Modal interface theories for communication-safe component assemblies

Rolf Hennicker, Alexander Knapp

Angaben zur Veröffentlichung / Publication details:

Hennicker, Rolf, and Alexander Knapp. 2011. "Modal interface theories for communication-safe component assemblies." *Lecture Notes in Computer Science* 6916: 135–53. https://doi.org/10.1007/978-3-642-23283-1_11.

Modal Interface Theories for Communication-Safe Component Assemblies*

Rolf Hennicker[1] and Alexander Knapp[2]

[1] Ludwig-Maximilians-Universität München
hennicke@pst.ifi.lmu.de
[2] Universität Augsburg
knapp@informatik.uni-augsburg.de

Abstract. We propose an extension of the abstract rules for independent implementability of reactive components proposed in interface theories to take into account interface assemblies. As a concrete instantiation we extend existing interface theories for modal I/O-transition systems to support assemblies, (greybox) assembly refinement and assembly encapsulation. We introduce a new notion of communication-safety for N-ary assemblies which overcomes problems with previous definitions of interface compatibility. We show that communication-safety can be checked incrementally. We also show that communication-safety is preserved by assembly refinement, that blackbox refinement of component interfaces is compositional w.r.t. greybox refinement of assemblies and, conversely, that assembly encapsulation maps greybox to blackbox refinement. The methodology of our approach is illustrated by a small case study.

1 Introduction

Reactive software components are commonly understood as encapsulated units which communicate with their environment via well-defined interfaces. Interface specifications provide a means to describe the visible behaviour of interacting components. They serve, on the one hand, to express what is expected from the environment for a correct functioning of a component, and, on the other hand, to specify what is offered by a component. For the development of component systems on the basis of interfaces we can identify three key issues: the ability to build larger specifications from smaller ones (by composition), the (stepwise) refinement of interface specifications, and compatibility requirements ensuring safe communication of components. Of course, it is important that the different aspects fit properly together, i.e. that refinement is preserved by composition and that compatibility of interfaces is preserved by refinement, thus guaranteeing independent implementability of components.

1.1 Interface Languages

These crucial requirements, that any concrete interface theory should obey, are nicely formulated by the rules of an interface language stated in [7]. It assumes a domain $\mathcal{F}$

* This work has been partially sponsored by the Bavarian Ministry for Economics, Infrastructure, Traffic and Technology under the IuK-project RAJA, IUK-0805-0005 and by the European Union under the FP7-project ASCENS, 257414.

of interfaces, a partial composition operator $\| : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$, a binary compatibility relation $\sim \subseteq \mathcal{F} \times \mathcal{F}$, and a refinement relation $\leq \subseteq \mathcal{F} \times \mathcal{F}$ relating concrete and abstract specifications. On this basis the principle of independent implementability reads as follows:

Independent implementability: For all $F, F', G, G' \in \mathcal{F}$,
if $F \sim G$ and $F' \leq F$, $G' \leq G$, then $F' \sim G'$ and $(F' \parallel G') \leq (F \parallel G)$.[1]

Particular instances of interface theories satisfying these requirements are formulated for interface automata in [7] and for modal I/O-transition systems (MIOs) in [10,2].

Following the ideas of an interface language, interfaces for complex components are constructed from smaller ones by interface composition, following, e.g., a synchronous communication scheme, like interface automata and MIOs, where shared actions of complementary types (input/output) are synchronised. The result of an interface composition yields again an interface describing the visible (blackbox) behaviour of a composite component. Hence, the architectural information behind interface composition is hidden and it is not possible to specify architectural requirements, which are important as well for the development of complex systems. In particular, the communication behaviour of interacting components is abstracted away during interface composition and can not be taken into account in further refinement, since interface refinement is inherently a blackbox refinement. For instance, assume that an interface for a credit card payment system is specified by composing an interface describing the customer's behaviour with another interface describing the clearing company's behaviour, which is responsible to verify the credit card. That the verification is really performed assumes a communication between the customer with the clearing company. But during interface composition this communication is made invisible and therefore the composite interface could be refined, for instance, by a primitive interface which has the same observable behaviour but which does not actually communicate with the clearing company.[2]

1.2 Interface Assemblies

As a consequence of this discussion, we claim that there is still a missing link in the notion of an interface language and in its concrete instantiations. What is still missing is the specification of architectural information and the explicit possibility to observe communications between components. Only in a next step, if the necessary communications are established, it should be legal to abstract them away and to construct a new interface specification by explicit encapsulation of the architectural one. To tackle this issue the most obvious approach is to consider interface networks as a distinguished concept. Interface networks are assemblies of connected interfaces thus providing architectural requirements for the planned system. Such assemblies can be refined only if the architectural requirements are respected. From the behavioural point of view this

[1] It is assumed that the composition of compatible interfaces is defined. Interface languages require also the property of incremental design which will be discussed later.

[2] A way out could be to require that refinements of composite interfaces not only can but must be performed piecewise; but this is not anchored in the formalism and anyway cannot be treated with alternating simulation refinement of interface automata since, in contrast to modal I/O-transition systems, abstract outputs can always be omitted there in a refinement.

means that not only the visible actions to the outside but also specified communications between components within the assembly must be taken into account when assemblies are refined. Following the terminology of [12] such refinements are also called greybox refinements. Hence, we can extend the abstract concept of an interface language by introducing, additionally to the domain $\mathcal{F}$ of interfaces, a domain $\mathcal{A}$ of assemblies. For the construction of assemblies we assume, for each natural number n, a partial (overloaded) operator $asm : \mathcal{F}^n \to \mathcal{A}$. Concerning compatibility of components, we extend the binary compatibility relation between interfaces to a communication-safety predicate $cs \subseteq \mathcal{A}$ on assemblies, which expresses that the interfaces belonging to an assembly can work together without communication errors. This could mean, e.g., that the assembly is deadlock-free, or, in the context of the I/O-transition systems considered below, that any output a component wants to issue to the remainder of the assembly is indeed accepted.[3] Concerning refinement, we distinguish between interface and assembly refinement, expressed by the binary relations $\preccurlyeq_{\mathrm{intf}} \subseteq \mathcal{F} \times \mathcal{F}$ and $\preccurlyeq_{\mathrm{asm}} \subseteq \mathcal{A} \times \mathcal{A}$ resp. Finally, we introduce an operation *pack* : $\mathcal{A} \to \mathcal{F}$ which allows to express encapsulation of an assembly into an interface (by hiding the architectural information of the assembly). The principle of independent implementability can now be rephrased as follows:

Independent implementability: For all $F_k, F'_k \in \mathcal{F}$ for $k = 1, \ldots, n$,
if $cs(asm(F_1, \ldots, F_n))$ and $F'_k \preccurlyeq_{\mathrm{intf}} F_k$ for $k = 1, \ldots, n$,
then $cs(asm(F'_1, \ldots, F'_n))$ and $asm(F'_1, \ldots, F'_n) \preccurlyeq_{\mathrm{asm}} asm(F_1, \ldots, F_n)$.

But obviously this does not suffice for extracting an interface from an assembly by encapsulation, i.e., by applying the *pack* operation. Then it must be ensured that packing two assemblies which are in refinement relation leads to interfaces which are in refinement relation again, which is expressed by the principle of refinement encapsulation:

Refinement encapsulation: For all $A, A' \in \mathcal{A}$,
if $cs(A)$ and $A' \preccurlyeq_{\mathrm{asm}} A$, then $pack(A') \preccurlyeq_{\mathrm{intf}} pack(A)$.

1.3 Modal I/O-Automata for Interface and Assembly Specifications

As a concrete instantiation of our approach we will build upon modal I/O-transition systems (MIOs) introduced by Larsen et al. [10,11]. We have chosen MIOs as our basic formalism since they allow us to distinguish between transitions which are optional (may) or mandatory (must) for refinements. We extend the interface theory of MIOs presented in [2] by introducing interface assemblies together with a new notion of communication-safety and assembly refinement. In our approach, interface specifications are simply MIOs with input and output actions only, while assemblies are MIO networks formally presented by finitely indexed sets of (syntactically composable) MIOs. To any assembly a greybox behaviour is associated which is again a MIO having, additionally to input and output actions, distinguished communication actions. It is computed by the synchronous composition of the interface MIOs contained in the assembly, but synchronisations on complementary input and output actions are not

[3] We will show in the technical part of the paper that this requirement is different from deadlock-freedom.

hidden, as in the original MIO approach, but considered as visible communication actions. Our notion of communication-safety is inspired by the notion of weak modal compatibility proposed in [2]. However, it goes beyond that because, first, it allows to study the compatibility of arbitrarily many interfaces, and, secondly, it generalises significantly the behavioural requirements of weak modal compatibility. The idea is that whenever one component wants to send an output it finds the communication partner in a state where it must eventually take the corresponding input. Before taking the input the communication partner can still perform silent must-transitions *and/or* mandatory communications with other components of the assembly *and also* outputs which are a must and are directed outside of the assembly. The latter is a significant generalisation of weak modal compatibility needed in many practical applications. We show that communication-safe assemblies can be built up in an incremental way, i.e., given a communication-safe assembly A and an interface F which is compatible to A, the new assembly extending A by F will be communication-safe again. This result is related to the property of incremental design of an interface language [7].

