IMT 2013
Lucca, Italy
September 2013

Part 1.3 Modal I/O Transition Systems as Semantics of UML4SOA

Martin Wirsing
LMU München

Ludwig-Maximilians-Universität München

in co-operation with Sebastian Bauer, Rolf Hennicker, Philip Mayer

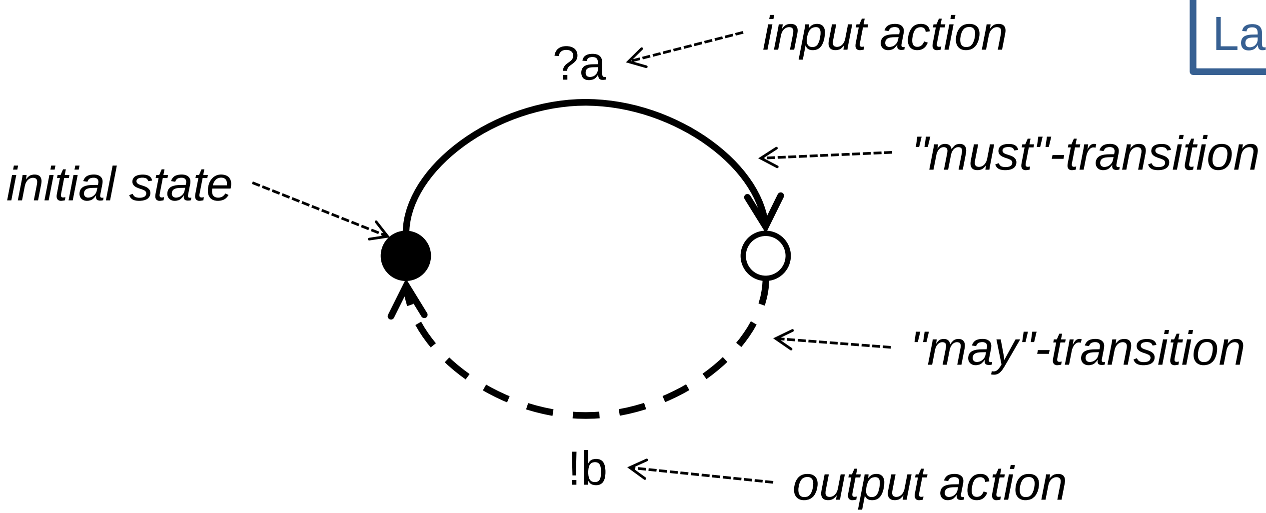
Modal I/O-Transition Systems (MIOs)

- Modalities ("may" and "must") for refinement (vertical relationship)
  - "must": what is required (~ bisimulation)
  - "may": what is optional (~ trace inclusion refinementt)
- Input/output for compatibility (horizontal relationship)
- Synchronous composition (shared actions are internalized)
- Output Compatibility (any outputs must be received)

Astrid Lindgren 1954
www.villa-galactica.de/

Modal I/O-Transition Systems (MIOs)

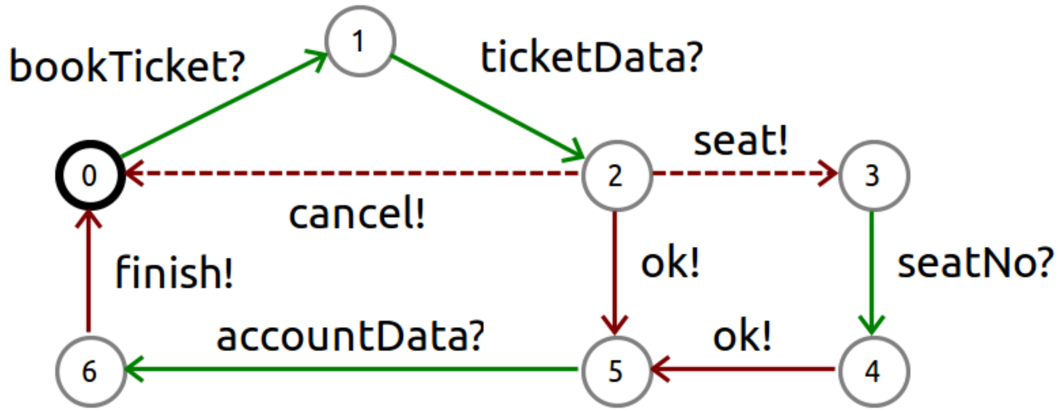
Larsen, Thomsen 1988
Larsen et al. 2007



- Formally: S = (states, start, act, $\dashrightarrow$, $\longrightarrow$)
  where
  - act = in ∪ out ∪ int(ernal)
  - $\longrightarrow \; \subseteq \; \dashrightarrow$ "every must is a may"

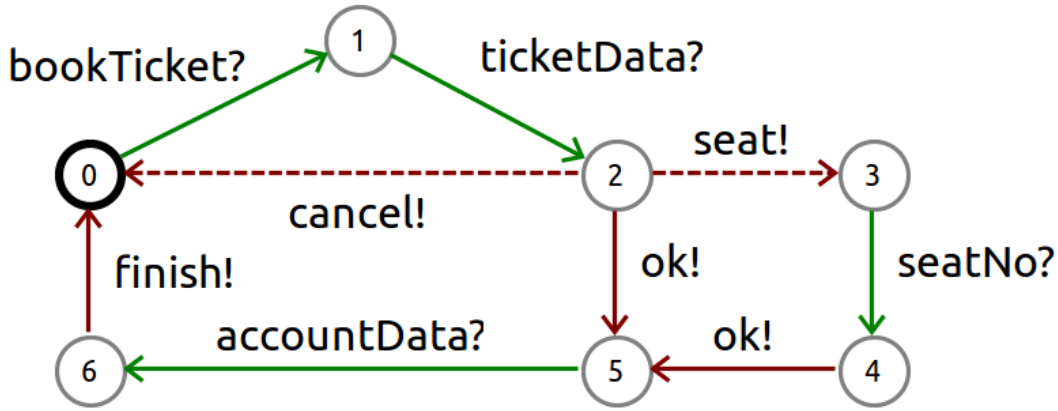
Example: Flight Booking Service

- **Server**

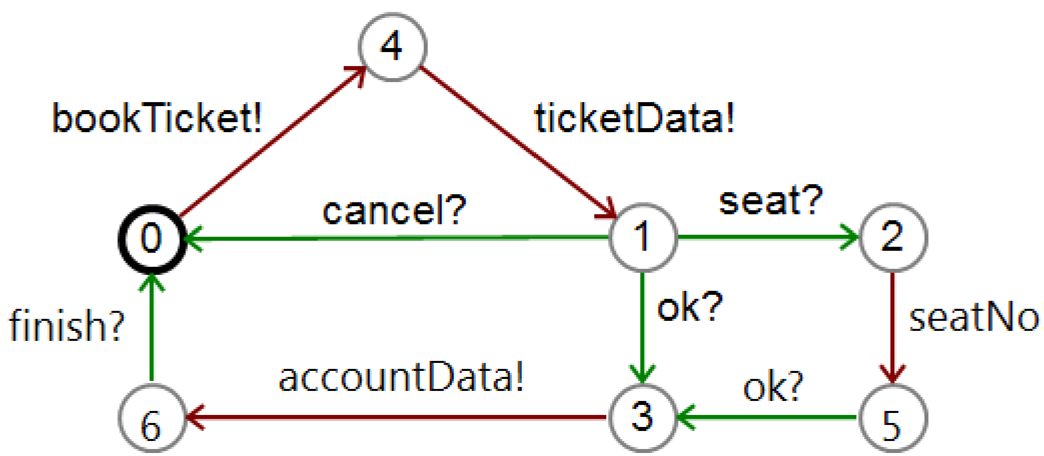


Example: Flight Booking Service

- **Server**

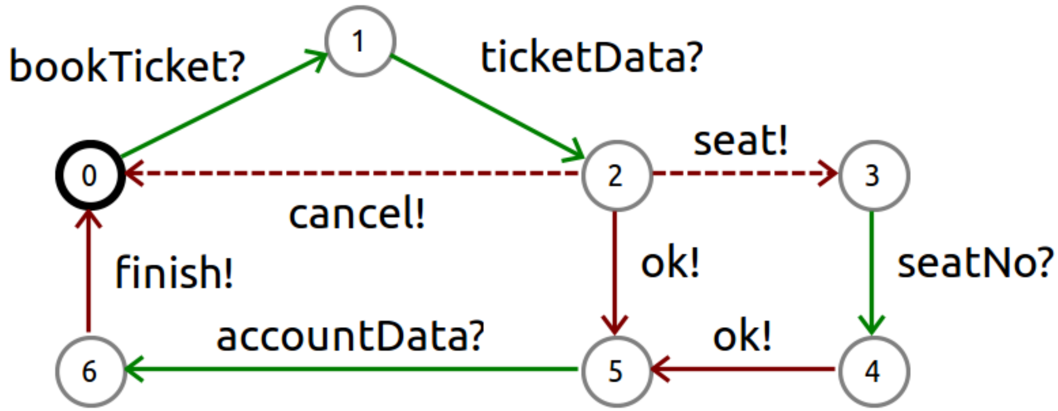


- **Client**

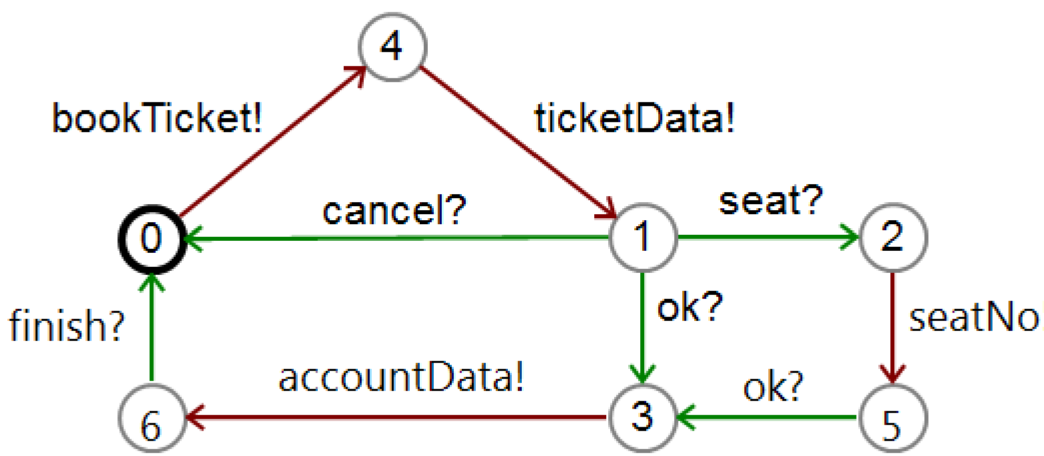


Flight Booking Service
(Client Server Synchronous Composition)

- **Server**



- **Client**



Composability

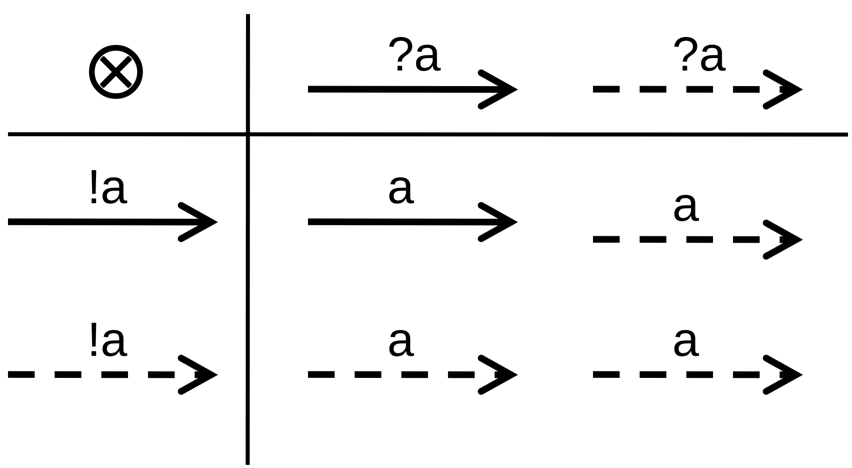
- Two MIOs are called composable if
  overlapping of actions only happens on complementary types:

Definition 4 (Composability [LNW07a]) *Two MIOs S and T are called composable if* $(in_S \cup int_S) \cap (in_T \cup int_T) = \emptyset$ *and* $(out_S \cup int_S) \cap (out_T \cup int_T) = \emptyset$.

- Server and Client are composable.

