Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 264 (2010) 125–139
www.elsevier.com/locate/entcs

A Strict-Observational Interface Theory for Analysing Service Orchestrations

Philip Mayer[1] Andreas Schroeder[2] Sebastian S. Bauer[3]

Lehrstuhl für Programmierung und Softwaretechnik
Institut für Informatik
Ludwig-Maximilians-Universität
München, Germany

Abstract

Service oriented computing is an accepted architectural style for developing large, distributed software systems. A particular promise of such architectures is service orchestration, i.e. the ability to combine existing services to create more complex functionality, thereby yielding new services. In this paper, we discuss application-level protocol compliance checking of service orchestrations and service protocols using the semantic domain of modal input/output automata (MIOs). Based on a practical example, we motivate and introduce new notions of refinement and compatibility, and prove that they constitute a valid interface theory. With this domain-specific interface theory, we provide a framework for application-level analysis of service orchestrations, thus complementing existing work on compatibility analysis. Our theory is tool-supported through the MIO Workbench, a verification tool for modal input/output automata.

Keywords: protocol compliance, interface theory, refinement, compatibility, modal input/output automata, SOA, services, service orchestrations, service protocols, protocol breach

1 Introduction

With the advent of Service-Oriented Architectures (SOAs), the enterprise software landscape has been transformed from interactions of proprietary, closed components to an open and standardised communication between individual, self-describing components called services. Of particular importance in this context is the composition of individual services to form a new service, which has come to be known as service orchestration. Using techniques from the model-driven community, service-based systems can be modelled in UML or other user-friendly, graphical or non-graphical languages and later be transformed to platform-specific models and code.

[1] E-Mail: mayer@pst.ifi.lmu.de
[2] E-Mail: schroeda@pst.ifi.lmu.de
[3] E-Mail: bauerse@pst.ifi.lmu.de

doi:10.1016/j.entcs.2010.07.009

Using models opens up the possibility of verifying system correctness early in the development process, where design problems can be more easily dealt with. In this paper, we are interested in one aspect of such verification, namely checking protocol compliance of service orchestrations – a question to be asked on the application layer of the SOA stack (which, in lower levels, also includes the network, operating system, and middleware).

Our aim is supporting end users, i.e. developers of SOA systems, by providing assistance with regard to two questions:

- Does an orchestration (implementation) follow a certain protocol?
- Are two protocols compatible?

For answering these questions, we introduce a formal framework – an interface theory based on modal input/output automata (MIOs) [19], which enables us to precisely specify both SOA systems and the questions asked of them. The theory must be able to address the following requirements:

A Focus on Orchestrations

A service orchestration is characterised by the fact that it describes the interactions of multiple existing services, thereby forming one or more new services. In general, an orchestration uses many existing services, requiring each of them to fulfil a certain protocol, which is thus a required protocol of the orchestration; secondly, it may also provide multiple services itself, leading to multiple provided protocols of an orchestration. We are interested in verifying if an orchestration adheres to each protocol individually, and whether two such protocols are compatible. This is in the same spirit as Knapp et al. [18], who apply the same approach in the context of component-based software engineering.

Application-Level Protocol Breach Analysis

Correctly implementing a given protocol in a service orchestration is not an easy task. Our aim, therefore, is supporting developers in this task by providing application-level protocol breach analysis. This type of analysis focusses exclusively on protocol actions, which corresponds to the layer of abstraction the developer is interested in. It explicitly does not cover lower-level implementation issues such as race conditions.

Assured Compositionality

Thirdly, the analysis of service protocols and orchestrations should have some tangible result for the developer. An important point of service-oriented architectures is the ability to switch between different services with a minimum amount of effort. Therefore, the analysis should give us the assurance that if an orchestration works with a certain protocol, it can work with any service implementation which corresponds to the same protocol. This property has come to be known as *independent implementability*.

Our contribution in this paper is the definition of an appropriate interface theory for checking protocol compliance of service protocols and orchestrations, which meets all the requirements mentioned above. We have successfully applied this theory to several case studies in the EU project SENSORIA [32]. As a running example, we therefore use a scenario from one of the case study of SENSORIA. The scenario we present here is taken from the *eUniversity* domain, and models thesis management, i.e. the process of coordinating a student thesis (bachelor, master, diploma).

