A Modal Interface Theory with Data Constraints

Sebastian S. Bauer[1], Rolf Hennicker[1], and Michel Bidoit[2]

[1] Ludwig-Maximilians-Universität München, Germany
[2] Laboratoire Spécification et Vérification,
CNRS & ENS de Cachan, France

Abstract. For the design of component-based software, the behavioral specification of component interfaces is crucial. We propose an extension of the theory of modal I/O-transition systems by Larsen et al. to cope with both control flow *and* data states of reactive components at the same time. In our framework, transitions model incoming or outgoing operation calls which are constrained by pre- and postconditions expressing the mutual assumptions and guarantees of the receiver and the sender of a message. We define a new interface theory by adapting synchronous composition, modal refinement and modal compatibility to the case of modal I/O-transition systems with data constraints. We show that in this formalism modal compatibility is preserved by refinement and modal refinement is preserved by composition which are basic requirements for any interface theory.

1 Introduction

A rigorous discipline of component-based software development relies strongly on interface specifications which describe the observable behavior of components [6]. Thereby, behavior is often understood from a control-flow oriented perspective considering the sequences of actions a component can perform when interacting with its environment. Of equal importance is, however, the aspect of changing data – owned by a component – when certain actions of the component are performed. We claim that the integrated treatment of control flow and data change in the context of concurrent, reactive components is not yet sufficiently understood and therefore needs further investigation.

Building on previous work on the semantics of behavior protocols for components with data states [3] we propose an interface theory on the basis of modal I/O-transition systems (MIOs) introduced in [9]. A particular advantage of modal transition systems is that they distinguish between "may" and "must" transitions which leads to a powerful refinement notion [11]: 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 a MIO with must-transitions only, is reached. Another aspect which can be conveniently formalized with modal I/O-transition systems 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 [4].

In this paper 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 messages 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 I/O-transition systems 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. Our main result shows that our framework satisfies the basic requirements of an interface theory: refinement is compositional and compatibility is preserved by refinement. Thus modal I/O-transition systems with data constraints support reusability and independent implementability of components via interface specifications.

Related Work. Specifications of control flow and of changing data states are often considered separately from each other. While transition systems are a popular formalism to specify the temporal ordering of messages, invariants and pre- and postconditions are commonly used to specify the effects of operations w.r.t. data states. Though approaches like CSP-OZ [8] offer means to specify both aspects, they still lack a fully integrated treatment since their expressive power is limited to cases where the effect of an operation on data states must be independent of the control-flow behavior of a component. Other related approaches are based on symbolic transition systems (STS) [7,1] 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 our work is the study of Mouelhi et al. [12] who consider an extension of the theory of interface automata [6] to data states. Their approach does, however, not take into account modalities of transitions and modal refinements. There is also no discrimination of different types of state variables (provided, required and internal) which, in our case, is a methodologically important ingredient to express the contract principle between interface specifications. In our previous work, we have proposed in [3] a formal semantics of behavior protocols based on model classes, but [3] does not consider modalities and a refinement notion between specifications. On the other hand, we have investigated in [4] various kinds of modal interface theories with refinements but without taking into account constraints on data states.

Outline. The paper is organized as follows. In Sect. 2 we introduce modal I/O-transition systems with data constraints (MIODs). Their refinement is considered in Sect. 3, and in Sect. 4 the composition and the compatibility of MIODs is defined. Moreover, we show in Sect. 4 that our framework satisfies the requirements of an interface theory concerning compositionality of refinement and preservation of compatibility by refinement. In Sect. 5 we finish with some concluding remarks.

Example 1. We use a *hexapod robot* as a running example. This insect-like, six-leg robot exhibits a non-trivial control-flow behavior with a significant impact of data states. When specifying such a complex subject, non-trivial synchronization and coordination problems arise which must be addressed in a behavioral specification. For the illustration of the formal definitions and concepts introduced hereafter, we will focus here only on the locomotion aspect of the robot's legs. We assume a simple component architecture: The locomotion of each leg is coordinated by a controller component, called *LegC*, and the leg motorization is wrapped in another component, named *Leg*. ■

2 Modal I/O-Transition Systems with Data Constraints

Modal I/O-transition systems (MIOs) have been introduced in [9,11] as a formalism to describe the behavior of reactive, concurrent components which interact via matching input and output actions. MIOs distinguish between "may" and "must" transitions where the former specify which transitions are allowed in a refinement while the latter specify which transitions are mandatory (i.e. must be preserved) in any refinement. Formally, a MIO S is given by a tuple $S = (states_S, start_S, acts_S, \Delta_S^{\mathsf{may}}, \Delta_S^{\mathsf{must}})$ of a set of states $states_S$, the initial state $start_S \in states_S$, a set of actions $acts_S = act_S^{in} \uplus act_S^{out} \uplus act_S^{int}$ being the disjoint union of sets of input, output and internal actions respectively, a may-transition relation $\Delta_S^{\mathsf{may}} \subseteq states_S \times acts_S \times states_S$, and a must-transition relation $\Delta_S^{\mathsf{must}} \subseteq \Delta_S^{\mathsf{may}}$. To deal with data constraints, we want to be able to specify pre- and postconditions for the actions on modal transitions. Hence, we must use more involved labels than simple action names.

Operations. Instead of actions, we consider *operations* which may have formal parameters. An operation op is of the form $opname(Par)$ where Par is a (possibly empty) set of formal parameters. We write $par(op)$ to refer to the formal parameters of an operation op. An *I/O-operation signature* $\mathcal{O} = \mathcal{O}^{prov} \uplus \mathcal{O}^{req} \uplus \mathcal{O}^{int}$

consists of pairwise disjoint sets $\mathcal{O}^{prov}$ of *provided* operations (for inputs), $\mathcal{O}^{req}$ of *required* operations (for outputs), and $\mathcal{O}^{int}$ 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 $op \in \mathcal{O}^{prov}$ we often write $?op$ and to indicate that $op \in \mathcal{O}^{req}$ we often write $!op$.

State variables. In order to equip operations with pre- and postconditions concerning the operations' formal parameters *and* the data states of a component, we must first provide a formal notation for data states. For this purpose we use state variables of different kinds which all belong to a 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 possible 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* $\mathcal{V} = \mathcal{V}^{prov} \uplus \mathcal{V}^{req} \uplus \mathcal{V}^{int}$ consists of pairwise disjoint sets $\mathcal{V}^{prov}$, $\mathcal{V}^{req}$, and $\mathcal{V}^{int}$ of provided, required and internal state variables, respectively.