Composition

- Composition of MIOs synchronises transitions with matching shared actions and same type of transition
  - E.g. a must-transition labeled with a shared action occurs in the composition if there exists a corresponding matching must-transition in the original MIOs
  - A may-transition labeled with a shared action occurs in the composition if there exists a corresponding matching (may- or must-) transition in the original MIOs



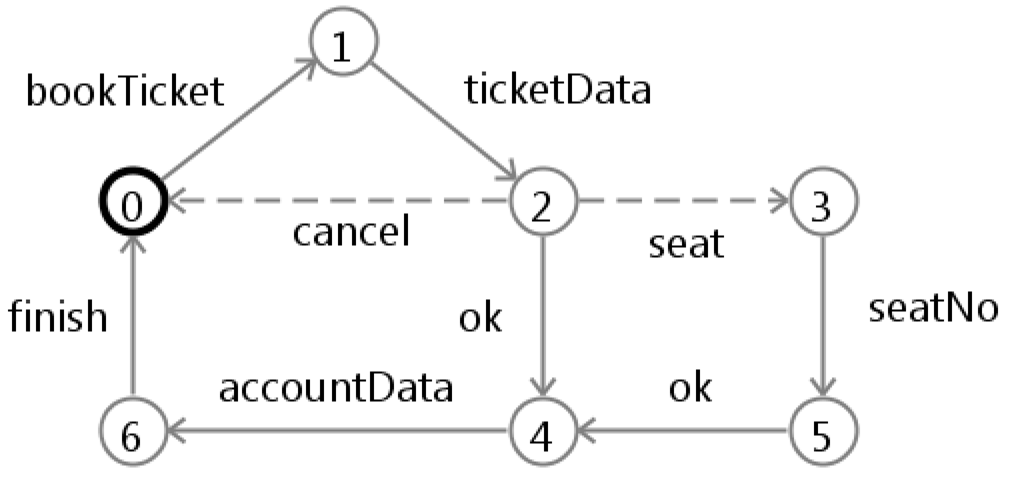
Synchronous Composition Formally

Definition 5 (Composition [LNW07a]) *Two composable MIOs* S_1 *and* S_2 *can be composed to a MIO* $S_1 \otimes S_2$ *defined by* $states_{S_1 \otimes S_2} = states_{S_1} \times states_{S_2}$, *the initial state is given by* $start_{S_1 \otimes S_2} = (start_{S_1}, start_{S_2})$, $in_{S_1 \otimes S_2} = (in_{S_1} \setminus out_{S_2}) \cup (in_{S_2} \setminus out_{S_1})$, $out_{S_1 \otimes S_2} = (out_{S_1} \setminus in_{S_2}) \cup (out_{S_2} \setminus in_{S_1})$, $int_{S_1 \otimes S_2} = int_{S_1} \cup int_{S_2} \cup (in_{S_1} \cap out_{S_2}) \cup (in_{S_2} \cap out_{S_1})$. *The transition relations* $\dashrightarrow_{S_1 \otimes S_2}$ *and* $\longrightarrow_{S_1 \otimes S_2}$ *are given by, for each* $\rightsquigarrow \in \{\dashrightarrow, \longrightarrow\}$,

- *for all* $i, j \in \{1, 2\}, i \neq j$, *for all* $a \in (act_{S_1} \cap act_{S_2})$, *if* $s_i \overset{a}{\rightsquigarrow}_{S_i} s_i'$ *and* $s_j \overset{a}{\rightsquigarrow}_{S_j} s_j'$ *then* $(s_1, s_2) \overset{a}{\rightsquigarrow}_{S_1 \otimes S_2} (s_1', s_2')$,
- *for all* $a \in act_{S_1}$, *if* $s_1 \overset{a}{\rightsquigarrow}_{S_1} s_1'$ *and* $a \notin act_{S_2}$ *then* $(s_1, s_2) \overset{a}{\rightsquigarrow}_{S_1 \otimes S_2} (s_1', s_2)$,
- *for all* $a \in act_{S_2}$, *if* $s_2 \overset{a}{\rightsquigarrow}_{S_2} s_2'$ *and* $a \notin act_{S_1}$ *then* $(s_1, s_2) \overset{a}{\rightsquigarrow}_{S_1 \otimes S_2} (s_1, s_2')$.

Composition Example

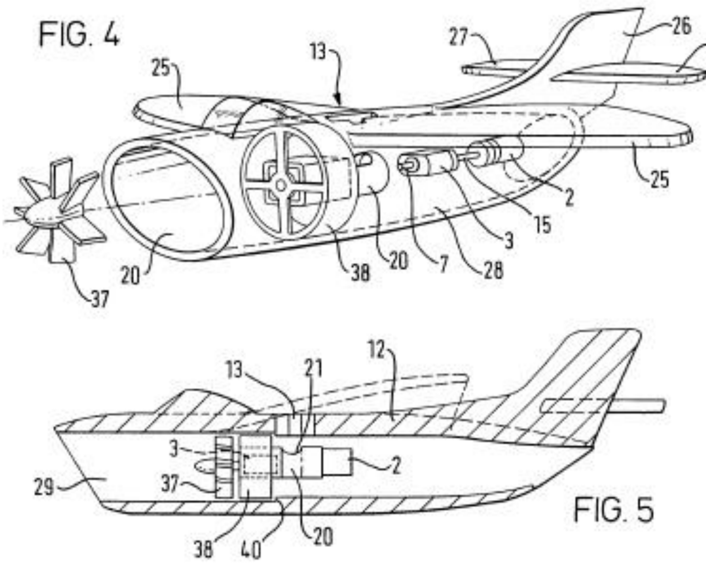
- **Server ⊗ Client =**



The MIO Workbench

Bauer et al. (TACAS) 2010

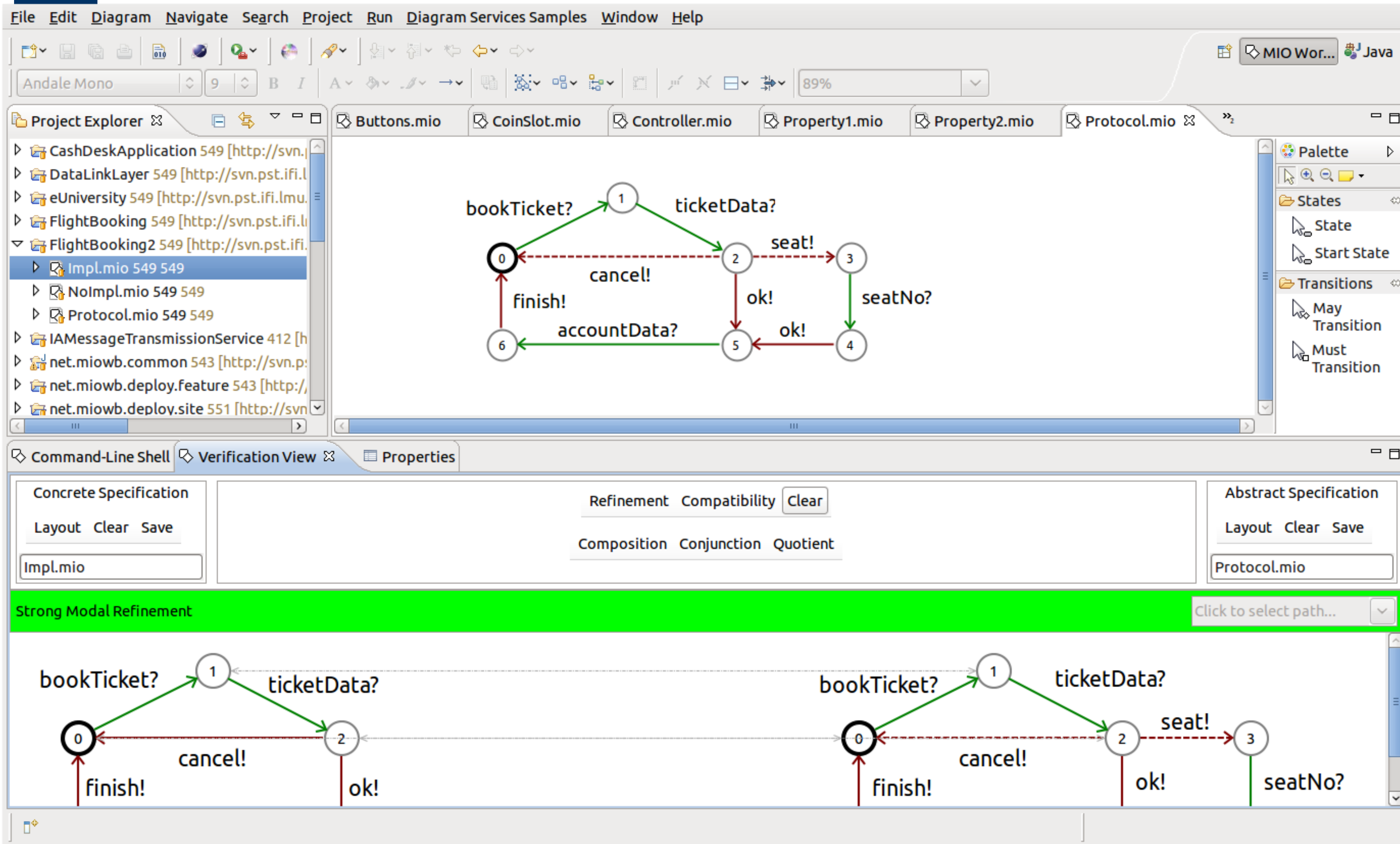
- The MIO Workbench is an Eclipse-based verification tool for Modal I/O-Transition Systems.
- Features:
  - Graphical editor for MIOs
  - Implementations of
    - Refinement: Strong, Weak, May-Weak
    - Compatibility: Strong, Weak, "Helpful Environment"
    - Composition
  - Graphical Relation and Error View
  - Easily extendable and easy installation via software manager inside Eclipse
- See http://www.miowb.net !



http://patent.kilu.de/

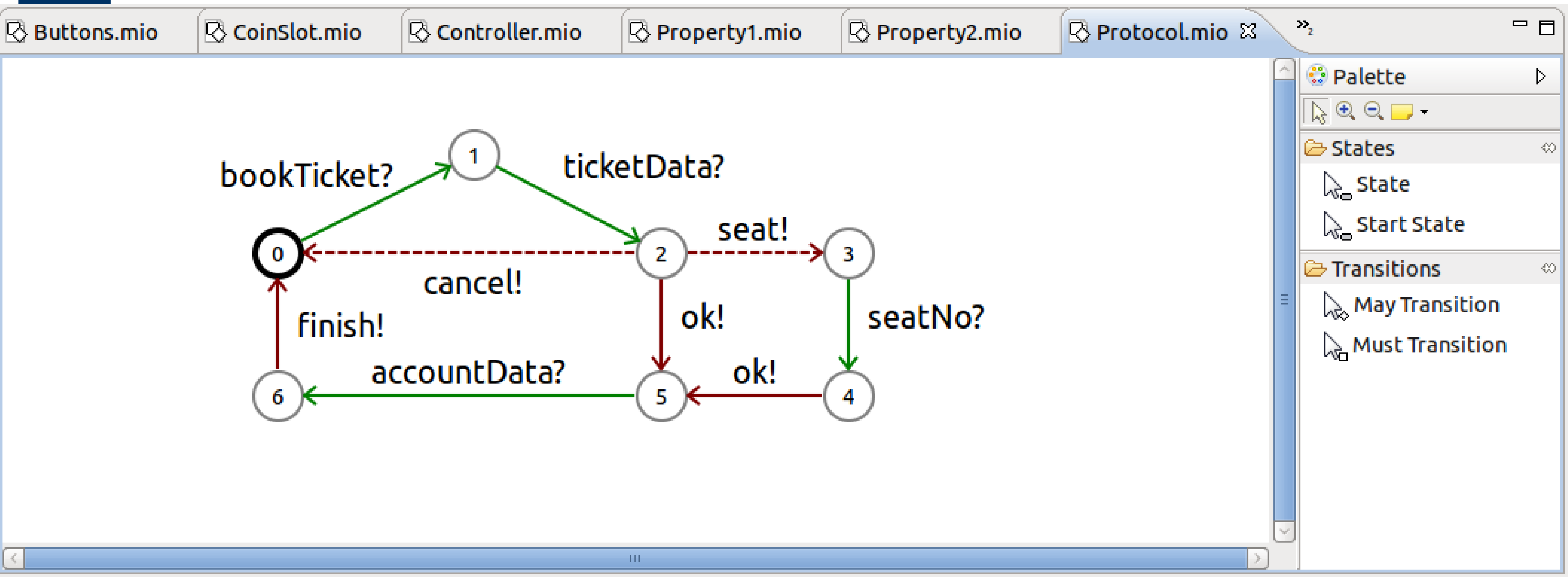
Mio's model airplane

MIO Workbench Perspective



MIO Workbench: Graphical Editor

- Graphical editor for creating and modifying MIOs



MIO Workbench: Textual Editor

- Textual editor for writing scripts that can create MIOs and execute operations and verication tasks

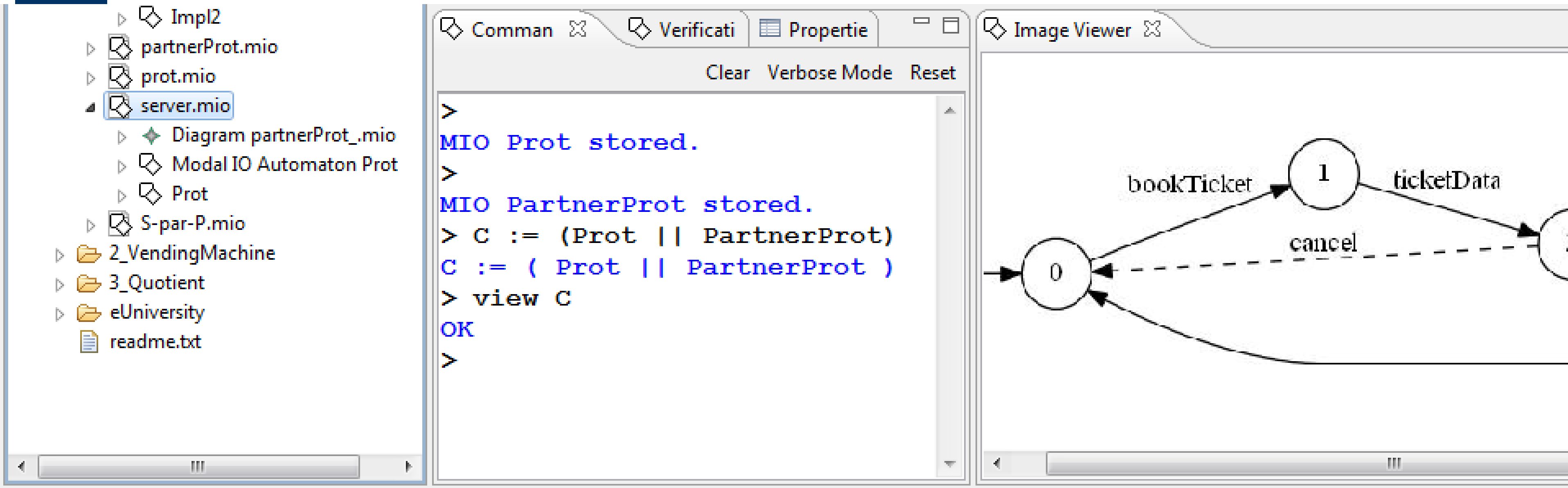
VendingMachine.miotx | CoinSlot.mio | Controller.mio | Property1.mio | Property2.mio | Protocol.mio

```
// Vending Machine Example
// Last Update: 26.08.2011

mio AbstractCore {
inputs coin, tea_selected, coffee_selected
outputs dispense_tea, dispense_coffee, return_coin
internals activate
states a0, a1, a2, a3, a4
start a0
mustTransitions
 a0 -> a1 [ coin ]
mayTransitions
 a1 -> a2 [ coffee_selected ],
 a1 -> a3 [ tea_selected ],
 a3 -> a4 [ dispense_tea ],
 a2 -> a4 [ dispense_coffee ],
 a1 -> a1 [ coin ],
 a3 -> a3 [ coin ],
```

