Interface Theories for Concurrency and Data[☆]

Sebastian S. Bauer, Rolf Hennicker, Martin Wirsing

Institut für Informatik, Ludwig-Maximilians-Universität München, Germany

Abstract

We present a compositional approach for specifying concurrent behavior of components with data states on the basis of interface theories. The dynamic aspects of a system are specified by modal input/output automata, whereas changing data states are specified by pre- and postconditions. The combination of the two formalisms leads to our notion of modal input/output automata with data constraints (MIODs). In this setting we study refinement and behavioral compatibility of MIODs. We show that compatibility is preserved by refinement and that refinement is compositional w.r.t. synchronous composition, thus satisfying basic requirements of an interface theory. We propose a semantic foundation of interface specifications where any MIOD is equipped with a model-theoretic semantics describing the class of its correct implementation models. Implementation models are formalized in terms of guarded input/output transition systems and the correctness notion is based on a simulation relation between an MIOD and an implementation model which relates not only abstract and concrete control states but also (abstract) data constraints and concrete data states. We show that our approach is compositional in the sense that locally correct implementation models of compatible MIODs compose to globally correct implementations, thus ensuring independent implementability.

Keywords: component-based design, interface theory, modal input/output automata, pre- and postcondition, compositionality, refinement, compatibility

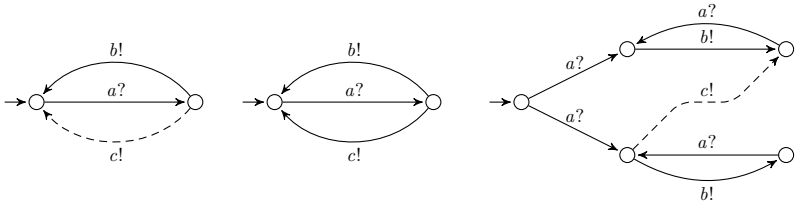
1. Introduction

Component-based software development has emerged as an important subdiscipline of software engineering. Software components represent functional units which collaborate with other components and their environment via interfaces. These interfaces usually distinguish between the required and provided operations of a component and, moreover, specify the observable behavior of components [12].

In a sequential environment the observable behavior is purely functional and can be adequately described by pre- and postconditions. In a concurrent environment the behavior is also determined by the component interactions. Most current work on interface specifications abstracts from the functional data requirements and focuses on the interaction behavior. E.g. Jan Bergstra and C. A. Middelburg propose so-called interface groups for studying the composition of interacting process components in the setting of process algebra [7]. We claim that the combination of interactions with functional behavior is far from being well understood. For instance, it is well-known that pre/postcondition style specifications do, in general, not work for systems of concurrent components, but we believe that it is still important to investigate how far one can go by using them in a concurrent environment. More specifically, this concerns the impact of integrated control flow and data flow on specifications, implementations, formal correctness and compatibility notions, composition, and, last not least, independent implementability.

[☆]This work has been partially sponsored by the EU project ASCENS, 257414. The first author has been partially supported by the German Academic Exchange Service (DAAD), grant D/10/46169.

Email addresses: bauerse@pst.ifi.lmu.de (Sebastian S. Bauer), hennicke@pst.ifi.lmu.de (Rolf Hennicker), wirsing@pst.ifi.lmu.de (Martin Wirsing)



(a) T (b) S_1 (c) S_2

Figure 1: A modal input/output automata T and possible refinements S_1 and S_2.

In this work we propose an interface theory on the basis of modal input/output automata (MIOs) introduced in [20]. A particular advantage of modal transition systems is that they distinguish between "may" and "must" transitions which leads to a powerful refinement notion [22]: the may transitions determine which actions are permitted in a refinement while the must transitions specify which actions must be present in a refinement and hence in any implementation. In this way it is possible to provide abstract, loose specifications in terms of may transitions and to fix in a stepwise way the must transitions until an implementation, represented by an MIO with must transitions only, is reached. Another aspect which can be conveniently formalized with modal input/output automata concerns the compatibility of interacting components: whenever an interface specification allows that a message *may* be issued, then the communication partner should be in a (control) state where it *must* be able to accept the message [20, 4].

Fig. 1 shows an example for MIO refinements. The MIO T provides a loose specification with two must transitions, drawn with solid arrows, and one may transition, drawn with a dashed arrow. The specification says that in the initial state, and whenever this state is reached again, any refinement of T *must* be able to input $a?$. Then there is a choice between a must transition for the output $b!$ and a may transition for the output $c!$. The "may" modality expresses that this transition is not mandatory for refinements and can be omitted or can be turned into a must transition as done in the refinement S_1 of T. Another possible refinement of T is given by the MIO S_2 which non-deterministically decides whether to switch to a mode where after each input $a?$ the only output is $b!$ or to a mode where an output $c!$ is possible once.

In our approach we extend MIOs by taking into account the specification of data constraints which enhance transitions with pre- and postconditions describing the admissible data states of a component before and after the execution of an operation. We distinguish, like in MIOs, between input, output and internal actions and, additionally, between provided, required and internal state variables. Provided and internal state variables are local to a component and describe the data states a component can adopt. In contrast to the internal state variables, provided state variables are visible to the user of a component. Required state variables belong also to the interface specification of a component, however, they are not related to the data states of the component itself but to the data states the component can observe in its environment. On this basis we study (synchronous) composition, refinement and compatibility of modal input/output automata with data constraints (MIODs). In addition to relationships between control states, we take special care of the relationships between data constraints in all these cases. For instance, considering compatibility, the condition concerning control flow compatibility is extended to take into account data states: the caller of an operation must ensure that the precondition of the operation provided by the callee is satisfied and, conversely, the callee must guarantee that after the execution of the operation the postcondition expected by the caller holds. Thus, the compatibility notion takes into account the mutual assumptions and guarantees of communicating components guided by the idea that specifications provide contracts which must match when components are composed. We show that MIODs satisfy the basic requirements of an interface theory: compatibility is preserved by refinement and refinement is preserved by synchronous composition of MIODs.

So far MIODs have been introduced in [2] as a specification formalism for concurrent, reactive components with encapsulated data states. We believe, however, that any specification S should be equipped with a formal semantics $[\![S]\!]$ which unambiguously defines the meaning of the specification, for instance for analysis and further reasoning. This is particularly important in our context due to the many subtleties which

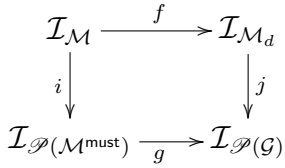


Figure 2: Interface theory morphisms.

arise when considering concurrently running components whose interactions have an effect on their data states. Since specifications are inherently loose, leaving freedom to design decisions in implementations, we will follow the loose semantics approach which, in the spirit of Hoare [18], considers the semantics of a specification as the class of all its correct implementations. In such a framework one gets for free notions like consistency, semantic equivalence of specifications etc. We take up this idea and propose a strict separation of specifications (MIODs) and implementations which are given by labeled I/O-transition systems whose states consist of a control part and of a concrete data state (formalized by an assignment of values to state variables). The labels of an implementation model represent concrete operation invocations with particular actual parameters and the transitions represent (atomic) executions of operations. Implementation models are called guarded input/output transition systems (GIOs) since all actions (sending, receiving of operation invocations and internal actions) can be guarded by concrete data states. Guards express conditions on the component's local data states and on the data states observable in the environment. An implementation model (given as a GIO) is correct w.r.t. a given MIOD if there exists a simulation relation between the two which relates control states and concrete data states of the model with control states and data constraints (i.e. pre/postconditions) of the MIOD. Then the semantics of a MIOD is given by the class of its correct implementations. Analogously to specifications, we define compatibility and synchronous composition of implementation models (GIOs) and show that our semantics is compositional and preserves compatibility. This means, that implementation models which are locally correct w.r.t. compatible MIODs are compatible as well and compose to a correct implementation model of the MIOD composition. As a consequence, our framework supports independent implementability of MIODs and substitutability of correct implementations.

Thus we get not only the syntax-directed interface theories $\mathcal{I}_{\mathcal{M}}$ for MIOs and $\mathcal{I}_{\mathcal{M}_d}$ for MIODs but also an interface theory $\mathcal{I}_{\mathscr{P}(\mathcal{G})}$ whose objects are classes of GIOs and where refinement is model class inclusion. We relate the various interface theories in terms of so-called interface theory morphisms, see Fig. 2. For instance, there is a morphism f between the interface theory $\mathcal{I}_{\mathcal{M}}$ of MIOs and the interface theory $\mathcal{I}_{\mathcal{M}_d}$ of MIODs which embeds modal input/output automata into MIODs (with trivial data constraints) such that composition, refinement and compatibility are preserved. We also show that the semantic function j associating the class of correct implementations to an MIOD is a (weak) interface theory morphism. This means, in particular, that refinement of MIODs expresses model class inclusion on the semantic level. A similar construction can be performed for the semantics of MIOs where implementations are MIOs as well but with must transitions only, see [21]. Then the semantic function i associating the class of correct implementations to an MIO is also a (weak) interface theory morphism. Finally, MIO implementations can be embedded into implementation models of MIODs by the interface theory morphism g such that the diagram in Fig. 2 commutes.

Related Work. Specifications of interaction behavior and of changing data states are often considered separately from each other. Complex interaction behavior can be well specified by process algebraic approaches [7, 24]; transition systems in form of sequence diagrams (see e.g. [9]) or basic message sequence charts (see e.g. [17]) are popular formalisms to specify the temporal ordering of messages, and pre/postconditions are commonly used to specify the effects of operations w.r.t. data states. Though approaches like CSP-OZ [15] or Circus [27, 30] offer means to specify interaction and data aspects, however they do not support modalities expressing allowed and required behavior. Other related approaches are based on symbolic transition systems (STS) [14, 1] but STS are mainly focussing on model checking and

not on interface theories supporting the (top down) development of concurrent systems by refinement. Most closely related to the concept of MIODs is the study of Mouelhi et al. [25] who consider an extension of the theory of interface automata [12] to data states. However, their approach does not take into account modal refinements and the contract principle between interface specifications which, in our case, is based on a careful and methodologically important separation of provided, internal and required state variables. Sociable interfaces [10] are another extension of interface automata which take into account data states in a similar way, however, they do not consider modalities for transitions. In our previous work, we have introduced MIODs in [2] which are further refined here and equipped with a formal semantics such that it is possible to relate specification refinement with semantic model class inclusion as sketched above. Our semantics is based on the ideas presented in [3] for behavior protocols (without modalities and without specification refinement). Existing work on modal transition systems and their use as specification formalism for component interfaces [20, 26] does not take into account explicit data states.

Personal note. In the beginning of the eighties Jan and MW (the third author of this paper) were both working on the idea to use algebraic methods for providing a sound theoretical basis to program construction. At that time Jan was starting the process algebra approach (with W. Klop) for formalizing the behavior of concurrent systems and was studying the computability of abstract data types (together with John Tucker) whereas MW (together with Manfred Broy) was developing the theory of hierarchical data types. By discussing our different approaches we came up with our (unique) common paper on the expressive power of algebraic specifications [6] in which we were able to characterize the expressivity of hierarchical and partial abstract data types. Then the common involvement in the EU project METEOR allowed us to continue this research on algebraic methods for several years in order to "provide techniques for data abstraction and the structured specification, validation and analysis of data structures" as we wrote together in the editorial of the LNCS volume 394 [29] on "Algebraic methods: Theory, Tools and Applications." It is pleasure to see that 30 years later algebraic techniques are still a cornerstone of formal software analysis; indeed, good (complementary) examples are Jan's new process algebraic theory of interface groups [7] and the interface theory approach of this paper for specifying and analyzing the behavior of interacting process components.

Outline. The paper is organized as follows. In Sect. 2 we consider the basic notions of an interface theory and interface theory morphism. The particular interface theory of MIOs with modal refinement and strong modal compatibility is recalled in Sect. 3. In Sect. 4 we introduce modal input/output automata with data constraints (MIODs). In Sect. 5 the semantics of MIODs is defined in terms of guarded input/output transition systems. In Sect. 6, Sect. 7 and Sect. 8, we define refinement, composition and compatibility of MIODs and GIOs, respectively. We show that refinement, composition and compatibility on the level of MIODs are sound with respect to their semantics formalized in terms of GIOs. Then, in Sect. 9, we relate the obtained interface theories for MIODs and GIOs by appropriate interface theory morphisms. In Sect. 10 we finish with some concluding remarks.

2. Interface Theories

A formal notion of an *interface theory* was, to our knowledge, first proposed by de Alfaro and Henzinger in [12]. In their work, an interface theory consists of an interface algebra together with a component algebra thus distinguishing between interface specifications and component implementations. Later, in [13], the authors have introduced the term *interface language* which simplifies the approach by considering just interfaces with the requirements that incremental design and independent implementability is possible. Interface theory and interface language are abstract concepts which can be instantiated by concrete formalisms. The (abstract) notion of an interface theory we shall use hereafter is close to an interface language but further simplified by concentrating on the rudimentary requirement of independent implementability. We deviate from [13] that we do not require *incremental design* to hold which is, in general, not satisfied in interface theories with a *pessimistic* compatibility notion, like the compatibility notion developed in this paper; cf. [11] for a discussion on *optimistic* and *pessimistic* approaches to compatibility.

In our study *interface theories* are required to define a class of *interface specifications* (or shortly *specifications*), together with their *composition*, *refinement* and *compatibility* which are key concepts for any

interface specification formalism. The composition operator allows to form larger specifications from smaller ones, refinement relates "concrete" and "abstract" specifications, and compatibility expresses that two specifications work properly together.

Definition 1 (Interface Theory). An *interface theory* is a tuple $(\mathcal{A}, \otimes, \leq, \leftrightarrows)$ consisting of a class $\mathcal{A}$ of interface specifications, a partial composition operator $\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$, a reflexive and transitive refinement relation $\leq \subseteq \mathcal{A} \times \mathcal{A}$, and a symmetric compatibility relation $\leftrightarrows \subseteq \mathcal{A} \times \mathcal{A}$, such that the following conditions are satisfied. Let $S, S', T, T' \in \mathcal{A}$ be specifications.

(1) **Compatibility implies composability**

If $S \leftrightarrows T$ then $S \otimes T$ is defined.

(2) **Compositional refinement**

If $S' \leq S$ and $T' \leq T$ and $S \otimes T$ is defined,
then $S' \otimes T'$ is defined and $S' \otimes T' \leq S \otimes T$.

(3) **Preservation of compatibility**

If $S \leftrightarrows T$ and $S' \leq S$ and $T' \leq T$, then $S' \leftrightarrows T'$.

In our notion of an interface theory, independent implementability of [13] is split into the conditions (2) and (3) in order to clearly identify the basic requirements of an interface theory. Condition (1) is required from an intuitive point of view since compatibility is only meaningful for interface specifications which can actually be composed.