Definition 1 (I/O-Signature). *An* I/O-signature *is a pair* $\Sigma = (\mathcal{V}, \mathcal{O})$ *consisting of an I/O-state signature* $\mathcal{V}$ *and an I/O-operation signature* $\mathcal{O}$.

Predicates on states. We assume given a global set LV of logical variables, disjoint from the state variables SV such that, for each operation op, $par(op) \subseteq \mathrm{LV}$. Moreover, we assume a set $\mathcal{S}(\mathrm{SV}, \mathrm{LV})$ of *state predicates* φ and a set $\mathcal{T}(\mathrm{SV}, \mathrm{LV})$ of *transition predicates* π with associated sets $var_{state}(\varphi) \subseteq \mathrm{SV}$ of state variables and $var_{log}(\varphi) \subseteq \mathrm{LV}$ of logical variables; analogously for π. State predicates refer to single states and transition predicates to pairs of states (pre- and poststates). For each $\mathcal{W} \subseteq \mathrm{SV}$ and $X \subseteq \mathrm{LV}$ we define

$$\mathcal{S}(\mathcal{W}, X) = \{\varphi \in \mathcal{S}(\mathrm{SV}, \mathrm{LV}) \mid var_{state}(\varphi) \subseteq \mathcal{W}, var_{log}(\varphi) \subseteq X\},$$
$$\mathcal{T}(\mathcal{W}, X) = \{\pi \in \mathcal{T}(\mathrm{SV}, \mathrm{LV}) \mid var_{state}(\pi) \subseteq \mathcal{W}, var_{log}(\pi) \subseteq X\}.$$

We require that both sets, $\mathcal{S}(\mathcal{W}, X)$ and $\mathcal{T}(\mathcal{W}, X)$, are closed under the usual logical connectives, like $\wedge, \Rightarrow$, etc. We assume that state predicates $\varphi \in \mathcal{S}(\mathcal{W}, X)$ and transition predicates $\pi \in \mathcal{T}(\mathcal{W}, X)$ are both equipped with a *satisfaction relation* $\models \varphi$ and $\models \pi$, resp., expressing universal validity of predicates w.r.t. some semantic domain of states and w.r.t. a domain of valuations for the logical variables. We will not go into further details here, but the reader may assume a predefined data universe $\mathcal{U}$ such that concrete data states are functions mapping state variables to values in $\mathcal{U}$ and, similarly, valuations are mappings from logical variables to values in $\mathcal{U}$. Then state predicates should be interpreted w.r.t. a single data state and a valuation of the logical variables, while transition predicates should be interpreted w.r.t. two data states (pre- and poststates) and a valuation of the logical variables. Universal validity $\models \varphi$ of a state predicate $\varphi \in \mathcal{S}(\mathcal{W}, X)$ then means that φ is valid for all data states, modeled by functions $\sigma : \mathcal{W} \to \mathcal{U}$, and for all valuations $\rho : X \to \mathcal{U}$ while universal validity $\models \pi$ of a transition predicate $\pi \in \mathcal{T}(\mathcal{W}, X)$ means that π is valid for any two data states and any valuation.

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. How this would work in the case of the Object Constraint Language OCL is shown in [5].

Example 2. We exemplify the use of signatures in our running example. The component *Leg* has the I/O-signature $\Sigma_{Leg} = (\mathcal{V}_{Leg}, \mathcal{O}_{Leg})$ where

$$\begin{aligned} \mathcal{V}_{Leg}^{prov} &= \{maxStep, currStep\} & \mathcal{O}_{Leg}^{prov} &= \{init(), lift(), swing(a), drop(), retract()\} \\ \mathcal{V}_{Leg}^{req} &= \{\} & \mathcal{O}_{Leg}^{req} &= \{update(a)\} \\ \mathcal{V}_{Leg}^{int} &= \{steps\} & \mathcal{O}_{Leg}^{int} &= \{\} \end{aligned}$$

Leg has as provided state variables *maxStep*, which models the maximal step size, and *currStep* for the current step size of the leg. The only internal state variable is *steps* which counts the number of steps the leg has made since initialization (provided operation $init()$). The provided operations also include operations for the different kinds of leg motion, i.e. $lift()$, $swing(a)$, $drop()$, and $retract()$; the only required operation is $update(a)$ which, after a step has made, informs the leg controller component *LegC* about the actual step size. A $(\mathcal{V}_{Leg}^{prov} \cup \mathcal{V}_{Leg}^{int})$-data state can be given, for instance, by the function $\sigma : (\mathcal{V}_{Leg}^{prov} \cup \mathcal{V}_{Leg}^{int}) \to \mathcal{U}$ with $\sigma(maxStep) = 50$, $\sigma(currStep) = 0$, $\sigma(steps) = 5$.

The controller *LegC* has the I/O-signature $\Sigma_{LegC} = (\mathcal{V}_{LegC}, \mathcal{O}_{LegC})$, where $\mathcal{V}_{LegC}^{prov} = \{distance\}$, $\mathcal{V}_{LegC}^{req} = \{maxStep, currStep\}$, and $\mathcal{V}_{LegC}^{int} = \{gone\}$. The provided state variable *distance* stores the total distance to be walked by its (connected) leg, the internal state variable *gone* models the distance that has already been moved by its leg. The meaning of the required state variables *maxStep* and *currStep* has already been explained above. The provided operations of *LegC* include $update(a)$. In the following, for the specification of the behavior of *LegC*, we will only consider transitions labeled with $swing(a) \in \mathcal{O}_{LegC}^{req}$ (which is a provided operation of the *Leg* component, hence shared with *LegC*). ■

We are now able to define which labels can occur in a modal I/O-transition system with data constraints. Given an I/O-signature $\Sigma = (\mathcal{V}, \mathcal{O})$, the set $\mathcal{L}(\Sigma)$ of Σ- labels consists of the following expressions:

1. $[\varphi]?m[\pi]$ with $m \in \mathcal{O}^{prov}$ a provided operation, $\varphi \in \mathcal{S}(\mathcal{V}^{prov}, par(m))$ the precondition, $\pi \in \mathcal{T}(\mathcal{V}^{prov} \cup \mathcal{V}^{int}, par(m))$ the postcondition,
2. $[\varphi]!n[\pi]$ with $n \in \mathcal{O}^{req}$ a required operation, $\varphi \in \mathcal{S}(\mathcal{V}^{prov} \cup \mathcal{V}^{req} \cup \mathcal{V}^{int}, par(n))$ the precondition, $\pi \in \mathcal{T}(\mathcal{V}^{req}, par(n))$ the postcondition,
3. $[\varphi]i[\pi]$ with $i \in \mathcal{O}^{int}$ an internal operation, $\varphi \in \mathcal{S}(\mathcal{V}^{prov} \cup \mathcal{V}^{req} \cup \mathcal{V}^{int}, par(i))$ the precondition, $\pi \in \mathcal{T}(\mathcal{V}^{prov} \cup \mathcal{V}^{int}, par(i))$ the postcondition.

There are three kinds of labels concerning reception (?), sending (!) and internal execution of an operation. In each case, labels are equipped with pre- and post-conditions represented by state and transition predicates over particular sets of state variables.

Input labels. A label $[\varphi]?m[\pi]$ models that if a provided operation m is invoked under the precondition φ then the postcondition π will hold after the execution of m. In this case φ expresses the *assumption* (of the specified component) that the environment will only call m when the component's data state fulfills φ. Hence φ must be a state predicate over the *provided* state variables of the component (possibly containing the operation's formal parameters as logical variables). In particular, no internal state variables are allowed in φ, since it must be possible for the environment to check whether the precondition φ is actually valid upon operation call. The postcondition π models the change of the component's data state caused by the execution of the operation m and hence it is a transition predicate over the *provided* and the *internal* state variables of the component (again possibly containing the operation's formal parameters as logical variables). In this case π expresses the *guarantee* (of the specified component) that after the execution of m the postcondition π holds (if the assumption φ was met)[1]. The assumptions and guarantees of a component concerning input labels are part of the contract principle behind data constraints as shown in Table 1.

Output labels. A label $[\varphi]!n[\pi]$ models that a component issues a call to a required operation n under the precondition φ and with postcondition π. Here φ describes the condition under which the operation call is performed by a component. Since the component has access to its own, provided and internal, state variables and can also query its required state variables, the condition φ can contain all kinds of state variables (together with the operation's formal parameters as logical variables). From the contract point of view the component *guarantees* to issue the call to the operation n only if φ holds[2]. The postcondition π of an output label formulates the expectations on the change of the data state performed by the environment after the invoked operation has been executed. Hence π must be a transition predicate over the *required* state variables which expresses an *assumption* on the environment. The assumptions and guarantees of a component concerning output labels accomplish the contract principle behind data constraints as shown in Table 1.

Internal labels. Finally, a label $[\varphi]i[\pi]$ stands for the execution of an internal operation i. In this case φ describes the condition under which the internal operation is executed which, like the precondition in output labels, can again depend on all kinds of state variables. The postcondition π models the change of the component's data state caused by the execution of the internal operation i

[1] The environment can in fact only see the observable consequences of the guarantee π w.r.t. the provided state variables.

[2] In fact, the environment can only see the observable consequences of the guarantee φ w.r.t. the required state variables.

Table 1. Data constraints as contracts

Component
$[\varphi]?m[\pi]$ assume φ guarantee π
$[\varphi]!n[\pi]$ guarantee φ assume π

and hence, like the postcondition in input labels, must be a transition predicate over the provided and the internal state variables.

Example 3. An example of an input label is the Σ_{Leg}-label

$$[a \leq \mathit{maxStep}]\ ?\mathit{swing}(a)\ [\mathit{currStep}' = a \wedge \mathit{steps}' = \mathit{steps} + 1]$$

where *maxStep* and *currStep* are provided state variables, and *steps* is an internal state variable of component *Leg*. The primed expression *currStep'* in the postcondition indicates that we refer to the value of *currStep* in the poststate. For component *LegC*, the expression

$$[a = \min(\mathit{distance} - \mathit{gone}, \mathit{maxStep})]\ !\mathit{swing}(a)\ [\mathit{currStep}' \leq a]$$

is a valid Σ_{LegC}-label. This label expresses that the output $\mathit{swing}(a)$ happens if the precondition $\varphi \equiv [a = \min(\mathit{distance} - \mathit{gone}, \mathit{maxStep})]$ is satisfied. Note that φ involves all three kinds of state variables: the provided state variable *distance*, the required state variable *maxStep* and the internal state variable *gone*. The postcondition $\pi \equiv [\mathit{currStep}' \leq a]$ involves (beside the parameter a) the required state variable *currStep*. ■

Definition 2 (MIOD). *A modal I/O-transition system with data constraints*

$$S = (\mathit{states}_S, \mathit{start}_S, \Sigma_S, \Delta_S^{\mathsf{may}}, \Delta_S^{\mathsf{must}})$$

consists of a set of states states_S, the initial state $\mathit{start}_S \in \mathit{states}_S$, an I/O-signature Σ_S, a may-transition relation $\Delta_S^{\mathsf{may}} \subseteq \mathit{states}_S \times \mathcal{L}(\Sigma_S) \times \mathit{states}_S$, and a must-transition relation $\Delta_S^{\mathsf{must}} \subseteq \Delta_S^{\mathsf{may}}$. The class of all MIODs is denoted by $\mathcal{MIOD}$. The set of the (syntactically) reachable states of a MIOD S is defined by $\mathcal{R}(S) = \bigcup_{n=0}^{\infty} \mathcal{R}_n(S)$ where $\mathcal{R}_0(S) = \{\mathit{start}_S\}$ and $\mathcal{R}_{n+1}(S) = \{s' \mid \exists (s, \ell, s') \in \Delta_S^{\mathsf{may}}$ such that $s \in \mathcal{R}_n(S)\}$.

Example 4. The MIOD S_{Leg} specifying the behavior of the *Leg* component is shown in Fig. 1. Must-transitions are drawn with solid arrows and may-transitions with dashed arrows. Preconditions are written above or to the left of operations;

postconditions are written below or to the right of operations. Pre- and post-conditions of the form [*true*] are omitted.