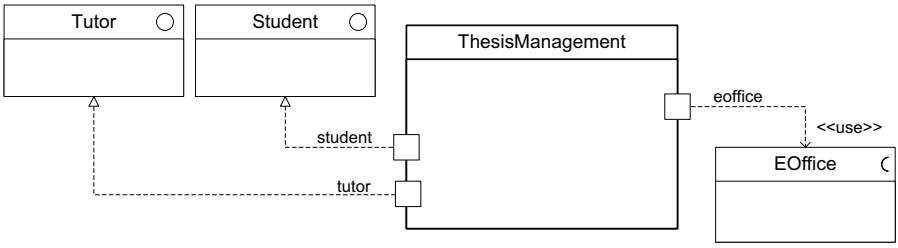

Figure 1. A service orchestration

Figure 1 shows the architectural sketch: An orchestration (`ThesisManagement`) provides two services (`Student` and `Tutor`) and requires one external service (`EOffice`).

The remaining sections are organised as follows. In Section 2, we recall the fundamental notion of interface theories and their most important properties, and moreover, modal I/O automata which serve as the formal basis of our approach. Then, in Section 3, we recapitulate the standard definition of weak modal refinement which is shown to be insufficient for our needs. As a remedy, we define a new refinement notion called *strict-observational* modal refinement which supports analysis of service orchestrations (Sect. 3.1). In the same fashion (Sect. 3.2), we address compatibility of protocols: an existing weak compatibility notion is shown not to be satisfactory; instead, we define a new notion of compatibility which is geared towards formulating application-level protocol conformance. Then (Sect. 3.3) we prove that the new notions of refinement and compatibility together with an adequate composition operator form a valid interface theory. In Section 4, we compare our approach with related work. We conclude in Section 5.

2 Interface Theories and Modal I/O Automata

In this section, we recall preliminaries on interface theories, their most important properties as well as modal I/O automata which will serve as specification domain throughout this paper.

2.1 Interface Theories

As our goal is checking protocol compliance of service protocols and service orchestrations, we need a precise way for specifying whether two service protocols match (protocol compatibility) and whether an implementation correctly implements its

protocol (refinement). The latter notion of refinement is also used to relate abstract and more concrete protocol specifications, i.e. it allows for a step-wise refinement of protocols towards a concrete implementation. Interface theories or interface languages are commonly used to precisely define these notions. Our notion of an interface theory is inspired by de Alfaro and Henzinger's work in [5,6].

Interface theories are tuples $\mathcal{I} = (\mathcal{A}, \leq, \sim, \otimes)$ consisting of a specification domain $\mathcal{A}$, a reflexive and transitive refinement relation $\leq \subseteq \mathcal{A} \times \mathcal{A}$, a symmetric compatibility relation $\sim \subseteq \mathcal{A} \times \mathcal{A}$, and a partial composition operator $\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$. If two interfaces are compatible then their composition is defined, i.e. for all $S, T \in \mathcal{A}$, if $S \sim T$ then $S \otimes T$ is defined. Moreover, interface theories impose the following requirements on their refinement and compatibility relation:

(1) Preservation of compatibility under refinement:

$$\text{for all } S, T, T' \in \mathcal{A},\quad \text{if } S \sim T \text{ and } T' \leq T \text{ then } S \sim T'.$$

(2) Compositionality:

$$\text{for all } S, T, T' \in \mathcal{A},\quad \text{if } S \otimes T \text{ defined and } T' \leq T \text{ then } S \otimes T' \text{ defined and } S \otimes T' \leq S \otimes T.$$

These properties imply *independent implementability*, which is the basis for a top-down design of services and service protocols: In order to refine a given composed interface $S \otimes T$ towards an implementation, it suffices to independently refine S and T, say, to S' and T', respectively; then the refinements S' and T' are compatible and their composition (which is defined) refines the interface $S \otimes T$.

