On Weak Modal Compatibility, Refinement, and the MIO Workbench

Sebastian S. Bauer, Philip Mayer, Andreas Schroeder and Rolf Hennicker

Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
`{bauerse,mayer,schroeda,hennicker}@pst.ifi.lmu.de`

Abstract. Building on the theory of modal I/O automata (MIOs) by Larsen et al. we introduce a new compatibility notion called weak modal compatibility. As an important property of behavioral interface theories we prove that weak modal compatibility is preserved under weak modal refinement. Furthermore, we organize and compare different notions of refinement and compatibility to give an easily-accessible overview. Finally, we describe the MIO Workbench, an Eclipse-based editor and verification tool for modal I/O automata, which implements various refinement, compatibility and composition notions and is able to depict the results of verification directly on the graphical representation of MIOs – relations or state pairs in the positive and erroneous paths in the negative case.

1 Introduction

Interface design has been a long-standing and important issue in the design of software systems. Various methods of interface specifications, both static and behavioral, have been suggested for software components [3,4]. We believe that behavioral specifications for components are of particular importance, and focus on such specifications in this paper.

Among the most widely accepted methods for specifying behavioral properties of interfaces are I/O automata [15,16], which have been introduced to specify the temporal ordering of events involving a component, explicitly taking communication aspects such as sending or receiving messages into consideration. Many variations of these automata have been introduced over the years; for example interface automata [5], timed interface automata [6], or resource automata [2]. At the same time, another aspect of interface behavior has been studied: Modal automata [13] explicitly address the difference between required and optional actions by using must and may transitions, which allow protocols and implementations to differ with regard to non-compulsory actions. Recently, both the input/output and the may/must aspects of behavioral specifications have been integrated [11], giving rise to modal I/O automata (MIOs).

Building on the basic formalisms for behavioral specifications such as MIOs, we can use notions of interface compatibility and correct interface implementation (refinement) to verify component behavior; for a survey on compatibility notions see [3]. Interface theories [4] are commonly used to precisely define these requirements. Interface theories are tuples $(\mathcal{A}, \leq, \sim, \otimes)$ of a semantic domain $\mathcal{A}$, a refinement relation $\leq \subseteq \mathcal{A} \times \mathcal{A}$, a (symmetric) compatibility relation $\sim \subseteq \mathcal{A} \times \mathcal{A}$, and a (possibly partial) composition function $\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ satisfying the following three properties: Let $S, T, T', T'' \in \mathcal{A}$.

(1) Preservation of compatibility: If $S \sim T$ and $T' \leq T$ then $S \sim T'$.
(2) Compositional refinement: If $T' \leq T$ then $S \otimes T' \leq S \otimes T$.
(3) Transitivity of refinement: If $T'' \leq T'$ and $T' \leq T$ then $T'' \leq T$.

These three properties imply *independent implementability*, which is the basis for top-down component-based design.

Independent implementability states that in order to refine a given composed interface $S \otimes T$ towards an implementation, it suffices to independently refine S and T, say, to S' and T', respectively; then the refinements S' and T' are compatible and their composition refines the interface $S \otimes T$. More formally, let $(\mathcal{A}, \leq, \sim, \otimes)$ be an interface theory. Independent implementability means that if $S \sim T$, $S' \leq S$ and $T' \leq T$ hold, both $S' \sim T'$ and $S' \otimes T' \leq S \otimes T$ follow.

In this paper, we elaborate on the challenges of component behavior specification. We introduce new interface theories in the same line as in de Alfaro and Henzinger [4] for modal I/O automata with appropriate refinement and compatibility notions. We introduce the notion of *weak modal compatibility* which allows for loose coupling between interfaces and prove its preservation under weak modal refinement.

Another result of this paper is the organization and comparison of different existing and new notions of refinement and compatibility. We give an easily-accessible overview of these notions and their relationships.

Although I/O automata and modal automata have a long history, there is little tool support. Therefore, another major contribution of this paper is a verification tool and editor for MIOs – the MIO Workbench – which includes implementations of various refinement and compatibility notions, including but not limited to the ones which are part of our interface theories. We believe that the ability to automatically verify these properties is useful for both discussing the theory of MIOs as well as a foundation for practical applications.

This paper is structured as follows. We discuss modal I/O transition systems in Sect. 2. In Sect. 3, we introduce our notion of weak modal compatibility, followed by an overview of the different notions of refinement and compatibility in Sect. 4. Tool support for MIOs is discussed in Sect. 5. We conclude in Sect. 6.