In the initial state 0, *Leg* can receive *init*() which has the effect of initializing the internal state variable *steps* with 0. In state 1 the leg is able to receive *lift*(), and then *swing*(*a*); in the latter case, the leg component assumes that the value of the parameter *a* does not exceed the limit *maxStep* more than ten percent; the maximal step size *maxStep* is a provided state variable, hence visible for the connected leg controller. The guarantee of the leg component is then, that the current step size is *a* and that the number of steps is increased by one. Next, in state 3, the leg is put on the ground when the operation *drop*() is received, and then, on reception of the operation call *retract*(), the leg is pulling the body forward. Finally, an update message is sent to inform the leg controller how far the leg has been moved forward. ■

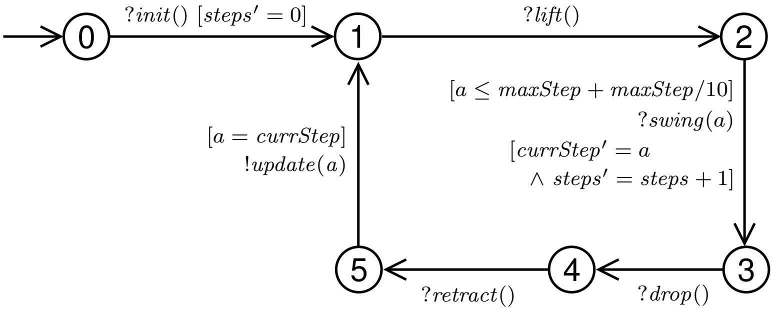
?init() [steps′ = 0]
?lift()
[a ≤ maxStep + maxStep/10] ?swing(a) [currStep′ = a ∧ steps′ = steps + 1]
[a = currStep] !update(a)
?drop()
?retract()
0
1
2
3
4
5

Fig. 1. Specification S_{Leg} of the *Leg* component

3 Refinement of MIODs

The basic idea of *modal* refinement is that any required (*must*) transition in the abstract specification must also be present in the concrete specification. Conversely, any allowed (*may*) transition in the concrete specification must be allowed by the abstract specification. Modal refinement has the following consequences: A concrete specification may omit allowed transitions, but is required to keep all must-transitions. Moreover, it is not allowed to perform more transitions than the abstract specification admits.

Concerning the impact of data constraints, let us first consider required (i.e. *must*) transitions of an abstract MIOD, say T, see Def. 3(1). Obviously, any such transition with a precondition φ_T must also be executable in a refinement S, whenever φ_T is valid. Hence there should be a corresponding must-transition in S whose precondition φ_S does not require more than φ_T does. 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 transition of T concerns input or internal labels, the corresponding transition of the refinement S should lead to a postcondition π_S which guarantees the postcondition π_T of T (expected, for instance, by a user of a provided operation who relies on the postcondition given in the abstract specification T, see 1(a) in Def. 3). However, if a 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, a refinement S should not expect more than the abstract specification does, i.e. π_S should be at most weaker than π_T, see 1(b) in Def. 3.

Let us now consider may-transitions of the concrete MIOD S, see Def. 3(2). Obviously, any such transition with a precondition φ_S must be allowed by the abstract specification T, whenever φ_S is valid. Hence, there should be a corresponding may-transition in T whose precondition φ_T is at most weaker than φ_S and this condition is again independent of the kind of the labels. As explained above the situation is different when considering postconditions since they are not related to the executability of transitions. Therefore, in this case, the kind of the transitions (may or must) is irrelevant and hence the requirements 2(a)(b) coincide with the requirements 1(a)(b) in Def. 3.

In summary, 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).

Definition 3 (Modal Refinement of MIODs). *Let S and T be two MIODs with the same I/O-signature $\Sigma = (\mathcal{V}, \mathcal{O})$. A binary relation $R \subseteq states_S \times states_T$ is a modal refinement between the states of S and T iff for all $(s,t) \in R$*

1. *(from abstract to concrete) for all $op \in \mathcal{O}$, if $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{must}}$, then there exist $s' \in states_S$ and a transition $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{must}}$ such that $(s',t') \in R$, $\models \varphi_T \Rightarrow \varphi_S$, and the following holds:*
   (a) if $op \in \mathcal{O}^{prov} \cup \mathcal{O}^{int}$ then $\models \pi_T \Leftarrow \pi_S$,
   (b) if $op \in \mathcal{O}^{req}$ then $\models \pi_T \Rightarrow \pi_S$.
2. *(from concrete to abstract) for all $op \in \mathcal{O}$, if $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{may}}$, then there exist $t' \in states_T$ and a transition $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{may}}$ such that $(s',t') \in R$, $\models \varphi_T \Leftarrow \varphi_S$, and the following holds:*
   (a) if $op \in \mathcal{O}^{prov} \cup \mathcal{O}^{int}$ then $\models \pi_T \Leftarrow \pi_S$,
   (b) if $op \in \mathcal{O}^{req}$ then $\models \pi_T \Rightarrow \pi_S$.

A state $s \in states_S$ refines a state $t \in states_T$, written $s \leq_m t$, iff there exists a modal refinement between the states of S and T containing (s,t). S is a modal refinement of T, written $S \leq_m T$, iff $start_S \leq_m start_T$.

In the following, we will illustrate the concepts of refinement and, later on, compatibility by means of our running example. We will focus here on the treatment of data constraints by considering small excerpts of corresponding MIODs.