An interface theory can be considered as an algebraic structure (cf. [28]). An *interface theory morphism*, similar to an algebraic homomorphism between algebraic structures, is a function between two interface theories preserving the composition operator and the refinement and compatibility relation.

Definition 2 (Interface Theory Morphism). Let $\mathcal{I} = (\mathcal{A}, \otimes, \leq, \leftrightarrows)$ and $\mathcal{I}' = (\mathcal{A}', \otimes', \leq', \leftrightarrows')$ be two interface theories. An *interface theory morphism* from $\mathcal{I}$ to $\mathcal{I}'$ is a function $f : \mathcal{A} \to \mathcal{A}'$ such that, for all $A, B \in \mathcal{A}$,

1. $f(A) \otimes' f(B) = f(A \otimes B)$,
2. if $A \leq B$ then $f(A) \leq' f(B)$,
3. if $A \leftrightarrows B$ then $f(A) \leftrightarrows' f(B)$.

If condition 1 is replaced by

1′. $f(A) \otimes' f(B)' \leq f(A \otimes B)$,

then f is called a *weak interface theory morphism*.

Establishing an interface theory morphism i from $\mathcal{I}$ to $\mathcal{I}'$ demonstrates that $\mathcal{I}'$ is at least as expressive as $\mathcal{I}$. The *weak* interface theory morphism is mainly motivated by the fact that the modal composition operator for modal transition systems is not complete w.r.t. implementation semantics. This will be discussed in the next section.

3. An Interface Theory for Modal Input/Output Automata

In this section, we give a short introduction to modal input/output automata (MIOs) and summarize previous work on their use as underlying specification domain in interface theories. In particular, we will consider implementation semantics of MIOs and define a weak interface theory morphism between an interface theory based on MIOs and an interface theory formed by their implementation classes (where implementations are MIOs with must transitions only). For a survey

Modal transition systems were introduced by Larsen and Thomsen in [22] as a general way of loosely specifying reactive, concurrent processes. Almost 20 years later, in [20], MIOs were proposed as a suitable interface language with composition and compatibility notions targeted on reasoning about component interfaces. MIOs distinguish between "may" and "must" transitions, where the former can be disregarded and the latter must be respected by refinements. MIOs specialize modal transition systems [22, 19] by the explicit discrimination of input, output and internal actions. An *action set* is a set Act of actions which is partitioned into disjoint sets of input, output and internal actions.

Definition 3 (MIO [20]). A *modal input/output automata (MIO)*

$$S = (Act, St, init, \Delta^{\mathsf{may}}, \Delta^{\mathsf{must}})$$

consists of an action set $Act = Act^{in} \uplus Act^{out} \uplus Act^{int}$ with pairwise disjoint sets Act^{in}, Act^{out}, Act^{int} of input, output, and internal actions resp., a set of states St, an initial state $init \in St$, a may transition relation $\Delta^{\mathsf{may}} \subseteq St \times Act \times St$, and a must transition relation $\Delta^{\mathsf{must}} \subseteq \Delta^{\mathsf{may}}$.

The condition $\Delta^{\mathsf{must}} \subseteq \Delta^{\mathsf{may}}$ is called *syntactic consistency*. A state $s \in St$ of a MIO S is called *reachable* if there exist may transitions $(s_0, a_0, s_1), (s_1, a_1, s_2), \ldots, (s_{n-1}, a_{n-1}, s_n) \in \Delta^{\mathsf{may}}$, $n \geq 0$, such that $s_0 = init$ and $s_n = s$. The set of the reachable states of S is denoted by $\mathcal{R}(S)$. An MIO satisfying $\Delta^{\mathsf{must}} = \Delta^{\mathsf{may}}$ is called an *implementation*. The class of all MIOs is denoted by $\mathcal{M}$, and the class of all implementations is denoted by $\mathcal{M}^{\mathsf{must}}$. In the following, given an MIO S, we will use subscripts to refer to the single constituent parts of S, e.g. St_S means the set of states of S.

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.

Definition 4 (Modal Refinement [22]). Let S and T be MIOs with the same action set Act. A binary relation $R \subseteq St_S \times St_T$ is a *modal refinement* between the states of S and T iff for all $(s,t) \in R$ and all $a \in Act$ it holds that

1. whenever $(t, a, t') \in \Delta_T^{\mathsf{must}}$ then there exists $s' \in St_S$ such that $(s, a, s') \in \Delta_S^{\mathsf{must}}$ and $(s', t') \in R$,
2. whenever $(s, a, s') \in \Delta_S^{\mathsf{may}}$ then there exists $t' \in St_T$ such that $(t, a, t') \in \Delta_T^{\mathsf{may}}$ and $(s', t') \in R$.

A state $s \in St_S$ *refines* a state $t \in St_T$, written $s \leq_m t$, iff there exists a modal refinement between the states of S and T which contains (s,t). S is a *modal refinement* of T, written $S \leq_m T$, iff $init_S \leq_m init_T$.

It can be easily verified that $\leq_m$ is a preorder, i.e. that $\leq_m$ is reflexive and transitive. If both S and T are implementations, i.e. if the must transition relation coincides with the may transition relation, then modal refinement coincides with (strong) bisimulation; if $\Delta_T^{\mathsf{must}} = \emptyset$ then it corresponds to simulation [23].

The implementation semantics of a MIO T, denoted by $[\![T]\!]$, consists of all modal refinements of T which are implementations (and therefore cannot be refined further, up to bisimulation). Thus, the implementation semantics of a MIO $T \in \mathcal{M}$ is given by $[\![T]\!] = \{I \in \mathcal{M}^{\mathsf{must}} \mid I \leq_m T\}$. It easily follows from transitivity of $\leq_m$ that refinement of MIOs implies inclusion of implementation classes.

Proposition 1. *For all $S, T \in \mathcal{M}$, if $S \leq_m T$ then $[\![S]\!] \subseteq [\![T]\!]$.*

It is well-known [21] that modal refinement is incomplete meaning that the converse of Proposition 1 is not true in general: there exist specifications S and T such that $[\![S]\!] \subseteq [\![T]\!]$ but $S \not\leq_m T$; a counterexample can be found, e.g., in [21]. Note that deterministic MIOs are complete [5].

MIOs can be composed to specify the behavior of concurrent systems of several interacting components. The composition operator synchronizes on shared actions yielding an internal action in the composition [20]. First, we need some syntactic restrictions under which two MIOs are composable. We require that overlapping of actions only happens on complementary types.

Definition 5 (Composability [20]). Two action sets Act_S, Act_T are *composable* if

$$Act_S \cap Act_T = (Act_S^{in} \cap Act_T^{out}) \uplus (Act_S^{out} \cap Act_T^{in}).$$

Two MIOs S and T are *composable* if their action sets are composable.

Definition 6 (Composition of Action Sets [20]). Let Act_S and Act_T be two composable action sets. Then their composition $Act_S \otimes Act_T$ is defined by

$(Act_S \otimes Act_T)^{in} = (Act_S^{in} \uplus Act_T^{in}) \setminus (Act_S \cap Act_T)$,
$(Act_S \otimes Act_T)^{out} = (Act_S^{out} \uplus Act_T^{out}) \setminus (Act_S \cap Act_T)$,
$(Act_S \otimes Act_T)^{int} = Act_S^{int} \uplus Act_T^{int} \uplus (Act_S \cap Act_T)$.

The (synchronous) parallel composition operator $\otimes$ is defined for composable MIOs in a straightforward way by synchronization on shared actions.

Definition 7 (Composition of MIOs [20]). The *composition* of two composable MIOs S and T is given by the MIO

$$S \otimes T = (Act_S \otimes Act_T, St_S \times St_T, (init_S, init_T), \Delta^{\mathsf{may}}_{S\otimes T}, \Delta^{\mathsf{must}}_{S\otimes T})$$

where the transition relations $\Delta^{\mathsf{may}}_{S\otimes T}$ and $\Delta^{\mathsf{must}}_{S\otimes T}$ are generated by the following rules:

$$\frac{(s,a,s') \in \Delta_S^\gamma \quad (t,a,t') \in \Delta_T^\gamma}{((s,t),a,(s',t')) \in \Delta^\gamma_{S\otimes T}} \quad \text{for } a \in (Act_S \cap Act_T), \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

$$\frac{(s,a,s') \in \Delta_S^\gamma \quad t \in St_T}{((s,t),a,(s',t)) \in \Delta^\gamma_{S\otimes T}} \quad \frac{(t,a,t') \in \Delta_T^\gamma \quad s \in St_S}{((s,t),a,(s,t')) \in \Delta^\gamma_{S\otimes T}} \quad \text{for } a \notin (Act_S \cap Act_T),\ \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

During composition of two composable MIOs a behavioral mismatch may occur if one of the two MIOs wants to send out a shared message which the other one cannot receive in its current state. The notion of *strong modal compatibility* rules out such erroneous situations. Two MIOs S and T are strongly modally compatible, denoted by $S \leftrightarrows T$, if they are composable and if for each reachable state (s,t) in the composition $S \otimes T$, if S *may* send out in state s an action shared with T, then T *must* be able to receive it in state t, and conversely. The difference to [13] and [20] is that we consider the "pessimistic" case, where MIOs should work properly together in *any* composable environment while the "optimistic" approach, pursued in [13] and [20], requires the existence of a (helpful) environment; for a discussion see [11].

Definition 8 (Strong Modal Compatibility [4]). Two composable MIOs S and T are *strongly modally compatible*, denoted by $S \leftrightarrows T$, iff for all reachable states $(s,t) \in \mathcal{R}(S \otimes T)$,

1. for all $a \in (Act_S^{out} \cap Act_T^{in})$, if $(s,a,s') \in \Delta_S^{\mathsf{may}}$ then there exists $t' \in St_T$ such that $(t,a,t') \in \Delta_T^{\mathsf{must}}$,
2. for all $a \in (Act_T^{out} \cap Act_S^{in})$, if $(t,a,t') \in \Delta_T^{\mathsf{may}}$ then there exists $s' \in St_S$ such that $(s,a,s') \in \Delta_S^{\mathsf{must}}$.

In [4], we have shown that MIOs, together with the synchronous composition operator $\otimes$, modal refinement $\leq_m$ and strong modal compatibility $\leftrightarrows$ satisfy all requirements of an interface theory, in particular compositional refinement and preservation of compatibility.

Theorem 2 ([4]). *$\mathcal{I}_{\mathcal{M}} = (\mathcal{M}, \otimes, \leq_m, \leftrightarrows)$ is an interface theory.*

Finally, we can define an interface theory $\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})}$ where the objects are classes of implementations. Let $\mathscr{P}(\mathcal{M}^{\mathsf{must}})$ denote the powerclass of the class $\mathcal{M}^{\mathsf{must}}$. Let $\widehat{\otimes}$ and $\widehat{\leftrightarrows}$ be the pointwise extensions of $\otimes$ and $\leftrightarrows$, respectively, to classes of MIOs in $\mathcal{M}^{\mathsf{must}}$, i.e. for $M, N \in \mathscr{P}(\mathcal{M}^{\mathsf{must}})$, $M \widehat{\otimes} N = \{S \otimes T \mid S \in M, T \in N\}$, and similarly, $M \widehat{\leftrightarrows} N$ iff $S \leftrightarrows T$ for all $S \in M$ and all $T \in N$. The proof of the compositional refinement and preservation of compatibility is trivial since refinement is just inclusion.

Theorem 3. *$\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})} = (\mathscr{P}(\mathcal{M}^{\mathsf{must}}), \widehat{\otimes}, \subseteq, \widehat{\leftrightarrows})$ is an interface theory.*

The interface theory $\mathcal{I}_{\mathcal{M}}$ and the interface theory $\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})}$ can be related by a weak interface theory morphism i mapping any $S \in \mathcal{M}$ to its implementation semantics $[\![S]\!] \subseteq \mathcal{M}^{\mathsf{must}}$ (i.e. the class of all its correct implementations).

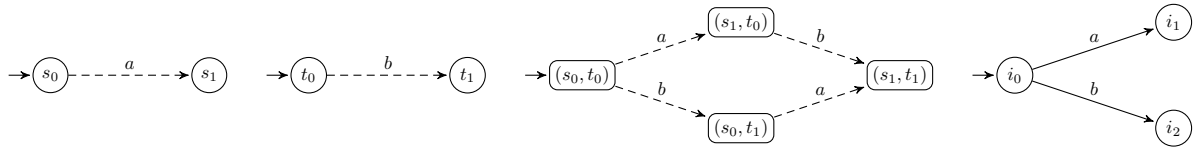
Theorem 4. *The mapping*

$$\begin{array}{rccc} i: & \mathcal{M} & \to & \mathscr{P}(\mathcal{M}^{\mathsf{must}}) \\ & S & \mapsto & [\![S]\!] \end{array}$$

is a weak interface theory morphism from $\mathcal{I}_{\mathcal{M}}$ to $\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})}$.

Proof. We have to prove all three conditions of a weak interface theory morphism. Condition 1', that is $[\![S]\!] \widehat{\otimes} [\![T]\!] \subseteq [\![S \otimes T]\!]$, follows from Theorem 2 since compositionality holds for all MIOs. Condition 2 is Proposition 1. Condition 3 follows again from Theorem 2.

It is not an interface theory morphism (in the strong sense) since, in general, $[\![S]\!] \widehat{\otimes} [\![T]\!] \subsetneq [\![S \otimes T]\!]$. This can be easily seen in the following example. Consider the MIOs S and T in Fig. 3, with $Act_S \cap Act_T = \emptyset$, and their composition $S \otimes T$. Obviously, the implementation I in Fig. 3(d) refines $S \otimes T$ and hence $I \in [\![S \otimes T]\!]$. But I is not in $[\![S]\!] \widehat{\otimes} [\![T]\!]$ since I cannot be obtained by composition of some implementation of S and some implementation of T.



(a) S (b) T (c) $S \otimes T$ (d) I

Figure 3: $I \in [\![S \otimes T]\!]$, but $I \notin [\![S]\!] \widehat{\otimes} [\![T]\!]$.

4. Modal Input/Output Automata with Data Constraints

In this section we extend MIOs to take into account interface specifications for components with encapsulated data states. For this purpose, we enrich the labels on transitions by pre- and postconditions to specify the evolution of data states caused by the execution of actions. We will introduce Modal Input/Output automata with Data constraints (MIODs) and study their composition, refinement and compatibility guided by the idea that specifications – in particular, pre- and postconditions – represent contracts describing assumptions and guarantees. A simplified approach to MIODs has been introduced in [2]. This approach is generalized here by considering labels with more general types of guards and by considering more general refinement and compatibility notions allowing case distinctions on data states.

To define the transition labels used hereafter we proceed in several steps. First, we enhance the concept of an action by introducing operations with parameters. Then we introduce various kinds of state variables

which together with operations are used to build I/O-signatures. State variables are also the basis for modeling concrete data states and for constructing state and transition predicates which will appear as pre- and postconditions on the transitions of MIODs.