Concerning refinement we distinguish, as explained above, between interface and assembly refinement. Interface refinement has to respect blackbox behaviours, it is formalised in terms of weak modal refinement of MIOs based on a simulation relation which abstracts from silent transitions (labelled with the invisible action τ). Assembly refinement has to respect the architectural requirements and the (greybox) behaviour of assemblies. Therefore it is defined as structure preserving pairwise refinement of the finitely indexed sets of interfaces which form the assemblies. The refinement of assembly behaviours abstracts away silent transitions (which stem from inner components), but transitions resulting from communications are considered as visible and therefore they are respected by the simulation relation. Interface refinement is also called blackbox refinement and assembly refinement corresponds to greybox refinement.

As central results of our approach we obtain that the property of independent implementability (second version from above) is satisfied. Indeed we even get stronger results, called compositionality of refinement and preservation of communication-safety, which read as follows and which obviously imply independent implementability:

Compositionality of refinement: For all $F_k, F'_k \in \mathcal{F}$ for $k = 1, \ldots, n$,
if $F'_k \preccurlyeq_{\text{intf}} F_k$ for $k = 1, \ldots n$,
then $asm(F'_1, \ldots, F'_n) \preccurlyeq_{\text{asm}} asm(F_1, \ldots, F_n)$.

Preservation of communication-safety: For all $A, A' \in \mathcal{A}$,
if $cs(A)$ and $A' \preccurlyeq_{\text{asm}} A$, then $cs(A')$.

Last but not least we consider the encapsulation of MIO assemblies into interfaces, i.e. the *pack* operation, which moves the visible communications of assemblies into silent transitions of interfaces. We show that encapsulation is compatible with assembly and interface refinement, i.e., that the principle of refinement encapsulation stated above is satisfied as well. Also in this case we can prove a stronger version which does not need the communication-safety assumption of the abstract assembly. Our methodology and our results will be demonstrated on a small, but detailed case study.

The paper is structured as follows: In Sect. 2 we summarise the basic notions of modal I/O-transition systems and in Sect. 3 we introduce our running example. Then, in

Sect. 4, we consider formally interfaces, assemblies, their greybox behaviour and their communication-safety; we also define interface and assembly refinement and prove our central compatibility and compositionality results. Our methodology is illustrated by our small case study in Sect. 5. Finally, in Sect. 6, we finish with some concluding remarks.

2 Modal I/O-Transition Systems: Basic Definitions

Modal I/O-transition systems (MIOs) have been introduced by Larsen et al. [10,11] as a formalism to specify the behaviour of reactive components. They allow to distinguish between transitions which are optional (*may*) or mandatory (*must*) for refinements. We will use MIOs for describing interface behaviours as well as assembly behaviours. Technically this is achieved by distinguishing not only input, output, and internal labels, but also communication labels expressing synchronous communication. In contrast to [10] internal actions are not explicitly named here but represented by the internal, invisible action τ while communication labels are newly introduced here. In the following of this section we recall the technical definitions used hereafter. These definitions will be illustrated by our running example introduced in Sect. 3.

An *I/O-labelling* $L = (I_L, O_L, T_L)$ consists of pairwise disjoint sets of *input labels* I_L, *output labels* O_L, and *communication labels* T_L, such that $\tau \notin I_L \cup O_L \cup T_L$. We write $\bigcup L$ for $I_L \cup O_L \cup T_L$.

A *modal I/O-transition system* (*MIO*) $M = (L_M, S_M, s_{0,M}, \dashrightarrow_M, \longrightarrow_M)$ consists of an I/O-labelling $L_M = (I_M, O_M, T_M)$, a set of *states* S_M, an *initial state* $s_{0,M} \in S_M$, a *may-transition relation* $\dashrightarrow_M \subseteq S_M \times (\bigcup L_M \cup \{\tau\}) \times S_M$, and a *must-transition relation* $\longrightarrow_M \subseteq \dashrightarrow_M$. M is called an *implementation* if $\longrightarrow_M = \dashrightarrow_M$. For $l \in \bigcup L_M$ we write $s \overset{\widehat{l}}{\dashrightarrow}_M s'$ for $s \overset{\tau}{\dashrightarrow}_M^m \cdot \overset{l}{\dashrightarrow}_M \cdot \overset{\tau}{\dashrightarrow}_M^n s'$ with $m, n \geq 0$, and $s \overset{\widehat{\tau}}{\dashrightarrow}_M s'$ for $s \overset{\tau}{\dashrightarrow}_M^n s'$ with $n \geq 0$ (and likewise for the must-transition relation). Furthermore, for $X \subseteq \bigcup L_M$, we write $s \overset{\widehat{X}}{\dashrightarrow}_M s'$ for $s \overset{\widehat{l_1}}{\dashrightarrow}_M \cdots \overset{\widehat{l_n}}{\dashrightarrow}_M s'$ with $n \geq 0$ and $l_1, \ldots, l_n \in X$. The *reachable states* of M are denoted by $\mathscr{R}(M)$ with $s \in \mathscr{R}(M)$ if, and only if there is a finite sequence of may-transitions from $s_{0,M}$ to s in M.

The *hiding* of communication labels of a MIO M is given by $M\xi = (L_M\xi, S_M, s_{0,M}, \dashrightarrow_{M\xi}, \longrightarrow_{M\xi})$ with $L_M\xi = (I_{L_M}, O_{L_M}, \emptyset)$, $\dashrightarrow_{M\xi} = \{(s, l, s') \mid s \overset{l}{\dashrightarrow}_M s', l \in I_M \cup O_M\} \cup \{(s, \tau, s') \mid s \overset{l}{\dashrightarrow}_M s', l \in T_M\}$ and likewise for $\longrightarrow_{M\xi}$.

Two I/O-labellings K and L are *composable* if their labels overlap only on complementary types, i.e. $\bigcup K \cap \bigcup L \subseteq (I_K \cap O_L) \cup (I_L \cap O_K)$. The *synchronous composition* of two composable I/O-labellings K and L moves corresponding input/output labels to the set of communication labels, i.e., it yields the I/O-labelling $K \otimes^{\mathrm{sy}} L = ((I_K \setminus O_L) \cup (I_L \setminus O_K), (O_K \setminus I_L) \cup (O_L \setminus I_K), T_K \cup T_L \cup (I_K \cap O_L) \cup (I_L \cap O_K))$.

Two MIOs M and N are *composable* if their I/O-labellings are composable. The *synchronous composition* of two composable MIOs M and N is denoted by $M \otimes^{\mathrm{sy}} N$ and defined as the usual product of automata with synchronisation on shared labels, which are communication labels of the product; a synchronisation transition in $M \otimes^{\mathrm{sy}} N$ is a must-transition if, and only if both synchronising transitions are must-transitions.

Composability and synchronous composition are straightforwardly extended to finitely indexed sets of I/O-labellings and MIOs: A finitely indexed set $(L_i)_{1\leq i\leq n}$ of I/O-labellings with $n \geq 1$ is *composable* if its I/O-labellings are pairwise composable. The *synchronous composition* of a composable finitely indexed set $(L_i)_{1\leq i\leq n}$ of I/O-labellings with $n \geq 1$ is inductively given by $\bigotimes^{\mathrm{sy}}_{1\leq i\leq n} L_i = L_1 \otimes^{\mathrm{sy}} \ldots \otimes^{\mathrm{sy}} L_n$. A finitely indexed set $(M_i)_{1\leq i\leq n}$ of MIOs with $n \geq 1$ is *composable* if the I/O-labellings of its MIOs are pairwise composable. The *synchronous composition* of a composable finitely indexed set $(M_i)_{1\leq i\leq n}$ of MIOs with $n \geq 1$ is inductively given by $\bigotimes^{\mathrm{sy}}_{1\leq i\leq n} M_i = M_1 \otimes^{\mathrm{sy}} \ldots \otimes^{\mathrm{sy}} M_n$. (Commutativity and associativity laws could be obtained up to strong bisimulation.)