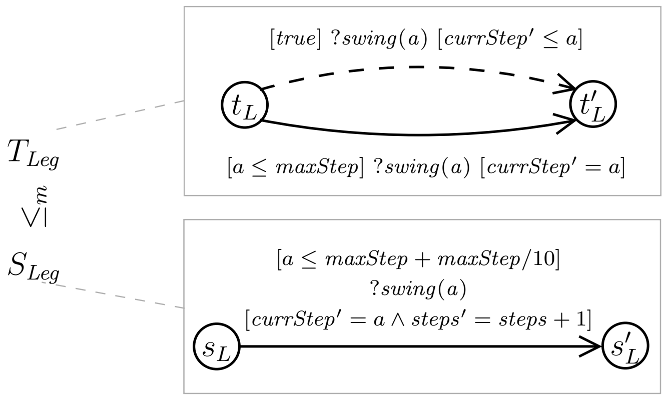
T_{Leg}
$\leq_m$
S_{Leg}
$[true]$ $?swing(a)$ $[currStep' \leq a]$
t_L
t'_L
$[a \leq maxStep]$ $?swing(a)$ $[currStep' = a]$
$[a \leq maxStep + maxStep/10]$
$?swing(a)$
$[currStep' = a \wedge steps' = steps + 1]$
s_L
s'_L

Fig. 2. Refinement S_{Leg} of an abstract specification T_{Leg}

Example 5. Fig. 2 shows excerpts of two MIODs specifying the component *Leg*, an abstract specification T_{Leg} and the more concrete specification S_{Leg}, see also Fig. 1, which refines T_{Leg}, i.e. $S_{Leg} \leq_m T_{Leg}$. The abstract specification T_{Leg} expresses that under the precondition $[a \leq maxStep]$ a call to the operation $swing(a)$ *must* be accepted in any implementation such that in the state after execution of the operation, the postcondition $[currStep' = a]$ is satisfied. The proper may-transition of T_{Leg} says that also such implementations are allowed which accept the operation call $swing(a)$ in states not satisfying $[a \leq maxStep]$, and then the postcondition $[currStep' \leq a]$ must be satisfied in the next state.

In the refinement S_{Leg} there is only a must-transition. The refinement relation holds since, as required by condition 1 in Def. 3, for the must-transition in T_{Leg} there is a corresponding must-transition in S_{Leg} such that the precondition is weakened (by allowing also values of the parameter a exceeding the maximal step size at most by ten percent) and, according to 1(a) in Def. 3, the postcondition is strengthened by requiring additionally $[steps' = steps + 1]$. Conversely, for the direction from S_{Leg} to T_{Leg}, condition 2 in Def. 3 requires that the must-transition (which is also a may-transition) in S_{Leg} is allowed by the abstract specification T_{Leg}. This is indeed the case because for the transition in S_{Leg} there is a corresponding may-transition in T_{Leg} such that the precondition is weakened in T_{Leg} and, according to 2(a) in Def. 3, the postcondition in S_{Leg} is stronger than the corresponding postcondition in T_{Leg}. ■

4 Composition and Compatibility of MIODs

MIODs can be composed to specify the behavior of concurrent systems of interacting components with data constraints. The composition operator extends the synchronous composition of modal I/O-transition systems [9]. Like for MIOs, we need some syntactic restrictions under which two MIODs are composable. First, we require that overlapping of operations only happens on complementary types and that the same holds for state variables.

Definition 4 (Composability of I/O-Signatures). *Two I/O-signatures $\Sigma_S = (\mathcal{V}_S, \mathcal{O}_S)$ and $\Sigma_T = (\mathcal{V}_T, \mathcal{O}_T)$ are* composable *if*

1. $\mathcal{V}_S \cap \mathcal{V}_T = (\mathcal{V}_S^{prov} \cap \mathcal{V}_T^{req}) \cup (\mathcal{V}_T^{prov} \cap \mathcal{V}_S^{req})$,
2. $\mathcal{O}_S \cap \mathcal{O}_T = (\mathcal{O}_S^{prov} \cap \mathcal{O}_T^{req}) \cup (\mathcal{O}_T^{prov} \cap \mathcal{O}_S^{req})$.

The set $\mathcal{V}_S \cap \mathcal{V}_T$ of shared state variables *will be denoted by $sh(\mathcal{V}_S, \mathcal{V}_T)$ and the set $\mathcal{O}_S \cap \mathcal{O}_T$ of* shared operations *will be denoted by $sh(\mathcal{O}_S, \mathcal{O}_T)$.*

When two composable I/O-signatures are actually composed, the shared state variables become internal. Likewise, the shared operations become internal representing the synchronization of provided and required operations.

Definition 5 (Composition of I/O-Signatures). *The composition of two composable I/O-signatures $\Sigma_S = (\mathcal{V}_S, \mathcal{O}_S)$ and $\Sigma_T = (\mathcal{V}_T, \mathcal{O}_T)$ is the I/O-signature $\Sigma_S \otimes \Sigma_T = (\mathcal{V}_S \otimes \mathcal{V}_T, \mathcal{O}_S \otimes \mathcal{O}_T)$ where*

$$\begin{aligned}
(\mathcal{V}_S \otimes \mathcal{V}_T)^{prov} &= (\mathcal{V}_S^{prov} \uplus \mathcal{V}_T^{prov}) \setminus sh(\mathcal{V}_S, \mathcal{V}_T),\\
(\mathcal{V}_S \otimes \mathcal{V}_T)^{req} &= (\mathcal{V}_S^{req} \uplus \mathcal{V}_T^{req}) \setminus sh(\mathcal{V}_S, \mathcal{V}_T),\\
(\mathcal{V}_S \otimes \mathcal{V}_T)^{int} &= \mathcal{V}_S^{int} \uplus \mathcal{V}_T^{int} \uplus sh(\mathcal{V}_S, \mathcal{V}_T),
\end{aligned}$$

$$\begin{aligned}
(\mathcal{O}_S \otimes \mathcal{O}_T)^{prov} &= (\mathcal{O}_S^{prov} \uplus \mathcal{O}_T^{prov}) \setminus sh(\mathcal{O}_S, \mathcal{O}_T),\\
(\mathcal{O}_S \otimes \mathcal{O}_T)^{req} &= (\mathcal{O}_S^{req} \uplus \mathcal{O}_T^{req}) \setminus sh(\mathcal{O}_S, \mathcal{O}_T),\\
(\mathcal{O}_S \otimes \mathcal{O}_T)^{int} &= \mathcal{O}_S^{int} \uplus \mathcal{O}_T^{int} \uplus sh(\mathcal{O}_S, \mathcal{O}_T).
\end{aligned}$$

Two MIODs are composable if their signatures are composable and if the labels occurring on their transitions satisfy certain syntactic constraints. Condition 1 in Def. 6 is technically necessary to ensure that the composition of MIODs defined below is well-formed. For instance, condition 1(a) requires that preconditions of non-shared, provided operations must not include shared state variables; otherwise the precondition of a provided operation in the composition would involve internal state variables which is not allowed. Condition 2 in Def. 6 is needed from an intuitive point of view. For instance, condition 2(a) requires that for shared, provided operations preconditions of one MIOD can only talk about provided variables which are known from the other MIOD as required variables.