In the following we assume given two disjoint global sets LV of logical variables and SV of state variables. We also assume a predefined data universe $\mathcal{U}$.

Operations. Instead of actions, we consider *operations op* which may have a (possibly empty) set of formal parameters $par(op) \subseteq \text{LV}$ treated as logical variables. An *I/O-operation signature* $O = O^{prov} \uplus O^{req} \uplus O^{int}$ consists of pairwise disjoint sets O^{prov} of *provided* operations (for inputs), O^{req} of *required* operations (for outputs), and O^{int} of *internal* operations. Provided operations are offered by a component and can be invoked by the environment; required operations are required from the environment and can be called by the component. To indicate that $op \in O^{prov}$ (O^{req}, O^{int}) we often write $op?$ ($op!$, $op;$).

State variables. In order to model data states and to equip operations with pre- and postconditions we use state variables of different kinds, which all belong to the given global set SV of state variables. *Provided* state variables describe the externally visible data states, while *internal* state variables describe the hidden data states of a component. Provided and internal state variables together model the local data states a component can adopt. There is, however, still a third kind of state variable which we call *required* state variable. Required state variables are used to refer to the data states a component expects to be visible in its environment. Formally, an *I/O-state signature* $V = V^{prov} \uplus V^{req} \uplus V^{int}$ consists of pairwise disjoint sets V^{prov}, V^{req}, and V^{int} of provided, required and internal state variables, respectively. The provided and the internal state variables together form the "local" variables denoted by $V^{loc} = V^{prov} \uplus V^{int}$.

Definition 9 (I/O-Signature). An *I/O-signature* is a pair $\Sigma = (V, O)$ consisting of an I/O-state signature V and an I/O-operation signature O.

Predicates on states. We use a generic, basic framework to deal with predicates and states. For any sets $W, W' \subseteq \text{SV}$ of state variables and set $X \subseteq \text{LV}$ of logical variables, we assume a set $\mathcal{S}(W, X)$ of *state predicates* and a set $\mathcal{T}(W, W', X)$ of *transition predicates*. State predicates, often denoted by φ, refer to single states and transition predicates, often denoted by π, to pairs of states (pre- and poststates). We require that $\mathcal{S}(W, X)$ and $\mathcal{T}(W, W', X)$ are monotonic w.r.t. set inclusion in all arguments, and that both sets are closed under the usual logical connectives like conjunction ($\wedge$) and implication ($\Rightarrow$).

Data states and satisfaction relation. For any $W \subseteq \text{SV}$, we define the set $\mathcal{D}(W)$ of W-data states to consist of all functions $\sigma : W \to \mathcal{U}$ assigning values to state variables; an element $\sigma \in \mathcal{D}(W)$ defines a concrete data state w.r.t. W. For each subset $X \subseteq \text{LV}$, we define the set $Val(X)$ of all valuations $\rho : X \to \mathcal{U}$. We assume that state predicates $\varphi \in \mathcal{S}(W, X)$ are equipped with a *satisfaction relation* $(\sigma; \rho) \models^X_W \varphi$ for states $\sigma \in \mathcal{D}(W)$ and valuations $\rho \in Val(X)$. If $X = \emptyset$ then we also write $\sigma \models^X_W \varphi$. Similarly, for transition predicates $\pi \in \mathcal{T}(W, W', X)$ we assume a satisfaction relation $(\sigma, \sigma'; \rho) \models^X_{W,W'} \pi$, for two states $\sigma \in \mathcal{D}(W)$ (prestate) and $\sigma' \in \mathcal{D}(W')$ (poststate) and valuations $\rho \in Val(X)$. Super- and subscripts of the satisfaction relation are omitted in the following if they are clear from the context. For $\varphi \in \mathcal{S}(W, X)$, we write $\models \varphi$ to express that φ is universally valid, i.e. $(\sigma; \rho) \models \varphi$ for all $\sigma \in \mathcal{D}(W)$ and all $\rho \in Val(X)$. φ is satisfiable if there exists $\sigma \in \mathcal{D}(W)$ and $\rho \in Val(X)$ such that $(\sigma; \rho) \models \varphi$. Universal validity and satisfiability of transition predicates are defined analogously. The logical connectives are interpreted as usual, e.g. $(\sigma; \rho) \models \varphi_1 \wedge \varphi_2$ iff $(\sigma; \rho) \models \varphi_1$ and $(\sigma; \rho) \models \varphi_2$. We require that the language contains a universally valid state (and transition) predicate *true*. We will frequently use state predicates in combination with transition predicates. Therefore, we require that every state predicate is also a transition predicate where state variables refer to the prestate only; i.e. given a state predicate $\varphi \in \mathcal{S}(W, X)$, we require that $\varphi \in \mathcal{T}(W, W', X)$ for any $W' \subseteq \text{SV}$ such that for all $\sigma \in \mathcal{D}(W)$, all $\sigma' \in \mathcal{D}(W')$ and all $\rho \in Val(X)$, $(\sigma, \sigma'; \rho) \models^X_{W,W'} \varphi$ iff $(\sigma; \rho) \models^X_W \varphi$

Finally, we require that a satisfaction condition, similar to institutions [16], holds. For transition predicates π, the satisfaction condition is as follows: For all $W_1 \subseteq W_1' \subseteq \text{SV}$, $W_2 \subseteq W_2' \subseteq \text{SV}$ and $X \subseteq X' \subseteq \text{LV}$, for all $\sigma \in \mathcal{D}(W_1')$ and $\sigma' \in \mathcal{D}(W_2')$ and $\rho \in Val(X')$, for all $\pi \in \mathcal{T}(W_1, W_2, X)$ it holds that

$$(\sigma, \sigma'; \rho) \models^{X'}_{W_1', W_2'} \pi \text{ if and only if } (\sigma|_{W_1}, \sigma'|_{W_2}; \rho|_X) \models^X_{W_1, W_2} \pi$$

where $f|_A$ denotes the usual restriction of a function f to a subset A of its definition domain. An analogous satisfaction condition is required for state predicates. The satisfaction condition is implicitly used throughout the proofs in this paper.

The above definitions are generic and sufficient for the following considerations. Therefore, we do not fix a particular syntax for signatures and predicates here, neither a particular definition of the satisfaction relation. We claim that our notions could be easily instantiated in the context of a particular assertion language based, e.g., on the equational or first-order logic calculus or on set-theoretic notations like in Z. How this would work in the case of the Object Constraint Language OCL is sketched in [8].

Example 1. Our running example is a simple system consisting of two components modeling a researcher and a coffee machine. In short, the researcher can drop coins into the machine's slot, and can request coffee or tea.

We start by exemplifying the use of signatures in our running example. The component *Researcher* has the I/O-signature $\Sigma_{Researcher} = (V_{Researcher}, O_{Researcher})$ where

$$\begin{array}{ll} V^{prov}_{Researcher} = \emptyset & O^{prov}_{Researcher} = \{wakeUp, coffee, tea\} \\ V^{req}_{Researcher} = \{cp, m\} & O^{req}_{Researcher} = \{publish, coin(x), selectCoffee, selectTea\} \\ V^{int}_{Researcher} = \{ct\} & O^{int}_{Researcher} = \{relax\} \end{array}$$

The component *Researcher* has as internal state variable ct modeling the number of coffees the researcher has drunk today. Required state variables are cp which models the machine's coffee price and m the machine's current credit. The provided operations include *wakeUp* to wake up the sleeping researcher, *coffee* and *tea* to receive a coffee or tea. The required operations are *publish* (write and publish a paper), $coin(x)$ (drop a coin with value x into the machine's coin slot), *selectCoffee* and *selectTea* (press the coffee and tea button, respectively). Finally, the researcher relaxes by performing the internal operation *relax*. The only operation having formal parameters is $coin(x)$.

The I/O-signature $\Sigma_{Machine} = (V_{Machine}, O_{Machine})$ of the component *Machine* is determined by the sets of variables $V^{prov}_{Machine} = \{cp, m\}$, $V^{req}_{Machine} = \{\}$ and $V^{int}_{Machine} = \{\}$, and the sets of operations are given by $O^{prov}_{Machine} = \{coin(x), selectCoffee, selectTea\}$, $O^{req}_{Machine} = \{coffee, tea\}$ and $O^{int}_{Machine} = \{\}$, ■

Transition Labels of MIODs. We are now able to define the kind of labels which can occur in a modal input/output automaton with data constraints. Given an I/O-signature $\Sigma = (V, O)$, the set $\mathcal{L}(\Sigma)$ of Σ-labels consists of the following expressions where operations (of any kind) are surrounded by pre- and postconditions which may contain the operation's formal parameters as logical variables.

- $[\varphi]op?[\pi]$ with $\varphi \in \mathcal{S}(V, par(op))$, $op \in O^{prov}$, $\pi \in \mathcal{T}(V, V^{loc}, par(op))$.
- $[\varphi]op![\pi]$ with $\varphi \in \mathcal{S}(V, par(op))$, $op \in O^{req}$, $\pi \in \mathcal{T}(V, V^{req}, par(op))$.
- $[\varphi]op;[\pi]$ with $\varphi \in \mathcal{S}(V, par(op))$, $op \in O^{int}$, $\pi \in \mathcal{T}(V, V^{loc}, par(op))$.

Note that the symbols "?" ("!",";") are just used as decorations in order to emphasize that op is a provided (required, internal) operation. Thus in the following, if we write $[\varphi]op[\pi]$ then op can be a provided, required or internal operation.

We have decided to consider explicit preconditions instead of considering postconditions only and relying on their weakest preconditions [10]. Explicit preconditions meet better our intuition about the contract principle of interfaces and the methodological ideas for the definitions of compatibility and refinement later on. Preconditions φ are state predicates which can refer to any kind of state variable, i.e. to variables local to a component as well as to required variables in the environment. This means that input, output and internal operations of a component can be guarded by a condition which can be checked in an implementation by inspecting the local data state of the component and/or by querying the visible data state of the environment.

An *input* label $[\varphi]op?[\pi]$ models that a provided operation op can be invoked under the precondition φ and then the postcondition π will hold after the execution of op. The postcondition π of an input is a

transition predicate which must only specify changes of data states for local state variables. Concerning the contract principle, the precondition φ expresses both, a guarantee and an assumption of the input. It guarantees that the operation is input-enabled if φ holds while it assumes that the operation is only called in a state where φ holds. For inputs, the postcondition π expresses just a guarantee, saying that the operation execution will lead to a state where π holds.

An *output* label $[\varphi]op![\pi]$ models that a component issues a call to a required operation *op* if the precondition φ is satisfied and after execution of the invoked operation the component expects that the postcondition π holds. The postcondition of an output is a transition predicate which must only specify the expected changes of the visible data states in the environment, i.e. for required state variables. Hence, outputs are not expected to alter the data state of the calling component itself. From the contract point of view, the precondition φ of an output expresses again a guarantee *and* an assumption. It guarantees that the operation call is issued only in a state where φ holds while it assumes that the environment will be ready (enabled) to take the operation call if φ is satisfied. For outputs, the postcondition π expresses just an assumption on the environment as explained above.

Finally, an *internal* label $[\varphi]op;[\pi]$ stands for the execution of an internal operation *op*. In this case φ describes the condition under which the internal operation is executed and π models the change of the component's local data state caused by the execution of the operation *op*. For internal operations the contract principle is not relevant.

The next definition extends modal input/output automata to take into account constraints on data states. The resulting transition systems, called MIODs, provide interface specifications for components with data states. They do not only specify the control flow of behaviors but also the effect on data states in terms of pre- and postconditions. Moreover, the modalities stemming from MIOs allow additionally to distinguish must and may transitions. In the context of modalities the assume/guarantee reasoning from above can even be refined, since preconditions on may transitions can only express assumptions but no guarantees. In particular, enabledness of an input can only be guaranteed by must transitions. Consider, for instance, a must transition starting from the initial state with label $[\varphi]op?[true]$ and a may transition from the initial state with label $[true]op?[true]$. Then any (correct) implementation will be input-enabled in the initial state if φ is satisfied. If φ is not satisfied there can be implementations which are not input-enabled but, according to the may transition, there can also be implementations which are always input-enabled. But this is not guaranteed by the interface specification.

Definition 10 (MIOD). A *modal I/O automaton with data constraints (MIOD)*

$$S = (\Sigma, St, init, \varphi^0, \Delta^{\mathsf{may}}, \Delta^{\mathsf{must}})$$

consists of an I/O-signature Σ, a finite set of states St, the initial (control) state $init \in St$, the initial (data) state predicate $\varphi^0 \in \mathcal{S}(V^{loc}, \emptyset)$, a finite may transition relation $\Delta^{\mathsf{may}} \subseteq St \times \mathcal{L}(\Sigma) \times St$, and a finite must transition relation $\Delta^{\mathsf{must}} \subseteq \Delta^{\mathsf{may}}$.

The class of all MIODs is denoted by $\mathcal{M}_d$.

Example 2. We continue our running example. The I/O-signatures have already been described in Ex. 1. In Fig. 4 the two interface specifications for *Researcher* and *Machine* are shown. The I/O-signature is shown in the diagram as follows. The sets of variables are written in the lower compartment of the surrounding box. The sets of operations are shown by drawing them at the border of the box:

- If an operation is above an *incoming* arrow then it is a provided operation.
- If an operation is above an *outgoing* arrow then it is a required operation.
- If an operation is next to a small bullet without any arrows then it is an internal operation.

For instance, the set of operations $O_{Researcher}$ contains *wakeUp* as a provided, $coin(x)$ as a required and *relax* as an internal operation.

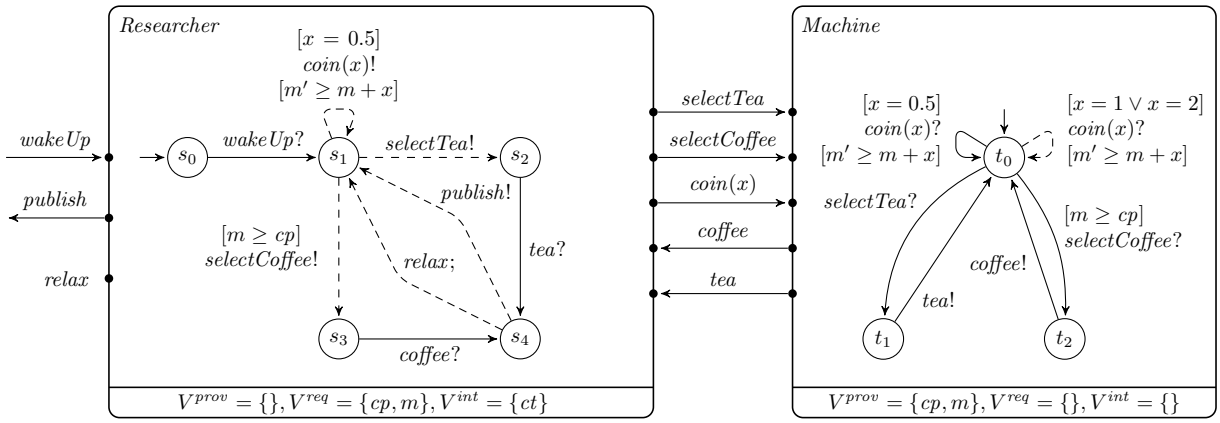


Figure 4: Abstract system specification: *Researcher* and *Machine*.