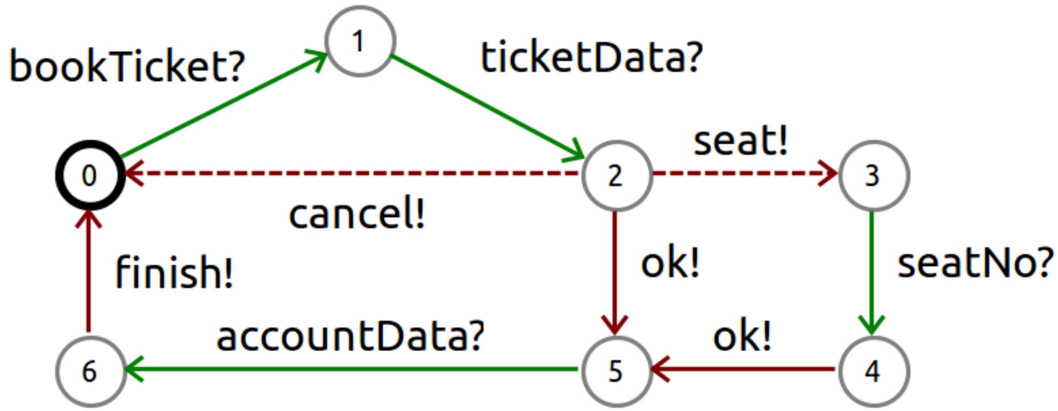
Command-Line Shell

- Interpreter for executing complex verification tasks
- Example: Composition of Server and Client

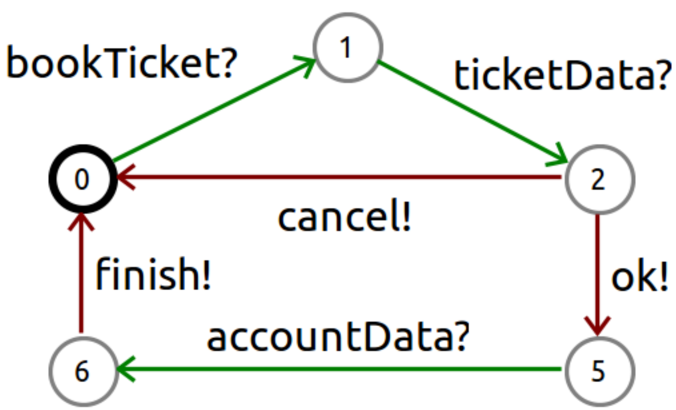


Refinement

- **Server**

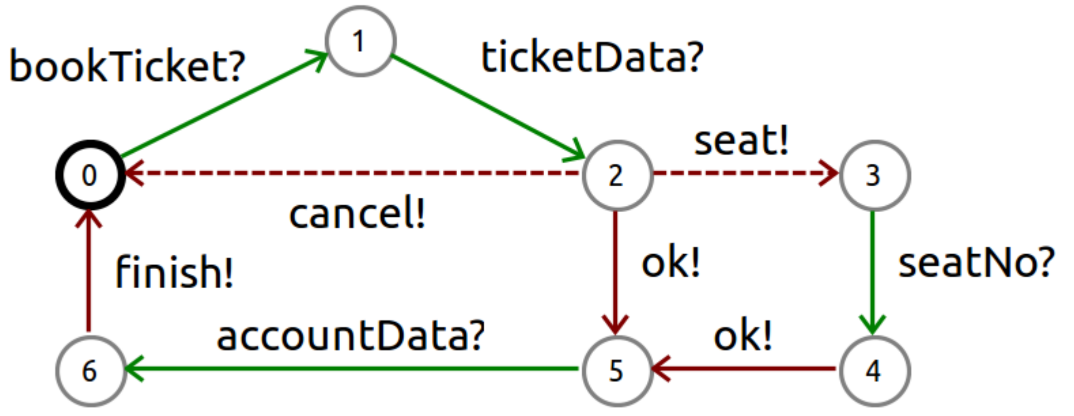


- **Possible Refinement**

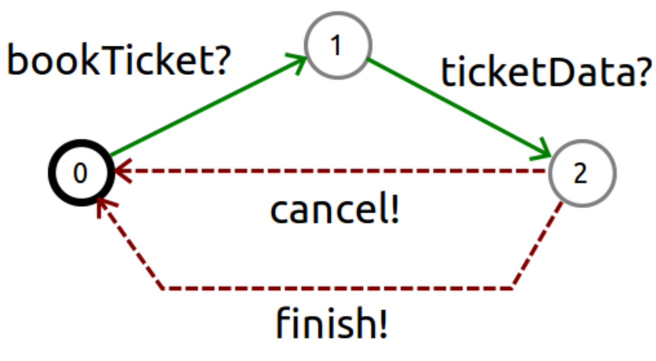


Wrong Refinement

- **Server**



- **Wrong Refinement**



Refinement Formally

- Idea
  1. any required (must) transition in the abstract specification must also occur in the concrete specification. Conversely,
  2. any allowed (may) transition in the concrete specification must be allowed by the abstract specification.
  3. in both cases the target states must conform to each other.

Definition 3 (Strong Modal Refinement [LT88b]) *Let S and T be MTSs (MIOs, resp.) with the same signature. A relation $R \subseteq states_S \times states_T$ is called strong modal refinement for S and T iff for all $(s,t) \in R$ and for all $a \in act_S$ it holds that*

1. *if $t \xrightarrow{a}_T t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{a}_S s'$ and $(s',t') \in R$,*
2. *if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \overset{a}{\dashrightarrow}_T t'$ and $(s',t') \in R$.*

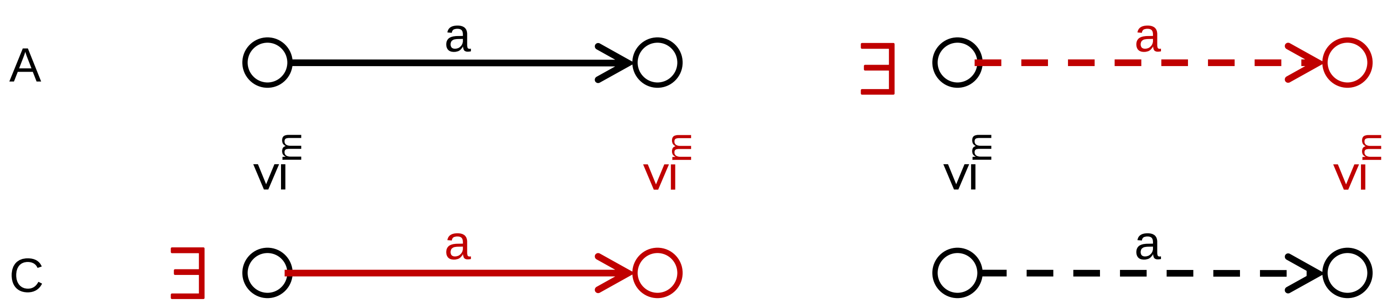
We say that S strongly modally refines T, written $S \leq_m T$, iff there exists a strong modal refinement for S and T containing $(start_S, start_T)$.

Refinement

- Idea

Larsen, Thomsen 1988

1. any required (must) transition in the abstract specification must also occur in the concrete specification. Conversely,
2. any allowed (may) transition in the concrete specification must be allowed by the abstract specification.
3. in both cases the target states must conform to each other.



Refinement Formally

Definition 3 (Strong Modal Refinement [LT88b]) *Let S and T be MTSs (MIOs, resp.) with the same signature. A relation $R \subseteq states_S \times states_T$ is called strong modal refinement for S and T iff for all $(s,t) \in R$ and for all $a \in act_S$ it holds that*

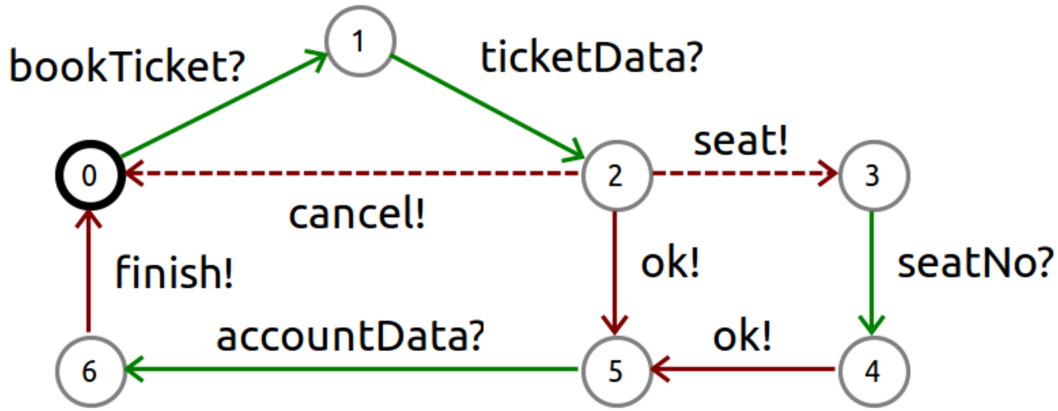
1. *if $t \xrightarrow{a}_T t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{a}_S s'$ and $(s',t') \in R$,*

2. *if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \overset{a}{\dashrightarrow}_T t'$ and $(s',t') \in R$.*

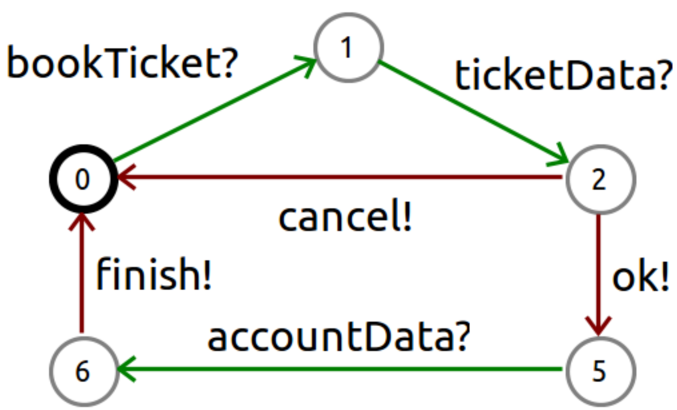
We say that S strongly modally refines T, written $S \leq_m T$, iff there exists a strong modal refinement for S and T containing $(start_S, start_T)$.