Definition 6 (Composability of MIODs). *Two MIODs S and T are* composable *if their signatures $\Sigma_S = (\mathcal{V}_S, \mathcal{O}_S)$ and $\Sigma_T = (\mathcal{V}_T, \mathcal{O}_T)$ are composable, if for all transitions $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{may}}$ we have*

1. *if $op \notin sh(\mathcal{O}_S, \mathcal{O}_T)$ then*
   (a) if $op \in \mathcal{O}_S^{prov}$ then $\varphi_S \in \mathcal{S}(\mathcal{V}_S^{prov} \setminus sh(\mathcal{V}_S, \mathcal{V}_T), par(op))$,
   (b) if $op \in \mathcal{O}_S^{req}$ then $\pi_S \in \mathcal{T}(\mathcal{V}_S^{req} \setminus sh(\mathcal{V}_S, \mathcal{V}_T), par(op))$,
2. *if $op \in sh(\mathcal{O}_S, \mathcal{O}_T)$ then*
   (a) if $op \in \mathcal{O}_S^{prov}$ then $\varphi_S \in \mathcal{S}(\mathcal{V}_S^{prov} \cap \mathcal{V}_T^{req}, par(op))$,
   (b) if $op \in \mathcal{O}_S^{req}$ then $\pi_S \in \mathcal{T}(\mathcal{V}_S^{req} \cap \mathcal{V}_T^{prov}, par(op))$,

and if the same holds symmetrically for all transitions $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{may}}$.

The composition 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 $[\varphi_S \wedge \varphi_T]op[\pi_S \wedge \pi_T]$ where the original preconditions (postconditions resp.) are combined by logical conjunction. Transitions whose labels concern shared operations which cannot be synchronized are dropped while all other transitions are interleaved in the composition. Concerning modalities we follow the MIO composition operator which yields a must-transition if two must-transitions are synchronized and a may-transition otherwise.

Definition 7 (Composition of MIODs). *The composition of two composable MIODs S and T is the MIOD*

$$S \otimes T = (states_S \times states_T, (start_S, start_T), \Sigma_S \otimes \Sigma_T, \Delta^{\mathsf{may}}_{S\otimes T}, \Delta^{\mathsf{must}}_{S\otimes T})$$

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

$$\frac{(s, [\varphi_S]op[\pi_S], s') \in \Delta^{\gamma}_S,\ (t, [\varphi_T]op[\pi_T], t') \in \Delta^{\gamma}_T}{((s,t), [\varphi_S \wedge \varphi_T]op[\pi_S \wedge \pi_T], (s',t')) \in \Delta^{\gamma}_{S\otimes T}} \quad \begin{array}{l} op \in sh(\mathcal{O}_S, \mathcal{O}_T), \\ \gamma \in \{\mathsf{may}, \mathsf{must}\} \end{array}$$

$$\frac{(s, [\varphi_S]op[\pi_S], s') \in \Delta^{\gamma}_S, \quad t \in states_T}{((s,t), [\varphi_S]op[\pi_S], (s',t)) \in \Delta^{\gamma}_{S\otimes T}} \quad op \notin sh(\mathcal{O}_S, \mathcal{O}_T),\ \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

$$\frac{(t, [\varphi_T]op[\pi_T], t') \in \Delta^{\gamma}_T, \quad s \in states_S}{((s,t), [\varphi_T]op[\pi_T], (s,t')) \in \Delta^{\gamma}_{S\otimes T}} \quad op \notin sh(\mathcal{O}_S, \mathcal{O}_T),\ \gamma \in \{\mathsf{may}, \mathsf{must}\}$$

Example 6. Assume given the MIOD S_{Leg} of Fig. 1 and a MIOD S_{LegC} describing the behavior of the controller component *LegC*. Fig. 3 shows the result of the synchronization of the two transitions of S_{LegC} and S_{Leg} concerning the shared

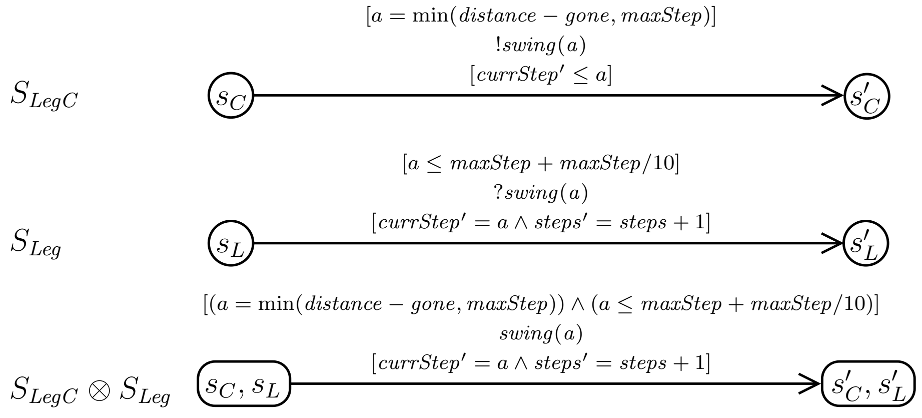
S_{LegC}
s_C
$[a = \min(distance - gone, maxStep)]$
$!swing(a)$
$[currStep' \leq a]$
s'_C
S_{Leg}
s_L
$[a \leq maxStep + maxStep/10]$
$?swing(a)$
$[currStep' = a \wedge steps' = steps + 1]$
s'_L
$S_{LegC} \otimes S_{Leg}$
s_C, s_L
$[(a = \min(distance - gone, maxStep)) \wedge (a \leq maxStep + maxStep/10)]$
$swing(a)$
$[currStep' = a \wedge steps' = steps + 1]$
s'_C, s'_L

Fig. 3. MIOD composition

operation *swing*(a) which is an internal operation in the composition $S_{LegC} \otimes S_{Leg}$. Similarly the shared state variable *maxStep* is an internal state variable in $S_{LegC} \otimes S_{Leg}$. ■

As discussed above, if we want to compose two MIODs 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 of MIOs as defined in [4]. From the control point of view (strong) compatibility requires that in any reachable state of the product $S \otimes T$ 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[3]. 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 (see Table 1) the two contracts according to S and T must match. More precisely, matching means that the mutual assumptions and guarantees of the two MIODs imply each other for any shared operation as illustrated in Table 2. Thus, by combining the control flow and the data state aspects of compatibility we obtain the following compatibility notion for MIODs.