The initial state of an MIOD is indicated by a state with an incoming arrow without source state. In our examples, we always assume *true* as the initial state predicates. Must transitions are drawn with solid arrows and may transitions with dashed arrows. May transitions underlying must transitions are not drawn. Preconditions are written above/in front of and postconditions below/after operation names. We use a very simple language for the predicates, with the usual arithmetic operations and relations with the usual interpretation. The primed variables in postconditions indicate that we refer to its value in the poststate. Pre- and postconditions of the form [*true*] are omitted.

The *Researcher*, after being woken up (*wakeUp*?), can throw 0.50€ coins into the slot of the machine (*coin*(*x*)!) while she can assume that the credit displayed increases accordingly. When the credit exceeds the coffee price, she may press a button to request a coffee (*selectCoffee*!). She may also press the tea button (*selectTea*!), even without throwing any coin into the machine's slot. After the machine has dispensed either coffee or tea (*coffee*?, *tea*?), she may relax (*relax*;) or write and publish a paper (*publish*!). The behavior of *Machine* is almost as expected. Note that in the initial state, it may also accept 1€ or 2€ coins. ■

Before we develop refinement, compatibility, and composition for MIODs we will first consider, in the next section, a semantic interpretation.

5. Implementation Semantics

We propose a formal semantics for MIODs which assigns, to any MIOD S, the class $[\![S]\!]$ of all correct implementations of S. For the definition of implementations (or implementation models) we use guarded input/output transition systems (GIOs) which are supposed to provide a suitable semantic formalization for the behavior of components implemented on the basis of concrete data states and concrete control states determining the current execution points of an implementation. Given an I/O-signature $\Sigma = (V, O)$, the state space of an implementation model is given by the cartesian product of a set C of control states and the set $\mathcal{D}(V^{loc})$ of local data states. Hence any state (c, σ) of an implementation is determined by a control state $c \in C$ and a local data state $\sigma \in \mathcal{D}(V^{loc})$. Implementation labels describe incoming, outgoing and internal operation calls with actual parameter values. Since the actual execution of all kinds of operations may depend on conditions on the environment, labels will be restricted by a guard $\nu \in \mathcal{D}(V^{req})$ which represents a visible data state of the environment. Guards express that the implementation will only execute the transition if the environment is in the state determined by the guard. This will, of course, be crucial when we consider the composition of implementations later on. In a concrete program the guard may require that the sender component performs in one atomic step a test on the visible data state of the environment and, depending on the result, performs the action.

The set $\mathcal{L}^{impl}(\Sigma)$ of implementation labels consists of the following expressions:

- A label of the form $[\nu](op, \rho)?$ expresses that if the visible state of the environment is ν, the provided operation op is enabled for the actual parameters determined by valuation $\rho \in Val(par(op))$. A transition labeled with $[\nu](op, \rho)?$ connects a (control) state where the operation is called with the state after execution of the operation. Hence the implementation models considered here assume atomic operation executions.

- A label of the form $[\nu](op, \rho)!$ expresses that the implementation issues an operation call of op with actual parameters determined by ρ provided that the visible data state of the environment is ν. The target state of a transition labeled by $[\nu](op, \rho)!$ is reached when the environment has finished the execution of the operation.

- Finally, internal operation calls are described with labels of the form $[\nu](op, \rho)$; which express an internal execution of an operation under the environment condition ν. The target state of a transition labeled by $[\nu](op, \rho)$; is reached when the operation has finished its execution.

Definition 11 (GIO). A *guarded input/output transition system (GIO)* $I = (\Sigma, Q, (c^0, \sigma^0), \Delta)$ consists of an I/O-signature Σ, a set of states $Q = C \times \mathcal{D}(V^{loc})$ where C is a set of control states, an initial state $(c^0, \sigma^0) \in Q$, and a transition relation $\Delta \subseteq Q \times \mathcal{L}^{impl}(\Sigma) \times Q$.

The class of all GIOs is denoted by $\mathcal{G}$. The set of the reachable states of I is denoted by $\mathcal{R}(I)$.

Let us now discuss implementation correctness for an implementation model I w.r.t. a given MIOD T. The implementor of T must ensure the guarantees provided by T if the assumptions are met (by the environment). For the formalization of the implementation notion we follow the simulation idea of MIO refinement and define an implementation relation between concrete and abstract states. First, we consider must transitions of T with labels of the form $[\varphi]op[\pi]$; cf. 1 in Def. 12. Any such transition in T must (at least) be implemented by a transition in I whenever φ is valid in the current data state (for any valuation of the parameters of op), and the implementing transition in I must stay in the implementation relation. If op is a provided or internal operation then the implementing transition must, additionally lead to a data state in which the postcondition π is satisfied. The condition 2 of Def. 12 is similarly. It formalizes the fact that each transition in the implementation is allowed by the specification.

Definition 12 (Implementation Relation). Let T be a MIOD and I be an GIO, both with the same I/O-signature $\Sigma = (V, O)$. A binary relation $R \subseteq Q_I \times St_T$ is an *implementation relation* between the states of I and T iff for all $((c, \sigma), t) \in R$,

1. **from specification to implementation**

   for all $(t, [\varphi]op[\pi], t') \in \Delta_T^{\mathsf{must}}$, all $\nu \in \mathcal{D}(V^{req})$, and all $\rho \in Val(par(op))$,
   if $(\sigma \cdot \nu; \rho) \vDash \varphi$ then there exists $((c, \sigma), [\nu](op, \rho), (c', \sigma')) \in \Delta_I$ such that
   - if $op \in O^{prov} \uplus O^{int}$ then $(\sigma \cdot \nu, \sigma'; \rho) \vDash \pi$,[1]
   - $((c', \sigma'), t') \in R$;

2. **from implementation to specification**

   if $((c, \sigma), [\nu](op, \rho), (c', \sigma')) \in \Delta_I$ then there exists $(t, [\varphi]op[\pi], t') \in \Delta_T^{\mathsf{may}}$ such that
   - $(\sigma \cdot \nu; \rho) \vDash \varphi$,
   - if $op \in O^{prov} \uplus O^{int}$ then $(\sigma \cdot \nu, \sigma'; \rho) \vDash \pi$,
   - $((c', \sigma'), t') \in R$;

[1] Here and in the following the notation $\sigma \cdot \nu$ denotes the union of the data states σ an ν which are defined on the disjoint sets of local and required variables resp.

A state $(c, \sigma) \in S_I$ *implements* a state $t \in St_T$, written $(c, \sigma) \triangleleft t$, iff there exists an implementation relation containing $((c, \sigma), t)$. I is an *implementation* of T (or I *implements* T), denoted by $I \triangleleft T$, iff $(c^0, \sigma^0) \triangleleft init_T$ and $\sigma^0 \vDash \varphi^0$.

The implementation semantics of a MIOD T is defined by $[\![T]\!] = \{I \in \mathcal{G} \mid I \triangleleft T\}$. A MIOD is called *consistent*, if $[\![T]\!] \neq \emptyset$.

Example 3. Assume a correct implementation model I of *Researcher*. If I is in some concrete state (c, σ) which is related by the implementation relation to the abstract state s_1 of *Researcher* (see Fig. 4), then I may perform a transition

$$((c, \sigma), [\nu](coin(x), \rho)!, (c', \sigma')) \in \Delta_I$$

for some required data state $\nu \in \mathcal{D}(\{m, cp\})$ and for some valuation $\rho \in Val(\{x\})$. This transition can only be allowed by the following transition in *Researcher*:

$$(s_1, [x = 0.5]coin(x)![m' \geq m + x], s_1) \in \Delta^{\mathsf{may}}_{Researcher}$$

Now correctness means that the precondition must be satisfied, i.e. $(\sigma \cdot \nu; \rho) \vDash x = 0.5$ which basically requires that $\rho(x) = 0.5$. The second correctness condition is that the target states (c', σ') and s_1 are again related by the implementation relation. Note that the postcondition of the *required* operation $coin(x)!$ is not taken into account in the implementation relation since it is an assumption on the change of the data states in the environment. ■

In the next sections, we will introduce refinement, (synchronous) composition, and compatibility for GIOs; we will also introduce their counterparts on the level of MIODs, prove their soundness and hence arrive at an interface theory which supports the desired properties of compositional refinement and preservation of compatibility.

6. Refinement of MIODs

We follow the basic idea of modal refinement [22] where must transitions of an abstract specification must be respected by the more concrete specification and, conversely, may transitions of the concrete specification must be allowed by the abstract one. Concerning the impact of data constraints, every must transitions of an abstract MIOD, say T, with a precondition φ_T must be simulated by a corresponding must transition of a more concrete MIOD, say S, whose precondition does not require more than φ_T does. In general, this idea can be relaxed since it is sufficient if the precondition on a must transition of T is matched by the disjunction of several preconditions distributed over different transitions of S which all maintain the simulation relation between states; see the first item of condition 1 in Def. 13. This condition is independent of the kind of the labels. Concerning postconditions the situation is different, because postconditions are not related to the executability of transitions but rather to the specification of admissible poststates after a transition has fired. In this case, if the must transition of T concerns input or internal labels, the corresponding must transition of the refinement S should lead to a postcondition which guarantees the postcondition π_T of T. This idea can again be relaxed by taking the splitting into different transitions in S into account; see item two of Def. 13(1). If a must transition of T concerns an output label, then the postcondition π_T expresses the expectation of T about the next state of the environment. Then, obviously, the postcondition of the refinement should be at most weaker than π_T which is formalized, for the general case of splitting transitions, in the third item of Def. 13(1).

When moving from concrete to abstract specifications concrete may transitions must be allowed by the abstract specification which is formalized in condition 2 of Def. 13. In this case, the simulation of a concrete may transition of S can be split into different allowed transitions of the abstract specification T. If we compare conditions 1 and 2 we can observe that the implication direction concerning preconditions in a refinement depends on the kind of the transitions (may or must) while the implication direction concerning postconditions in a refinement depends on the kind of the labels (input, internal, or output). This fits to our contract principle where postconditions of inputs are guarantees which must also hold in refinements while postconditions of outputs are assumptions which must be valid in accordance with the abstract specification.

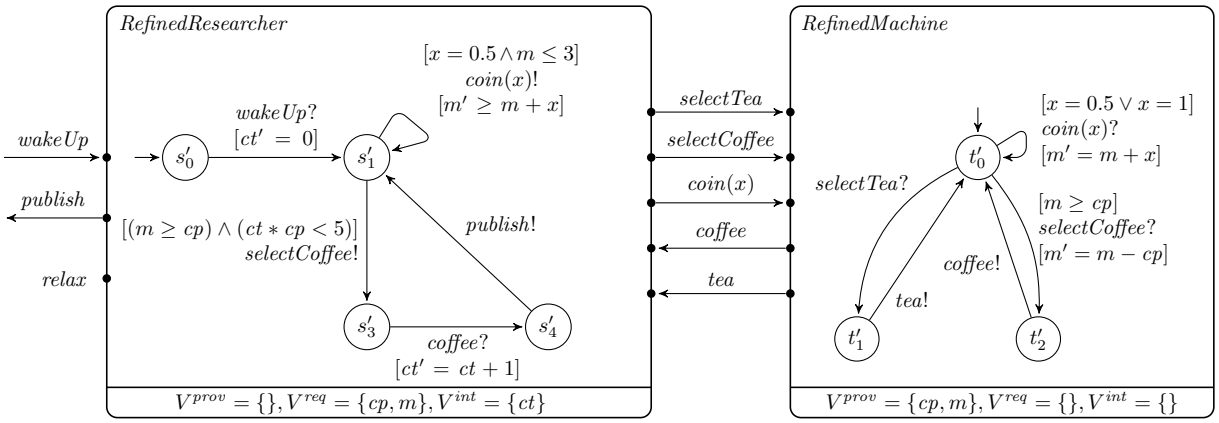


Figure 5: Refined system specification: *RefinedResearcher* and *RefinedMachine*.

Definition 13 (Modal Refinement). Let S and T be two MIODs with the same I/O-signature. A binary relation $R \subseteq St_S \times St_T$ is a *modal refinement* between the states of S and T iff for all $(s,t) \in R$,

1. **from abstract to concrete**

   if $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{must}}$ and φ_T is satisfiable
   then there exists $N \geq 0$ and transitions $(s, [\varphi_{S,i}]op[\pi_{S,i}], s'_i) \in \Delta_S^{\mathsf{must}}$, $0 \leq i \leq N$, such that

   - $\models \varphi_T \Rightarrow \bigvee_i \varphi_{S,i}$
   - for all i, if $op \in O^{prov} \uplus O^{int}$ then $\models \varphi_T \wedge \varphi_{S,i} \wedge \pi_{S,i} \Rightarrow \pi_T$
   - for all i, if $op \in O^{req}$ then $\models \varphi_T \wedge \varphi_{S,i} \wedge \pi_T \Rightarrow \pi_{S,i}$
   - for all i, $(s'_i, t') \in R$

   are satisfied.

2. **from concrete to abstract**

   if $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{may}}$ and φ_S is satisfiable
   then there exists $N \geq 0$ and $(t, [\varphi_{T,i}]op[\pi_{T,i}], t'_i) \in \Delta_T^{\mathsf{may}}$, $0 \leq i \leq N$, such that

   - $\models \varphi_S \Rightarrow \bigvee_i \varphi_{T,i}$
   - for all i, if $op \in O^{prov} \uplus O^{int}$ then $\models \varphi_S \wedge \varphi_{T,i} \wedge \pi_S \Rightarrow \pi_{T,i}$
   - for all i, if $op \in O^{req}$ then $\models \varphi_S \wedge \varphi_{T,i} \wedge \pi_{T,i} \Rightarrow \pi_S$
   - for all i, $(s', t'_i) \in R$

   are satisfied.

A state $s \in St_S$ *refines* a state $t \in St_T$, written $s \leq_{md} t$, iff there exists a modal refinement between the states of S and T containing (s,t). S is a *modal refinement* of T, written $S \leq_{md} T$, iff $init_S \leq_{md} init_T$ and $\models \varphi_S^0 \Rightarrow \varphi_T^0$.

It can be easily verified that $\leq_{md}$ is a preorder on $\mathcal{M}_d$ (the class of all MIODs).