2 Modal (I/O) Transition Systems

This first section is devoted to a short introduction to modal transition systems, and in particular modal *input/output* transition systems. A modal transition system is characterized by the fact that it has *two* transition relations, indicating allowed (*may*) and required (*must*) behavior. In this paper, we consider an extended version of the original modal transition systems [13] by including a signature which distinguishes between *internal* and *external* actions.

Definition 1 (Modal Transition System). *A modal transition system (MTS)* $S = (states_S, start_S, (ext_S, int_S), \dashrightarrow_S, \longrightarrow_S)$ *consists of a set of states* $states_S$*, an initial state* $start_S \in states_S$*, disjoint sets* ext_S *and* int_S *of external and internal actions where* $act_S = ext_S \cup int_S$ *denotes the set of (all) actions, a may-transition relation* $\dashrightarrow_S \subseteq states_S \times act_S \times states_S$*, and a must-transition relation* $\longrightarrow_S \subseteq states_S \times act_S \times states_S$*. The pair* (ext_S, int_S) *is called the signature of* S*.*

An MTS S is called *syntactically consistent* if every required transition is also allowed, i.e. it holds that $\longrightarrow_S \subseteq \dashrightarrow_S$. From now on we only consider syntactically consistent MTSs. Moreover, we call an MTS S an *implementation* if the two transition relations coincide, i.e. $\longrightarrow_S = \dashrightarrow_S$.

Modal I/O transition systems [11] further differentiate between two kinds of external actions, namely *input* and *output* actions.

Definition 2 (Modal I/O Transition System). *A modal I/O transition system (MIO)* S *is an MTS with the set of external actions* ext_S *partitioned into two disjoint sets* in_S*,* out_S *of input and output actions, respectively. The triple* (in_S, out_S, int_S) *is called the signature of* S*.*

The notions of syntactic consistency and implementation also apply for MIOs.

Example 1. Our running example in this paper is the specification (and implementation) of a flight booking service. In Fig. 1, the MIO T_0 specifying the service provider is depicted. For improving readability, output actions are suffixed with an exclamation mark (*!*) whereas input actions are suffixed with a question mark (*?*). Internal actions do not have any suffix.

In the initial state (indicated by a filled circle) the session is initiated by receiving *bookTicket?*, followed by the reception of the data of requested tickets in *ticketData?*. Then, a service implementation may ask the client for choosing a seat number (*seat!*) which it must be able to receive afterwards (*seatNo?*). The reservation of the tickets may be cancelled by the service provider (*fail!*) if the requested flight is fully booked, or the request is confirmed by sending *ok!*, which is followed by receiving the account data (*accountData?*) of the client. ■

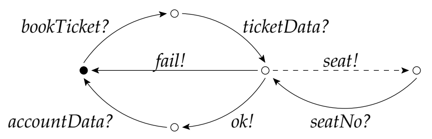


Fig. 1. Specification T_0 of a flight booking service.

In the following, we recall the standard definition of refinement for modal transition systems, cf. [13]. The notion of refinement aims at capturing the relation between an abstract specification of an interface and a more detailed one, possibly an implementation of that interface. Thus, it allows for a stepwise refinement of an abstract specification towards an implementation.

The basic idea of *modal* refinement is that any required (*must*) transition in the abstract specification must also occur in the concrete specification. Conversely, any allowed (*may*) transition in the concrete specification must be allowed by the abstract specification. Moreover, in both cases the target states must conform to each other. Modal refinement has the following consequences: A concrete specification may leave out allowed transitions, but is required to keep all must transitions, and moreover, it is not allowed to perform more transitions than the abstract specification admits. The following definition of modal refinement is called *strong* since every transition that is taken into account must be simulated "immediately", i.e. without performing internal actions before.

Definition 3 (Strong Modal Refinement [13]). *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 $(start_S, start_T)$.

If both S and T are implementations, i.e. the must-transition relation coincides with the may-transition relation, then strong modal refinement coincides with (strong) bisimulation; if $\longrightarrow_T = \emptyset$ then it corresponds to simulation [17].

Example 2. In Fig. 2, a (possible) implementation T_1 of the flight booking service specified by the MIO T_0 in Fig. 1 is shown. In this particular implementation of the specification T_0, the optional output for asking the client for a particular seat number is never taken. However, all must-transitions of T_0 are retained in the implementation T_1, hence we have $T_1 \leq_m T_0$. ■

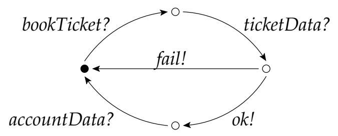


Fig. 2. Implementation T_1 of T_0.