3 Modelling Component Systems with MIOs: Example

We introduce a running example to explain our notions of interface and assembly, its greybox behaviour and encapsulation. We consider a simple cash desk application, inspired by [13]. Figure 1 shows the assembly CashDeskAssembly, which is a set of three interfaces, CashDeskGUI, CashDeskController and ClearingCompany. Each interface has input and output actions indicated by incoming and outgoing arrows on the frame of the interface which shows the interface's signature. Interfaces are connected by shared actions of complementary types. For instance, newSale, itemReady and finish are the shared actions of CashDeskGUI and CashDeskController where, e.g., newSale is an input for CashDeskGUI and an output of CashDeskController. The assembly itself has also a signature with communication actions given by the shared actions of the interfaces and

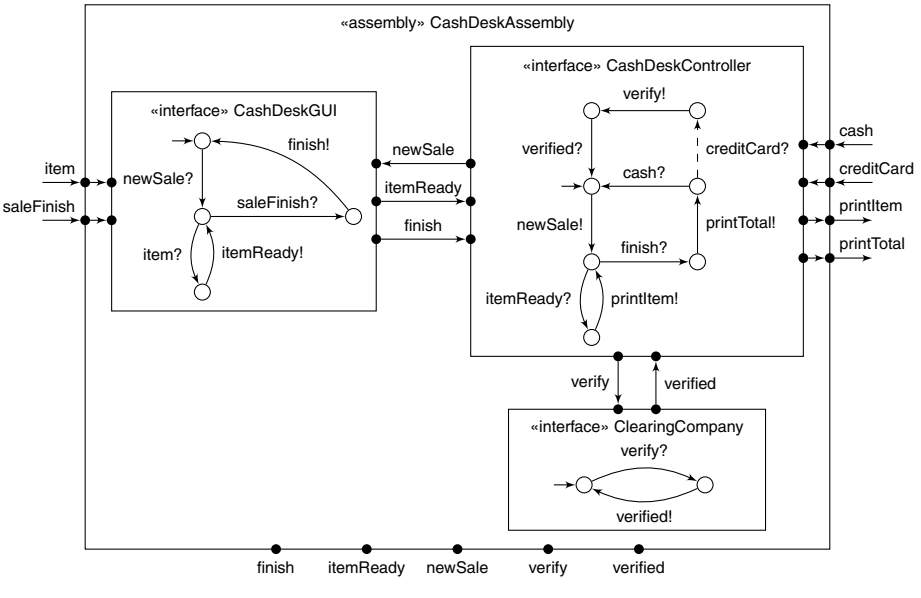


Fig. 1. Cash desk assembly with contained interfaces

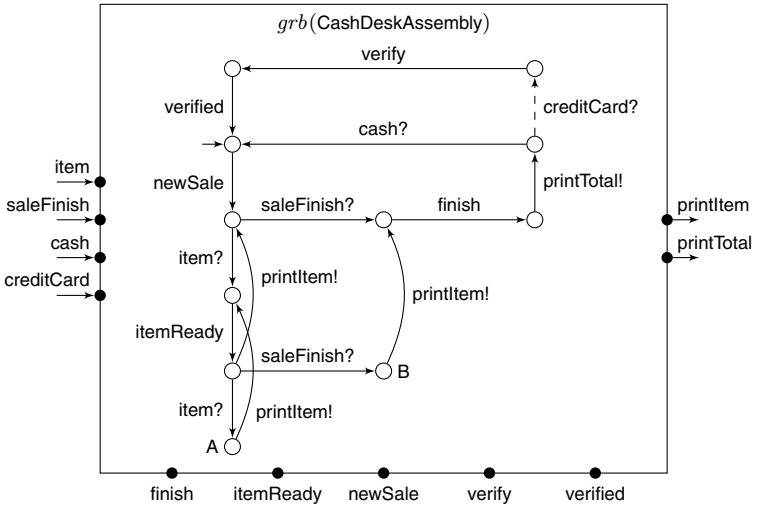


Fig. 2. Greybox behaviour of CashDeskAssembly in Fig. 1

with input/output actions given by the non-connected (open) actions of the interfaces. The communication actions are indicated by bullets on the frame of the assembly.

Each interface has a behaviour specification represented by a modal I/O-transition system with input/output labels but without communication labels. The I/O-labelling of the MIO determines the signature of the interface (shown on its frame). There may also be silent transitions labelled with τ, but those do not occur in the three example interfaces. In the drawings of the MIOs, labels suffixed with ? indicate inputs, those suffixed with ! outputs of the interface. The CashDeskGUI interface behaviour waits for a newSale? from the environment, then reacts to incoming item?s by issuing corresponding itemReady!s until a saleFinish? arrives, upon which it signals finish!. The CashDeskController interface behaviour starts each sale by issuing newSale! and then answers each itemReady? by printItem! until a finish? arrives, upon which a printTotal! is issued and either cash? or creditCard? is accepted. Only creditCard? is a may-transition, such that in a refinement of CashDeskController it may be absent or turned into a must-transition. The ClearingCompany waits for a verify? and then reacts with a verified!. For simplicity of presentation we have only specified the positive case where a credit card is validated.

To each assembly we associate a behaviour which is presented as a MIO with input, output, *and* communication labels. It is also called greybox behaviour since the communication labels are still visible, only τ actions possibly occurring in the contained interfaces of the assembly are hidden. Figure 2 shows the greybox behaviour of the CashDeskAssembly. It is constructed by the synchronous composition (see Sect. 2) of the three interface MIOs where communication happens if shared labels of complementary types match. The resulting communication labels are still visible but not usable for further input or output; i.e., we follow the binary communication scheme of interface automata and MIOs. Pictorially, the communication labels are drawn without a suffix on the transitions. Note that the signature of an assembly is determined by the I/O-labelling of its greybox view.

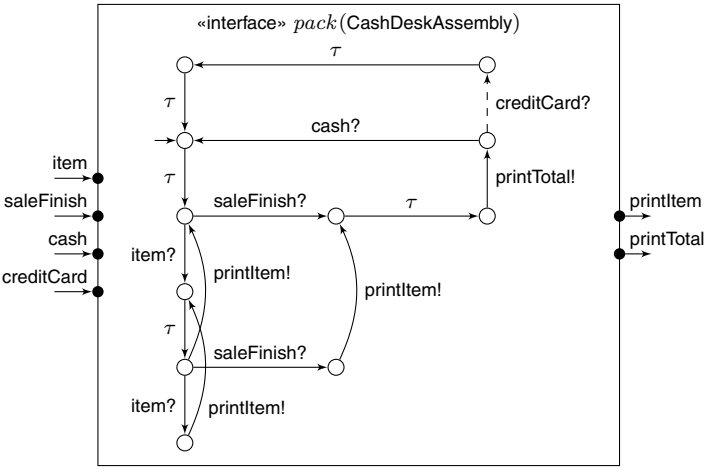


Fig. 3. Interface resulting from packing CashDeskAssembly in Fig. 1

An assembly can be encapsulated by "packing" its greybox view, i.e. by hiding all communication labels; see Sect. 2. Technically, this is achieved by replacing each transition with a communication label by a silent τ-transition. The result of assembly encapsulation yields an interface. Hence, assembly encapsulation is an important step for hierarchical system development. In our example, packing of the greybox behaviour of the CashDeskAssembly results in the interface shown in Fig. 3.

4 Interfaces and Assemblies

We now turn to a formal presentation of the notions of interface and assembly as motivated in the introduction and illustrated by the example. We discuss communication-safe assemblies and the refinement of interfaces and assemblies. In particular, we justify compositionality of refinement, preservation of communication-safety, and refinement encapsulation in our approach. We build on the definitions of MIOs, their composability and synchronous composition, as well as hiding of communication labels recalled in Sect. 2. All claims are reduced to corresponding lemmas for MIOs and their refinement; the proofs of these lemmas can be found in App. A.