Example 4. The abstract specifications (see Fig. 4) are now refined as shown in Fig. 5. Concerning the control flow, we have left out the may transitions with label *selectTea*! and *relax*; and the other may transitions have been refined to must transitions. Concerning the data constraints, in *RefinedResearcher* the postcondition of *wakeUp*? has been strengthened by initializing the internal state variable *ct* (modeling the number of coffees she had today) by 0. The precondition of *coin*(x)! is strengthened such that she will stop throwing coins into the machine's slot if the displayed credit is greater than 3. *RefinedResearcher*

is refined in such a way that the request of coffee also depends on the number of coffees she has already drunk today ($ct * cp < 5$, i.e. she requests coffee if she did not have enough coffee today, or the coffee is very cheap). When the machine dispenses a coffee the number of coffees is increased by one. The relation demonstrating the refinement *RefinedMachine* $\leq_{md}$ *Machine* is $R = \{(s'_i, s_i) \mid i \in \{0, 1, 3, 4\}\}$. For instance, the may transition

$$(s'_1, [(m \geq cp) \wedge (ct * cp < 5)]selectCoffee![true], s'_3) \in \Delta^{\mathsf{may}}_{RefinedResearcher}$$

is allowed by the may transition

$$(s_1, [m \geq cp]selectCoffee![true], s_3) \in \Delta^{\mathsf{may}}_{Researcher}$$

and we have to check whether $\vDash ((m \geq cp) \wedge (ct * cp < 5)) \Rightarrow (m \geq cp)$ which is obviously satisfied. The condition for the postconditions are trivially satisfied, and the next states s'_3 and s_3 are again related by R.

Now consider the refined specification *RefinedMachine*. The two input transitions for operation $coin(x)$? has been refined to a single (must) transition. Concerning predicates, the postconditions of the transitions labeled with $coin(x)$? and *selectCoffee*? have been strengthened. The relation demonstrating the refinement *RefinedMachine* $\leq_{md}$ *Machine* is $R' = \{(t'_i, t_i) \mid i \in \{0, 1, 2\}\}$. For instance, the must transition

$$(t_0, [x = 0.5]coin(x)?[m' \geq m + x], t_0) \in \Delta^{\mathsf{must}}_{Machine}$$

is matched by

$$(t'_0, [x = 0.5 \vee x = 1]coin(x)?[m' = m + x], t'_0) \in \Delta^{\mathsf{must}}_{RefinedMachine}$$

and, obviously, $\vDash x = 0.5 \Rightarrow (x = 0.5 \vee x = 1)$, and for all may transitions labeled with $coin(x)$? in *RefinedMachine*, postconditions must match, i.e. $\vDash x = 0.5 \wedge (x = 0.5 \vee x = 1) \wedge (m' = m + x) \Rightarrow (m' \geq m + x)$ is satisfied. There is also a may transition in *RefinedMachine*,

$$(t'_0, [x = 0.5 \vee x = 1]coin(x)?[m' = m + x], t'_0) \in \Delta^{\mathsf{may}}_{RefinedMachine}$$

which must be allowed by *Machine*. We can find the two transitions in *Machine*,

$$(t_0, [x = 0.5]coin(x)?[m' \geq m + x], t_0) \in \Delta^{\mathsf{may}}_{Machine}$$
$$(t_0, [x = 1 \vee x = 2]coin(x)?[m' \geq m + x], t_0) \in \Delta^{\mathsf{may}}_{Machine}$$

for which $\vDash (x = 0.5 \vee x = 1) \Rightarrow x = 0.5 \vee (x = 1 \vee x = 2)$, and for the postconditions,

$$\vDash (x = 0.5 \vee x = 1) \wedge x = 0.5 \wedge (m' = m + x) \Rightarrow (m' \geq m + x)$$
$$\vDash (x = 0.5 \vee x = 1) \wedge (x = 1 \vee x = 2) \wedge (m' = m + x) \Rightarrow (m' \geq m + x)$$

are satisfied. ■

As a first result we can prove that modal refinement implies inclusion of implementation semantics. Thus modal refinement of MIODs is sound; refinement means less implementations. Since, in general, we cannot get completeness (which was already shown for MIOs), refinement of MIODs will remain an approximation. Obviously, the refinement relation proposed above is a better approximation than the simpler form of MIOD refinement in [2] which was appropriate to show the intuition (without incorporating splitting of transitions as done above).

Proposition 5. *Let S and T be two MIODs with the same I/O-signature. Then $S \leq_{md} T$ implies $[\![S]\!] \subseteq [\![T]\!]$.*

Proof. Let $I \in [\![S]\!]$ be an implementation of S. We have to show that $I \in [\![T]\!]$. We define a relation $R \subseteq Q_I \times St_T$ by

$$R = \{((c, \sigma), t) \mid \exists s \in St_S : (c, \sigma) \triangleleft s \text{ and } s \leq_{md} t\}.$$

We show that R is an implementation relation between I and T. Let $((c,\sigma),t) \in R$. By definition of R we can assume a state $s \in St_S$ such that $(c,\sigma) \triangleleft s$ and $s \leq_{md} t$.

Condition 1 of Def. 12: Assume $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{must}}$ and let $\nu \in \mathcal{D}(V^{req})$ and $\rho \in Val(par(op))$ such that

$$(\sigma \cdot \nu; \rho) \vDash \varphi_T. \tag{1}$$

We know by assumption that $s \leq_{md} t$, hence by definition of modal refinement, there exists $N \geq 0$ and transitions $(s, [\varphi_{S,i}]op[\pi_{S,i}], s_i') \in \Delta_S^{\mathsf{must}}$, for $0 \leq i \leq N$, such that $\vDash \varphi_T \Rightarrow \bigvee_i \varphi_{S,i}$. From (1) it follows that there is some $0 \leq j \leq N$ such that

$$(\sigma \cdot \nu; \rho) \vDash \varphi_{S,j}. \tag{2}$$

Additionally, we know from $s \leq_{md} t$ that for the target state s_j' it holds that $s_j' \leq_m t'$, and

$$\text{if } op \in O^{prov} \uplus O^{int} \text{ then } \vDash \varphi_T \wedge \varphi_{S,j} \wedge \pi_{S,j} \Rightarrow \pi_T. \tag{3}$$

From $(c,\sigma) \triangleleft s$ it follows that there exists a transition $((c,\sigma),[\nu](op,\rho),(c',\sigma')) \in \Delta_I$ such that $(c',\sigma') \triangleleft s_j'$, and

$$\text{if } op \in O^{prov} \uplus O^{int} \text{ then } (\sigma \cdot \nu, \sigma'; \rho) \vDash \pi_{S,j}. \tag{4}$$

We still have to show that if $op \in O^{prov} \uplus O^{int}$ then $(\sigma \cdot \nu, \sigma'; \rho) \vDash \pi_T$. However this follows from (1), (2), (3), and (4).

Condition 2 can be shown in a similar way.

Finally, from $S \leq_{md} T$ it follows that $init_S \leq_{md} init_T$ and $\vDash \varphi_S^0 \Rightarrow \varphi_T^0$. Moreover, from $I \in [\![S]\!]$ we know that $(c^0,\sigma^0) \triangleleft init_S$ and $\sigma^0 \vDash \varphi_S^0$. Then $((c^0,\sigma^0), init_T) \in R$ since there is $init_S \in St_S$ such that $(c^0,\sigma^0) \triangleleft init_S$ and $init_S \leq_{md} init_T$; and $\sigma^0 \vDash \varphi_S^0$ and $\vDash \varphi_S^0 \Rightarrow \varphi_T^0$ imply $\sigma^0 \vDash \varphi_T^0$. Thus I is an implementation of T. □

7. Composition

MIODs can be composed to specify the behavior of concurrent systems of interacting components with data states. The composition operator extends the synchronous composition of modal input/output automata [20, 4].

For defining the composition operator, we need some syntactic restrictions under which two I/O-signatures are composable. We require that overlapping of operations only happens on complementary types and that the same holds for state variables. More precisely, two I/O-signatures Σ_S and Σ_T are *composable* if $O_S \cap O_T = (O_S^{prov} \cap O_T^{req}) \uplus (O_T^{prov} \cap O_S^{req})$ and $V_S \cap V_T = (V_S^{prov} \cap V_T^{req}) \uplus (V_T^{prov} \cap V_S^{req})$. Two MIODs (GIOs resp.) are called *composable* if their signatures are composable.

Two composable I/O-signatures $\Sigma_S = (V_S, O_S)$ and $\Sigma_T = (V_T, O_T)$ can be composed to $\Sigma_S \otimes \Sigma_T = (O_S \otimes O_T, V_S \otimes V_T)$ where shared variables as well as shared operations are internalized:

$$\begin{array}{ll}
(O_S \otimes O_T)^{prov} = (O_S^{prov} \uplus O_T^{prov}) \setminus (O_S \cap O_T) & (V_S \otimes V_T)^{prov} = (V_S^{prov} \uplus V_T^{prov}) \setminus (V_S \cap V_T) \\
(O_S \otimes O_T)^{req} = (O_S^{req} \uplus O_T^{req}) \setminus (O_S \cap O_T) & (V_S \otimes V_T)^{req} = (V_S^{req} \uplus V_T^{req}) \setminus (V_S \cap V_T) \\
(O_S \otimes O_T)^{int} = O_S^{int} \uplus O_T^{int} \uplus (O_S \cap O_T) & (V_S \otimes V_T)^{int} = V_S^{int} \uplus V_T^{int} \uplus (V_S \cap V_T)
\end{array}$$

Hence, $(V_S \otimes V_T) = V_S \cup V_T$ and $(V_S \otimes V_T)^{loc} = V_S^{loc} \cup V_T^{loc}$.

First, we define a composition operator on the syntactic level of MIODs. The synchronous composition $S \otimes T$ of two MIODs S and T synchronizes transitions whose labels refer to shared operations. For instance, a transition with label $[\varphi_S]op![\pi_S]$ of S is synchronized with a transition with label $[\varphi_T]op?[\pi_T]$ of T which results in a transition with label $[\varphi_S \wedge \varphi_T]op[\pi_T]$ where the original preconditions are combined by logical conjunction and only the postcondition π_T of the input is kept. Since the postcondition π_S of the output expresses an assumption on the environment and since input and output actions synchronize to an internal action, π_S is irrelevant for the composition. Transitions whose labels concern shared operations which cannot be synchronized are dropped (as usual) while all other transitions are interleaved in the composition. Concerning modalities we follow the usual modal composition operator [20] which yields a must transition if two must transitions are synchronized and a may transition otherwise.

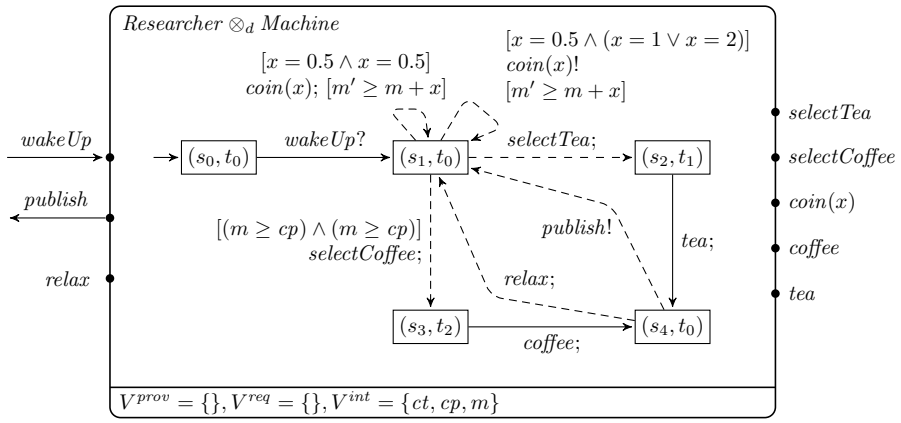


Figure 6: Abstract system specifications composed: *Researcher* $\otimes_d$ *Machine*.

Definition 14 (Composition of MIODs). The *composition* of two composable MIODs S and T is defined by the MIOD

$$S \otimes_d T = (\Sigma_S \otimes \Sigma_T, St_S \times St_T, (init_S, init_T), \varphi_S^0 \wedge \varphi_T^0, \Delta^{\mathsf{may}}_{S\otimes_d T}, \Delta^{\mathsf{must}}_{S\otimes_d T})$$

where the transition relations $\Delta^{\mathsf{may}}_{S\otimes_d T}$ and $\Delta^{\mathsf{must}}_{S\otimes_d T}$ are generated by the following rules:

$$\frac{(s, [\varphi_S]op![\pi_S], s') \in \Delta_S^\gamma,\ (t, [\varphi_T]op?[\pi_T], t') \in \Delta_T^\gamma}{((s,t), [\varphi_S \wedge \varphi_T]op; [\pi_T], (s', t')) \in \Delta^\gamma_{S\otimes_d T}} \quad op \in O_S^{req} \cap O_T^{prov}, \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

$$\frac{(s, [\varphi_S]op?[\pi_S], s') \in \Delta_S^\gamma,\ (t, [\varphi_T]op![\pi_T], t') \in \Delta_T^\gamma}{((s,t), [\varphi_S \wedge \varphi_T]op; [\pi_S], (s', t')) \in \Delta^\gamma_{S\otimes_d T}} \quad op \in O_S^{prov} \cap O_T^{req}, \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

$$\frac{(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^\gamma, \quad t \in St_T}{((s,t), [\varphi_S]op[\pi_S], (s', t)) \in \Delta^\gamma_{S\otimes_d T}} \quad op \notin O_S \cap O_T, \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

$$\frac{(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^\gamma, \quad s \in St_S}{((s,t), [\varphi_T]op[\pi_T], (s, t')) \in \Delta^\gamma_{S\otimes_d T}} \quad op \notin O_S \cap O_T, \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

The composition is well-defined for composable MIODs; i.e. it respects the conditions on the state variables of labels.

Example 5. Fig. 6 shows the composition

Researcher $\otimes_d$ *Machine*

of the two abstract specifications *Researcher* and *Machine* (see also Fig. 4). Preconditions of synchronized transitions are conjoined whereas the new postcondition is the postcondition of the synchronized input transition. Shared variables and shared operations become internal in the composition. Note also that, concerning the modalities, a transition in the composition labeled with a shared operation is only a must transition if both synchronized input and output transitions were must transitions.

The next theorem shows that modal refinement is a precongruence with respect to the composition of MIODs which provides our first compositionality result.

Theorem 6. *Let* S, S', T, T' *be MIODs and let* S *and* T *be composable. Then* $S' \leq_{md} S$ *and* $T' \leq_{md} T$ *imply* $S' \otimes_d T' \leq_{md} S \otimes_d T$.