Next, we introduce a binary (synchronous) composition operator on MIOs. When two protocols (implementations), each one describing a particular component, can communicate by synchronous message passing, we are interested in computing the resulting protocol (implementation) of the composed system. Although composition can obviously be defined for MTSs, we directly give a definition for MIOs as this is our main interest.

It is convenient to restrict the composition operator to *composable* MIOs by requiring that overlapping of actions only happens on complementary types.

Definition 4 (Composability [11]). *Two MIOs S and T are called composable if $(in_S \cup int_S) \cap (in_T \cup int_T) = \emptyset$ and $(out_S \cup int_S) \cap (out_T \cup int_T) = \emptyset$.*

We now define composition of MIOs in a straightforward way by a binary partial function $\otimes$ synchronizing on matching (shared) actions.

Definition 5 (Composition [11]). *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 of MIOs only synchronizes transitions with matching shared actions and same type of transition, i.e. a must-transition labeled with a shared action only occurs in the composition if there exist corresponding matching must-transitions in the original MIOs.

A well-known problem occurs when composing arbitrary MIOs S and T: If for a reachable state (s, t) in $S \otimes T$, S in state s is able to send out a message a shared with T, and T in state t is not able to receive a then this is considered as a compatibility problem since S may get stuck in this situation. We want to rule out this erroneous behavior by requiring that S and T must be *compatible*.

The following definition of strong compatibility is strongly influenced by [5] and [11]. Intuitively, two MIOs S and T are compatible if for every reachable state in the product $S \otimes T$, if S is able to provide an output which is shared with T, i.e. is in the input alphabet of T, then T must "immediately" be able to receive this message (and vice versa).

Definition 6 (Strong Modal Compatibility). *Let S and T be composable MIOs. S and T are called strongly modally compatible, denoted by $S \sim_{sc} T$, iff for all reachable states (s, t) in $S \otimes T$,*

1. *for all $a \in (out_S \cap in_T)$, if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \xrightarrow{a}_T t'$,*
2. *for all $a \in (out_T \cap in_S)$, if $t \overset{a}{\dashrightarrow}_T t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{a}_S s'$.*

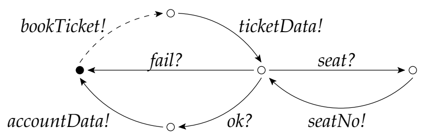


Fig. 3. Protocol S.

Example 3. In Fig. 3, a specification S of a client of the flight booking service is shown. It is easily provable that indeed S and T_0 and also S and T_1 are strongly modally compatible, i.e. $S \sim_{sc} T_0$ and $S \sim_{sc} T_1$. ■

For MIOs equipped with $\sim_{sc}$ and $\leq_m$, we obtain a valid interface theory.

Theorem 1. *Let S, T, T', T'' be MIOs, and let S and T be composable.*

1. (Preservation) *If $S \sim_{sc} T$ and $T' \leq_m T$ then $S \sim_{sc} T'$.*
2. (Compositionality) *If $T' \leq_m T$ then $S \otimes T' \leq_m S \otimes T$.*
3. (Transitivity) *If $T'' \leq_m T'$ and $T' \leq_m T$ then $T'' \leq_m T$.*

The proof of statement 1 is given in [1]; statement 2 is a consequence of a result in [13] (where it has been proved that every static construct – for which $\otimes$ is a special case – is compositional for $\leq_m$) and statement 3 follows directly from the definition of $\leq_m$.

Remark 1. The compatibility notions used in this paper follow a *pessimistic* approach: two MIOs S and T are only compatible if no communication error between S and T can occur in *any* environment of $S \otimes T$. A different approach to compatibility is the *optimistic* one, cf. [11,5]: two MIOs S and T are compatible if they are compatible for any "helpful" environment in the sense that the environment never provides outputs that would cause the product $S \otimes T$ to run in a state (s, t) with incompatible states s and t.

3 Weak Modal Compatibility

The refinement presented in the last section is strong in the sense that every must-transition in the protocol must be *immediately* simulated in the implementation and conversely, every may-transition in the implementation must be *immediately* simulated in the protocol. This definition can be weakened by including the notion of *weak transitions*.

For denoting weak transitions, given a MIO S and an action $a \in ext_S$, we write $s \xrightarrow{a}^*_S s'$ iff there exist states $s_1, s_2 \in states_S$ such that

$$s(\xrightarrow{\tau}_S)^* s_1 \xrightarrow{a}_S s_2 (\xrightarrow{\tau}_S)^* s'$$

where $t(\xrightarrow{\tau}_T)^* t'$ stands for finitely many transitions with internal actions leading from t to t'; including no transition and in this case $t = t'$. The label τ always denotes an arbitrary internal action. Moreover, we write

$$s \xrightarrow{\hat{a}}^*_S s' \text{ iff either } s \xrightarrow{a}^*_S s' \text{ and } a \in ext_S, \text{ or } s(\xrightarrow{\tau}_S)^* s' \text{ and } a \notin ext_S.$$

Both notations are analogously used for may-transitions.