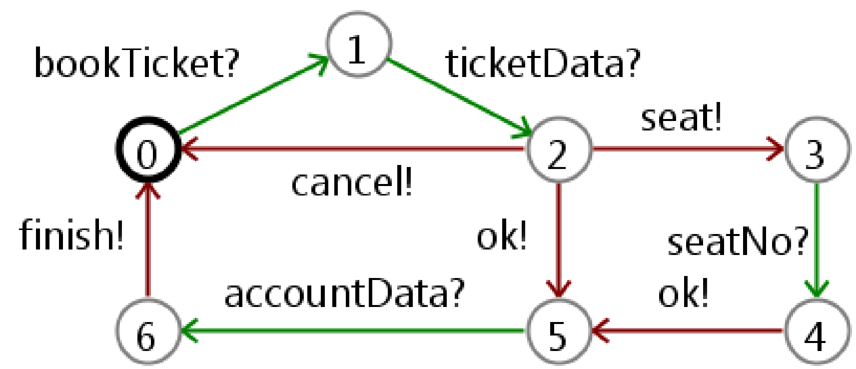
Refinement Examples

- **Server**



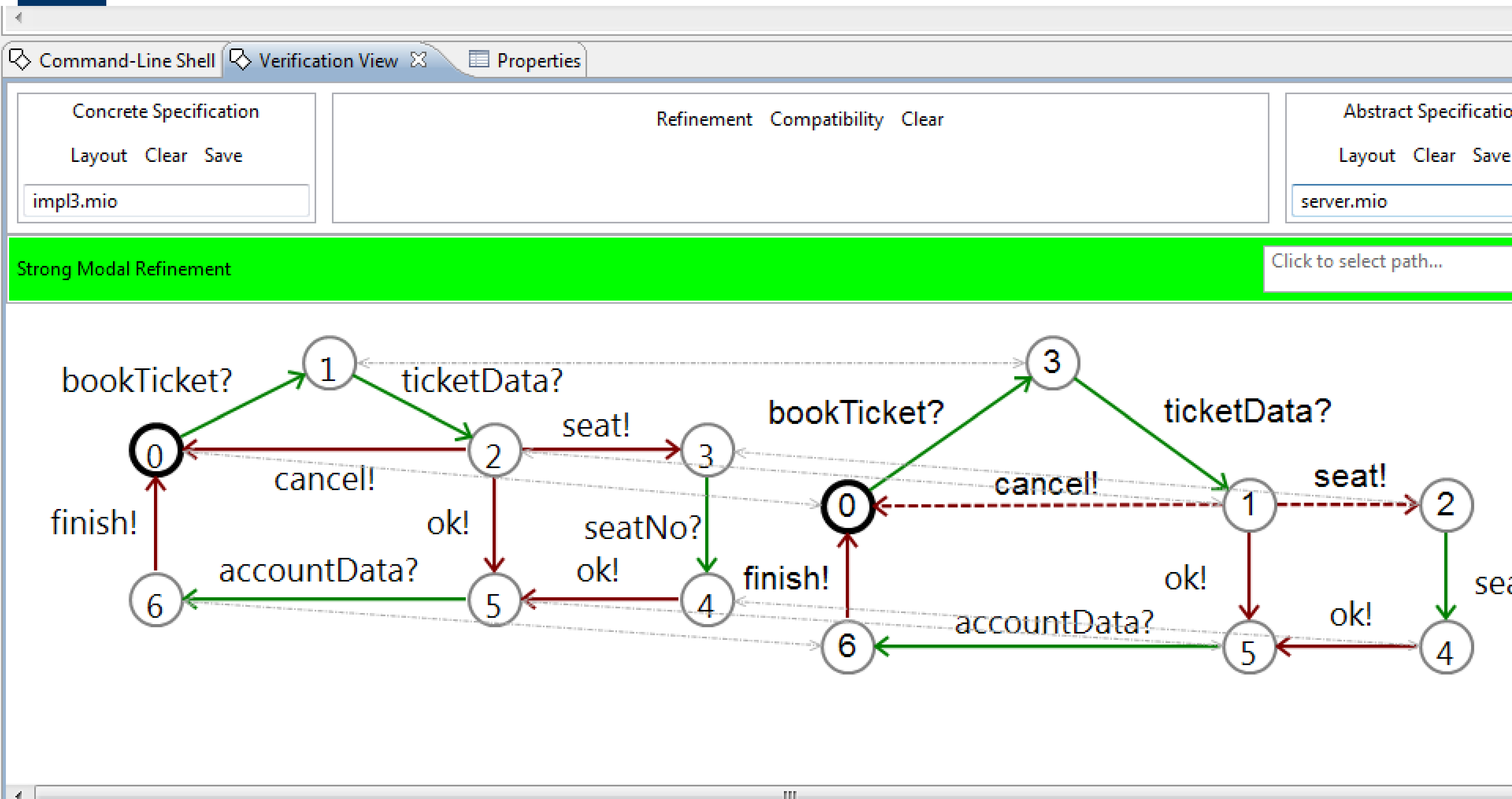
- **Strong Modal Refinements**





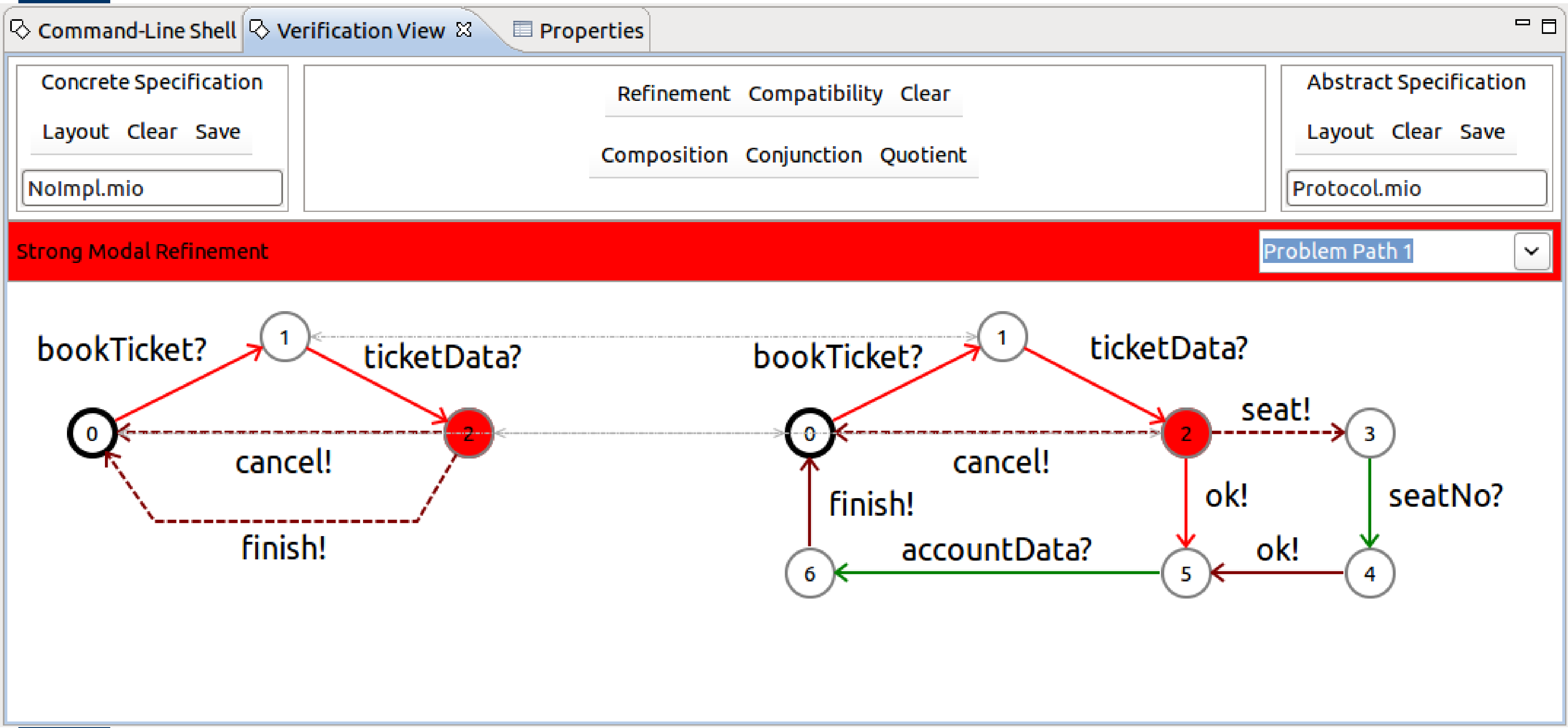
MIO Workbench: Verification View

- The verification view provides a way to visually execute individual operations and depict the results graphically.



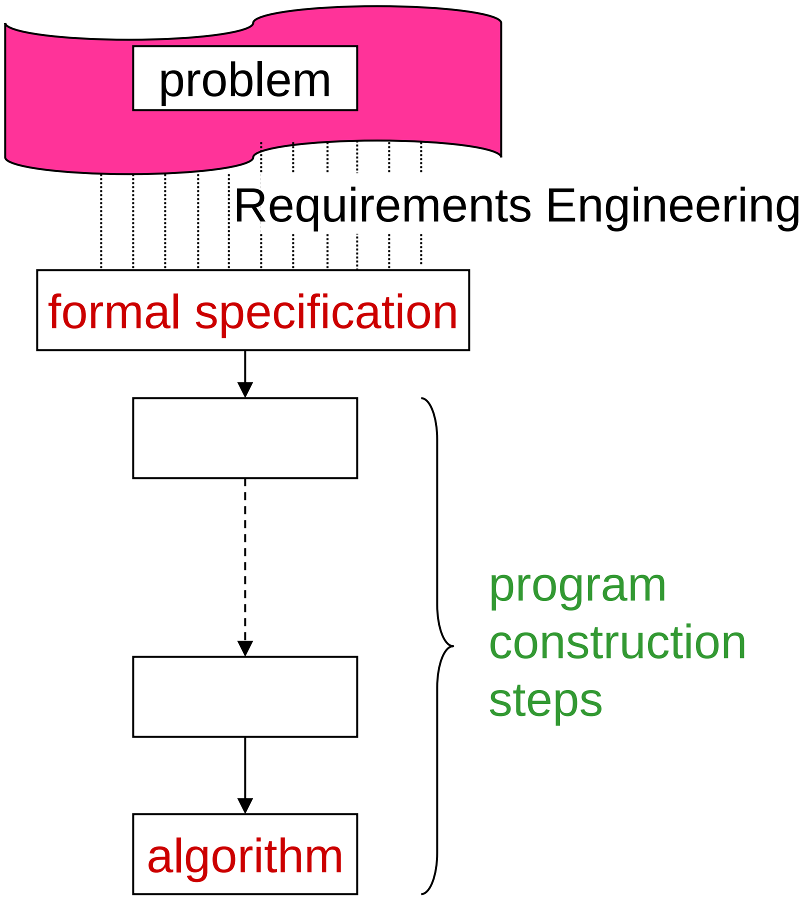
MIO Workbench: Verication View Example

- Wrong Refinement



Excursion Program Development and Interface Theories

- Formal Program Development
  - **from specifications**
  - **to programs**
  - **by transformations**
- Approaches
  - **CIP: Computer-aided Intuition-guided Programming [Bauer, Samelson 75]**
  - **Recursion elimination transformations [Burstall, Darlington ~75]**
  - **Model-based development with Z [Suffrin, Abrial 78] and B [Abrial ~80]**

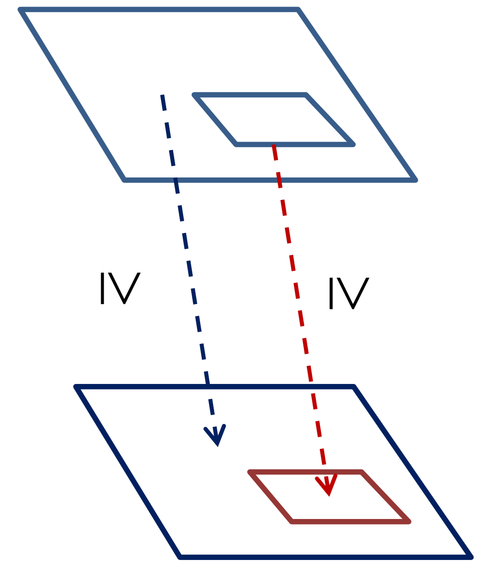


Excursion:
Compositional program development

- Refinement
  - $SP \geq SP_1$
- Vertical composition (Transitivity)
  - from abstract to more concrete specifications
    $SP \geq SP_1 \geq \ldots \geq SP_n$
- Horizontal composition (Monotonicity)
  - $SP \geq SP_1$ and $P \geq P_1$
    $\Rightarrow$
    $P[SP] \geq P_1[SP_1]$

  [Ehrig, Kreowski 83, Ehrich 82,

  Sannella, W 83, Maibaum 85, …]



M. Wirsing

Excursion: Interface Theories

- An **interface theory** is a tuple (**A**, ⊗, ≤, **~**) consisting of
  - a class **A** of specifications
  - a partial composition operator ⊗ : **A** x **A** →**A**
  - a binary refinement preorder ≤
  - a symmetric compatibility relation ~

  satisfying

  1. **compositional refinement**:
     If C ≤ A, C' ≤ A', and A ⊗ A' is defined,
     then C ⊗ C' is defined and C ⊗ C' ≤ A ⊗ A'.
  2. **preservation of compatibility**:
     If A ~ A' and C ≤ A and C' ≤ A', then C ~ C'.

de Alfaro, Henzinger 2001
Fiadeiro ~ 2000
Maibaum ~1995

www.grovelandscapearchitecture.com

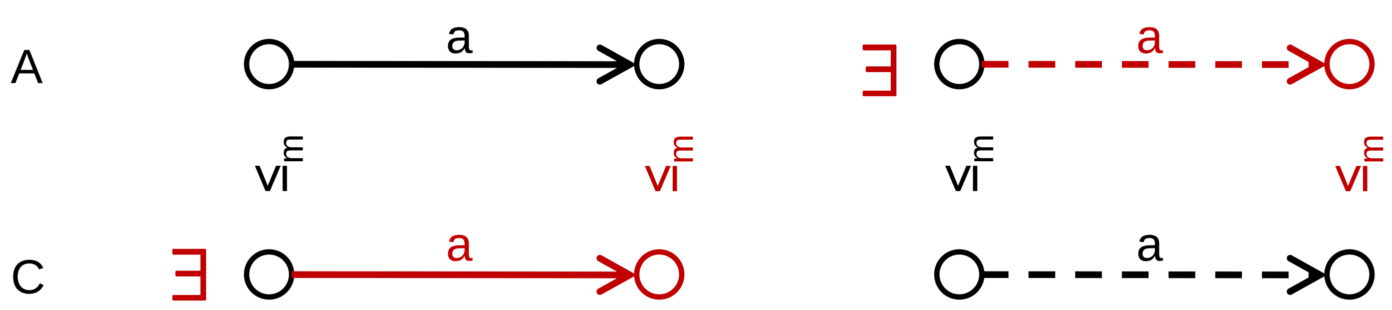
Interface Theory for MIOs

- (**MIO**, ⊗, $\leq_m$, $\sim_{sc}$) is an interface theory.
  - ⊗ is the synchronous composition operator on MIOs
  - $\leq_m$ is strong modal refinement
  -

    $C \leq_m A$ if
    - every **must**-transition in A is simulated by C
    - every **may**-transition in C is simulated by A

Bauer et al. (TACAS) 2010

Larsen, Thomsen 1988



Interface Theory for MIOs

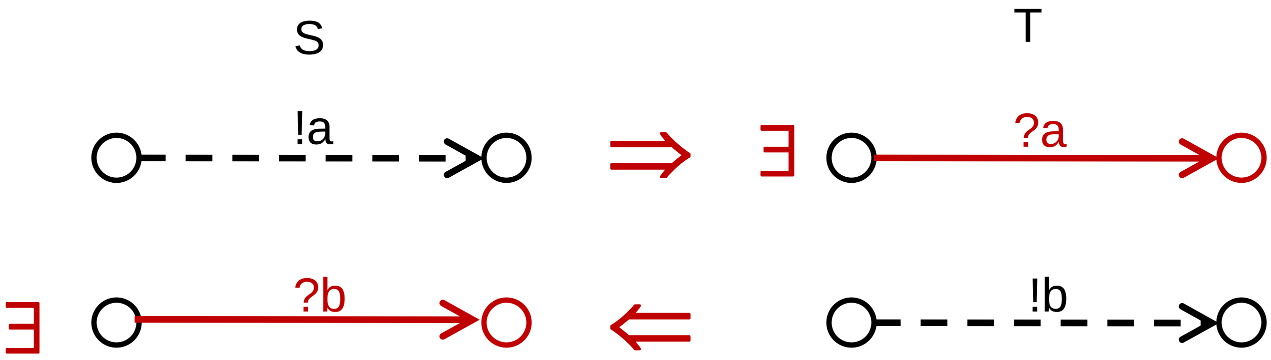
- ($\mathbf{MIO}$, ⊗, $\leq_m$, $\sim_{sc}$) is an interface theory.
  - ⊗ is the synchronous composition operator on MIOs
  - $\leq_m$ is strong modal refinement
  - $\sim_{sc}$ is strong output compatibility (partner must be input enabled)