PROOF. Composability of S and T implies that $S \otimes_d T$ is defined. Since modal refinement does not change I/O-signatures, S' and T' are again composable and hence $S' \otimes_d T'$ is defined, too. We define a relation $R \subseteq (St_{S'} \times St_{T'}) \times (St_S \times St_T)$ by

$$R = \{((s', t'), (s, t)) \mid s' \leq_{md} s \text{ and } t' \leq_{md} t\}.$$

We show that R is a modal refinement between the states of $S' \otimes_d T'$ and $S \otimes_d T$.

Let $((s', t'), (s, t)) \in R$, so we can assume that $s' \leq_{md} s$ and $t' \leq_{md} t$. Let

$$((s, t), [\varphi] op [\pi], (\dot{s}, \dot{t})) \in \Delta^{\mathsf{must}}_{S \otimes_d T} \tag{5}$$

be a must transition in $S \otimes_d T$. The only interesting case is when op is a shared operation of S and T, i.e. $op \in O_S \cap O_T$; w.l.o.g., let $op \in O_S^{req} \cap O_T^{prov}$. From (5) and the rules of composition it follows that there exists

$$(s, [\varphi_S] op! [\pi_S], \dot{s}) \in \Delta^{\mathsf{must}}_S \quad \text{and} \quad (t, [\varphi_T] op? [\pi_T], \dot{t}) \in \Delta^{\mathsf{must}}_T$$

such that $\varphi \equiv \varphi_S \wedge \varphi_T$ and $\pi \equiv \pi_T$. From $s' \leq_{md} s$ and $t' \leq_{md} t$ we can conclude that there exists $N \geq 0$ and

$$(s', [\varphi_{S',i}] op! [\pi_{S',i}], \dot{s}'_i) \in \Delta^{\mathsf{must}}_{S'}, \; 0 \leq i \leq N, \text{ such that } \vDash \varphi_S \Rightarrow \bigvee_i \varphi_{S',i}$$

and for all i, $\dot{s}'_i \leq_{md} \dot{s}$. Moreover, there exists $M \geq 0$ and

$$(t', [\varphi_{T',k}] op? [\pi_{T',k}], \dot{t}'_k) \in \Delta^{\mathsf{must}}_{T'}, \; 0 \leq k \leq M, \text{ such that } \vDash \varphi_T \Rightarrow \bigvee_k \varphi_{T',k}$$

and for all k, $\dot{t}'_k \leq_{md} \dot{t}$ and $\vDash \varphi_T \wedge \varphi_{T',k} \wedge \pi_{T',k} \Rightarrow \pi_T$.

Then, for each i and k, we have

$$((s', t'), [\varphi_{S',i} \wedge \varphi_{T',k}] op; [\pi_{T',k}], (\dot{s}'_i, \dot{t}'_k)) \in \Delta^{\mathsf{must}}_{S' \otimes_d T'}$$

and we know (since $\varphi \equiv \varphi_S \wedge \varphi_T$) that $\vDash \varphi \Rightarrow (\bigvee_i \varphi_{S',i}) \wedge (\bigvee_k \varphi_{T',k})$ which implies

$$\vDash \varphi \Rightarrow \bigvee_{i,k} (\varphi_{S',i} \wedge \varphi_{T',k}).$$

And for all i and k, it is satisfied that $\vDash \varphi \wedge (\varphi_{S',i} \wedge \varphi_{T',k}) \wedge \pi_{T',k} \Rightarrow \pi$ and $((\dot{s}'_i, \dot{t}'_k), (\dot{s}, \dot{t})) \in R$.

The other direction of modal refinement (condition 2 of Def. 13, from concrete to abstract) is very similar to the proof above.

Finally, $S' \otimes_d T' \leq_{md} S \otimes_d T$ is satisfied: R is a modal refinement between the states of $S' \otimes_d T'$ and $S \otimes_d T$, and

- $((init_{S'}, init_{T'}), (init_S, init_T)) \in R$ since $S' \leq_{md} S$ implies $init_{S'} \leq_{md} init_S$ and $T' \leq_{md} T$ implies $init_{T'} \leq_{md} init_T$;
- $\vDash \varphi^0_{S'} \wedge \varphi^0_{T'} \Rightarrow \varphi^0_S \wedge \varphi^0_T$ since $S' \leq_{md} S$ implies $\varphi^0_{S'} \Rightarrow \varphi^0_S$ and $T' \leq_{md} T$ implies $\varphi^0_{T'} \Rightarrow \varphi^0_T$.

□

Example 6. Fig. 7 shows the composition *RefinedResearcher* $\otimes_d$ *RefinedMachine* of the refined system specifications *RefinedResearcher* and *RefinedMachine* (see also Fig. 5 for their individual specifications). Thanks to Theorem 6 we can infer

$$(RefinedResearcher \otimes_d RefinedMachine) \leq_{md} (Researcher \otimes_d Machine)$$

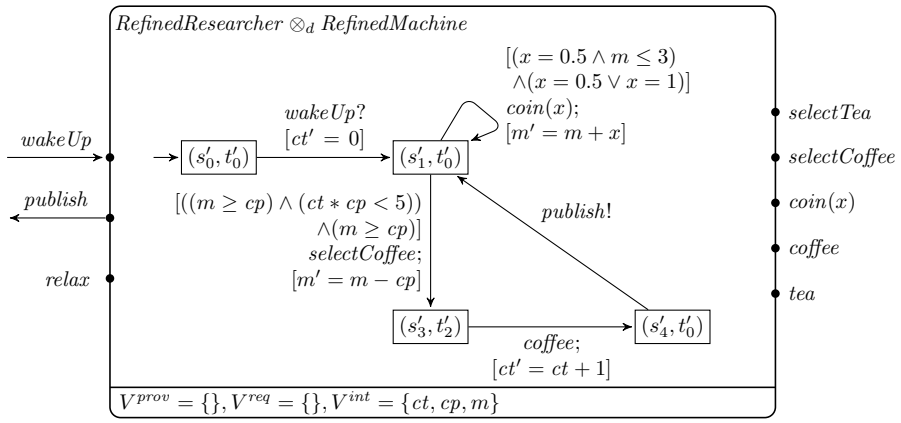


Figure 7: Refined system specifications composed: *RefinedResearcher* $\otimes_d$ *RefinedMachine*.

just by verifying *RefinedResearcher* $\leq_{md}$ *Researcher* and *RefinedMachine* $\leq_{md}$ *Machine*. This property is fundamental in component-based design: Once we have proven that the composed abstract specifications refines some other MIOD (which expresses some desired property of the composed system), this property will be satisfied in any composition of refined specifications; according to Theorem 6 the refinement relation need to be established for the individual components only. In our example, such a property could be that coffee is only requested when there is enough money in the machine; this can be easily expressed by a MIOD. Another simple (data-independent) property could be that a publication is only possible after drinking either tea or coffee. ■

Next we define a semantic composition operator for implementation models. Given two composable GIOs I and J, their composition $I \otimes_{\mathcal{G}} J$ synchronizes transitions whose labels refer to shared operations: a transition with label $[\nu_I](op, \rho)!$ of I is synchronized with a transition with label $[\nu_J](op, \rho)?$ of J if the current data state of J matches ν. More precisely, if σ_J is the source data state in J, matching means that $\nu(x) = \sigma_J(x)$ for all $x \in dom(\nu)$. If ν does not match σ_J no output should be issued and the output transition is dropped. All other transitions are interleaved in the composition.

Definition 15 (Composition of GIOs). Let I and J be two composable GIOs.The *composition* of I and J is defined by the GIO

$$I \otimes_{\mathcal{G}} J = (\Sigma_I \otimes \Sigma_J, (C_I \times C_J) \times \mathcal{D}((V_I \otimes V_J)^{loc}), ((c_I^0, c_J^0), \sigma_I^0 \cdot \sigma_J^0), \Delta_{I \otimes_{\mathcal{G}} J})$$

where the transition relation $\Delta_{I \otimes_{\mathcal{G}} J}$ is defined by

$$\frac{((c_I, \sigma_I), [\nu_I](op, \rho), (c_I', \sigma_I')) \in \Delta_I \quad ((c_J, \sigma_J), [\nu_J](op, \rho), (c_J', \sigma_J')) \in \Delta_J \quad (\sigma_I \cdot \nu_I)|_{(V_I \cap V_J)} = (\sigma_J \cdot \nu_J)|_{(V_I \cap V_J)}}{(((c_I, c_J), \sigma_I \cdot \sigma_J), [(\nu_I \cdot \nu_J)|_{(V_I \otimes V_J)^{req}}](op, \rho), ((c_I', c_J'), \sigma_I' \cdot \sigma_J')) \in \Delta_{I \otimes_{\mathcal{G}} J}} \quad op \in O_I \cap O_J$$

$$\frac{((c_I, \sigma_I), [\nu_I](op, \rho), (c_I', \sigma_I')) \in \Delta_I \quad (c_J, \sigma_J) \in Q_J \quad \nu_J \in \mathcal{D}(V_J^{req}) \quad (\sigma_I \cdot \nu_I)|_{(V_I \cap V_J)} = (\sigma_J \cdot \nu_J)|_{(V_I \cap V_J)}}{(((c_I, c_J), \sigma_I \cdot \sigma_J), [(\nu_I \cdot \nu_J)|_{(V_I \otimes V_J)^{req}}](op, \rho), ((c_I', c_J), \sigma_I' \cdot \sigma_J)) \in \Delta_{I \otimes_{\mathcal{G}} J}} \quad op \notin O_I \cap O_J$$

$$\frac{((c_J, \sigma_J), [\nu_J](op, \rho), (c_J', \sigma_J')) \in \Delta_J \quad (c_I, \sigma_I) \in Q_I \quad \nu_I \in \mathcal{D}(V_I^{req}) \quad (\sigma_I \cdot \nu_I)|_{(V_I \cap V_J)} = (\sigma_J \cdot \nu_J)|_{(V_I \cap V_J)}}{(((c_I, c_J), \sigma_I \cdot \sigma_J), [(\nu_I \cdot \nu_J)|_{(V_I \otimes V_J)^{req}}](op, \rho), ((c_I, c_J'), \sigma_I \cdot \sigma_J')) \in \Delta_{I \otimes_{\mathcal{G}} J}} \quad op \notin O_I \cap O_J$$

The next result shows that our framework supports independent implementability of composable MIODs and therefore substitutability of correct implementations.

Theorem 7. *Let S and T be two composable MIODs. If $I \in [\![S]\!]$ and $J \in [\![T]\!]$ then $I \otimes_{\mathcal{G}} J \in [\![S \otimes_d T]\!]$.*

PROOF. We define a relation $R \subseteq Q_{I\otimes_{\mathcal{G}} J} \times St_{S\otimes_d T}$ by

$$R = \{(((c_I, c_J), \sigma_I \cdot \sigma_J), (s, t)) \mid (c_I, \sigma_I) \lhd s \text{ and } (c_J, \sigma_J) \lhd t\}.$$

We show that R is an implementation relation between the states of $I \otimes_{\mathcal{G}} J$ and $S \otimes_d T$. Let $(((c_I, c_J), \sigma_I \cdot \sigma_J), (s, t)) \in R$ so we can assume that $(c_I, \sigma_I) \lhd s$ and $(c_J, \sigma_J) \lhd t$ are satisfied.

Condition 1 of Def. 12: Assume $((s, t), [\varphi]op[\pi], (s', t')) \in \Delta^{\mathsf{must}}_{S\otimes_d T}$, and let $\nu \in \mathcal{D}((V_S \otimes V_T)^{req})$ and $\rho \in Val(par(op))$ such that

$$(\sigma_I \cdot \sigma_J \cdot \nu; \rho) \vDash \varphi. \tag{6}$$

If $op \notin O_S \cap O_T$ then, w.l.o.g., it is a transition originating from a transition $(s, [\varphi]op[\pi], s') \in \Delta^{\mathsf{must}}_S$ and $t = t'$. Then, from (6) it follows that $(\sigma_I \cdot \sigma_J|_{(V_J^{prov} \cap V_I^{req})} \cdot \nu|_{V_I^{req}}; \rho) \vDash \varphi$. From $(c_I, \sigma_I) \lhd s$ it follows that there exists

$$((c_I, \sigma_I), [\sigma_J|_{(V_J^{prov} \cap V_I^{req})} \cdot \nu|_{V_I^{req}}](op, \rho), (c'_I, \sigma'_I)) \in \Delta_I$$

such that $(c'_I, \sigma'_I) \lhd s'$, and

$$\text{if } op \in O_S^{prov} \uplus O_S^{int} \text{ then } (\sigma_I \cdot \sigma_J|_{(V_J^{prov} \cap V_I^{req})} \cdot \nu|_{V_I^{req}}, \sigma'_I; \rho) \vDash \pi. \tag{7}$$

Then we have

$$(((c_I, c_J), \sigma_I \cdot \sigma_J), [\nu](op, \rho), ((c'_I, c_J), \sigma'_I \cdot \sigma_J)) \in \Delta_{I\otimes_{\mathcal{G}} J}$$

such that $(((c'_I, c_J), \sigma'_I \cdot \sigma_J), (s', t)) \in R$, and if $op \in O_S^{prov} \uplus O_S^{int}$ then, by (7), $(\sigma_I \cdot \sigma_J \cdot \nu, \sigma'_I \cdot \sigma_J; \rho) \vDash \pi$.

Now assume that $op \in O_S \cap O_T$. Then the transition $((s, t), [\varphi]op[\pi], (s', t')) \in \Delta^{\mathsf{must}}_{S\otimes_d T}$ must come from a synchronization of (w.l.o.g. $op \in O_S^{req} \cap O_T^{prov}$)

$$(s, [\varphi_S]op![\pi_S], s') \in \Delta^{\mathsf{must}}_S \quad \text{and} \quad (t, [\varphi_T]op?[\pi_T], t') \in \Delta^{\mathsf{must}}_T$$

such that $\varphi \equiv \varphi_S \wedge \varphi_T$ and $\pi \equiv \pi_T$. Hence, by (6), we get that

$$(\sigma_I \cdot \sigma_J|_{(V_J^{prov} \cap V_I^{req})} \cdot \nu|_{V_I^{req}}; \rho) \vDash \varphi_S \quad \text{and} \quad (\sigma_J \cdot \sigma_I|_{(V_I^{prov} \cap V_J^{req})} \cdot \nu|_{V_J^{req}}; \rho) \vDash \varphi_T.$$

From $(c_I, \sigma_I) \lhd s$ and $(c_J, \sigma_J) \lhd t$ it follows that there exists

$$((c_I, \sigma_I), [\sigma_J|_{(V_J^{prov} \cap V_I^{req})} \cdot \nu|_{V_I^{req}}](op, \rho), (c'_I, \sigma'_I)) \in \Delta_I$$

and

$$((c_J, \sigma_J), [\sigma_I|_{(V_I^{prov} \cap V_J^{req})} \cdot \nu|_{V_J^{req}}](op, \rho), (c'_J, \sigma'_J)) \in \Delta_J$$

such that $(c'_I, \sigma'_I) \lhd s'$, $(c'_J, \sigma'_J) \lhd t'$ and $(\sigma_J \cdot \sigma_I|_{(V_I^{prov} \cap V_J^{req})} \cdot \nu|_{V_J^{req}}, \sigma'_J; \rho) \vDash \pi_T$. Then there exists

$$(((c_I, c_J), \sigma_I \cdot \sigma_J), [\nu](op, \rho), ((c'_I, c'_J), \sigma'_I \cdot \sigma'_J)) \in \Delta_{I\otimes_{\mathcal{G}} J}$$

such that $(\sigma_I \cdot \sigma_J \cdot \nu, \sigma'_I \cdot \sigma'_J; \rho) \vDash \pi$ and $(((c'_I, c'_J), \sigma'_I \cdot \sigma'_J), (s', t')) \in R$.

