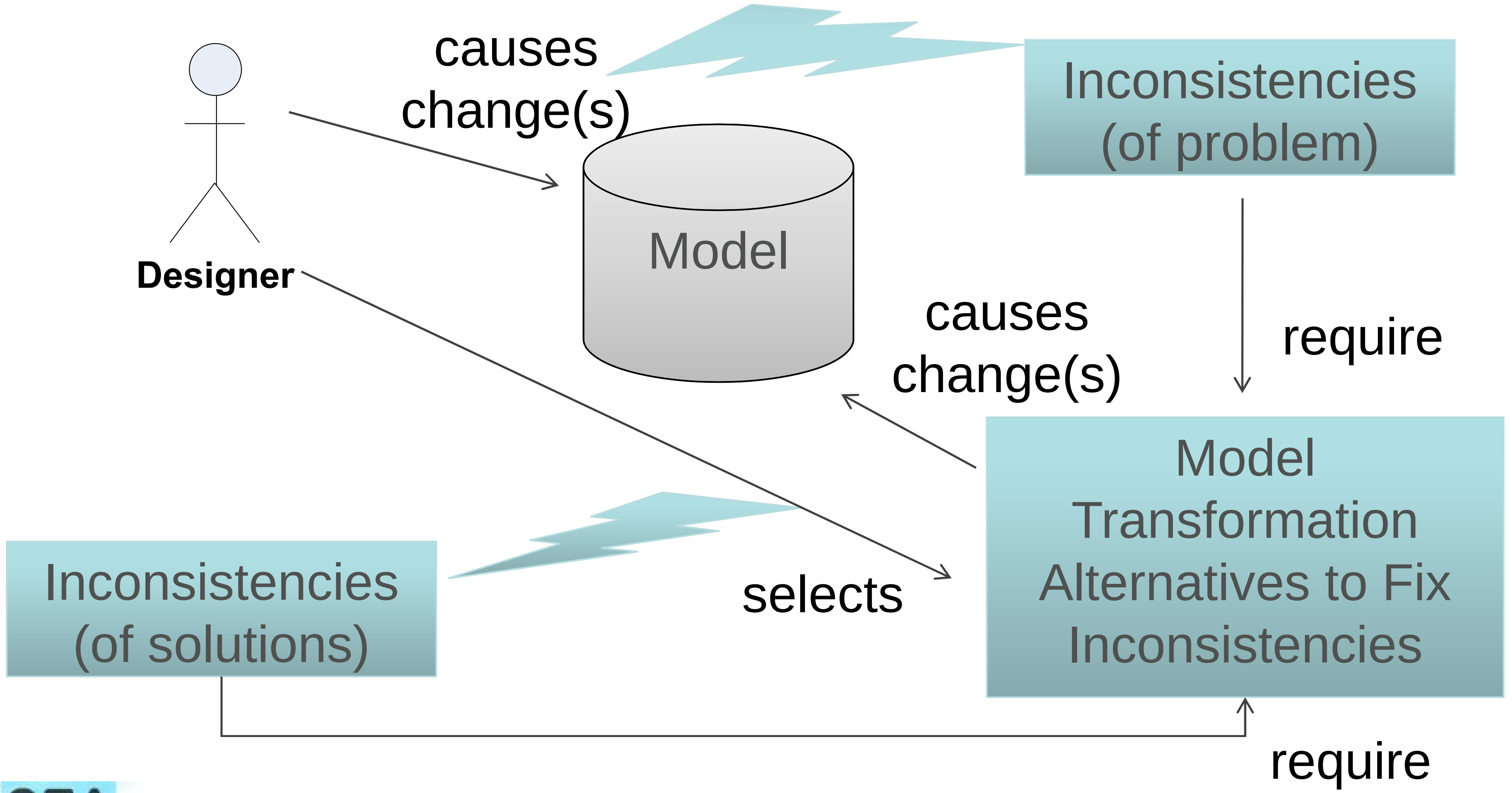
(c)

N-angulation


All Possible Choices for Change Propagation
6
7
5
4
CM2
1
2
3
10
9
7
8
/CM3
Constraint from Model 1 (CM1)

Single Change Possible

The Many Uses of Constraints during Transformation



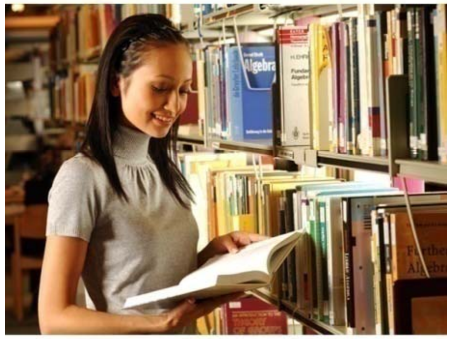
causes change(s)
Inconsistencies (of problem)
Designer
Model
causes change(s)
require
Model Transformation Alternatives to Fix Inconsistencies
Inconsistencies (of solutions)
selects
require

Know When To Transform

- Constraints are the guards to define when to transform
- Constraints are also the utility functions to gauge a transformations success
  - A failure caused during transformation implies wrong transformation or incomplete transformation
- Exploring Alternatives requires a toolbox of transformations
  - Small and (perhaps) larger ones

We would like to gratefully acknowledge the Austrian Science Fund (FWF) grants P21321-N15 and M1268-N23, and the EU Marie Curie Actions – Intra European Fellowship (IEF) project 254965

JOHANNES KEPLER UNIVERSITY LINZ | JKU



Contact me at alexander.egyed@jku.at

Johannes Kepler University, Linz (JKU)

http://www.sea.jku.at