The domain of *interfaces* $\mathcal{F}$ is constructed from all those MIOs whose I/O-labellings do not show communication labels. We write $F = intf(M)$ for an interface with underlying MIO M with I/O-labelling $L_M = (I_M, O_M, \emptyset)$. The *signature* of F is given by the I/O-labelling L_M and pictorially indicated in the examples on the frame of an interface. The labelling restriction reflects the blackbox characteristics of interfaces abstracting from communication. Two interfaces are *composable* if their underlying MIOs are composable, and a finitely indexed set $(F_i)_{1 \leq i \leq n}$ of interfaces is *composable* if the F_i are pairwise composable.

The domain of *assemblies* $\mathcal{A}$ is constructed from the composable finitely indexed sets of interfaces, and we write $A = asm((F_i)_{1 \leq i \leq n})$ for an assembly consisting of the

interfaces $F_1, \ldots, F_n$. Each assembly A is assigned its *greybox behaviour* $grb(A)$ which is the synchronous composition of the MIOs underlying the constituting interfaces of the assembly, formally

$$grb(asm((intf(M_i))_{1\leq i\leq n})) = \bigotimes^{\mathrm{sy}}_{1\leq i\leq n} M_i \ .$$

In particular, such a greybox behaviour leaves communications visible. The *signature* of an assembly A is given by the I/O-labelling $\bigotimes^{\mathrm{sy}}_{1\leq i\leq n} L_i$ of the MIO $\bigotimes^{\mathrm{sy}}_{1\leq i\leq n} M_i$.

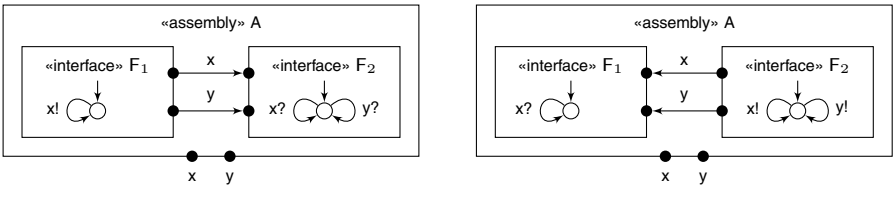
We also construct an interface $pack(A)$ from an assembly A which abstracts from the communication labels in the greybox view. For this purpose we use the hiding operator ξ on MIOs defined in Sect. 2:

$$pack(A) = intf(grb(A)\xi) \ .$$

According to the hiding operator in the signature of $pack(A)$ all communication labels of A have become τ.

4.1 Communication-Safe Assemblies

Our notion of communication-safe assembly is inspired by the notion of weak modal compatibility in [2]. This compatibility notion, as well as the compatibility notions in [6,7] and [10], relies on the assumption that outputs are autonomous and must be accepted by a communication partner while inputs are subject to external choice and need not to be served. Hence the discrimination of inputs and outputs is essential here. For instance, the two interfaces shown in the assembly in Fig. 4(a) are (strongly) compatible, since the output x! issued by the interface on the left-hand side will be (immediately) accepted by the input x? of the MIO on the right-hand side. However, if we consider the assembly in Fig. 4(b), then the two interfaces are not compatible, since the interface on the right-hand side can autonomously decide to output y! which cannot be accepted by the interface on the left-hand side.



(a) Compatible interfaces (b) Incompatible interfaces

Fig. 4. Autonomy of outputs

Strong modal compatibility is based on the idea that whenever one component wants to send an output it finds the communication partner in a state where it must take the corresponding input immediately. Weak modal compatibility is more liberal since it is sufficient if the communication partner must accept the message possibly after performing first some silent must-transitions. But simple examples show, see e.g. Ex. 1 below,

that this compatibility requirement is still too strong. Therefore we generalise weak compatibility further and allow the communication partner to take the input only after performing silent must-transitions *and/or* mandatory communications with other components of the assembly *and also* outputs which are a must and are directed outside of the assembly. This works well because, assuming communication-safe developments, these (open) outputs are again guaranteed to be taken (possibly after a delay) when an assembly is further extended. We show that communication-safe assemblies can indeed be built up in an incremental way. Moreover, our notion of communication-safety goes beyond the compatibility notions because it allows to study the compatibility of arbitrarily many interfaces within an assembly.

We base our definition of communication-safety on the corresponding notion of output compatibility for MIO-families. Assume given a composable finitely indexed set $(M_i)_{1\leq i\leq n}$ of MIOs and let M_j be an arbitrary MIO of the family. Then the rest of the family after omitting M_j, let us call it E_j, plays the role of the environment for M_j. We must ensure that in any reachable state of the product of the family, whenever M_j wants to send an output, then E_j must be able to take the corresponding input possibly after some autonomous must-transitions of E_j which do not concern communication with M_j. These autonomous transitions can be silent must-transitions or must-communication transitions or must-outputs of E_j which are not shared with the inputs of M_j.

Definition 1. *Let $(M_i)_{1\leq i\leq n}$ be a composable finitely indexed set of MIOs. For each j with $1 \leq j \leq n$, let $E_j = \bigotimes^{\mathrm{sy}}_{1\leq i\neq j\leq n} M_i$. The MIO-family $(M_i)_{1\leq i\leq n}$ is* output compatible *if for each $1 \leq j \leq n$, each $(s_1, \ldots, s_n) \in \mathscr{R}(\bigotimes^{\mathrm{sy}}_{1\leq i\leq n} M_i)$, and each $l \in O_{M_j} \cap I_{E_j}$ the following holds with $X_j = T_{E_j} \cup (O_{E_j} \setminus I_{M_j})$.*[4]

$$\exists s'_j \in S_{M_j} \,.\, s_j \xrightarrow{l}_{M_j} s'_j \Rightarrow \exists (s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_n) \in \mathscr{R}(E_j) \,.$$
$$(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n) \xrightarrow{\widehat{X}_j}_{E_j} \cdot \xrightarrow{l}_{E_j} (s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_n) \,. \qquad \square$$

An assembly $A = asm((intf(M_i))_{1\leq i\leq n})$ is *communication-safe*, denoted by $cs(A)$, if the family of MIOs $(M_i)_{1\leq i\leq n}$ is output compatible .

Example 1. Consider the CashDeskAssembly in Fig. 1. To check communication-safety we have to consider the assembly's greybox behaviour shown in Fig. 2. Crucial states are the states A and B. For instance, in state A the CashDeskGUI has reached its lowest state in Fig. 1 where it wants to send out itemReady! and the CashDeskController has also reached its lowest state in Fig. 1 where it can perform the open output printItem! to the environment of the assembly. Only after this output it can input, as requested, itemReady?. This would not be allowed by weak compatibility since printItem! is not silent. On the other hand, sending printItem! before accepting itemReady? is not a problem, because we can expect that the whole assembly will only be put in a communication-safe context, where we can again assume that the output printItem! will eventually be

[4] Note that T_{E_j} is the set of communication labels of E_j and $(O_{E_j} \setminus I_{M_j})$ is the set of output labels of E_j which are not shared with the input labels of M_j, i.e. not used for communication between E_j and M_j. The silent must-transitions are anyway subsumed in the notation $\xrightarrow{\widehat{X}_j}_{E_j}$; see Sect. 2.

accepted. A similar situation concerns state B of the assembly, where the CashDeskController accepts an output finish! of the CashDeskGUI only if it has performed an output of printItem! before. □

Let us still point out that communication-safety does not coincide with deadlock-freedom. Indeed deadlock-freedom is neither necessary nor sufficient. For instance, the assembly in Fig. 4(b) is not communication-safe but deadlock-free. On the other hand, the assembly in Fig. 5 is communication-safe but does deadlock immediately since none of the inputs is served which is not required by our notion of communication-safety. (Of course one can also imagine other variants of communication correctness where inputs must be served.)