Table 2. Data compatibility of MIODs

S	T	S $\sim$ T
$[\varphi_S^m]?m[\pi_S^m]$	$[\varphi_T^m]!\mathrm{m}[\pi_T^m]$	
assume φ_S^m	guarantee φ_T^m	$\varphi_S^m \Leftarrow \varphi_T^m$
guarantee π_S^m	assume π_T^m	$\pi_S^m \Rightarrow \pi_T^m$
$[\varphi_S^n]!n[\pi_S^n]$	$[\varphi_T^n]?\mathrm{m}[\pi_T^n]$	
guarantee φ_S^n	assume φ_T^n	$\varphi_S^n \Rightarrow \varphi_T^n$
assume π_S^n	guarantee π_T^n	$\pi_S^n \Leftarrow \pi_T^n$

Definition 8 (Compatibility of MIODs). *Let S and T be two composable MIODs. S and T are* compatible*, denoted by $S \sim T$, if for all reachable states $(s,t) \in \mathcal{R}(S \otimes T)$,*

1. *if $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{may}}$ and $op \in (\mathcal{O}_S^{req} \cap \mathcal{O}_T^{prov})$, then there exists a must-transition $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{must}}$ such that $\models \varphi_S \Rightarrow \varphi_T$, and for all may-transitions $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{may}}$ it holds that $\models \pi_S \Leftarrow \pi_T$;*
2. *if $(t, [\varphi_T]op[\pi_T], t') \in \Delta_T^{\mathsf{may}}$ and $op \in (\mathcal{O}_T^{req} \cap \mathcal{O}_S^{prov})$, then there exists a must-transition $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{must}}$ such that $\models \varphi_T \Rightarrow \varphi_S$, and for all may-transitions $(s, [\varphi_S]op[\pi_S], s') \in \Delta_S^{\mathsf{may}}$ it holds that $\models \pi_T \Leftarrow \pi_S$.*

[3] This concept follows a "pessimistic" approach where two components should be compatible in any environment, in contrast to the "optimistic" approach pursued in [6,9] which relies on the existence of a "helpful" environment.

Example 7. In $S_{LegC} \otimes S_{Leg}$, the state (s_C, s_L) is a reachable state, see Fig. 3. Compatibility holds because for the operation call $swing(a)$ issued by S_{LegC} in state s_C there exists a single must-transition of S_{Leg} which guarantees the reception of $swing(a)$ in state s_L such that the following holds (for any usual satisfaction relation):

$$\models (a = \min(distance - gone, maxStep)) \Rightarrow (a \leq maxStep + maxStep/10),$$
$$\models (currStep' = a \wedge steps' = steps + 1) \Rightarrow (currStep' \leq a). \qquad \blacksquare$$

We are now able to state our central result which says that compatibility is preserved by refinement and that refinement is compositional (for compatible MIODs). Thus our framework supports independent implementability.

Theorem 1 (Independent Implementability). *Let S, S', T, and T' be MIODs, and assume that S, T as well as S', T' are composable. If $S \sim T$, $S' \leq_m S$ and $T' \leq_m T$, then $S' \sim T'$ and $S' \otimes T' \leq_m S \otimes T$.*

Proof. We first prove preservation of compatibility. Let $(s', t') \in \mathcal{R}(S' \otimes T')$ be a reachable state in $S' \otimes T'$. It follows from condition 2 of Def. 3 that there exists a reachable state $(s, t) \in \mathcal{R}(S \otimes T)$ such that $t' \leq_m t$ and $s' \leq_m s$. Assume that there exists a transition $(s', [\varphi_{S'}]op[\pi_{S'}], \hat{s}') \in \Delta^{\mathsf{may}}_{S'}$ such that $op \in \mathcal{O}^{req}_{S'} \cap \mathcal{O}^{prov}_{T'}$. From $s' \leq_m s$ it follows that there exists $(s, [\varphi_S]op[\pi_S], \hat{s}) \in \Delta^{\mathsf{may}}_S$ such that $\models \varphi_S \Leftarrow \varphi_{S'}$ and $\models \pi_S \Rightarrow \pi_{S'}$. From $S \sim T$ it follows that there exists $(t, [\varphi_T]op[\pi_T], \hat{t}) \in \Delta^{\mathsf{must}}_T$ such that $\models \varphi_S \Rightarrow \varphi_T$. Since $t' \leq_m t$ we can conclude that there exists $(t', [\varphi_{T'}]op[\pi_{T'}], \hat{t}') \in \Delta^{\mathsf{must}}_{T'}$ such that $\models \varphi_T \Rightarrow \varphi_{T'}$. It follows that $\models \varphi_{S'} \Rightarrow \varphi_{T'}$. Then, we must additionally show that for all accepting may-transitions of T', the postcondition matches $\pi_{S'}$ which is again straightforward.

For the proof of compositionality of modal refinement we define $\leq'_m \subseteq (states_{S'} \times states_{T'}) \times (states_S \times states_T)$ by

$$(s', t') \leq'_m (s, t) \text{ iff } s' \leq_m s, t' \leq_m t, (s', t') \in \mathcal{R}(S' \otimes T') \text{ and } (s, t) \in \mathcal{R}(S \otimes T).$$

Obviously, $(start_{S'}, start_{T'}) \leq'_m (start_S, start_T)$ and it remains to show that $\leq'_m$ is a modal refinement between the states of $S' \otimes T'$ and the states of $S \otimes T$. The detailed proof can be found in [2].