Similar to the generalization of bisimulation to weak bisimulation [17], one can introduce a notion of modal refinement in a weak form: Every (non-weak) must-transition in the protocol must be simulated in the implementation by a weak must-transition, and conversely, every (non-weak) may-transition in the implementation must be simulated by a weak may-transition in the protocol. This form of weak modal refinement was originally introduced in [10].

Definition 7 (Weak Modal Refinement [10]). *Let S and T be MTSs (MIOs, resp.) with the same signature.*[1] *A relation $R \subseteq states_S \times states_T$ is called a weak modal refinement for S and T iff for all $(s,t) \in R$, 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 $s \xrightarrow{\hat{a}}^*_S s'$ and $(s',t') \in R$,*
2. *if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \overset{\hat{a}}{\dashrightarrow}^*_T t'$ and $(s',t') \in R$.*

We say that S weakly modally refines T, denoted by $S \leq^*_m T$, iff there exists a weak modal refinement for S and T containing $(start_S, start_T)$.

Example 4. In Fig. 4, another implementation T_2 of T_0 is presented which, after receiving *bookTicket?*, performs an internal action *log* with the meaning of executing an internal logging operation. T_0 does not specify any internal actions, so we have $T_2 \not\leq_m T_0$, but weak modal refinement allows to postpone the further execution of *ticketData?* (according to protocol T_0) until some internal (must)-transitions are passed through. It follows that $T_2 \leq^*_m T_0$. ∎

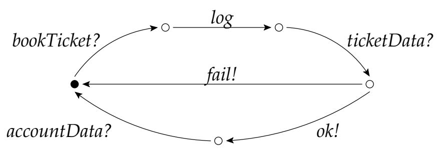


Fig. 4. Implementation T_2 of T_0.

Let us recall our goal. We want to find appropriate notions of refinement and compatibility for component behavior specifications. In order to obtain a valid interface theory involving weak modal refinement we have to make sure that, given a suitable compatibility notion, compatibility is preserved under refinement. The following example shows that strong compatibility is *not* preserved by weak modal refinement.

Example 5. S and T_0 are strongly compatible and $T_2 \leq^*_m T_0$. But S and T_2 are not strongly compatible, since S is able to send out the message *ticketData* to T_2, but T_2, being in the state before performing *log*, is not able to receive the message immediately. ∎

[1] More generally, in the weak case, one could also allow that S and T have arbitrary (non related) internal actions.

Obviously, internal actions are not adequately considered in the definition of compatibility. Dealing with this problem requires a new definition of compatibility, which we call *weak modal compatibility.* The intuition behind weak modal compatibility follows from the previous example: If a MIO *may* send out a certain message to its partner, we consider this transaction as compatible if the other MIO *must* be able to receive it, possibly performing some internal must steps in between. Note that the *must* modality is essential here: If internal *may* transitions would be allowed then this path could be omitted in further refinements and therefore compatibility of implementations would not be ensured anymore.

In the following, given a MIO S and an action $a \in ext_S$, we write $s \xrightarrow{a}{}^{\triangleleft}_{S} s'$ iff there exists a state $s'' \in states_S$ such that

$$s(\xrightarrow{\tau}_S)^* s'' \xrightarrow{a}_S s'.$$

Moreover, $s \xrightarrow{\hat{a}}{}^{\triangleleft}_{S} s'$ denotes $s \xrightarrow{a}{}^{\triangleleft}_{S} s'$ if $a \in ext_S$, otherwise $s(\xrightarrow{\tau}_S)^* s'$. Both notations are analogously used for may-transitions.

Definition 8 (Weak Modal Compatibility). *Let S and T be composable MIOs. S and T are called weakly modally compatible, denoted by $S \sim_{wc} T$, iff for all reachable states (s,t) in $S \otimes T$,*

1. *for all $a \in (out_S \cap in_T)$, if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \xrightarrow{a}{}^{\triangleleft}_{T} t'$,*
2. *for all $a \in (out_T \cap in_S)$, if $t \overset{a}{\dashrightarrow}_T t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{a}{}^{\triangleleft}_{S} s'$.*

Obviously, it holds that $S \sim_{sc} T$ implies $S \sim_{wc} T$.