Bauer et al. (TACAS) 2010

Larsen, Thomsen 1988

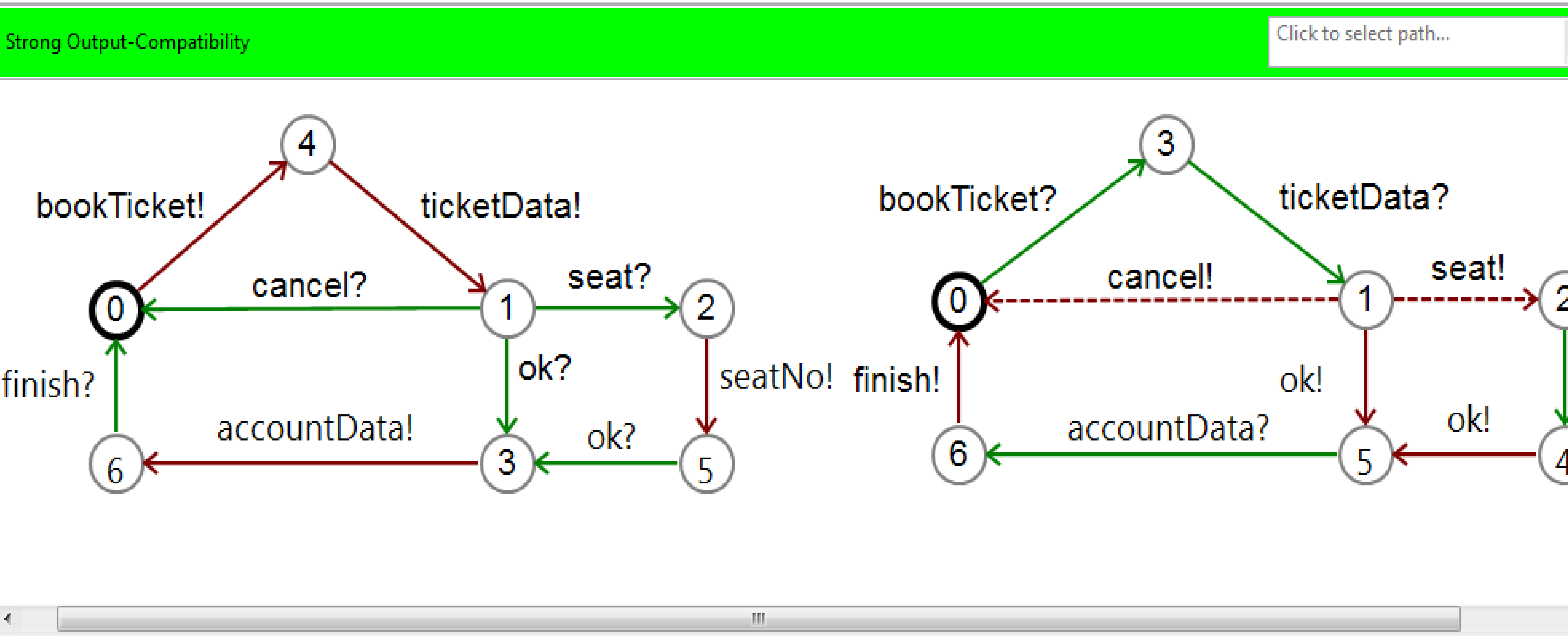
S $\sim_{sc}$ T if for every reachable state in S ⊗ T,

- if S **may** send an output shared with T, then T **must** be able to receive it, and conversely.



MIO Workbench: Strong Output Compatibility

- Example: Strong Output Compatibility of Client and Server



MIO Semantics for UML4SOA

- Denotational Semantics (compositional)
- Defines a function $\mathrm{mio}[\![\ldots]\!]$ which translates from UML4SOA behaviours and protocols to MIOs

- MIOs are a good match for the semantics of UML4SOA as:
  - Native support for input and output, which match the send and receive operations in UML4SOA
  - Distinguish between required and optional operations. Optional transitions (mays) in protocols are required to be able to verify optional implementation behaviour, for example compensation calls which might or might not be necessary

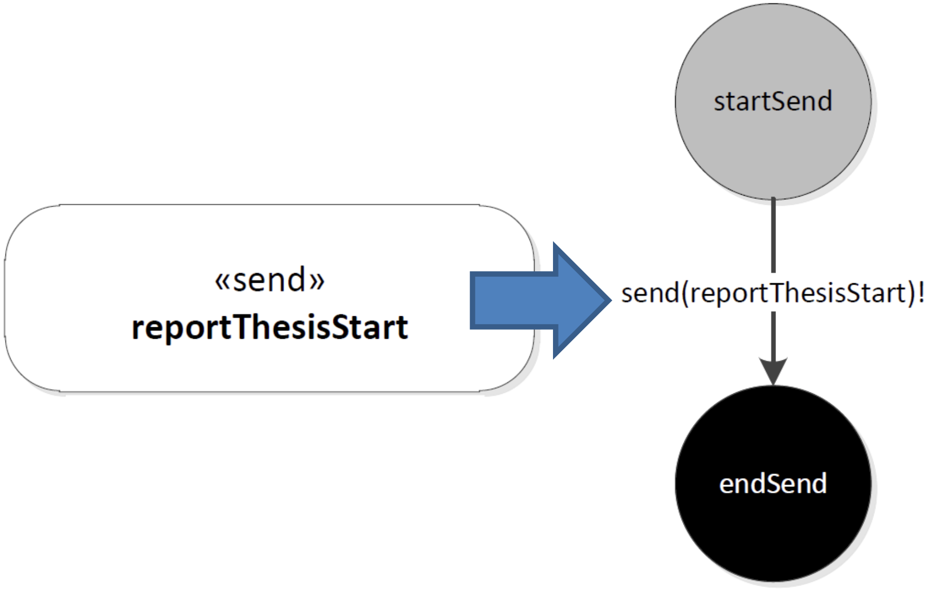
Semantics of activities

- **Simple actions** (like communication) are converted to transitions with an appropriate label

- **Structured actions** (like loops or decisions) are converted to their counterparts
  - Loop => back link
  - Decision => two outgoing transitions from previous state
  - Parallel => product automaton (interleaving composition)

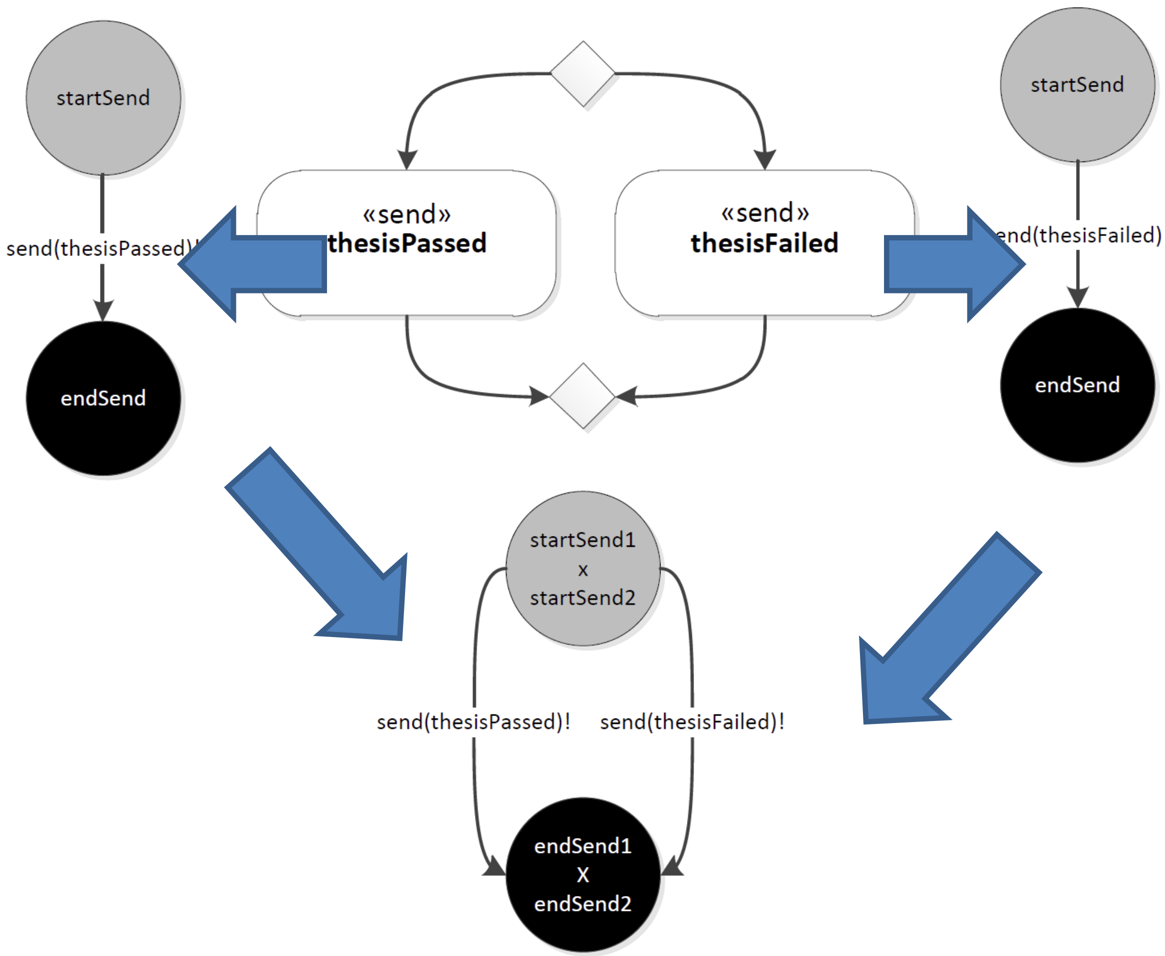
Example: Send

- All basic actions of UML4SOA are converted to transitions
  - Send/Reply => output action
  - Receive => input action
  - Send&Receive => both (in the appropriate order)



Example: Decision

- Each branch is converted first
- Afterwards, they are assembled



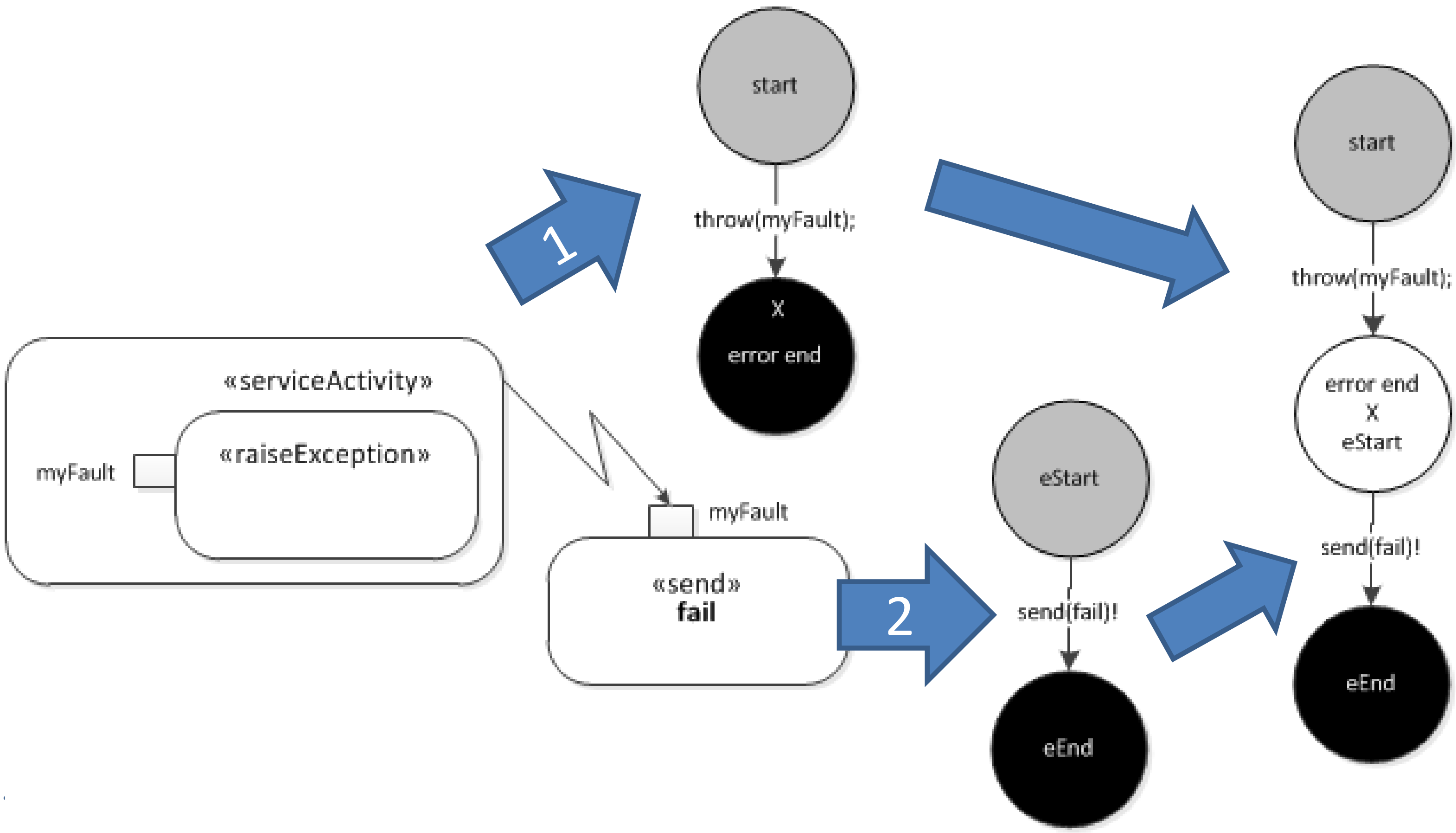
Service Activities

- Service Activities and handlers are more difficult
  - Service activity concept (grouping) does not exist on the MIO level – MIOs are flat

- **Event handlers** are added using standard interleaving (with an added loop, as they may be called more than once)
- **Compensation handlers** are converted to MIOs when encountered, then stored and added at the compensation site (i.e. the “compensate” call)
- **Exception Handlers** are likewise handled in a two-stage process, but inverse to compensation handers: When encountering a throw, a preliminary “throw” transition is added, to which the MIO of an exception handler is later appended