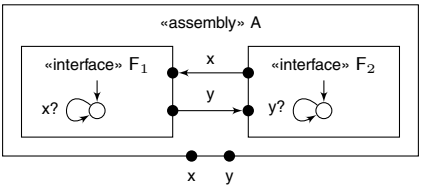


Fig. 5. Deadlocking interfaces

Communication-safety of an assembly can be shown incrementally, i.e. by enlarging the assembly by one interface at a time, each time checking that the assembly from the packed assembly up to now and the additional interface is communication-safe:

Proposition 1. *Let* $A = asm((F_i)_{1\leq i\leq n+1})$ *be an assembly. If* $A' = asm((F_i)_{1\leq i\leq n})$ *and* $asm(pack(A'), F_{n+1})$ *are communication-safe, then* A *is communication-safe.*

This claim follows immediately from a corresponding lemma for the underlying MIOs ensuring incremental checking of output compatibility:

Lemma 1. *Let* $(M_i)_{1\leq i\leq n+1}$ *be a composable finitely indexed set of MIOs. If* $(M_i)_{1\leq i\leq n}$ *and* $(\bigotimes^{\mathrm{sy}}_{1\leq i\leq k} M_i, M_{n+1})$ *are output compatible, then* $(M_i)_{1\leq i\leq n+1}$ *is output compatible.*

However, for guaranteeing communication-safety of an assembly $asm((F_i)_{1\leq i\leq n})$ it does not suffice to check pairwise communication-safety in the sense that each $asm(F_i, F_j)$ with $1 \leq i \neq j \leq n$ is communication-safe. Consider the assembly A in Fig. 6 consisting of three interfaces F_1, F_2, F_3. The three assemblies $asm(\mathsf{F}_1, \mathsf{F}_2)$, $asm(\mathsf{F}_1, \mathsf{F}_3)$, and $asm(\mathsf{F}_2, \mathsf{F}_3)$ are communication-safe; but $asm(\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3)$ is not communication-safe. For instance in the initial state of the whole assembly, F_1 wants to send x!, but the product of F_2 and F_3 does never take x? after autonomous actions which are not shared with F_1 (note that the output z! is shared).

4.2 Refinement of Interfaces and Assemblies

Refinement of interfaces and assemblies relies on the notion of weak modal refinement for MIOs [9]. The basic idea of *modal* refinement is that required (*must*) transitions

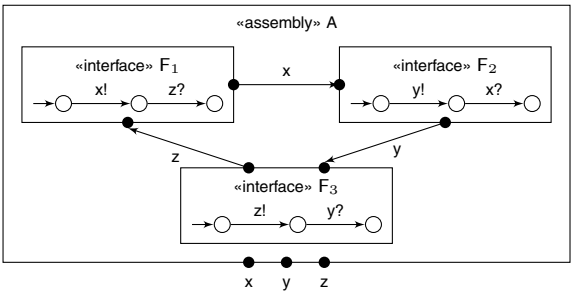


Fig. 6. Communication-safety does not follow from pairwise communication-safety

of an abstract specification must also occur in the concrete specification. Conversely, allowed (*may*) transitions of the concrete specification must be allowed by the abstract specification. The refinement relation is weak, because it supports observational abstraction, i.e., silent τ-transitions can be dropped and inserted as long as the modalities and the simulation relation are preserved. We first extend the notion of weak modal refinement given in [9] in a straightforward way to MIOs with communication labels. Like in [9], weak modal refinement abstracts from internal actions, but transitions with communication labels must be respected in the same way as input/output actions.

Definition 2. *Let M and N be MIOs with the same I/O-labelling. M* weakly modally refines *N, written $M \leq^*_{\mathrm{m}} N$, if there exists a relation $R \subseteq S_M \times S_N$ containing $(s_{0,M}, s_{0,N})$ such that for all $(s_M, s_N) \in R$:*

1. $\forall l \in \bigcup L_N, s'_N \in S_N \,.\, s_N \xrightarrow{l}_N s'_N \Rightarrow \exists s'_M \in S_M \,.\, s_M \xrightarrow{\widehat{l}}_M s'_M \wedge (s'_M, s'_N) \in R$
2. $\forall s'_N \in S_N \,.\, s_N \xrightarrow{\tau}_A s'_N \Rightarrow \exists s'_M \in S_M \,.\, s_M \xrightarrow{\widehat{\tau}}_M s'_M \wedge (s'_M, s'_N) \in R$
3. $\forall l \in \bigcup L_M, s'_M \in S_M \,.\, s_M \xrightarrow{l}_M s'_M \Rightarrow \exists s'_N \in S_N \,.\, s_N \xrightarrow{\widehat{l}}_N s'_N \wedge (s'_M, s'_N) \in R$
4. $\forall s'_M \in S_M \,.\, s_M \xrightarrow{\tau}_M s'_M \Rightarrow \exists s'_N \in S_N \,.\, s_N \xrightarrow{\widehat{\tau}}_N s'_N \wedge (s'_M, s'_N) \in R$

A MIO M co-simulates *a MIO N, written $M =^*_{\mathrm{m}} N$, if $M \leq^*_{\mathrm{m}} N$ and $N \leq^*_{\mathrm{m}} M$.* □

Note that $\leq^*_{\mathrm{m}}$ is reflexive and transitive. If all transitions of M are must-transitions co-simulation corresponds to weak bisimulation; if all transitions of M are may-transitions it is classical co-simulation.

Since interface refinement has to respect blackbox behaviours, and since interfaces do not show any communication labels, the notion of weak modal refinement is directly applicable to define interface refinement. An interface $F = intf(M)$ *refines* an interface $G = intf(N)$, written as $F \preccurlyeq_{\mathrm{intf}} G$, if $M \leq^*_{\mathrm{m}} N$. The interfaces F and G are *equivalent*, written $F \approx_{\mathrm{intf}} G$, if $F \preccurlyeq_{\mathrm{intf}} G$ and $G \preccurlyeq_{\mathrm{intf}} F$, i.e., $M =^*_{\mathrm{m}} N$.

Example 2. The interface $pack(\mathsf{CashDeskAssembly})$ in Fig. 3 was obtained by hiding the communication labels from the greybox behaviour of the CashDeskAssembly. Figure 7 shows an equivalent but "smaller" interface behaviour. For this purpose one has to prove two refinement relations $pack(\mathsf{CashDeskAssembly}) \preccurlyeq_{\mathrm{intf}} min(pack(\mathsf{CashDeskAssembly}))$ and vice versa. Indeed both directions have been verified with the MIO-Workbench [2]. Note that for the first direction observational abstraction w.r.t. τ-transitions on the refinement side, i.e. on $pack(\mathsf{CashDeskAssembly})$,

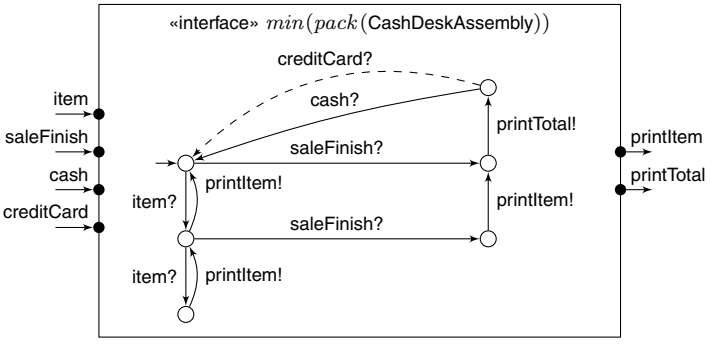


Fig. 7. Interface equivalent to the result from packing CashDeskAssembly in Fig. 3

is necessary which would not work for alternating simulation of interface automata. We believe that the interface $min(pack(\text{CashDeskAssembly}))$, as the name suggests, is minimal. However, whether minimal behaviours for equivalent interfaces exist and how they can be computed is an open issue for future research. □

Assembly refinement has to respect the architectural requirements and the greybox behaviour of assemblies. The first requirement amounts to relate the interfaces of assemblies pairwise by interface refinement; the latter amounts to relate the greybox behaviours of assemblies by means of weak modal refinement, which abstracts away silent transitions (which stem from inner interfaces), but transitions resulting from communications are considered as visible and indeed they are respected by our generalised notion of weak modal refinement. In summary, an assembly $A = asm((F_i)_{1\leq i\leq m})$ *refines* an assembly $B = asm((G_i)_{1\leq i\leq n})$, written as $A \preccurlyeq_{\text{asm}} B$, if (1) $m = n$ and $F_i \preccurlyeq_{\text{intf}} G_i$ for all $1 \leq i \leq m$ and (2) $grb(A) \leq^*_{\text{m}} grb(B)$. In fact, the first condition implies already the second one which is a consequence of the following lemma:

Lemma 2. *Let $(M_i)_{1\leq i\leq n}$ and $(N_i)_{1\leq i\leq n}$ be composable finitely indexed sets of MIOs such that $M_i \leq^*_{\text{m}} N_i$ for all $1 \leq i \leq n$. Then $\bigotimes^{\text{sy}}_{1\leq i\leq n} M_i \leq^*_{\text{m}} \bigotimes^{\text{sy}}_{1\leq i\leq n} N_i$.*

Moreover, the lemma shows that our claim of *compositionality of refinement* in Sect. 1.3 — refinements of the interfaces constituting assemblies induce assembly refinements — is indeed valid in our approach:

Proposition 2. *Let $(F_i)_{1\leq i\leq n}$ and $(G_i)_{1\leq i\leq n}$ be finitely indexed sets of interfaces with $F_i \preccurlyeq_{\text{intf}} G_i$ for all $1 \leq i \leq n$. Then $asm((F_i)_{1\leq i\leq n}) \preccurlyeq_{\text{asm}} asm((G_i)_{1\leq i\leq n})$.*

The rule of *preservation of communication-safety* stated in Sect. 1.3 requires that each refinement of a communication-safe assembly is again communication-safe. Indeed, also this rule is valid here. The proof relies on the fact that must-transitions are preserved by refinements.