Example 6. Looking back to our examples, it can be easily verified that S is weakly modally compatible with both T_0 and T_2 since the reception of *ticketData* in T_2 must take place after the internal must transition labeled with *log*. ■

Based on the MIO formalism, weak modal compatibility $\sim_{wc}$ and weak modal refinement $\leq^*_m$ satisfy the desired properties of an interface theory.

Theorem 2. *Let S, T, T', T'' be MIOs, and let S and T be composable.*

1. (Preservation) *If $S \sim_{wc} T$ and $T' \leq^*_m T$ then $S \sim_{wc} T'$.*
2. (Compositionality) *If $T' \leq^*_m T$ then $S \otimes T' \leq^*_m S \otimes T$.*
3. (Transitivity) *If $T'' \leq^*_m T'$ and $T' \leq^*_m T$ then $T'' \leq^*_m T$.*

The proof of statement 1 is given in [1]; statement 2 is a consequence of a result in [10] (where it has been proved that $\otimes$ is a binary, τ-insensitive operator on MTSs with input/output labels and therefore $\otimes$ is compositional for $\leq^*_m$) and statement 3 follows directly from the definition of $\leq^*_m$.

4 Overview of Refinement and Compatibility Notions

In order to complete the picture of existing notions of modal refinements for modal transition systems and their relationships to the notions of compatibility defined here, we also consider may-weak modal refinement, which has been defined in [12] (and, under the name of observational modal refinement, in [11]) to generalize alternating simulation [5]. May-weak modal refinement keeps the strong requirement for required (must-)transitions (as in strong modal refinement, but restricted to external actions), but has a weak condition for allowed (may-)transitions: every allowed transition in the more concrete MTS must be simulated by an allowed transition in the abstract MTS, possibly preceded by finitely many internal transitions.

Definition 9 (May-Weak Modal Refinement [12]). *Let S and T be MTSs (MIOs, resp.) with the same signature. A relation $R \subseteq states_S \times states_T$ is called may-weak modal refinement for S and T iff for all $(s,t) \in R$ it holds that*

1. *for all $a \in ext_T$, 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. *for all $a \in act_S$, if $s \overset{a}{\dashrightarrow}_S s'$ then there exists $t' \in states_T$ such that $t \overset{\hat{a}}{\dashrightarrow}{}^{\triangleleft}_T t'$ and $(s',t') \in R$.*

We say that S may-weakly modally refines T, denoted by $S \leq_m^{\triangleleft} T$, iff there exists a may-weak modal refinement for S and T containing $(start_S, start_T)$.

Given MIOs as the underlying formalism, may-weak modal refinement together with strong modal compatibility forms a valid interface theory.

Theorem 3. *Let S, T, T', T'' be MIOs, and let S and T be composable.*

1. (Preservation) *If $S \sim_{sc} T$ and $T' \leq_m^{\triangleleft} T$ then $S \sim_{sc} T'$.*
2. (Compositionality) *If $T' \leq_m^{\triangleleft} T$ then $S \otimes T' \leq_m^{\triangleleft} S \otimes T$.*
3. (Transitivity) *If $T'' \leq_m^{\triangleleft} T'$ and $T' \leq_m^{\triangleleft} T$ then $T'' \leq_m^{\triangleleft} T$.*

The proof of Thm. 3 is given in [1].

So far, we have considered three modal refinement notions. Obviously, for any two modal transition systems (or MIOs) S and T we have

Fact 1. if $S \leq_m T$ then $S \leq_m^* T$;
Fact 2. if $S \leq_m T$ then $S \leq_m^{\triangleleft} T$.

The converses of the above implications do obviously not hold; moreover, it is also obvious that weak modal refinement does not imply may-weak modal refinement. However, also may-weak modal refinement does not imply weak modal refinement since condition 1 in Def. 9 only considers external actions; for instance, for T and T' in Fig. 5, $T' \leq_m^{\triangleleft} T$ but $T' \not\leq_m^* T$ since the internal must transition of T is not respected by T' which would be required for weak modal refinement.

Table 1. Overview of preservation of compatibility under refinement.

	Strong Compatibility $\sim_{sc}$	Weak Compatibility $\sim_{wc}$
Strong Refinement $\leq_m$	✓(Thm. 1)	✓(Fact 1 & Thm. 2)
Weak Refinement $\leq_m^*$	✗(Ex. 5)	✓(Thm. 2)
May-Weak Refinement $\leq_m^{\triangleleft}$	✓(Thm. 3)	✗(Ex. 7)

We have shown that all modal refinements are compositional w.r.t. $\otimes$, but they substantially differ when preservation of strong/weak compatibility is considered. Table 1 summarizes the relationships between modal refinement and compatibility notions; a checkmark indicates that compatibility is preserved under refinement.