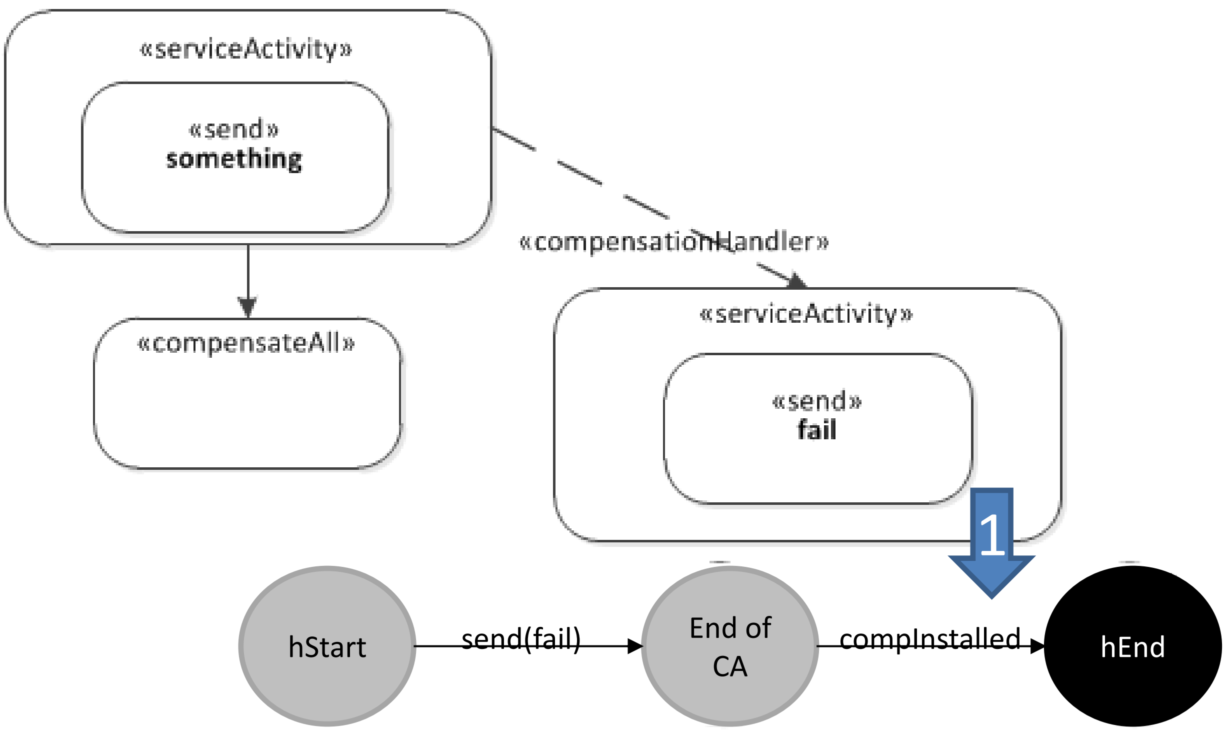
Example: Exception Handling

- Two-Stage process
  - First, a RaiseExceptionAction is added as a throw transition
  - If an exception handler is encountered later, it is attached to the automaton after the throw transition



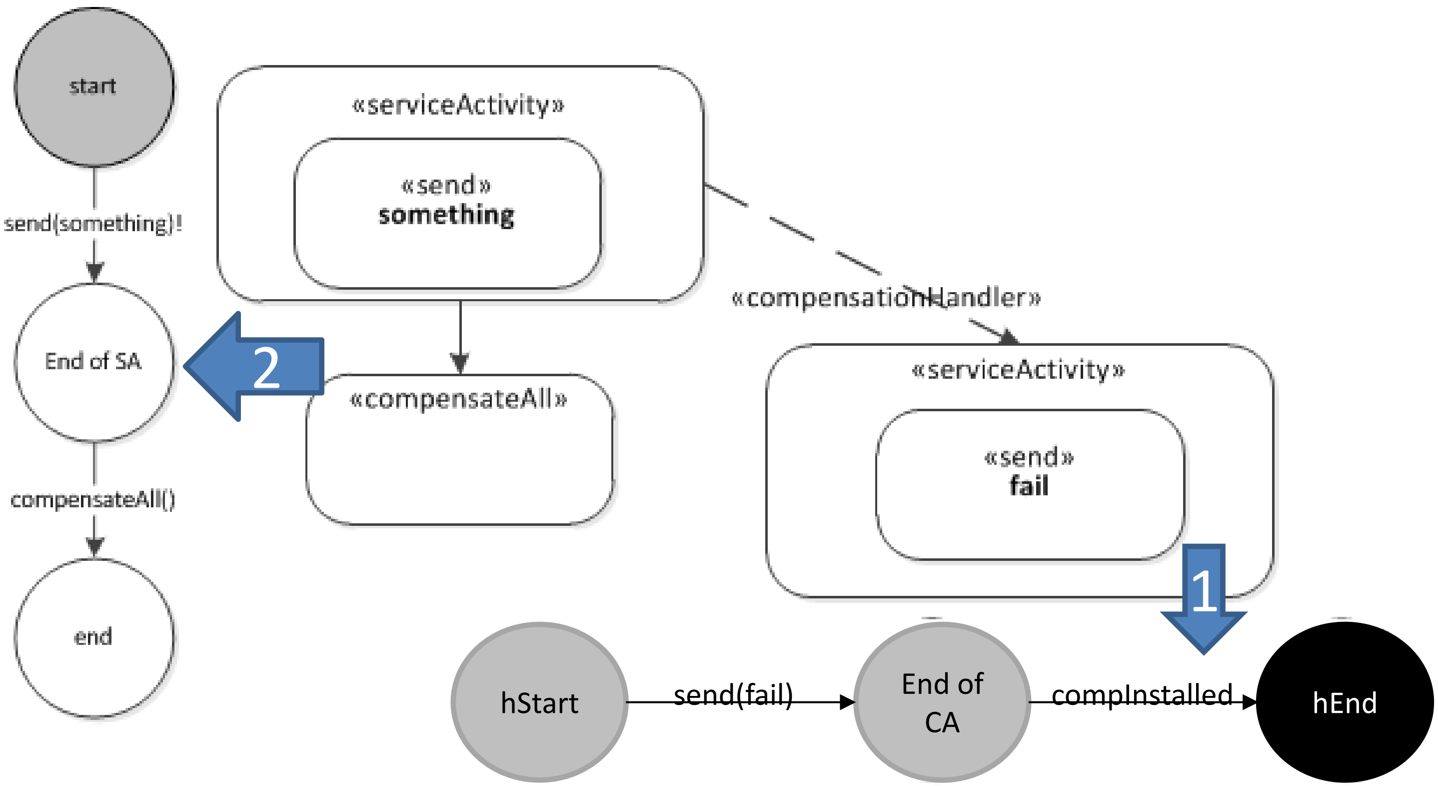
Example: Compensation Handling

- Two-Stage as before, but creating the handler first

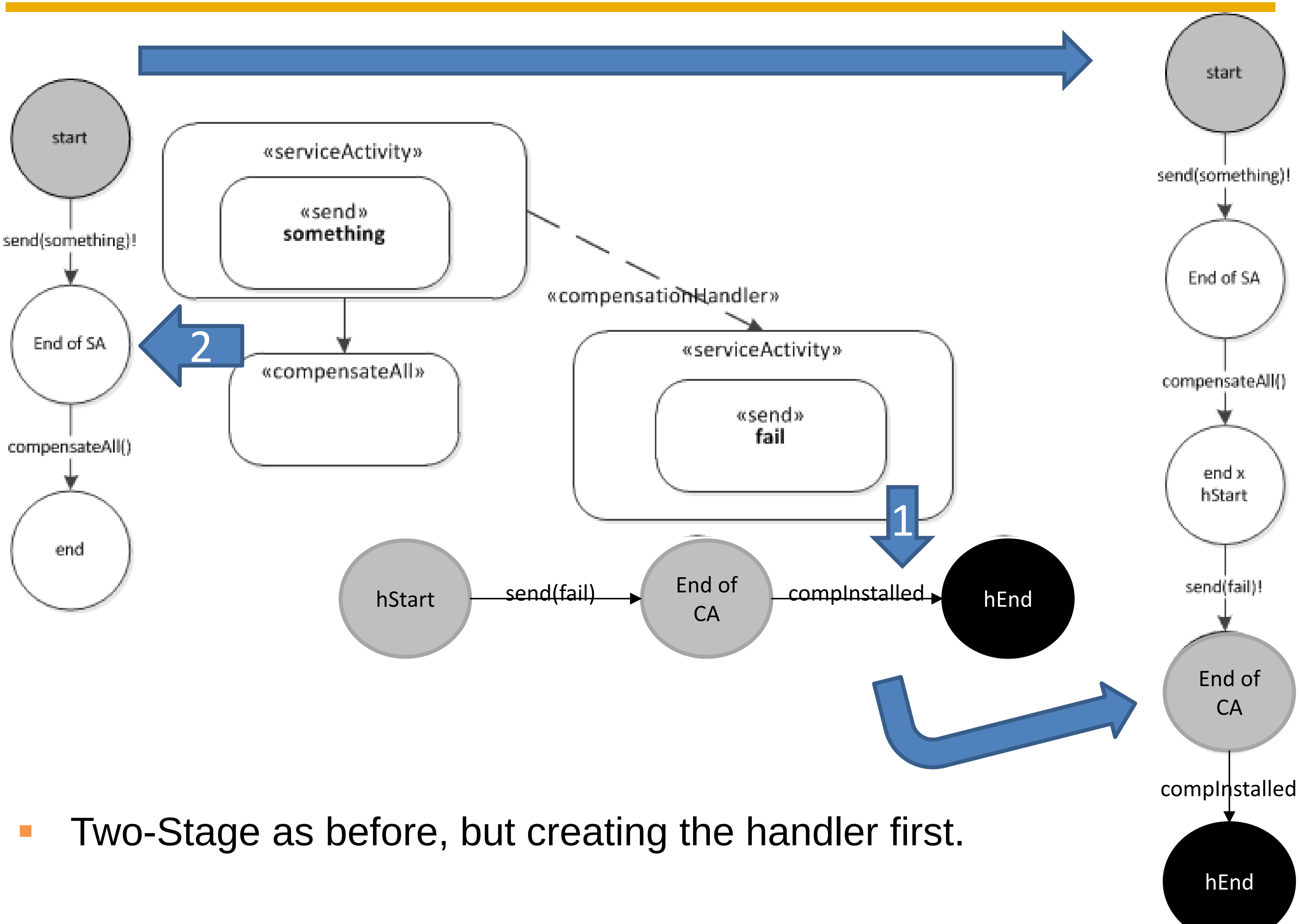


Example: Compensation Handling

- Two-Stage as before, but creating the handler first



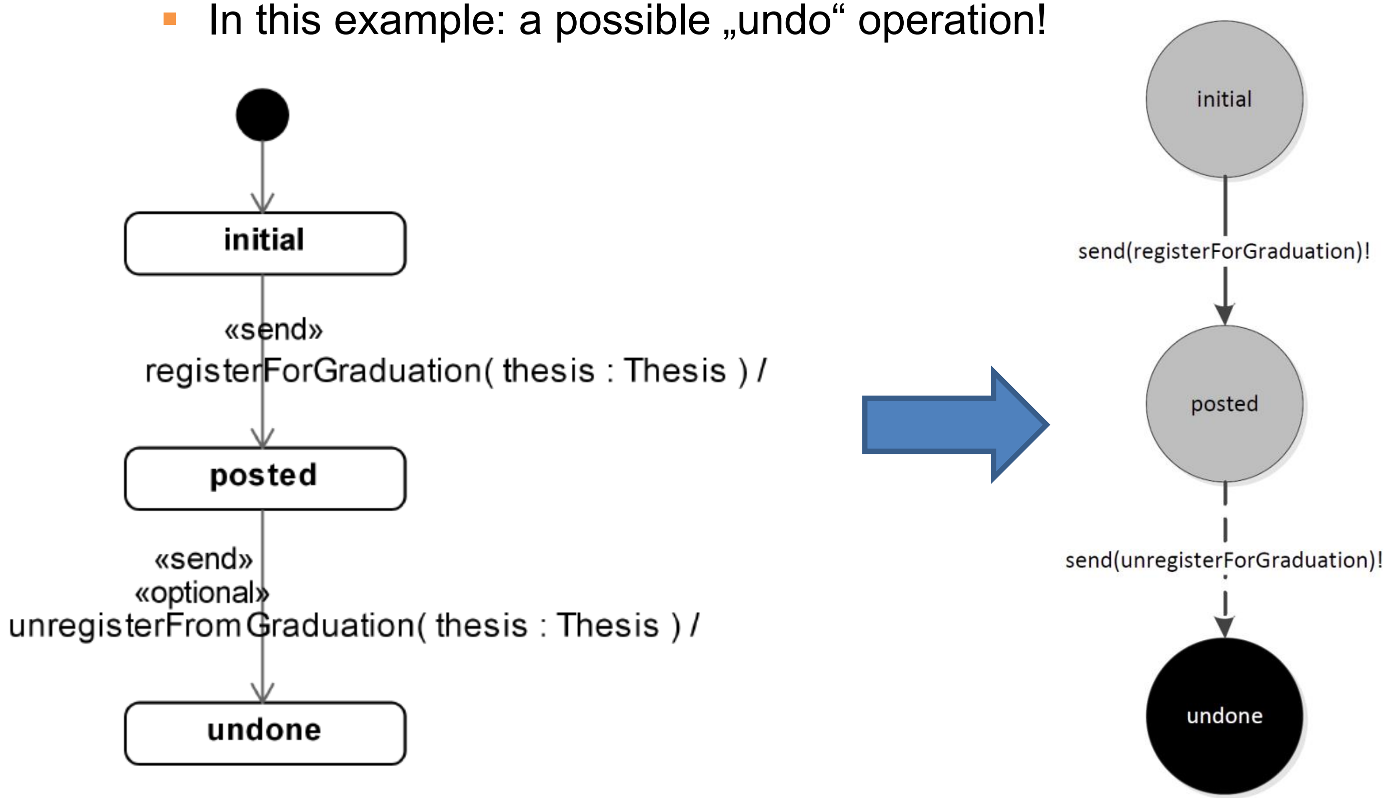
Example: Compensation Handling



- Two-Stage as before, but creating the handler first.

