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:

\[ S = (\text{states}, \text{start}, \text{act}, \rightarrow , \rightarrow ) \]

where

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

Larsen, Thomsen 1988
Larsen et al. 2007
Example: Flight Booking Service

- Server

![Diagram of flight booking process]

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

  \[
  \text{Definition 4 (Composability [LNW07a]) Two MIOs } S \text{ and } T \text{ are called composable if } (\text{in}_S \cup \text{int}_S) \cap (\text{in}_T \cup \text{int}_T) = \emptyset \text{ and } (\text{out}_S \cup \text{int}_S) \cap (\text{out}_T \cup \text{int}_T) = \emptyset.
  \]

- 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
Definition 5 (Composition \cite{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 $\rightarrow \in \{\rightarrow_{S_1 \otimes S_2}, \rightarrow_{S_1 \otimes S_2}\}$,

- for all $i, j \in \{1, 2\}, i \neq j$, for all $a \in (\text{act}_{S_1} \cap \text{act}_{S_2})$, if $s_i \overset{a}{\rightarrow}_{S_i} s'_i$ and $s_j \overset{a}{\rightarrow}_{S_j} s'_j$ then $(s_1, s_2) \overset{a}{\rightarrow}_{S_1 \otimes S_2} (s'_1, s'_2),$

- for all $a \in \text{act}_{S_1}$, if $s_1 \overset{a}{\rightarrow}_{S_1} s'_1$ and $a \notin \text{act}_{S_2}$ then $(s_1, s_2) \overset{a}{\rightarrow}_{S_1 \otimes S_2} (s'_1, s_2),$

- for all $a \in \text{act}_{S_2}$, if $s_2 \overset{a}{\rightarrow}_{S_2} s'_2$ and $a \notin \text{act}_{S_1}$ then $(s_1, s_2) \overset{a}{\rightarrow}_{S_1 \otimes S_2} (s_1, s'_2).$
Composition Example

- Server $\otimes$ Client =

![Diagram showing the interaction between Server and Client with nodes and edges representing operations like bookTicket, ticketData, cancel, seat, finish, accountData, ok, seatNo.](image-url)
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 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 verification tasks.
Command-Line Shell

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

- **Server**

![Diagram of the server process]

- **Possible Refinement**

![Diagram of the possible refinement process]
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 $R \subseteq \text{states}_S \times \text{states}_T$ 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 $t \xrightarrow{a} t'$ then there exists $s' \in \text{states}_S$ such that $s \xrightarrow{a} s'$ and $(s', t') \in R$,
2. if $s \xrightarrow{a} s'$ then there exists $t' \in \text{states}_T$ such that $t \xrightarrow{a} t'$ 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

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.

\[ A \xrightarrow{a} C \]

\[ \exists \xrightarrow{a} \]
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 $t \xrightarrow{a}^T t'$ then there exists $s' \in \text{states}_S$ such that $s \xrightarrow{a}^S s'$ and $(s', t') \in R$,  

2. if $s \xrightarrow{a}^S s'$ then there exists $t' \in \text{states}_T$ such that $t \xrightarrow{a}^T t'$ and $(s', t') \in R$.

We say that $S$ strongly modally refines $T$, written $S \preceq_m T$, iff there exists a strong modal refinement for $S$ and $T$ containing $(\text{start}_S, \text{start}_T)$. 
Refinement Examples

- **Server**

- **Strong Modal Refinements**
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 $P \geq P_1$
    - $P[SP] \geq P_1[SP_1]$

[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 \(C \sim C'\).
Interface Theory for MIOs

(MIO, ⊗, ≤_m, ~_sc) is an interface theory.
- ⊗ is the synchronous composition operator on MIOs
- ≤_m is strong modal refinement

C ≤_m A \iff
- every must-transition in A is simulated by C
- every may-transition in C is simulated by A

Larsen, Thomsen 1988

Bauer et al. (TACAS) 2010
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)

\[ S \sim_{sc} T \quad \text{if for every reachable state in } S \otimes T, \]
- if \(S\) may send an output shared with \(T\),
  then \(T\) must be able to receive it, and conversely.

Larsen, Thomsen 1988

Bauer et al. (TACAS) 2010
MIO Workbench: Strong Output Compatibility

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

- Denotational Semantics (compositional)
- Defines a function $\text{mio}[...]$ 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 => 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
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!

![Diagram of a protocol state machine with transitions](image)
Translation Example: Vacation Booking

- UML4SOA
Example Vacation Booking

- «send&receive» bookFlight (IFlightService::)
- «send» hotelBookingFailed (ICustomerCallback::)
- «send» bookingOK (ICustomerCallback::)
- «send» flightBookingFailed (ICustomerCallback::)
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

- SO (Strict-Observational)
  - Loose
  - Strong
  - Weak

- Hard
  - Non-Race Deadlocks
  - Race Conditions
  - Strict Implementation
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

\[ C \leq_w A \text{ if} \]
- 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 \( a \) \( \in \) \( A \)

- special treatment for \( \tau \)-actions (if \( a = \tau \), then the other automaton may also not move at all (\( \varepsilon \)))
Weak Refinement Example

- **Server**

  ![Server Diagram]

- **Weak Refinement**

  ![Weak Refinement Diagram]
A 2nd Interface Theory for MIOs

- \((\text{MIO}, \otimes, \leq_m, \sim_{sc})\) 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)

\[ S \sim_{wc} T \text{ if for every reachable state in } S \otimes T, \]
- if \(S\) may send an output shared with \(T\),
  then \(T\) must be able to receive it, and conversely.

\[ S \quad !a \quad \Rightarrow \quad \exists \quad ?a \]
\[ \exists \quad ?b \quad \Leftarrow \quad \exists \quad !b \]
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.