Example 7. Weak compatibility is not preserved under may-weak modal refinement, as shown in Fig. 5. ■

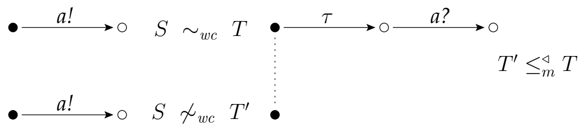


Fig. 5. Counterexample.

5 The MIO Workbench

In the previous sections, we have illustrated the sometimes subtle distinctions between different definitions of refinement and compatibility. During our work, we have come to appreciate the help of implementations of our formal notions, which were an unflinching partner in finding inconsistencies and confirming counterexamples. To aid ourselves and others, we have implemented a complete set of verification notions and surrounding functionality for working with modal I/O automata – the MIO Workbench, an Eclipse-based editor and verification tool for modal I/O automata, which we present here for the first time.

Workbench Features The most direct and intuitive way to work with MIOs is using a graphical editing facility based on a graph of nodes (states) and edges (transitions) as well as accompanying labels. The first feature provided by the workbench is thus a

(1) *Graphical Editor*, allowing to create new or change existing MIOs.

The implementation of the different notions of refinement and compatibility are the next features of the MIO Workbench:

(2) *Refinement Verification.* These include strong, may-weak, and weak modal refinement.
(3) *Compatibility Verification.* We support the notions of strong (with and without "helpful" environment, cf. [5]), and weak modal compatibility.

Furthermore, it is interesting to see an actual composition of composable MIOs:

(4) *Composition Operation* on MIOs.

The output of a composition operation is either the composed MIO or a list of problematic actions which caused the composition to fail.

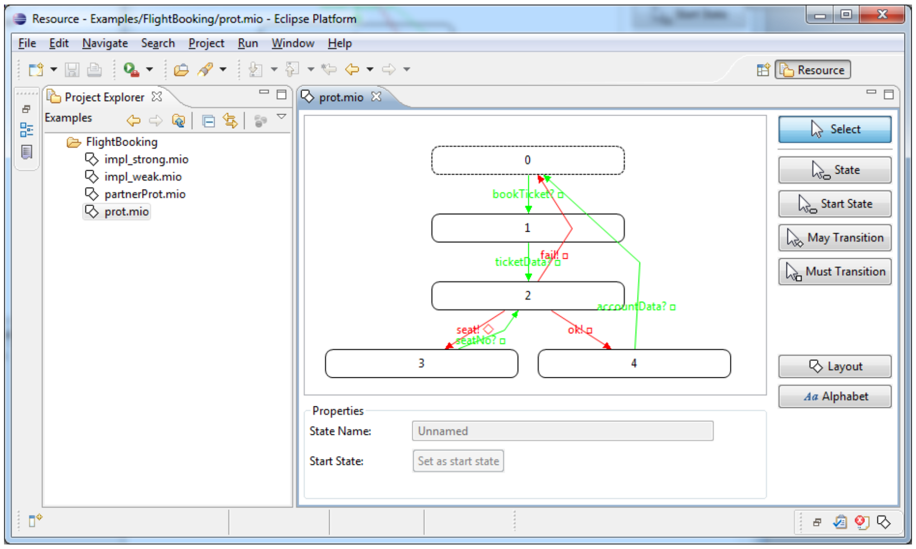
Considering refinement and compatibility verification, we can get two very important, but very different results. First, if refinement or compatibility is possible, we get refinement relation(s) and matching states for compatibility, respectively. However – and this is even more important – if the verification fails, we get the error states and the error transitions in the two automata, i.e. the exact position(s) which led to the erroneous outcome.

Visualizing these results in a graphical way is very important. Therefore, the workbench also includes:

(5) *Refinement relation and state match view.* If a refinement or compatibility verification was successful, the workbench graphically displays the relation or the matching states side-by-side between the two input MIOs.
(6) *Problem view including error states and unmatched actions.* If a refinement or compatibility verification was not successful, the workbench graphically displays, side-by-side, the path which led to an erroneous state, and the transition possible in one automaton, but not in the other.

On the technical side, the MIO Workbench is based on the Eclipse platform. We use an Eclipse Modeling Framework (EMF)-based metamodel for MIOs, which enables persistence and simple access to concrete automata. The workbench integrates into Eclipse by adding MIO-specific file handling and the new MIO editor as well as the verification view. The MIO Workbench is extensible with regard to new notions of refinement, compatibility, and composition, by means of standard Eclipse extension points.