2.2 Modal I/O Automata

Modal I/O automata (MIOs) have first been introduced by Larsen et al. [19]. We have chosen MIOs as the specification domain of our interface theory; they provide us with a formal basis to support the analysis of service protocols and service orchestrations. Despite the fact that modalities in our interface specifications are not necessarily required in this work, the may- and must-modalities for transitions modelling allowed and required behaviour, respectively, is useful for representing standardised contracts in our theory.

Definition 2.1 [Modal I/O Transition System [19]] A modal I/O transition system (MIO) $S = (states_S, start_S, (in_S, out_S, int_S), \dashrightarrow_S, \longrightarrow_S)$ consists of a set of states $states_S$, an initial state $start_S \in states_S$, disjoint sets in_S, out_S, int_S of input, output, and internal transitions, respectively, where $ext_S = in_S \uplus out_S$ denotes the set of external actions and $act_S = ext_S \uplus int_S$ denotes the set of all actions; furthermore, a may-transition relation $\dashrightarrow_S \subseteq states_S \times act_S \times states_S$, and a must-transition relation $\longrightarrow_S \subseteq states_S \times act_S \times states_S$ such that both transition relations satisfy $\longrightarrow_S \subseteq \dashrightarrow_S$ (syntactic consistency). S is called implementation if $\longrightarrow_S = \dashrightarrow_S$.

The external alphabet (or signature) of a MIO S is denoted by $\alpha_{ext}(S) = (in_S, out_S)$. Alphabet equality and inclusion of two given MIOs S and T is defined by component-wise equality and inclusion, respectively. Modal I/O transition systems are particular modal transition systems which, in their original form introduced by Larsen and Thomsen [21], do not distinguish between input, output, and internal actions.

A syntactical requirement for two MIOs that are supposed to communicate with each other is that overlapping of actions only happens on complementary types.

Definition 2.2 [Composability [19]] Two MIOs S and T are called composable if $(in_S \cup int_S) \cap (in_T \cup int_T) = \emptyset$ and $(out_S \cup int_S) \cap (out_T \cup int_T) = \emptyset$.

The set of shared labels of two composable MIOs S and T is defined by $shared(S,T) = (out_S \cap in_T) \uplus (in_S \cap out_T)$.

Existing interface theories, using MIOs as the specification domain, include various notions of refinement and compatibility [2,19,20,11]. However, as we point out in the following section, none of these are adequate for the requirements listed above. We therefore complement these existing notions with a new interface theory which is adapted for checking service orchestrations.

3 An Interface Theory for Checking Service Protocol Compliance

In this section we introduce existing notions of modal refinement and compatibility and give our new definitions of strict-observational modal refinement and compatibility. We motivate this again by our running example from the *eUniversity* domain introduced above, this time modelled as MIOs, as shown in Fig. 2. Input labels are suffixed with a question mark (*?*) and output labels with an exclamation mark (*!*).

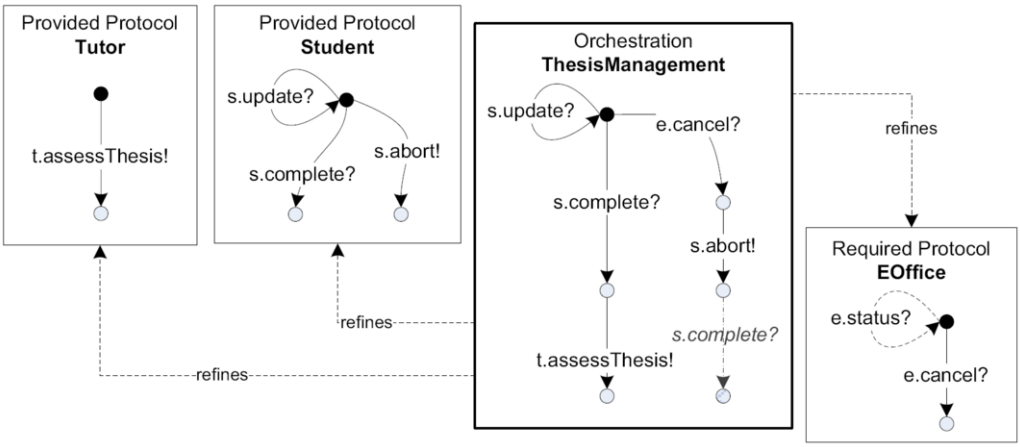

Figure 2. The eUniversity example

In the centre-right of the figure, the orchestration `ThesisManagement` is shown, which provides two services (left) which each have their own protocol (`Tutor` and