Example 8. In Fig. 4, the principle of independent implementability is illustrated in terms of our running example showing small excerpts of four MIODs. Starting from the abstract MIODs T_{LegC} and T_{Leg} we first check their compatibility, i.e. $T_{LegC} \sim T_{Leg}$. Then we refine T_{LegC} and T_{Leg} independently of each other by the MIODs S_{LegC} and S_{Leg}, respectively. Thm. 1 guarantees, first, that S_{LegC} and S_{Leg} are compatible (as explained explicitly in Ex. 7) and secondly, that $S_{LegC} \otimes S_{Leg} \leq_m T_{LegC} \otimes T_{Leg}$ holds. $\blacksquare$

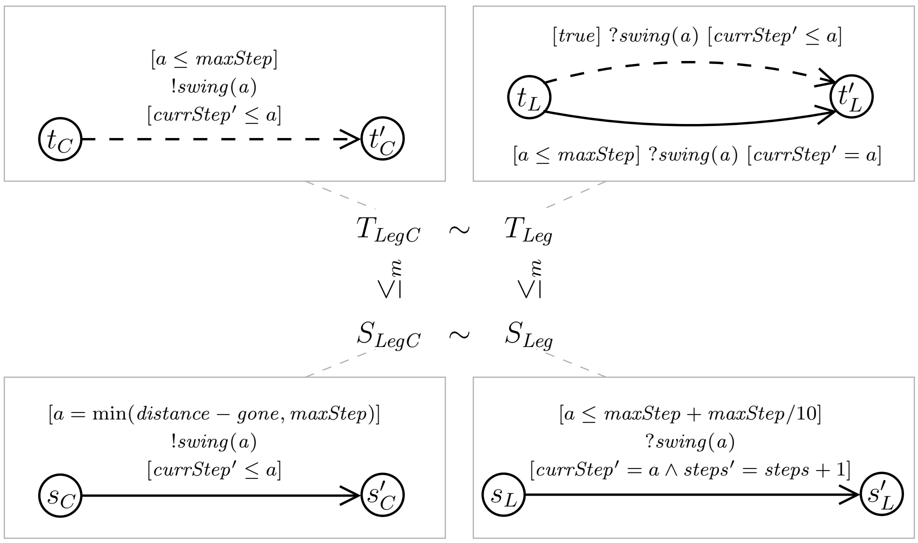
$[a \leq maxStep]$
$!swing(a)$
$[currStep' \leq a]$
t_C
t'_C
$[true]\ ?swing(a)\ [currStep' \leq a]$
t_L
t'_L
$[a \leq maxStep]\ ?swing(a)\ [currStep' = a]$
$T_{LegC} \sim T_{Leg}$
$\leq_m$
$S_{LegC} \sim S_{Leg}$
$[a = \min(distance - gone, maxStep)]$
$!swing(a)$
$[currStep' \leq a]$
s_C
s'_C
$[a \leq maxStep + maxStep/10]$
$?swing(a)$
$[currStep' = a \wedge steps' = steps + 1]$
s_L
s'_L

Fig. 4. Compositional refinement of compatible components *LegC* and *Leg*

5 Conclusion

Modal I/O-transition systems (MIOs) provide a flexible framework for the specification and refinement of interacting, concurrent components. In this work we have extended MIOs to take into account data constraints which allow to specify the behavior of components with regard to changing data states. We have shown that our proposal satisfies the requirements of an interface theory guaranteeing preservation of compatibility and compositionality of refinements when moving from abstract to more concrete specifications of interacting components.

We have tried to keep the formalism for refinement as simple as possible at the cost of some restrictions. For instance, one would like to be able to refine a single (abstract) must-transition with some precondition φ by several (concrete) transitions which distribute the precondition over several cases φ_i for $i = 1, \ldots, n$, such that $\varphi \Rightarrow \bigvee_i \varphi_i$. Similarly, one could relax the requirements for may-transitions. Moreover, it would not be a problem to integrate state invariants in our framework.

In this paper we have worked on the level of specifications and their refinement and compatibility which are clearly dependent on the syntactical representation of the specification. We have not yet provided a formal semantics which would allow us to define semantic notions of refinement and compatibility. A formal semantics could be delivered in terms of the class of all correct implementations of a specification as done for MIOs in [10] and for behavior protocols without modalities but with data states in [3]. An implementation would then be modeled by a transition system with must-transitions only and with concrete data states given by valuations of the state variables. Concerning the interpretation of specification transitions with pre- and postconditions, the implementation must guarantee that single specification transitions of a component are executed in an atomic way.

Verification techniques for refinement and compatibility are clearly an important issue for future work. Further next steps are, from the theoretical point of view, 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 [10,4], but also internal state variables. From the practical point of view we plan to integrate data constraints in our tool, the MIO Workbench [4], for modal refinement and compatibility checking.

References

1. Barros, T., Ameur-Boulifa, R., Cansado, A., Henrio, L., Madelaine, E.: Behavioural models for distributed Fractal components. Annales des Télécommunications 64(1-2), 25–43 (2009)
2. Bauer, S.S., Hennicker, R., Bidoit, M.: A modal interface theory with data constraints. Technical Report 1005, Ludwig-Maximilians-Universität München, Germany (2010)
3. Bauer, S.S., Hennicker, R., Janisch, S.: Behaviour protocols for interacting stateful components. Electr. Notes Th. Comp. Sci. 263, 47–66 (2010)
4. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On weak modal compatibility, refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer, Heidelberg (2010)
5. Bidoit, M., Hennicker, R., Knapp, A., Baumeister, H.: Glass-box and black-box views on object-oriented specifications. In: Proc. SEFM 2004, Beijing, China, pp. 208–217. IEEE Comp. Society Press, Los Alamitos (2004)
6. de Alfaro, L., Henzinger, T.A.: Interface Theories for Component-Based Design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001)
7. Fernandes, F., Royer, J.-C.: The STSLib project: Towards a formal component model based on STS. Electr. Notes Th. Comp. Sci. 215, 131–149 (2008)
8. Fischer, C.: CSP-OZ: a combination of Object-Z and CSP. In: Proc. FMOODS, Canterbury, UK, pp. 423–438. Chapman and Hall, Boca Raton (1997)
9. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
10. Larsen, K.G., Nyman, U., Wasowski, A.: On Modal Refinement and Consistency. In: Caires, L., Li, L. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 105–119. Springer, Heidelberg (2007)
11. Larsen, K.G., Thomsen, B.: A Modal Process Logic. In: 3rd Annual Symp. Logic in Computer Science, LICS 1988, pp. 203–210. IEEE Computer Society, Los Alamitos (1988)
12. Mouelhi, S., Chouali, S., Mountassir, H.: Refinement of interface automata strengthened by action semantics. Electr. Notes Theor. Comput. Sci. 253(1), 111–126 (2009)