User Interface Fig. 6 shows the MIO editor inside the Eclipse workbench. On the left-hand side, the project explorer shows MIOs stored on the file system as .mio files; on the right-hand side, the editor for one of these MIOs is displayed. A MIO is displayed in the classical way by using nodes as states and edges as transitions. Each transition has a type (must or may), which is indicated by a square or diamond, respectively. Furthermore, each transition also stands for an internal, input, or output action. An input action is colored green and is suffixed with a question mark (*?*). An output action is colored red and is suffixed with



State Name: Unnamed

Start State: Set as start state

Fig. 6. MIO Workbench editor.

an exclamation mark (*!*). Finally, an internal action is gray and does not have a suffix. The MIO editor offers all the usual operations such as adding new nodes, moving them around, changing labels, types, and re-layouting.

The verification view of the MIO Workbench is the central access point to the verification functionality. It features a side-by-side view of two modal I/O automata, which can then be analyzed for refinement or compatibility, or composed.

Fig. 7 shows verification of the protocol T_0 (left) and implementation T_2 (right) from Ex. 4 using weak modal refinement, such that $T_2 \leq_m^* T_0$, which is indicated by the green top and the green arrows between related states.

As said above, the most interesting results are negative cases, i.e. if a refinement does not exist or compatibility does not hold. In this case, the MIO Workbench displays the possible error paths, each indicating a state pair in violation and the corresponding erroneous action.

Fig. 8 shows the visualization of Ex. 4 again, but this time using strong modal refinement. Thus, we can take the *bookTicket?* action on both sides, marked in dark red, arriving at the state pair *(1,5)*. Here, the protocol T_0 can take the *ticketData?* action, also marked in dark red, which the implementation T_2 is unable to follow. Since it is a *must*, there is no relation in this case, which is indicated by the red top and the dark red actions.

The MIO Workbench contains additional helpful features such as automatically laying out MIOs, adjusting an alphabet of a MIO by hiding non-shared labels for a compatibility or refinement check, and more.

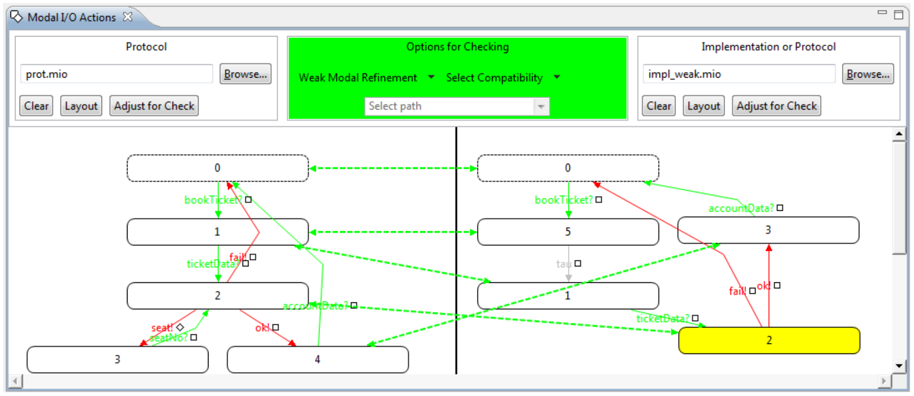


Fig. 7. MIO Workbench refinement view.

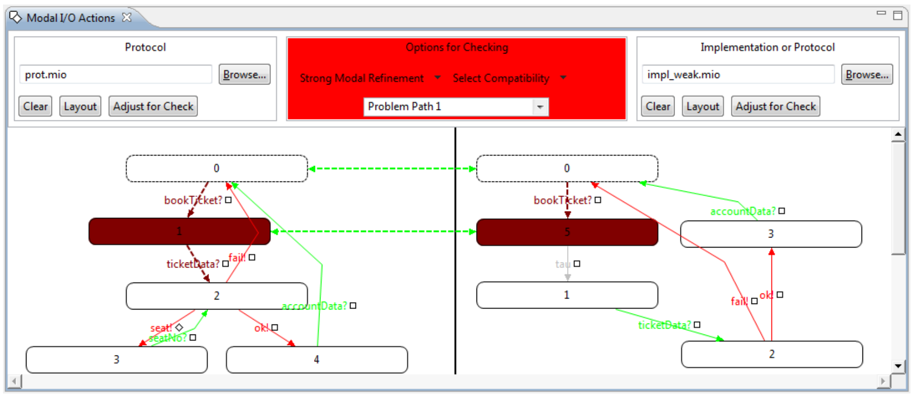


Fig. 8. MIO Workbench showing refinement problem.