Proposition 3. *If $cs(B)$ and $A \preccurlyeq_{\text{asm}} B$, then $cs(A)$.*

The proof of this proposition is reduced to a corresponding lemma for the preservation of output compatibility w.r.t. weak modal refinements.

Lemma 3. *Let $(M_i)_{1\leq i\leq n}$ and $(N_i)_{1\leq i\leq n}$ be composable finitely indexed sets of MIOs such that $M_i \leq^*_{\mathrm{m}} N_i$ for all $1 \leq i \leq n$. If $(N_i)_{1\leq i\leq n}$ is output compatible, then also $(M_i)_{1\leq i\leq n}$ is output compatible.*

Finally, we also obtain the (strong) version of *refinement encapsulation* discussed in Sect. 1.3 that assembly refinements induce interface refinements of their packings:

Proposition 4. *If $A \preccurlyeq_{\mathrm{asm}} B$, then $pack(A) \preccurlyeq_{\mathrm{intf}} pack(B)$.*

The proof of this proposition relies on Lem. 2 and the following simple observation that hiding preserves weak modal refinement:

Lemma 4. *If $C \leq^*_{\mathrm{m}} A$, then $C\xi \leq^*_{\mathrm{m}} A\xi$.*

5 Case Study

We will illustrate how our techniques work in terms of a (small) top-down development of the cash desk application. Figure 8 gives an overview of the different steps and their proof obligations. We start by an abstract requirements specification of the whole system which is given by the interface CashDesk in Fig. 9. The specification is rather loose having only a single must-transition requiring cash payment to be possible in any system implementation whenever a printTotal! has been performed before. The other transitions are may-transitions. At the start of a sale arbitrarily many items may be taken and printed; note that only as many printItem!s should be performed as item?s have been taken before, but this cannot be specified with finite state. Also a saleFinish? request may be accepted, possibly followed by printing items (that have not been printed yet) and then printing the total. Instead of cash payment, payment by credit card may be allowed by an implementation.

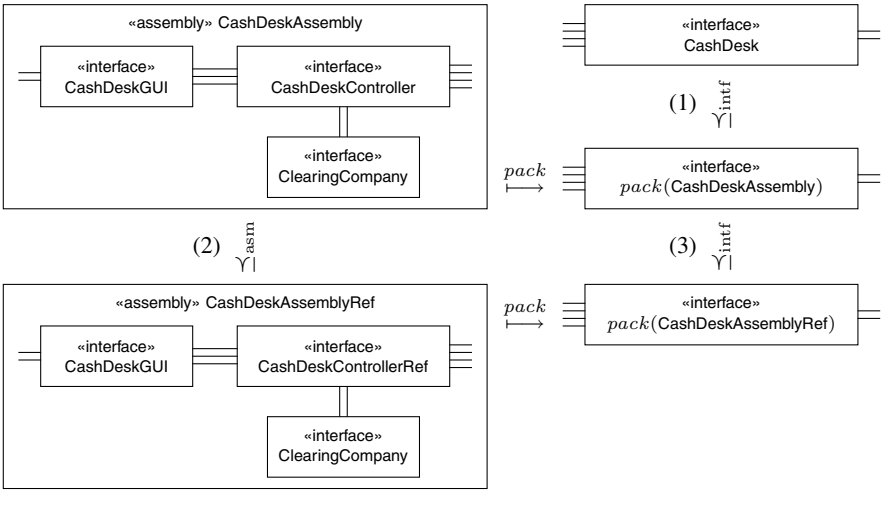


Fig. 8. Overview of top-down development of the cash desk application

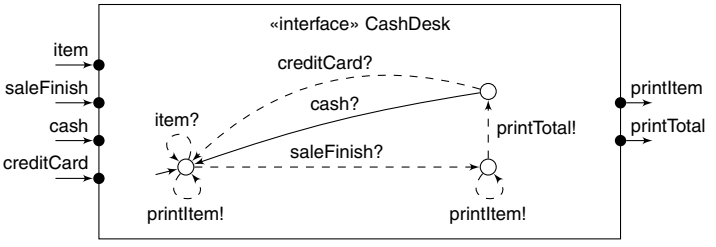


Fig. 9. Interface CashDesk

In the next step we specify an architecture for the intended system, which is given by the CashDeskAssembly known from Sect. 3, Fig. 1. As a first proof obligation, we have to show that the behaviour induced by the proposed assembly fits to the abstract requirements specification of the system. Formally this means that the interface of the encapsulated assembly is an interface refinement of CashDesk, i.e., that (1) in Fig. 8 is satisfied. For the proof it is obviously sufficient (cf. Sect. 4.2) to consider the minimised version of the interface shown in Fig. 7 and to prove $min(pack(\text{CashDeskAssembly})) \preccurlyeq_{\text{intf}} \text{CashDesk}$. We have verified this statement with the MIO-Workbench.[5]

The CashDeskAssembly introduces architectural requirements *and* behavioural requirements in terms of the greybox behaviour of the assembly shown in Fig. 2. In our third step this assembly is refined by the assembly CashDeskAssemblyRef where the interface CashDeskController is replaced by the interface CashDeskControllerRef. The latter has the same behaviour specification as CashDeskController (see Fig. 1) but the previous may-transition for creditCard? is turned into a must-transition. Obviously, CashDeskControllerRef $\preccurlyeq_{\text{intf}}$ CashDeskController and therefore, by compositionality of refinement as stated in Sect. 4.2, we get the proof obligation (2) in Fig. 8. Since CashDeskAssembly is communication-safe, (2) implies that CashDeskControllerRef is communication-safe as well; cf. Sect. 4.2. Moreover, encapsulation of assemblies turns assembly (greybox) refinement into interface (blackbox) refinement (cf. Sect. 4.2), and therefore we obtain (3) in Fig. 8. Now we can utilise that interface refinement is transitive to be sure that the visible behaviour of the encapsulated assembly CashDeskAssemblyRef is conform to the system's interface specification.

Finally, we would like to emphasise the significance of proper assembly refinement, i.e. the importance of respecting communications during assembly refinement. Imagine that we would use instead of CashDeskAssemblyRef an assembly CashDeskAssemblyRef' where the interface CashDeskControllerRef is replaced by the interface CashDeskControllerRef' shown in Fig. 10. This interface accepts creditCard? without initiating a subsequent verification of the card with the clearing company. Obviously, CashDeskControllerRef' is not an interface refinement of CashDeskController and also CashDeskAssemblyRef' is not an assembly refinement of CashDeskAssembly since the required communications with the clearing company, see Fig. 2, do not

[5] It may be interesting to remark, that the refinement would not hold, if at least one of the remaining inputs of the CashDesk interface would be a must-transition; hence we could also not get an alternating simulation relation here.

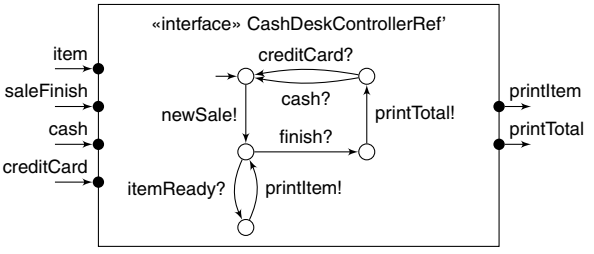


Fig. 10. Interface CashDeskControllerRef'

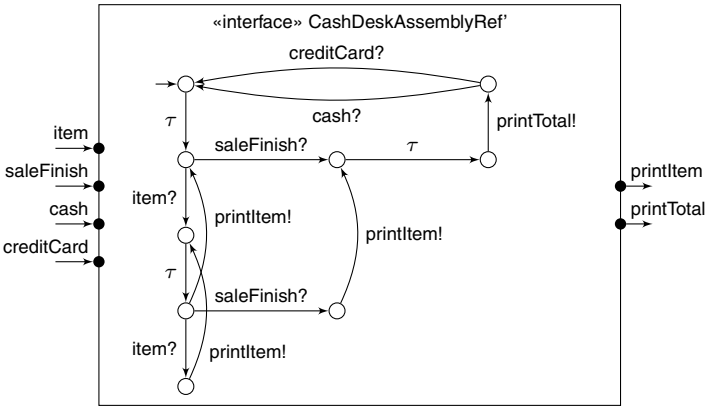


Fig. 11. Interface resulting from packing CashDeskAssemblyRef'

happen. However, if we would not check the assembly refinement but hide immediately all communications, which happens in approaches based on interface composition, then we obtain the interface $pack(\mathsf{CashDeskAssemblyRef'})$ shown in Fig. 11 which is obviously an interface refinement of $pack(\mathsf{CashDeskAssembly})$; cf. Fig. 3. Thus the assembly CashDeskAssemblyRef' with no credit card verification could be used for the implementation of the system, which is certainly not intended.