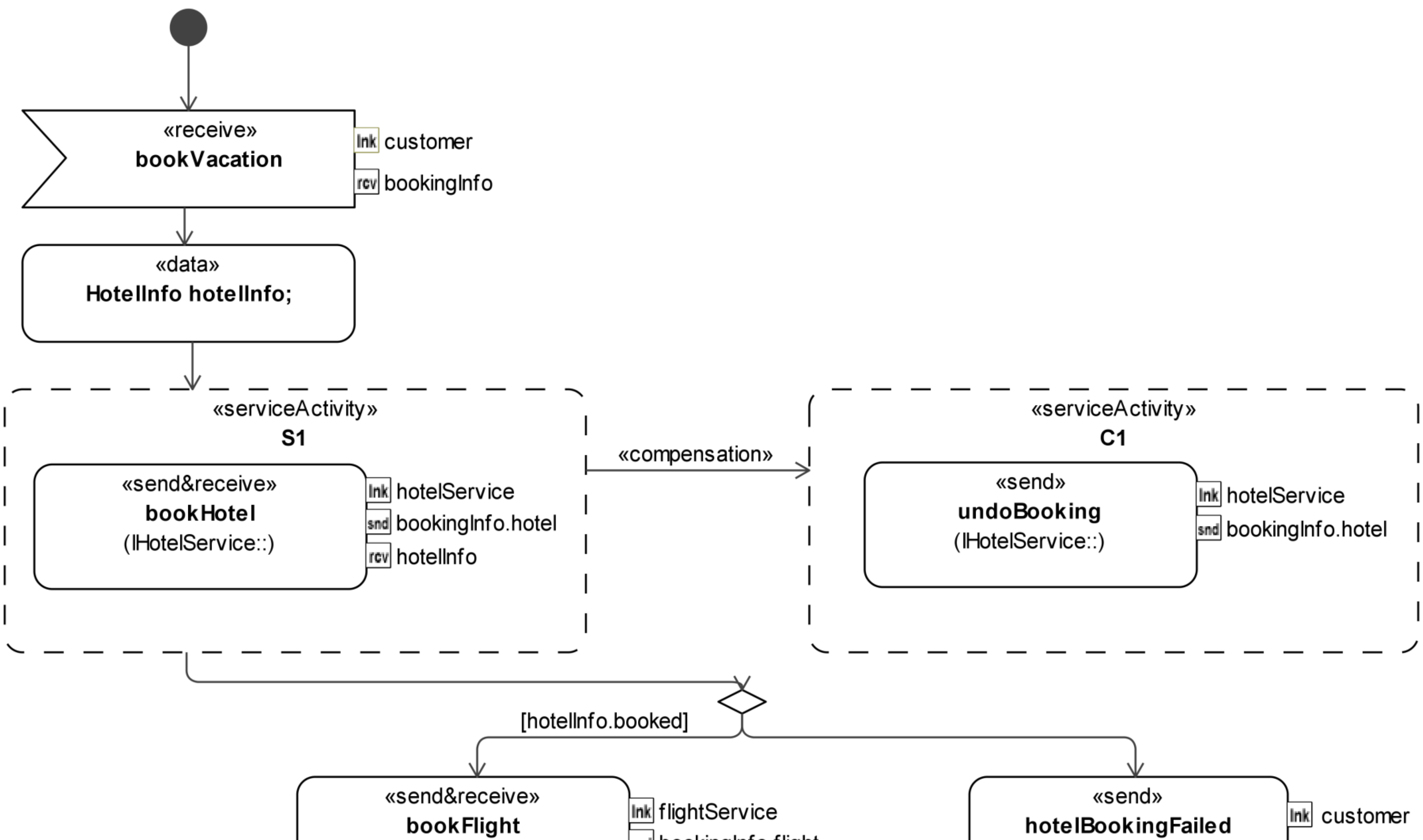
Semantics of Protocols

- UML4SOA Protocol State Machines are already close to MIOs
  - Send & Receive transitions can directly be translated to in- and output
  - Optional transitions are mays
    - In this example: a possible „undo“ operation!

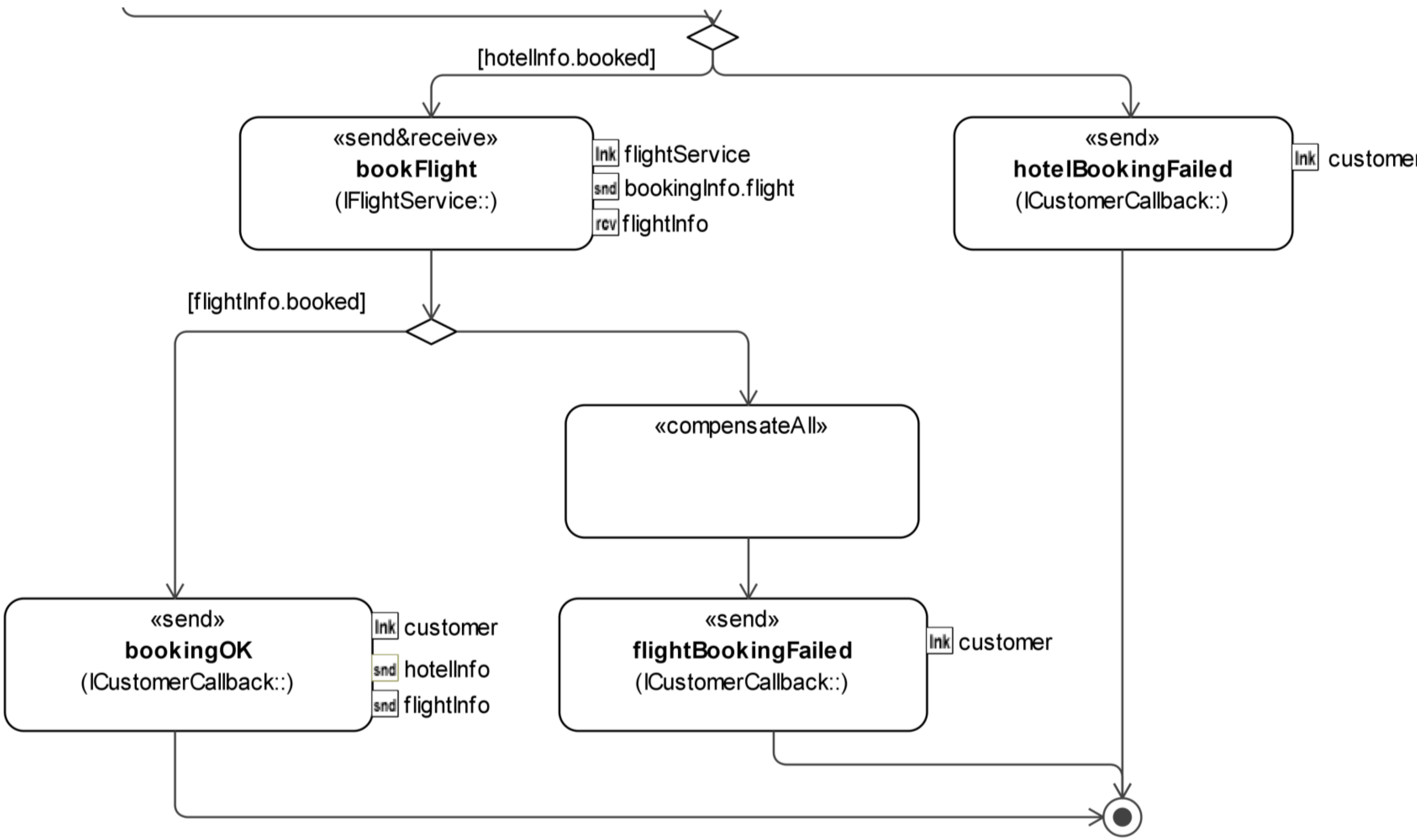


Translation Example: Vacation Booking

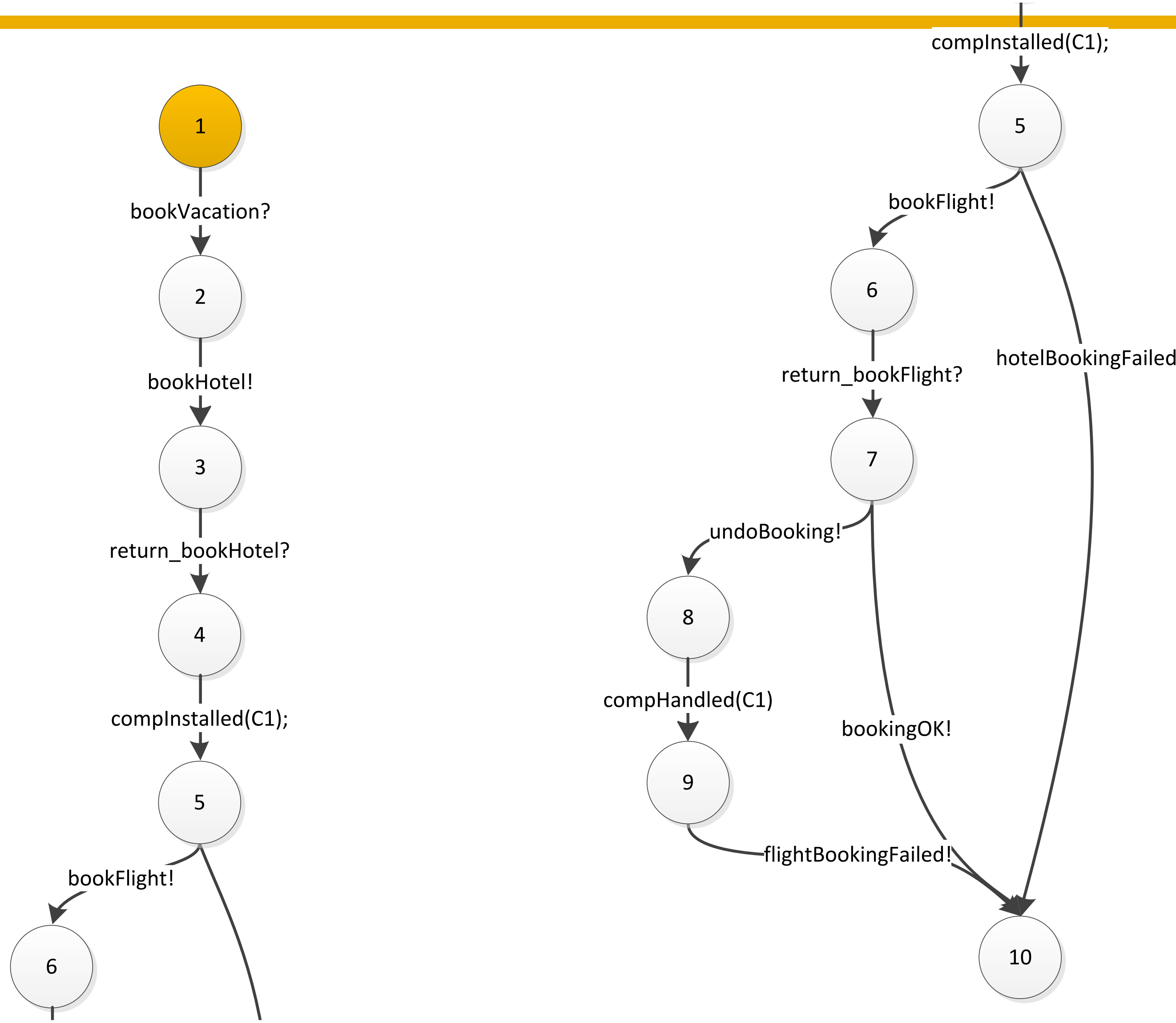
- UML4SOA



Example Vacation Booking



Vacation Example: MIO Translation



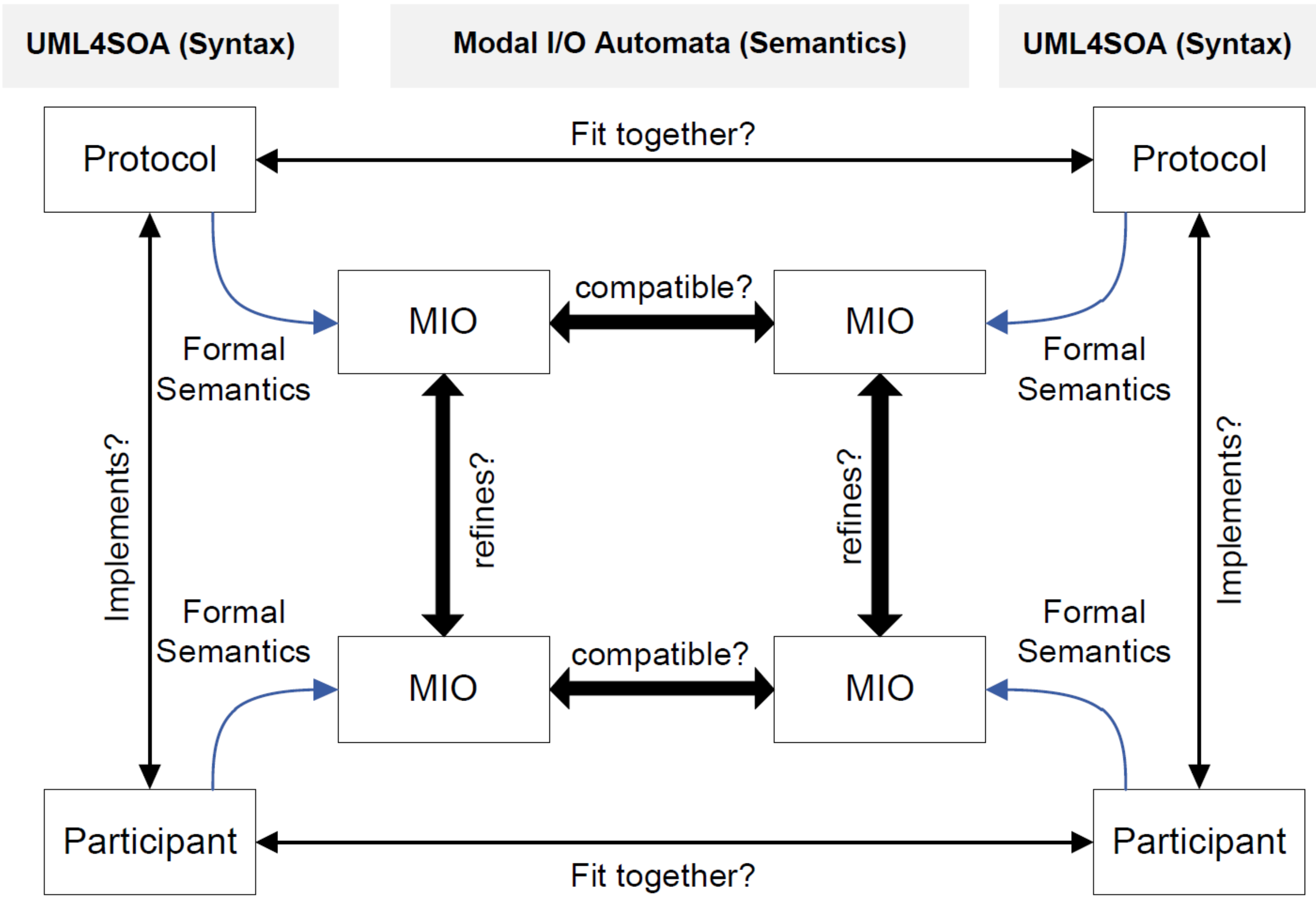
Using the Semantics

- The UML4SOA semantics can be used for formal analysis of UML models (by means of MIOs, and interface theories)

- In particular:
  - Refinement (i.e. does a service behaviour really implement the protocol it is supposed to fulfil?)
  - Compatibility (i.e. do two protocols really fit together?)