The second condition of Def. 12 follows the same schema and can be proven analogously.

Thus we have shown that R is an implementation relation between the states of $I \otimes_{\mathcal{G}} J$ and $S \otimes_d T$. From $I \in [\![S]\!]$ and $J \in [\![T]\!]$ it follows that $(c_I^0, \sigma_I^0) \lhd init_S$, $\sigma_I^0 \vDash \varphi_S^0$, $(c_J^0, \sigma_J^0) \lhd init_T$ and $\sigma_J^0 \vDash \varphi_T^0$. We can infer that $(((c_I^0, c_J^0), \sigma_I^0 \cdot \sigma_J^0), (init_S, init_T)) \in R$ and $\sigma_I^0 \cdot \sigma_J^0 \vDash \varphi_S^0 \wedge \varphi_T^0$ which finishes the proof. □

8. Compatibility

When we want to compose two MIODs we have seen that it is first necessary to check composability which is a purely syntactic condition. But then it is of course important that the two components work properly together, i.e. are behaviorally compatible. The following compatibility notion builds upon (strong) modal compatibility as defined in [4] and reconsidered in Sect. 3. From the control point of view (strong) compatibility requires that in any reachable state of the product $S \otimes_d T$ of two MIODs S and T, if one MIOD *may* issue an output (in its current control state) then the other MIOD is in a control state where it *must* be able to take the corresponding input.[2] In the context of data states we have the additional requirement that the data constraints of the two MIODs S and T must be compatible. Since the data constraints imposed by a MIOD can be considered as a contract, the two contracts according to S and T must match. This means that if a shared operation may be sent out under a certain precondition, the sender assumes that the communication partner *must* be enabled to take the operation call in its current state. Conversely, the receiver assumes that its operation *may* only be called in a state where the precondition of the receiver is valid. Moreover, the sender assumes that its expected postcondition is fulfilled after the operation execution which must be guaranteed by the receiver. These considerations suggest condition 1(a) in Def. 16, where S plays the role of the sender and T plays the role of the receiver. Here the condition is again relaxed to take into account a possible splitting of transitions on the side of the receiver which allows to express a case distinction for accepting inputs. Condition 1(b) additionally requires that also any other possible reception of the input leads to a state where the expected postcondition is satisfied.

For practical verification of compatibility of MIODs, we go through all syntactically reachable states of $S \otimes_d T$ and check whether the pre- and postconditions of synchronizing transitions match. The set of the syntactically reachable states of S is given by $\mathcal{R}(S) = \bigcup_{n=0}^{\infty} \mathcal{R}_n$ where $\mathcal{R}_0(S) = \{init_S\}$ and $\mathcal{R}_{n+1}(S) = \{s' \mid s \in \mathcal{R}_n(S), (s, \ell, s') \in \Delta_S^{\mathsf{may}}\}$. Note that taking the syntactically reachable states is, of course, an over-approximation of the (semantically) reachable states in the composition of implementation models. Hence non compatible MIODs may still admit compatible implementations but not the other way round as shown in Theorem 11 below.

Definition 16 (Compatibility of MIODs [2]). Let S and T be two composable MIODs. S and T are *compatible*, denoted by $S \leftrightarrows_d T$, iff for all reachable states $(s,t) \in \mathcal{R}(S \otimes_d T)$,

1. for all $op \in O_S^{req} \cap O_T^{prov}$, whenever $(s, [\varphi_S]op![\pi_S], s') \in \Delta_S^{\mathsf{may}}$ and φ_S is satisfiable then
   (a) there exists $(t, [\varphi_{T,i}]op?[\pi_{T,i}], t_i') \in \Delta_T^{\mathsf{must}}$, $0 \le i \le N$, such that $\vDash \varphi_S \Rightarrow \bigvee_i \varphi_{T,i}$, and
   (b) for all $(t, [\varphi_T]op?[\pi_T], t') \in \Delta_T^{\mathsf{may}}$, it holds that $\vDash \varphi_S \wedge \varphi_T \wedge \pi_T \Rightarrow \pi_S$;
2. symmetrically for all $op \in O_T^{req} \cap O_S^{prov}$.

Condition 1(a) of Def. 16 expresses that the operation call to *op* which is issued by S under the condition that φ_S holds, must be accepted by T, hence there must exist accepting transitions in T such that the disjunction of their preconditions is at most weaker than φ_S. Condition 1(b) of Def. 16 requires that the postcondition π_S (the assumption) of the caller S is respected: for any may transition with a corresponding input label the assumption π_S is at most weaker than the guarantee π_T.

Example 7. Consider our refined system specifications shown in Fig. 5 and their composition shown in Fig. 7. Compatibility of the specifications *RefinedResearcher* and *RefinedMachine* means, for instance, that in the (syntactically) reachable state (s_1', t_0'), every call to $coin(x)$ of *RefinedResearcher* must be accepted by *RefinedMachine*. In state s_1' of *RefinedResearcher*, there is the may transition

$$(s_1', [x = 0.5 \wedge m \le 3]coin(x)![m' \ge m + x], s_1') \in \Delta^{\mathsf{may}}_{RefinedResearcher}$$

[2] We still follow the "pessimistic" approach to compatibility as discussed in Sect. 3.

We can find a must transition in *RefinedMachine*,

$$(t_0', [x = 0.5 \vee x = 1]coin(x)?[m' = m + x], t_0') \in \Delta^{\mathsf{must}}_{RefinedMachine}$$

and we have to check the conditions on the predicates. But both

$$\vDash (x = 0.5 \wedge m \leq 3) \Rightarrow (x = 0.5 \vee x = 1)$$

and

$$\vDash (x = 0.5 \wedge m \leq 3) \wedge (x = 0.5 \vee x = 1) \wedge (m' = m + x) \Rightarrow (m' \geq m + x)$$

are satisfied. The other transitions with shared operations can be checked analogously. Thus

$$RefinedResearcher \leftrightarrows_d RefinedMachine$$

and one can also verify that the abstract specifications are compatible as well, i.e. *Researcher* $\leftrightarrows_d$ *Machine*.

Lemma 8. *Let S, S', T be MIODs such that S and T are composable. If $S' \leq_{md} S$ then for each reachable state $(s', t) \in \mathcal{R}(S' \otimes_d T)$ there exists a state $s \in St_S$ such that $(s, t) \in \mathcal{R}(S \otimes_d T)$ and $s' \leq_{md} s$.*

Proof. Reachability of (s', t) in $S' \otimes_d T$ implies that there exist transitions

$$((s_0', t_0), \ell_0', (s_1', t_1)), ((s_1', t_1), \ell_1', (s_2', t_2)), \ldots, ((s_{n-1}', t_{n-1}), \ell_{n-1}', (s_n', t_n)) \in \Delta^{\mathsf{may}}_{S' \otimes_d T},\ n \geq 0,$$

such that $s_0' = init_{S'}$, $t_0 = init_T$, $s_n' = s'$ and $t_n = t$. Then, by the rules of composition, there exist transitions

$$(s_0', k_0', s_1'), (s_1', k_1', s_2'), \ldots, (s_{n-1}', k_{n-1}', s_n') \in \Delta^{\mathsf{may}}_{S'}$$

such that, for all $0 \leq i \leq n$, ℓ_i' and k_i' involve the same operations. From our assumption $S' \leq_{md} S$ it follows that $init_{S'} \leq_{md} init_S$. By induction on the length $n \geq 0$, and there exist transitions

$$(s_0, k_0, s_1), (s_1, k_1, s_2), \ldots, (s_{n-1}, k_{n-1}, s_n) \in \Delta^{\mathsf{may}}_{S}$$

such that $s_0 = init_S$, and for all $0 \leq i \leq n$, $s_i' \leq_{md} s_i$ and k_i and k_i' involve the same operations. It follows that there exist transitions

$$((s_0, t_0), \ell_0, (s_1, t_1)), ((s_1, t_1), \ell_1, (s_2, t_2)), \ldots, ((s_{n-1}, t_{n-1}), \ell_{n-1}, (s_n, t_n)) \in \Delta^{\mathsf{may}}_{S \otimes_d T}.$$

Hence $s_n \in St_S$ demonstrates that there exists $s \in St_S$ such that (s, t) is reachable in $S \otimes_d T$ and $s' \leq_{md} s$. □

Compatibility of MIODs is preserved by refinement:

Theorem 9. *Let $S, S', T, T' \in \mathcal{M}_d$ be MIODs such that S and T are composable. Then $S \leftrightarrows_d T$, $S' \leq_{md} S$ and $T' \leq_{md} T$ imply $S' \leftrightarrows_d T'$.*

Proof. Obviously, it suffices to prove that $S \leftrightarrows_d T$ and $S' \leq_{md} S$ imply $S' \leftrightarrows_d T$.

Let $(s', t) \in \mathcal{R}(S' \otimes_d T)$ be a reachable state in $S' \otimes_d T$, and assume that there exists a transition $(s', [\varphi]op![\pi]S', \dot{s}') \in \Delta^{\mathsf{may}}_{S'}$ with $op \in O^{req}_{S'} \cap O^{prov}_{T'}$. By Lemma 8, there exists a state $s \in St_S$ $(s, t) \in \mathcal{R}(S \otimes_d T)$ and $s' \leq_{md} s$. From $s' \leq_{md} s$ it follows that there exists $N \geq 0$ and transitions $(s, [\varphi_{S,i}]op![\pi_{S,i}], \dot{s}_i) \in \Delta^{\mathsf{may}}_{S}$, $0 \leq i \leq N$, such that $\vDash \varphi_{S'} \Rightarrow \bigvee_i \varphi_{S,i}$ and for all i, $\vDash \varphi_{S'} \wedge \varphi_{S,i} \wedge \pi_{S,i} \Rightarrow \pi_{S'}$. We already know $(s, t) \in \mathcal{R}(S \otimes_d T)$, hence by compatibility of S and T, for each i, there exists $M_i \geq 0$ and transitions $(t, [\varphi_{T,k}]op?[\pi_{T,k}], \dot{t}_k) \in \Delta^{\mathsf{must}}_{T}$, $0 \leq k \leq M_i$, such that $\vDash \varphi_{S,i} \Rightarrow \bigvee_k \varphi_{T,k}$ and for all $(t, [\varphi_T]op?[\pi_T], \dot{t}) \in \Delta^{\mathsf{may}}_{T}$, $\vDash \varphi_{S,i} \wedge \varphi_T \wedge \pi_T \Rightarrow \pi_{S,i}$. We can conclude that

$$\vDash \varphi_{S'} \Rightarrow \bigvee_{0 \leq i \leq N} \bigvee_{0 \leq k \leq M_i} \varphi_{T,k}.$$

Moreover, for any $(t, [\varphi_T] op?[\pi_T], \dot{t}) \in \Delta_T^{\mathsf{may}}$, we already know that

$$\begin{aligned}
&\vDash \varphi_{S'} \Rightarrow \bigvee_i \varphi_{S,i} \\
&\vDash \varphi_{S'} \wedge \varphi_{S,i} \wedge \pi_{S,i} \Rightarrow \pi_{S'} \quad \text{for all } i \\
&\vDash \varphi_{S,i} \wedge \varphi_T \wedge \pi_T \Rightarrow \pi_{S,i} \quad \text{for all } i
\end{aligned}$$

and thus we can conclude that $\vDash \varphi_{S'} \wedge \varphi_T \wedge \pi_T \Rightarrow \pi_{S'}$ is satisfied.

The second part can be proven in a similar way. □

Example 8. This result allows us to infer compatibility of the refined specifications *RefinedResearcher* and *RefinedMachine* (see Fig. 5) by just proving (1) compatibility of the abstract specifications (see Fig. 4), and (2), that there is a refinement relation between each abstract specification and its refinement.

We now define a semantic compatibility notion. Compatibility between GIOs requires that in any reachable state of the product $S \otimes_{\mathcal{G}} T$ of two GIOs I and J, if one GIO wants to issue an output (in its current control state) with guard ν, and ν matches with the data state of the receiving GIO, then the receiving GIO is in a state where it is able to take the corresponding input.

Definition 17 (Compatibility of GIOs). Let I and J be two composable GIOs. I and J are *compatible*, denoted by $I \leftrightarrows_{\mathcal{G}} J$, iff for all reachable states $((c_I, c_J), \sigma_I \cdot \sigma_J) \in \mathcal{R}(I \otimes_{\mathcal{G}} J)$,

1. for all $op \in O_I^{req} \cap O_J^{prov}$ and all $\nu_J \in \mathcal{D}(V_J^{req})$, if there is a transition $((c_I, \sigma_I), [\nu_I](op, \rho)!, (c_I', \sigma_I')) \in \Delta_I$ and $(\sigma_I \cdot \nu_I)|_{(V_I \cap V_J)} = (\sigma_J \cdot \nu_J)|_{(V_I \cap V_J)}$ then there exists $((c_J, \sigma_J), [\nu_J](op, \rho)?, (c_J', \sigma_J')) \in \Delta_J$;
2. symmetrically for all $op \in O_J^{req} \cap O_I^{prov}$.

For showing preservation of compatibility by the implementation relation we first have to prove that any reachable state in a correct implementation is related to a state in its specification.

Lemma 10. *Let S and T be MIODs such that S and T are composable. Let I and J be GIOs such that $I \in [\![S]\!]$ and $J \in [\![T]\!]$. Then for each reachable state $((c_I, c_J), \sigma_I \cdot \sigma_J) \in \mathcal{R}(I \otimes_{\mathcal{G}} J)$ there exist states $s \in St_S$ and $t \in St_T$ such that $(c_I, \sigma_I) \triangleleft s$ and $(c_J, \sigma_J) \triangleleft t$.*

Theorem 11. *Let S and T be composable MIODs, and let $I \in [\![S]\!]$ and $J \in [\![T]\!]$ be GIOs. Then $S \leftrightarrows_d T$ implies $I \leftrightarrows_{\mathcal{G}} J$.*

Proof. Let $((c_I, c_J), \sigma_I \cdot \sigma_J)$ be a reachable state in $I \otimes_{\mathcal{G}} J$. By Lemma 10 there exist states $s \in St_S$ and $t \in St_T$ such that $(c_I, \sigma_I) \triangleleft s$ and $(c_J, \sigma_J) \triangleleft t$.