6 Conclusion

Our study is motivated by an extension of the abstract rule of independent implementability of interface languages [7] to take into account architectural information given by interface assemblies. We have deliberately kept our approach to assemblies simple, not involving further constructs like connectors, but we believe that our fundamental research can be successfully applied to component models as well. As a concrete formalism we have chosen modal I/O-transition systems which we have adopted to take into account not only interface specifications and interface (black-box) refinement but also interface assemblies and assembly (grey-box) refinement. To our knowledge such an extension of the interface theory of MIOs did not exist yet. Also our notion of communication-safe assembly is an extension of previous work on compatibility of

interfaces which we claim is strongly needed in practical examples. Of course many approaches in the literature support assemblies, i.e. networks of interface specifications and hierarchical constructions, in one or the other way. Among those based on labelled transition systems we want to mention interaction automata [5], pNets [1], symbolic transition systems (STS) [8], PADL [3], and communicating finite state machines (CFSMs) [4]. CFSMs are based on asynchronous communication and introduce a notion of unspecified reception which is related to (a stronger version of) communication-safety not allowing "open" outputs. Otherwise usually deadlock checks are performed for assemblies which, however, are neither sufficient nor necessary for communication-safety, at least in our sense. Assembly refinement, however, is not a concern in these approaches. Most closely related to our result on compositional refinement is the interaction automata approach which studies substitutability of components w.r.t. a behavioural equivalence which can be tuned, e.g., to keep communications visible. In future work we want to investigate to what extent our notion of assembly refinement can be relaxed such that architectures can change in a controlled way. We are also interested to transfer our results to asynchronous communication, hybrid systems, and interfaces for components with data states.

Acknowledgements. We want to thank Sebastian Bauer for checking the examples with the MIO-Workbench, and Stephan Janisch for fruitful discussions on communication-safety.

References

1. Barros, T., Ameur-Boulifa, R., Cansado, A., Henrio, L., Madelaine, E.: Behavioural models for distributed Fractal components. Ann. Télécom. 64(1-2), 25–43 (2009)
2. 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)
3. Bernardo, M., Ciancarini, P., Donatiello, L.: Architecting families of software systems with process algebras. ACM Trans. Softw. Eng. Methodol. 11(4), 386–426 (2002)
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
5. Cerná, I., Vareková, P., Zimmerova, B.: Component substitutability via equivalencies of component-interaction automata. Electr. Notes Theor. Comput. Sci. 182, 39–55 (2007)
6. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proc. 9th ACM SIGSOFT Ann. Symp. Foundations of Software Engineering (FSE 2001), pp. 109–120 (2001)
7. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, C.A.R. (eds.) Engineering Theories of Software-intensive Systems. NATO Science Series: Mathematics, Physics, and Chemistry, vol. 195, pp. 83–104. Springer, Heidelberg (2005)
8. 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)
9. Hüttel, H., Larsen, K.G.: The use of static constructs in a modal process logic. In: Meyer, A.R., Taitslin, M.A. (eds.) Logic at Botik. LNCS, vol. 363, pp. 163–180. Springer, Heidelberg (1989)

10. Larsen, K.G., Nyman, U., Wąsowski, 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)
11. Larsen, K.G., Thomsen, B.: A modal process logic. In: Proc. 3rd Ann. Symp. Logic in Computer Science (LICS 1988), pp. 203–210. IEEE Computer Society, Los Alamitos (1988)
12. Plášil, F., Višňovský, S.: Behavior protocols for software components. IEEE Trans. Software Eng. 28(11), 1056–1076 (2002)
13. Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.): The Common Component Modeling Example. LNCS, vol. 5153. Springer, Heidelberg (2008)

A Proofs

Proof (of Lem. 1). For each $1 \leq j \leq n+1$ let $E_j = \bigotimes^{\mathrm{sy}}_{1 \leq i \neq j \leq n+1} M_i$. For each $1 \leq j \leq n$ let $E'_j = \bigotimes^{\mathrm{sy}}_{1 \leq i \leq j \leq n}$. Let $(s_1, \ldots, s_{n+1}) \in \mathscr{R}(\bigotimes^{\mathrm{sy}}_{1 \leq i \leq n+1} M_i)$ and $s_j \xrightarrow{l}_{M_j} s'_j$ for $l \in O_{M_j} \cap I_{E_j}$.

Let $j = n+1$. Then $E_j = \bigotimes^{\mathrm{sy}}_{1 \leq i \leq n} M_i$, and the claim follows since E_{n+1} and M_{n+1} are output compatible by assumption. Now let $1 \leq j \leq n$.

If $l \in O_{M_j} \cap I_{M_{n+1}}$, then $s_{n+1} \xrightarrow{\widehat{X}}_{M_{n+1}} \cdot \xrightarrow{l}_{M_{n+1}} s'_{n+1}$ with $X = T_{M_{n+1}} \cup (O_{M_{n+1}} \setminus I_{E_{n+1}})$ since E_{n+1} and M_{n+1} are output compatible. Thus $(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{n+1}) \xrightarrow{\widehat{X}}_{E_j} \cdot \xrightarrow{l}_{E_j} (s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s'_{n+1})$ and $X \subseteq T_{E_j} \cup (O_{E_j} \setminus I_{M_j})$, since $T_{M_{n+1}} \subseteq T_{E_j}$, $O_{M_{n+1}} \setminus I_{E_{n+1}} \subseteq O_{E_j}$, and $(O_{M_{n+1}} \setminus I_{E_{n+1}}) \cap I_{M_k} = \emptyset$ for all $1 \leq k \leq n$.

If $l \in O_{M_j} \cap (\bigcup_{1 \leq i \neq j \leq n} I_i)$, then $(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n) \xrightarrow{\widehat{X}'_j}_{E'_j} \cdot \xrightarrow{l}_{E'_j} (s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_n)$ with $X'_j = T_{E'_j} \cup (O_{E'_j} \setminus I_{M_j})$ by the output compatibility of $(M_i)_{1 \leq i \leq n}$. Let $(s''_1, \ldots, s''_{j-1}, s''_{j+1}, \ldots, s''_n) \xrightarrow{l'}_{E'_j} (s'''_1, \ldots, s'''_{j-1}, s'''_{j+1}, \ldots, s'''_n)$ with $l' \in O_{M_k} \cap I_{M_{n+1}}$ for some $1 \leq k \leq n$ be the first transition to occur in this sequence. Then $s_{n+1} \xrightarrow{\widehat{X}}_{M_{n+1}} \cdot \xrightarrow{l'}_{M_{n+1}} s'_{n+1}$ with $X = T_{M_{n+1}} \cup (O_{M_{n+1}} \setminus I_{E_{n+1}})$ since E_{n+1} and M_{n+1} are output compatible. As argued in the previous case $(s''_1, \ldots, s''_{j-1}, s''_{j+1}, \ldots, s_{n+1}) \xrightarrow{\widehat{X}}_{E_j} \cdot \xrightarrow{l'}_{E_j} (s''_1, \ldots, s''_{j-1}, s''_{j+1}, \ldots, s'_{n+1})$ and thus inductively $(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{n+1}) \xrightarrow{\widehat{X}_j}_{E_j} \cdot \xrightarrow{l}_{E_j} (s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_{n+1})$ with $X_j = T_{E_j} \cup (O_{E_j} \setminus I_{M_j})$. □

Proof (of Lem. 2). We show that if M, N, and L are MIOs such that N and L are composable and $M \leq^*_{\mathrm{m}} N$, then $M \otimes^{\mathrm{sy}} L \leq^*_{\mathrm{m}} N \otimes^{\mathrm{sy}} L$. Then the claim follows by induction and symmetry.