The Workbench in Context To our knowledge, the MIO Workbench is the first tool for modal I/O automata which includes a full set of refinement, compatibility, and composition notions as well as a (graphical) MIO editor.

Perhaps the closest related tool is MTSA [7], which includes a (text-based) editor and implementations of refinement as well as composition of modal automata. As the MIO Workbench is based on modal *input/output* automata as first class citizens, it differs by including compatibility verification based on I/O information; furthermore, it includes a graphical editor and a side-by-side graphical result view. There are also tools for I/O automata (i.e. without modality), for example the command-line based IOA toolset [9] for plain I/O automata or the Eclipse-based Tempo Toolkit [14] which deals with timed I/O automata; however, none of those considers both modality and communication aspects available in MIOs.

6 Conclusion

In this paper, we have presented an overview of modal I/O automata and various notions of modal refinement and compatibility. We have motivated the need for a new compatibility notion called *weak modal compatibility*, which allows the passing of internal actions. We have shown this compatibility notion to hold under weak modal refinement, and we have given an overview of the relationships between modal refinements and compatibility notions introduced in this paper.

On the practical side, we have presented a verification tool and graphical editor for modal I/O automata called the MIO Workbench, which implements various refinement and compatibility notions based on MIOs. We believe that tool support is of great help for discussing modal I/O automata and may serve in research, teaching, and as a prototype for industrial applications. The MIO Workbench can be freely downloaded from www.miowb.net.

As future work, we plan to extend our notions of compatibility to new use cases identified from practical service specifications and from distributed systems with asynchronous communication. It also looks interesting to investigate compatibility in the context of a new semantics for MTSs introduced recently in [8].

Acknowledgements. We would like to thank Alexander Knapp who has pointed out the need for a more flexible compatibility relation between interface specifications. This research has been partially supported by the GLOWA-Danube project 01LW0602A2 sponsored by the German Federal Ministry of Education and Research, the EC project SENSORIA, IST-2005-016004, and the EC project REFLECT, IST-2007-215893.

References

1. Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On weak modal compatibility, refinement, and the MIO Workbench. Technical Report

1001, Institut für Informatik, Ludwig-Maximilians-Universität München, January 2010.

2. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *LNCS*, pages 117–133. Springer, 2003.
3. Edmund M. Clarke, Natasha Sharygina, and Nishant Sinha. Program compatibility approaches. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *FMCO*, volume 4111 of *LNCS*, pages 243–258. Springer, 2005.
4. Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In Thomas A. Henzinger and Christoph M. Kirsch, editors, *EMSOFT*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.
5. Luca de Alfaro and Thomas A. Henzinger. Interface-based Design. In Manfred Broy, Johannes Grünbauer, David Harel, and C. A. R. Hoare, editors, *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.
6. Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed interfaces. In Alberto L. Sangiovanni-Vincentelli and Joseph Sifakis, editors, *EMSOFT*, volume 2491 of *LNCS*, pages 108–122. Springer, 2002.
7. Nicolás D'Ippolito, Dario Fischbein, Howard Foster, and Sebastián Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In Li-Te Cheng, Alessandro Orso, and Martin P. Robillard, editors, *ETX*, pages 6–10. ACM, 2007.
8. Dario Fischbein, Víctor A. Braberman, and Sebastián Uchitel. A sound observational semantics for modal transition systems. In Martin Leucker and Carroll Morgan, editors, *ICTAC*, volume 5684 of *LNCS*, pages 215–230. Springer, 2009.
9. Stephen J. Garland and Nancy Lynch. Using I/O automata for developing distributed systems. In *Gary T. Leavens and Murali Sitaraman, editors, Foundations of Component-Based Systems*, pages 285–312. Cambridge University Press, 2000.
10. Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik*, volume 363 of *LNCS*, pages 163–180. Springer, 1989.
11. Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O automata for interface and product line theories. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.
12. Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. On modal refinement and consistency. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 105–119. Springer, 2007.
13. Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.
14. Nancy Lynch, Laurent Michel, and Alexander Shvartsman. Tempo: A toolkit for the timed input/output automata formalism. In *SIMUTools*, 2008. Industrial Track: Simulation Works. Conference Proc. CD, paper 3105, 8 pages, Marseille, France.
15. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC*, pages 137–151, 1987.
16. Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.
17. Robin Milner. *Communication and Concurrency.* Prentice Hall (International Series in Computer Science), 1989.