- An interface theory then guarantees that *compatibility is ensured under refinement*

Overview of Analysis Approach



Different Interface Theories

- Different interface theories can be used for analysis, depending on the use case

Loose

SO
(Strict-Observational)

Non-Race Deadlocks

Weak

Race Conditions

Strong

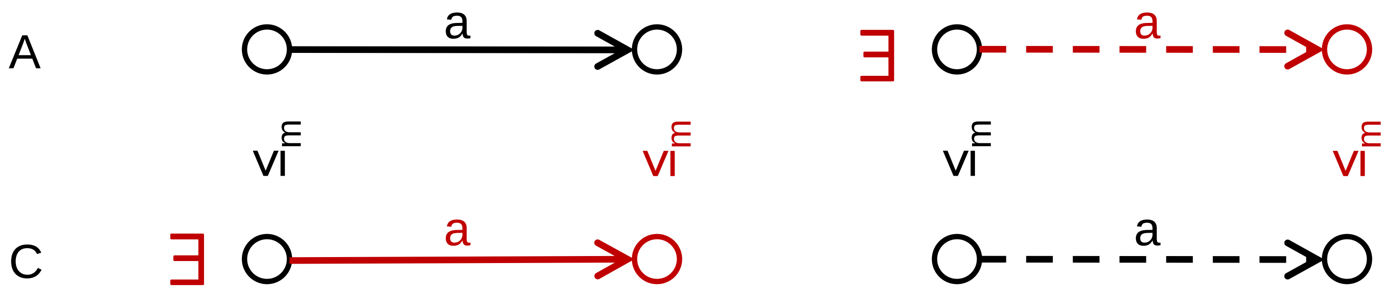
Strict Implementation

Hard

Strong Refinement for MIOs

$C \leq_m A$ if

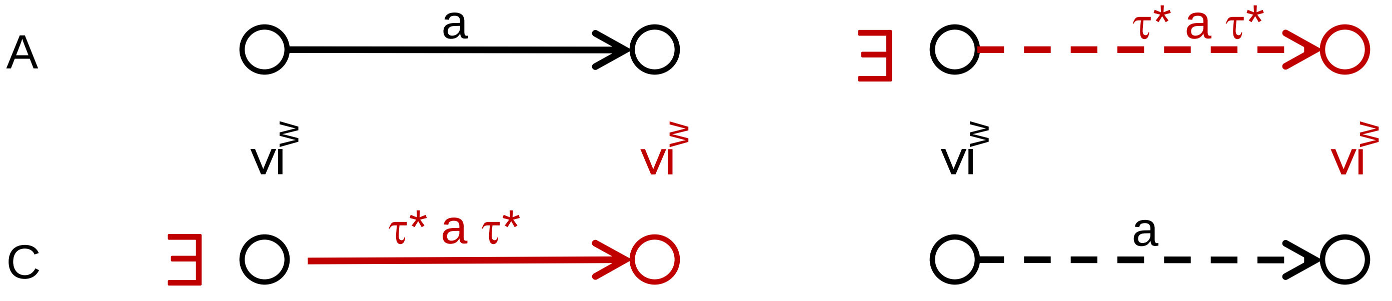
- every **must**-transition a in A is simulated by a in C
- every **may**-transition a in C is simulated by a in A



Weak Refinement for MIOs

$C \leq_w A$ if

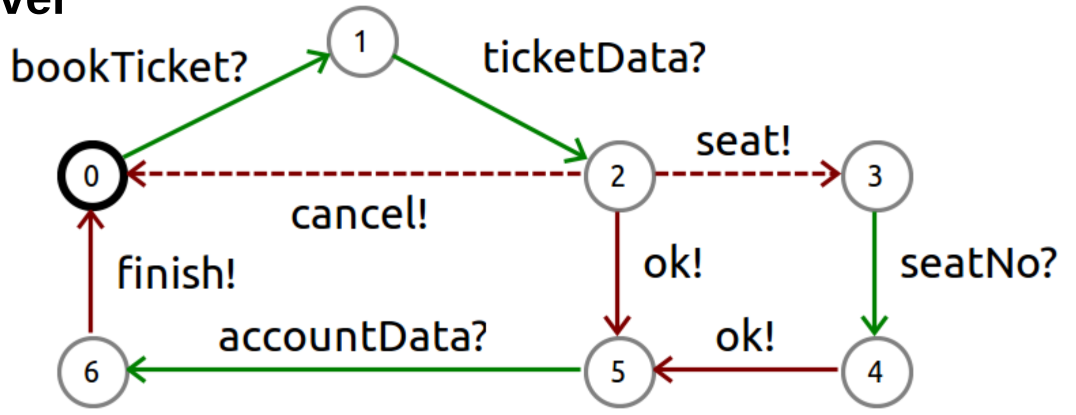
- every **must**-transition a in A is simulated by tau-embedded action a in C
- every **may**-transition a in C is simulated by tau-embedded action a A



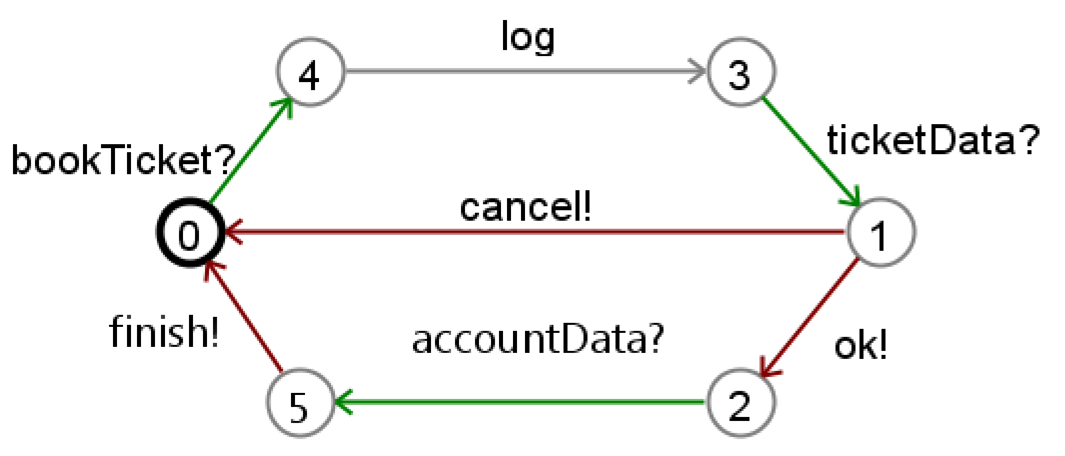
– special treatment for τ–actions (if a=τ, then the other automaton may also not move at all (ε))

Weak Refinement Example

- **Server**



- **Weak Refinement**



A 2nd Interface Theory for MIOs

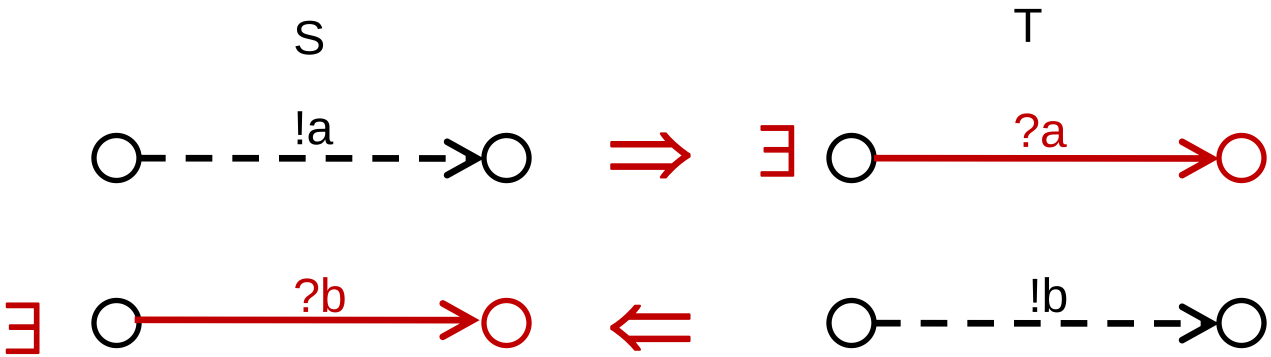
- ($\mathbf{MIO}$, $\otimes$, $\leq_m$, $\sim_{sc}$) is an interface theory.
  - $\otimes$ is the synchronous composition operator on MIOs
  - $\leq_{wm}$ is weak modal refinement
  - $\sim_{wc}$ is weak output compatibility (partner must be input enabled)

Bauer et al. (TACAS) 2010

$S \sim_{wc} T$ if for every reachable state in $S \otimes T$,

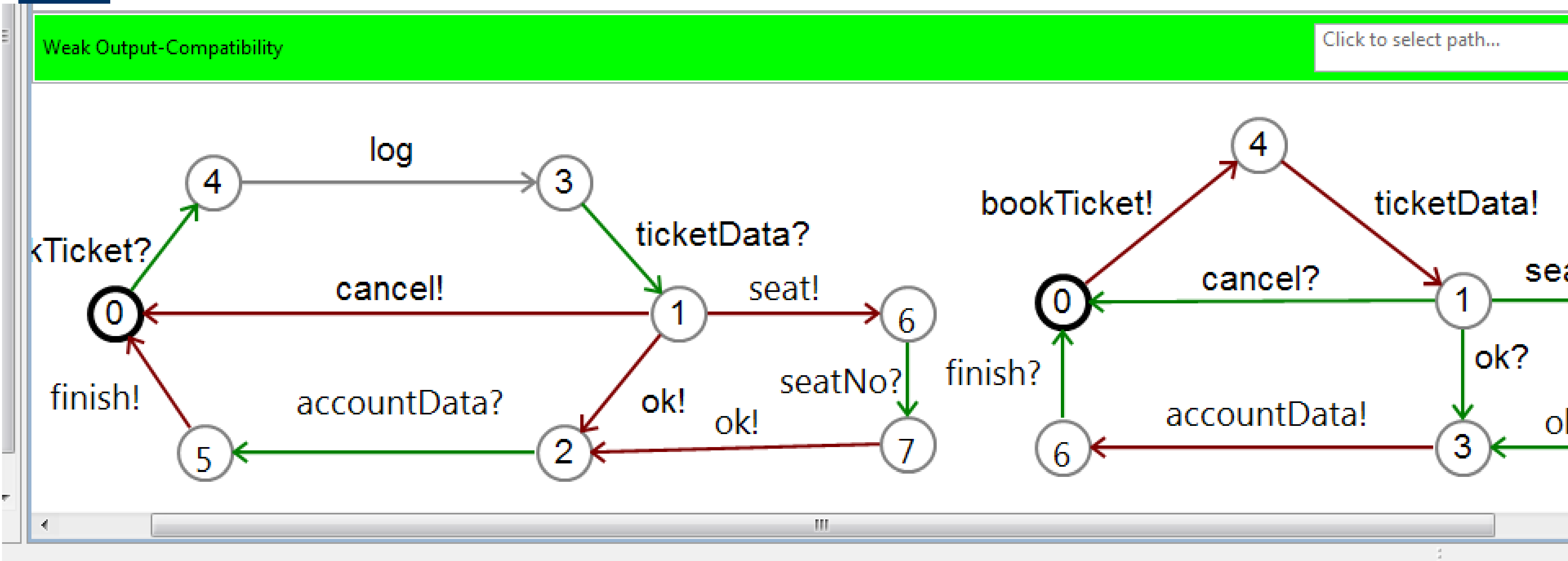
- if S **may** send an output shared with T, then T **must** be able to receive it, and conversely.

Larsen, Thomsen 1988



MIO Workbench: Weak Output Compatibility

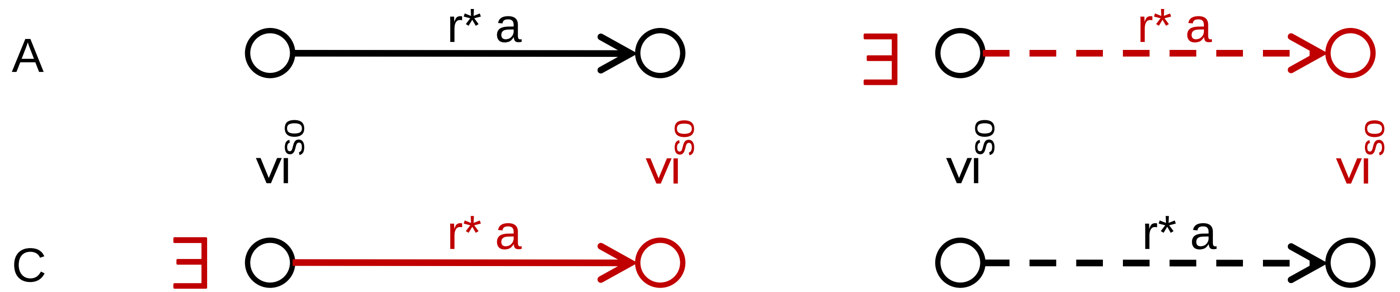
- Example: Weak Output Compatibility of Client and Weak Server Implementation



Strict-Observational-Refinement for MIOs

$C \leq_{so} A$ if

- every **must**-transition a (possibly prefixed with r's) in A is simulated by a (possibly prefixed with r's) in C
- every **may**-transition a (possibly prefixed with r's) in C is simulated by a (possibly prefixed with r's) in A



– An r is either tau (internal) or any action not defined in A (the protocol)

Summary: Formal Analysis of UML4SOA with MIOs

- MIOs form interface theories and thus are appropriate for compositional model development
- The Mio Workbench supports the formal analysis of Mios for several refinement and compatibility notions
- MIOs are an appropriate framework for formalizing and analyzing the dynamic behaviour of UML4SOA models.
  - UML4SOA analysis can be done by using an automated translation from UML4SOA to MIOs, then checking with refinement and compatibility
  - The result is back-annotated to the UML
- **Thus, we enable (early) checking of UML models with formal methods.**