Assume that there is a state $\nu_J \in \mathcal{D}(V_J^{req})$ and a transition

$$((c_I, \sigma_I), [\nu_I](op, \rho)!, (c_I', \sigma_I')) \in \Delta_I$$

such that $(\sigma_I \cdot \nu_I)|_{(V_I \cap V_J)} = (\sigma_J \cdot \nu_J)|_{(V_I \cap V_J)}$ and $op \in O_S^{req} \cap O_T^{prov}$. From $(c_I, \sigma_I) \triangleleft s$ it follows that there exists a transition $(s, [\varphi_S] op![\pi_S], s') \in \Delta_S^{\mathsf{may}}$ such that $(\sigma_I \cdot \nu_I; \rho) \vDash \varphi_S$ and $(c_I', \sigma_I') \triangleleft s'$. Then, by compatibility $S \leftrightarrows_d T$, there exists $N \geq 0$ and transitions $(t, [\varphi_{T,i}] op?[\pi_{T,i}], t_i') \in \Delta_T^{\mathsf{must}}$, $0 \leq i \leq N$, such that $\vDash \varphi_S \Rightarrow \bigvee_i \varphi_{T,i}$. We already know that $(\sigma_I \cdot \nu_I; \rho) \vDash \varphi_S$, then also $(\sigma_I \cdot \sigma_J \cdot (\nu_I \cdot \nu_J)|_{(V_I \otimes V_J)^{req}}; \rho) \vDash \varphi_S$. Hence there is some j, $0 \leq j \leq N$, such that $(\sigma_I \cdot \sigma_J \cdot (\nu_I \cdot \nu_J)|_{(V_I \otimes V_J)^{req}}; \rho) \vDash \varphi_{T,j}$. From $(c_J, \sigma_J) \triangleleft t$ it follows that there exists a transition

$$((c_J, \sigma_J), [\nu_J](op, \rho)?, (c_J', \sigma_J')) \in \Delta_J.$$

This was to be shown. The other direction is symmetric. □

9. Relating Interface Theories

In the following we collect the results of the previous sections and define interface theories and their relations. The first result is that our framework of MIODs is compositional; hence MIODs together with composition, refinement, and compatibility form an interface theory.

Corollary 12. *$(\mathcal{M}_d, \otimes_d, \leq_{md}, \leftrightarrows_d)$ is an interface theory.*

Proof. Compatibility trivially implies composability: $S \leftrightarrows_d T$ implies that S and T are composable, hence $S \otimes_d T$ is defined. Compositional refinement has been proven in Theorem 6 and preservation of compatibility has been proven in Theorem 9. □

Next, the class of all GIOs together with set inclusion as refinement relation, and pointwise composition and compatibility, as defined in Sect. 7 and 8, form an interface theory.

Theorem 13. *$(\mathscr{P}(\mathcal{G}), \widehat{\otimes}_{\mathcal{G}}, \subseteq, \widehat{\leftrightarrows}_{\mathcal{G}})$ is an interface theory.*

Proof. We have to show all three requirements an interface theory has to satisfy. The first condition, compatibility implies composability, it satisfied: $M \widehat{\leftrightarrows}_{\mathcal{G}} N$, for $M, N \in \mathscr{P}(\mathcal{G})$, implies that every $I \in M$, $J \in N$ are composable, hence $M \widehat{\otimes}_{\mathcal{G}} N$ is defined. The second and third conditions are trivially satisfied since refinement is set inclusion. □

Now we relate the introduced interface theories and establish (weak) interface theory morphisms between them. The interface theory $\mathcal{I}_{\mathcal{M}_d}$ with MIODs as specification domain can be related to their formal semantics by a weak interface morphism to $\mathcal{I}_{\mathscr{P}(\mathcal{G})}$, mapping any MIOD S to the class $[\![S]\!]$ of all correct implementations of S.

Corollary 14. *The mapping*

$$\begin{array}{rccc} j: & \mathcal{M}_d & \to & \mathscr{P}(\mathcal{G}) \\ & S & \mapsto & [\![S]\!] \end{array}$$

is a weak interface theory morphism from $\mathcal{I}_{\mathcal{M}_d}$ to $\mathcal{I}_{\mathscr{P}(\mathcal{G})}$.

Proof. This follows from Proposition 5, Theorem 7 and Theorem 11.

To give an overview of the introduced interface theories we study their correspondences by defining interface theory morphisms between them. First, for the embedding of MIOs into MIODs, we have to define how a set of actions (partitioned into input, output and internal actions) are mapped to operations. Given such a set of actions $Act = Act^{in} \uplus Act^{out} \uplus Act^{int}$, the I/O-signature for Act, denoted by $\Sigma(Act)$, is $(\emptyset, Act)$ which is defined by $O^{prov} = Act^{in}$, $O^{req} = Act^{out}$, $O^{int} = Act^{int}$, and $par(a) = \emptyset$ for each $a \in Act$.

We define an interface theory morphism f for the mapping of MIOs to MIODs. f is defined as follows: $f : \mathcal{M} \longrightarrow \mathcal{M}_d$ maps any $S \in \mathcal{M}$ to $f(S) \in \mathcal{M}_d$ such that

$$f(S) = (\Sigma_{f(S)}, St_{f(S)}, init_{f(S)}, \varphi^0_{f(S)}, \Delta^{\mathsf{may}}_{f(S)}, \Delta^{\mathsf{must}}_{f(S)})$$

where $\Sigma_{f(S)} = \Sigma(Act_S)$, $St_{f(S)} = St_S$, $init_{f(S)} = init_S$, $\varphi^0_{f(S)} = true$, and

$$\Delta^{\gamma}_{f(S)} = \{(s, [true]a[true], s') \mid (s, a, s') \in \Delta^{\gamma}_S\}, \text{ for } \gamma \in \{\mathsf{may}, \mathsf{must}\},$$

where *true* is the universally valid state predicate (transition predicate, resp.) over $\mathcal{S}(\emptyset, \emptyset)$ ($\mathcal{T}(\emptyset, \emptyset, \emptyset)$, resp.).

Lemma 15. *f is an interface theory morphism from $\mathcal{I}_{\mathcal{M}}$ to $\mathcal{I}_{\mathcal{M}_d}$.*

Next, we define an interface theory morphism g from $\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})}$ to $\mathcal{I}_{\mathscr{P}(\mathcal{G})}$. Let ε denote the function with empty domain. The function $g : \mathscr{P}(\mathcal{M}^{\mathsf{must}}) \longrightarrow \mathscr{P}(\mathcal{G})$ is defined by $g(M) = \{I_\varepsilon \mid I \in M\}$ where $I_\varepsilon = (\Sigma(Act_I), St_I \times \mathcal{D}(\emptyset), (init_I, \varepsilon), \Delta_{I_\varepsilon})$, and

$$\Delta_{I_\varepsilon} = \{((s,\varepsilon),[\varepsilon](a,\varepsilon),(s',\varepsilon)) \mid (s,a,s') \in \Delta_I\}.$$

Lemma 16. *g is an interface theory morphism from $\mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})}$ to $\mathcal{I}_{\mathscr{P}(\mathcal{G})}$.*

Finally, we get the diagram in Fig. 8. The interface theory morphism i has been introduced in Sect. 3 (see Theorem 4) and maps any MIO S to its implementation semantics $[\![S]\!]$.

$$\begin{array}{ccc} \mathcal{I}_{\mathcal{M}} & \xrightarrow{\ f\ } & \mathcal{I}_{\mathcal{M}_d} \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle j} \\ \mathcal{I}_{\mathscr{P}(\mathcal{M}^{\mathsf{must}})} & \xrightarrow[\ g\ ]{} & \mathcal{I}_{\mathscr{P}(\mathcal{G})} \end{array}$$

Figure 8: Relating interface theories by interface theory morphisms.

Theorem 17. *The diagram in Fig. 8 commutes, i.e. $j \circ f = g \circ i$.*

Proof. For proving $j \circ f = g \circ i$ one has to show that, given a MIO S, modal refinement of S (and adding empty data states according to g) coincides with the semantics of $[\![f(S)]\!]$ consisting of all implementations of $f(S)$. This is, however, easy to prove.

10. Conclusion

We have proposed a formalism for the specification and implementation of interfaces for interacting, concurrent components which integrates the aspects of control flow and evolving data states. Specifications are represented by modal I/O-transition systems with data constraints (MIODs), implementations are formalized in terms of guarded input/output transition systems (GIOs) which involve concrete data states. We have studied refinement and compatibility of specifications by taking into account a contract-oriented view and we have shown that MIODs form an interface theory: compatibility is preserved by refinement and refinement is preserved by synchronous composition of MIODs. Since modal specifications are inherently loose, we have adopted a loose semantics for MIODs where any MIOD is interpreted by the class of its correct implementations. The correctness notion is defined by a particular simulation relation between MIODs and GIOs which relates not only control states but also data constraints of a specification with concrete data states of an implementation. We have shown that our semantics is compositional in the sense that locally correct implementations of compatible MIODs are compatible as well and compose to a globally correct implementation of the composed MIODs. On the specification level, our approach is independent from a particular assertion language for pre- and postconditions and, on the implementation level, it is independent from a particular programming language notation. Of course, the instantiation to appropriate subsets of concrete languages, like OCL for assertions, UML for protocols, and concurrent Java for implementations is an interesting objective of further research.

Moreover, we are interested in better symbolic approximations of the semantic refinement and compatibility notions. Concerning compatibility, for instance, we want to investigate techniques to remove reachable states of MIODs and MIOD compositions which are not related to semantically reachable states. Concerning the expressive power of MIODs it would be desirable to integrate critical regions which would allow stepwise verification of data constraints along transitions within critical parts.

Further important issues are to extend our framework by taking into account weak versions of refinement and compatibility abstracting away not only internal actions, as done for MIOs in [21, 4], but also internal state variables and the application of our theory to a particular component model.

References

[1] Tomás Barros, Rabéa Ameur-Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed fractal components. *Annales des Télécommunications*, 64(1-2):25–43, 2009.

[2] Sebastian S. Bauer, Rolf Hennicker, and Michel Bidoit. A modal interface theory with data constraints. In *Proc. SBMF 2010*, Lect. Notes Comp. Sci. Springer, 2010. To Appear.

[3] Sebastian S. Bauer, Rolf Hennicker, and Stephan Janisch. Behaviour protocols for interacting stateful components. *Electr. Notes Theor. Comput. Sci.*, 263:47–66, 2010.

[4] Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On weak modal compatibility, refinement, and the MIO Workbench. In *Proc. TACAS 2010*, volume 6015 of *Lect. Notes Comp. Sci.*, pages 175–189. Springer, 2010.

[5] Nikola Benes, Jan Kretínský, Kim Guldstrand Larsen, and Jirí Srba. On determinism in modal transition systems. *Theor. Comput. Sci.*, 410(41):4026–4043, 2009.

[6] Jan A. Bergstra, Manfred Broy, J. V. Tucker, and Martin Wirsing. On the power of algebraic specifications. In Jozef Gruska and Michal Chytil, editors, *MFCS*, volume 118 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 1981.

[7] Jan A. Bergstra and C. A. Middelburg. An interface group for process components. *Fundam. Inform.*, 99(4):355–382, 2010.

[8] Michel Bidoit, Rolf Hennicker, Alexander Knapp, and Hubert Baumeister. Glass-box and black-box views on object-oriented specifications. In *Proc. SEFM'04, Beijing, China*, pages 208–217. IEEE Comp. Society Press, 2004.

[9] Maria Victoria Cengarle, Alexander Knapp, and Heribert Mühlberger. Interactions. In Kevin Lano, editor, *UML 2 Semantics and Applications*, pages 205–248. 2009.

[10] Luca de Alfaro, Leandro Dias da Silva, Marco Faella, Axel Legay, Pritam Roy, and Maria Sorea. Sociable interfaces. In Bernhard Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 81–105. Springer, 2005.

[11] Luca de Alfaro and Thomas A. Henzinger. Interface automata. *Software Engineering Notes*, pages 109–120, 2001.

[12] Luca de Alfaro and Thomas A. Henzinger. Interface Theories for Component-Based Design. In *EMSOFT 2001*, pages 148–165, 2001.

[13] 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.

[14] Fabrício Fernandes and Jean-Claude Royer. The STSLib project: Towards a formal component model based on STS. *Electr. Notes Theor. Comput. Sci.*, 215:131–149, 2008.

[15] Clemens Fischer. CSP-OZ: a combination of Object-Z and CSP. In H. Bowman and J. Derrick, editors, *Proc. FMOODS*, pages 423–438, Canterbury, UK, 1997. Chapman and Hall, London.

[16] Joseph Goguen and Rod Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[17] D. Harel and P.S. Thiagarajan. Message sequence charts. In Luciano Lavagno, Grant Martin, and Bran Selic, editors, *UML for Real: Design of Embedded Real-time Systems*. Kluwer Academic Publishers, 2003.

[18] C.A.R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[19] Hans Hüttel and Kim Guldstrand Larsen. The Use of Static Constructs in A Modal Process Logic. In *Logic at Botik 1989*, pages 163–180, 1989.

[20] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *ESOP 2007*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.

[21] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. On Modal Refinement and Consistency. In *CONCUR 2007*, volume 4703 of *LNCS*, pages 105–119. Springer, 2007.

[22] Kim Guldstrand Larsen and Bent Thomsen. A Modal Process Logic. In *3rd Annual Symp. Logic in Computer Science, LICS 1988*, pages 203–210. IEEE Computer Society, 1988.

[23] Robin Milner. *Communication and Concurrency*. Prentice Hall (International Series in Computer Science), 1989.

[24] F. Montesi and D. Sangiorgi. A model of evolvable components. In *Proc. TGC'10*, Lect. Notes Comp. Sci. Springer, 2010. To Appear.

[25] Sebti Mouelhi, Samir Chouali, and Hassan Mountassir. Refinement of interface automata strengthened by action semantics. *Electr. Notes Theor. Comput. Sci.*, 253(1):111–126, 2009.

[26] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, and Roberto Passerone. Why Are Modalities Good for Interface Theories? In *9th Int. Conf. Application of Concurrency to System Design, ACSD 2009*, pages 119–127, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[27] Augusto Sampaio, Jim Woodcock, and Ana Cavalcanti. Refinement in Circus. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2002.

[28] Martin Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 675–788. 1990.

[29] Martin Wirsing and Jan A. Bergstra, editors. *Algebraic Methods: Theory, Tools and Applications [papers from a workshop in Passau, Germany, June 9-11, 1987]*, volume 394 of *Lecture Notes in Computer Science*. Springer, 1989.

[30] Jim Woodcock and Ana Cavalcanti. The semantics of Circus. In Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors, *ZB*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2002.