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

Martin Wirsing
LMU 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 refinement)
- Input/output for compatibility (horizontal relationship)
- Synchronous composition (shared actions are internalized)
- Output Compatibility (any outputs must be received)
Modal I/O-Transition Systems (MIOs)

Formally: \( S = (\text{states}, \text{start}, \text{act}, \longrightarrow, \longrightarrow) \)

where
- \( \text{act} = \text{in} \cup \text{out} \cup \text{int(ernal)} \)
- \( \longrightarrow \subseteq \longrightarrow \) "every must is a may"

Larsen, Thomsen 1988
Larsen et al. 2007
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

\[
\begin{array}{c|c}
\otimes & \rightarrow \text{?a} & \rightarrow \text{?a} \\
\rightarrow \text{!a} & \rightarrow \text{a} & \rightarrow \text{a} \\
\rightarrow \text{!a} & \rightarrow \text{a} & \rightarrow \text{a} \\
\end{array}
\]
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 $S_1 \otimes S_2 = \text{states}_{S_1} \times \text{states}_{S_2}$, the initial state is given by $\text{start}_{S_1 \otimes S_2} = (\text{start}_{S_1}, \text{start}_{S_2})$, $\text{ins}_{S_1 \otimes S_2} = (\text{ins}_{S_1} \setminus \text{out}_{S_2}) \cup (\text{in}_{S_2} \setminus \text{out}_{S_1})$, $\text{out}_{S_1 \otimes S_2} = (\text{out}_{S_1} \setminus \text{in}_{S_2}) \cup (\text{out}_{S_2} \setminus \text{in}_{S_1})$, $\text{int}_{S_1 \otimes S_2} = \text{int}_{S_1} \cup \text{int}_{S_2} \cup (\text{in}_{S_1} \cap \text{out}_{S_2}) \cup (\text{in}_{S_2} \cap \text{out}_{S_1})$. The transition relations $\rightarrow_{S_1 \otimes S_2}$ and $\rightarrow_{S_1 \otimes S_2}$ are given by, for each $\sim \in \{\rightarrow, \rightarrow_{S_1 \otimes S_2}\}$,

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

- for all $a \in \text{act}_{S_1}$, if $s_1 \overset{a}{\sim}_{S_1} s_1'$ and $a \notin \text{act}_{S_2}$ then $(s_1, s_2) \overset{a}{\sim}_{S_1 \otimes S_2} (s_1', s_2')$,

- for all $a \in \text{act}_{S_2}$, if $s_2 \overset{a}{\sim}_{S_2} s_2'$ and $a \notin \text{act}_{S_1}$ then $(s_1, s_2) \overset{a}{\sim}_{S_1 \otimes S_2} (s_1, s_2')$. 

Composition Example

- Server $\otimes$ Client =

```
0 ----> 1
  |     |    |
  v     v    v
1 ----> 2 ----> 3
  |     |     |
  v     v     v
2 ----> 4 ----> 5
  |     |    |
  v     v    v
4 ----> 6
    |    |
    v    v
6 ----> 0
```

- Book Ticket
- Ticket Data
- Cancel
- Seat
- Account Data
- Finish
- OK
- Seat No
The MIO Workbench

- 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!
MIO Workbench: Graphical Editor

- Graphical editor for creating and modifying MIOs
Command-Line Shell

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

- **Server**

  ![Diagram of the server process]

  - bookTicket?
  - ticketData?
  - cancel!
  - finish!
  - ok!
  - seat!
  - seatNo?

- **Possible Refinement**

  ![Diagram of the possible refinement process]

  - bookTicket?
  - ticketData?
  - cancel!
  - finish!
  - ok!
  - accountData?
Wrong Refinement

- Server

- Wrong Refinement
Refinement

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.

Larsen, Thomsen 1988

\[ A \xrightarrow{a} B \quad \exists \quad C \xrightarrow{a} D \]
Definition 3 (Strong Modal Refinement [LT88b]) Let $S$ and $T$ be MTSs (MIOs, resp.) with the same signature. A relation $R \subseteq \text{states}_S \times \text{states}_T$ is called strong modal refinement for $S$ and $T$ iff for all $(s, t) \in R$ and for all $a \in \text{act}_S$ it holds that

1. if $t \stackrel{a}{\rightarrow}_T t'$ then there exists $s' \in \text{states}_S$ such that $s \stackrel{a}{\rightarrow}_S s'$ and $(s', t') \in R$,

2. if $s \stackrel{a}{\rightarrow}_S s'$ then there exists $t' \in \text{states}_T$ such that $t \stackrel{a}{\rightarrow}_T t'$ 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 $(\text{start}_S, \text{start}_T)$. 
Refinement Examples

- **Server**

  ![Diagram of a server process flow](image1.png)

- **Strong Modal Refinements**

  ![Diagram of strong modal refinements](image2.png)
MIO Workbench: Verification View