Let such M, N, and L be given, let $ML = M \otimes^{\mathrm{sy}} L$, $NL = N \otimes^{\mathrm{sy}} L$, and let R_{MN} be a relation witnessing $M \leq^*_{\mathrm{m}} N$ with $(s_{0,M}, s_{0,N}) \in R_{MN}$. Let

$$R = \{((s_M, s_L), (s_N, s_L)) \mid (s_M, s_N) \in R_{MN}\}.$$

Then $((s_{0,M}, s_{0,L}), (s_{0,N}, s_{0,L})) \in R$. Let $((s_M, s_L), (s_N, s_L)) \in R$; we check the conditions of weak modal refinement for R:

(1) Let $l \in I_{NL} \cup O_{NL} \cup T_{NL}$ and $(s_N, s_L) \xrightarrow{l}_{NL} (s'_N, s'_L)$. If $l \in (I_N \setminus O_L) \cup (O_N \setminus I_L) \cup T_N$, then $s_N \xrightarrow{l}_N s'_N$ and $s_L = s'_L$. By (1) for R_{MN} there is an $s'_M \in S_M$ with $s_M \xrightarrow{\widehat{l}}_M s'_M$ and, in particular, $(s_M, s_L) \xrightarrow{\widehat{l}}_{ML} (s'_M, s_L)$; thus $((s'_M, s_L), (s'_N, s_L)) \in R$. — If $l \in (I_L \setminus O_N) \cup (O_L \setminus I_N) \cup T_L$, then $s_N = s'_N$ and $s_L \xrightarrow{l}_L s'_L$. Thus $(s_M, s_L) \xrightarrow{l}_{ML} (s_M, s'_L)$ and hence $((s_M, s'_L), (s_N, s'_L)) \in R$. — If $l \in (I_N \cap O_L) \cup (O_N \cap I_L)$, then $s_N \xrightarrow{l}_N s'_N$ and $s_L \xrightarrow{l}_L s'_L$ for some $s'_N \in S_N$ and $s'_L \in S_L$. By (1) for R_{MN} there is an $s'_M \in S_M$ with $s_M \xrightarrow{\widehat{l}}_M s'_M$ and, in particular, $(s_M, s_L) \xrightarrow{\widehat{l}}_{ML} (s'_M, s'_L)$; thus $((s'_M, s'_L), (s'_N, s'_L)) \in R$.

(2) Let $(s_N, s_L) \xrightarrow{\tau}_{NL} (s'_N, s'_L)$. If $s_N \xrightarrow{\tau}_N s'_N$ and $s_L = s'_L$, then $s_M \xrightarrow{\widehat{\tau}}_M s'_M$ for some $s'_M \in S_M$ by (2) for R_{MN}; in particular, $(s'_M, s_L) \xrightarrow{\widehat{\tau}}_{ML} (s'_M, s_L)$ and hence $((s'_M, s_L), (s'_N, s_L)) \in R$. — If $s_N = s'_N$ and $s_L \xrightarrow{\tau}_L s'_L$ then $(s_M, s'_L) \xrightarrow{\widehat{\tau}}_{ML} (s_M, s'_L)$ and hence $((s_M, s'_L), (s_N, s'_L)) \in R$.

(3) Let $l \in I_{ML} \cup O_{ML} \cup T_{ML}$ and $(s_M, s_L) \dashrightarrow_{ML}^{l} (s'_M, s'_L)$. If $l \in (I_M \setminus O_L) \cup (O_M \setminus I_L) \cup T_M$, then $s_M \dashrightarrow_N^{l} s'_M$ and $s_L = s'_L$. By (3) for R_{MN} there is an $s'_N \in S_N$ with $s_N \dashrightarrow_N^{\widehat{l}} s'_N$ and, in particular, $(s_N, s_L) \dashrightarrow_{NL}^{\widehat{l}} (s'_N, s_L)$; thus $((s'_M, s_L), (s'_N, s_L)) \in R$. — If $l \in (I_L \setminus O_M) \cup (O_L \setminus I_M) \cup T_L$, then $s_M = s'_M$ and $s_L \dashrightarrow_L^{l} s'_L$. Thus $(s_N, s_L) \dashrightarrow_{NL}^{l} (s_M, s'_L)$ and hence $((s_M, s'_L), (s_N, s'_L)) \in R$. — If $l \in (I_M \cap O_L) \cup (O_M \cap I_L)$, then $s_M \dashrightarrow_M^{l} s'_M$ and $s_L \dashrightarrow_L^{l} s'_L$ for some $s'_M \in S_M$ and $s'_L \in S_L$. By (3) for R_{MN} there is an $s'_N \in S_N$ with $s_N \xrightarrow{\widehat{l}}_N s'_N$ and, in particular, $(s_N, s_L) \dashrightarrow_{NL}^{\widehat{l}} (s'_N, s'_L)$; thus $((s'_M, s'_L), (s'_N, s'_L)) \in R$.

(4) Let $(s_M, s_L) \xrightarrow{\tau}_{ML} (s'_M, s'_L)$. If $s_M \dashrightarrow_M^{\tau} s'_M$ and $s_L = s'_L$, then $s_N \dashrightarrow_N^{\widehat{\tau}} s'_N$ for some $s'_N \in S_N$ by (4) for R_{MN}; in particular, $(s'_N, s_L) \dashrightarrow_{NL}^{\widehat{\tau}} (s'_N, s_L)$ and hence $((s'_M, s_L), (s'_N, s_L)) \in R$. — If $s_M = s'_M$ and $s_L \dashrightarrow_L^{\tau} s'_L$ then $(s_N, s'_L) \dashrightarrow_{ML}^{\widehat{\tau}} (s_N, s'_L)$ and hence $((s_M, s'_L), (s_N, s'_L)) \in R$. □

Proof (of Lem. 3). Let $R_1, \ldots, R_n$ be the weakly modal refinement relations witnessing $M_1 \leq^*_{\mathrm{m}} N_1, \ldots, M_n \leq^*_{\mathrm{m}} N_n$, respectively. Let $D_j = \bigotimes^{\mathrm{sy}}_{1 \leq i \neq j \leq n} M_i$ and $E_j = \bigotimes^{\mathrm{sy}}_{1 \leq i \neq j \leq n} N_i$ for all $1 \leq i \leq n$. Let $(s_1, \ldots, s_n) \in \mathscr{R}(\bigotimes^{\mathrm{sy}}_{1 \leq i \leq n} M_i)$ and let $l \in O_{M_j} \cap I_{D_j}$ for some $1 \leq j \leq n$ such that $s_j \dashrightarrow_{M_j}^{l} s''_j$ for some $s''_j \in S_{M_j}$. Then there are $s'_1 \in S_{N_1}, \ldots, s'_n \in S_{N_n}$ with $(s_1, s'_1) \in R_1, \ldots, (s_n, s'_n) \in R_n$ and $(s'_1, \ldots, s'_n) \in \mathscr{R}(\bigotimes^{\mathrm{sy}}_{1 \leq i \leq n} N_i)$ by conditions (3) and (4) of weakly modal refinement relations. Moreover, $s'_j \dashrightarrow_{N_j}^{l} s'''_j$ for some $s'''_j \in S_{N_j}$ by condition (3). Thus, $(s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_n) \xrightarrow{\widehat{Y}_j}_{E_j} \cdot \xrightarrow{l}_{E_j} (s'''_1, \ldots, s'''_{j-1}, s'''_{j+1}, \ldots, s'''_n)$ with $Y_j = T_{E_j} \cup (O_{E_j} \setminus I_{N_j})$ by output compatibility of $(N_i)_{1 \leq i \leq n}$. Now $Y_j = T_{D_j} \cup (O_{D_j} \setminus I_{M_j})$. From conditions (1) and (2) of weakly modal refinement relations it follows that $(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n) \xrightarrow{\widehat{Y}_j}_{D_j} \cdot \xrightarrow{l}_{D_j} (s''_1, \ldots, s''_{j-1}, s''_{j+1}, \ldots, s''_n)$. □

Proof (of Lem. 4). Let R be a relation witnessing $C \leq^*_{\mathrm{m}} A$. Then R is also a relation witnessing $C\xi \leq^*_{\mathrm{m}} A\xi$. □