- The verification view provides a way to visually execute individual operations and depict the results graphically.
MIO Workbench: Verification 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
  - $\text{SP} \geq \text{SP}_1$

- Vertical composition (Transitivity)
  - from abstract to more concrete specifications
  - $\text{SP} \geq \text{SP}_1 \geq \ldots \geq \text{SP}_n$

- Horizontal composition (Monotonicity)
  - $\text{SP} \geq \text{SP}_1$ and $\text{P} \geq \text{P}_1$
  - $\Rightarrow \text{P[SP]} \geq \text{P}_1[\text{SP}_1]$

[Ehrig, Kreowski 83, Ehrich 82, Sannella, W 83, Maibaum 85, …]
Excursion: Interface Theories

- An **interface theory** is a tuple \((A, \otimes, \leq, \sim)\) consisting of
  - a class \(A\) of specifications
  - a partial composition operator \(\otimes : A \times A \rightarrow A\)
  - a binary refinement preorder \(\leq\)
  - a symmetric compatibility relation \(\sim\)

  satisfying
  
  1. **compositional refinement:**
     - If \(C \leq A, C' \leq A'\), and \(A \otimes A'\) is defined,
     - then \(C \otimes C'\) is defined and \(C \otimes C' \leq A \otimes A'\).

  2. **preservation of compatibility:**
     - If \(A \sim A'\) and \(C \leq A\) and \(C' \leq A'\), then \(C \sim C'\).

de Alfaro, Henzinger 2001
Fiadeiro ~ 2000
Maibaum ~1995

www.grovelandscapearchitecture.com
Interface Theory for MIOs

\[(\text{MIO}, \otimes, \leq_m, \sim_{sc})\] is an interface theory.

- \(\otimes\) 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

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

$S \sim_{sc} T$ 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.
MIO Workbench: Strong Output Compatibility

- Example: Strong Output Compatibility of Client and Server
MIO Semantics for UML4SOA

- Denotational Semantics (compositional)
- Defines a function $mio[...]$ 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 $\Rightarrow$ back link
  - Decision $\Rightarrow$ two outgoing transitions from previous state
  - Parallel $\Rightarrow$ 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)
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 handlers: 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
- 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

```
<<receive>>
bookVacation
```

```
<<data>>
HotelInfo hotelInfo;
```

```
<<serviceActivity>>
S1
```

```
<<send&receive>>
bookHotel
(IHotelService::)
```

```
hotelService
bookingInfo.hotel
hotelInfo
```

```
<<compensation>>
C1
```

```
<<send>>
undoBooking
(IHotelService::)
```

```
hotelService
bookingInfo.hotel
```

```
[hotelInfo.booked]
```

```
<<send&receive>>
bookFlight
```

```
flightService
bookingInfo-flight
```

```
<<send>>
hotelBookingFailed
```

```
customer
```
Example Vacation Booking

[Diagram of a UML flowchart for vacation booking, showing steps such as bookFlight, hotelBookingFailed, bookingOK, and flightBookingFailed.]
Vacation Example: MIO Translation

1. bookVacation?

2. bookHotel!

3. return_bookHotel?

4. compInstalled(C1);

5. bookFlight!

6. return_bookFlight?

7. undoBooking!

8. compHandled(C1)

9. bookingOK!

10. flightBookingFailed!
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
Weak Interface Theories

In many cases one would like to abstract from internal actions.

- Weak refinement allows the embedding of actions into sequences of internal ($\tau$-) actions in the refined system ($\tau^* a \tau^*$).
- Weak compatibility allows several internal ($\tau$-) actions before executing an input action.
Strong Refinement for MIOs

\[ C \leq_m A \text{ if} \]
- every \textbf{must}-transition \( a \) in \( A \) is simulated by \( a \) in \( C \)
- every \textbf{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 a tau-embedded action $$a$$ in $$C$$
- every **may**-transition $$a$$ in $$C$$ is simulated by a tau-embedded action $$a$$ in $$A$$

---

- special treatment for $$\tau$$-actions (if $$a=\tau$$, then the other automaton must perform $$\tau^*$$; in particular, it may not move at all ($$\varepsilon$$))
Weak Refinement Example

- **Server**

  - bookTicket?
  - ticketData?
  - seat!
  - cancel!
  - finish!
  - accountData?
  - ok!
  - seatNo?

- **Weak Refinement**

  - bookTicket?
  - ticketData?
  - cancel!
  - finish!
  - accountData?
  - ok!
A 2nd Interface Theory for MIOs

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

\[S \sim_{wc} T\] 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 after some \(\tau\)-actions, and conversely.

\[
\begin{align*}
S & \\
!a & \Rightarrow & \exists \tau^*?a \\
\exists \tau^*?b & \Leftarrow & \exists !b
\end{align*}
\]
MIO Workbench: Weak Output Compatibility

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