`Student`), and requires one external service `EOffice` (right; the examination office). The workflow is as follows. A student works on a thesis, providing updates. Once she is done, the thesis is marked complete and the tutor assesses it. There is one catch, though – the examination office may cancel the thesis if some formal requirements are not met. In this case, the student is notified of the problem.

Note that the `EOffice` protocol additionally specifies a call which is not implemented in the orchestration – a liberty given by the protocol as the call is modelled as a may-transition.

3.1 Strict-Observational Modal Refinement

We first try to analyse this service orchestration with existing interface theories for modal I/O automata. The closest to our own definitions of refinement and compatibility for MIOs are weak refinement and compatibility, complemented by the notion of hiding. We shall introduce these now, before moving on to the discussion of our interface theory.

Modal refinement [21] is usually defined in a contravariant way. The basic idea of *modal* refinement is that any required (*must*-)transition in the abstract specification must also occur in the concrete specification. Conversely, any allowed (*may*-)transition in the concrete specification must be allowed by the abstract specification. Moreover, in both cases the target states must conform to each other. Modal refinement has the following consequences: A concrete specification may leave out may-transitions, but is required to keep all must-transitions, and moreover, it is not allowed to perform more transitions than the abstract specification admits.

The basic form of modal refinement requires that every transition that is taken into account must be simulated *immediately*. There are many application areas in which this definition is too strong [17]. It can be weakened by distinguishing between external and internal actions and allowing an external action to be enclosed in internal actions. In this case, we speak of *weak transitions*.

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

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

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

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

Both notations are analogously used for may-transitions. Using this notion of weak transitions, we can define weak modal refinement. [4]

[4] We have adapted the definition of weak modal refinement [17] to MIOs; moreover, we have slightly generalised the definition by allowing the considered MIOs to differ with respect to their sets of internal actions.

Definition 3.1 [Weak Modal Refinement, adapted from [17]] Let S and T be MIOs such that $\alpha_{ext}(S) = \alpha_{ext}(T)$. S weakly modally refines T, denoted by $S \leq_m^* T$, iff there exists a relation $R \subseteq states_S \times states_T$ containing $(start_S, start_T)$ such that for all $(s,t) \in R$, for all $a \in act_S \cup act_T$,

(i) if $t \xrightarrow{a}_T t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{\hat{a}}{}^*_S s'$ and $(s',t') \in R$,

(ii) if $s \dashrightarrow{}^{a}_S s'$ then there exists $t' \in states_T$ such that $t \dashrightarrow{}^{\hat{a}}{}^*_T t'$ and $(s',t') \in R$.

Now, let us consider checking our *eUniversity* system: We want to assess whether the orchestration really adheres to all of the specified protocols. A first attempt at checking refinement fails due to incompatible alphabets – the orchestration, implementing more than one service, will always have more actions than the service protocol being checked. A possible solution for this problem is hiding (see, e.g., [28]), i.e. internalising all actions not belonging to the protocol under consideration.

However, this still does not capture our intention of application-level verification. Instead of detecting errors that *may* lead to a deadlock under *one* combination of "non-relevant" actions, we are interested in protocol breaches that *must* lead to a deadlock under *all* combinations of "non-relevant" actions – i.e., situations in which the protocol *cannot* be followed by the orchestration.

The interface theory we are looking for must therefore differ from the ones discussed above. This is seen in the example: The orchestration does not refine the student protocol under weak refinement, even with hiding, as *s.complete?* is no longer possible after *e.cancel?* – an action that is not relevant for the student protocol – has been received. The refinement analysis will report this problem, and hide the more significant error which is shown in dashed/dotted grey: The second *s.complete?* message in the implementation is a breach of the application-level protocol, as *s.complete?* is not allowed after *s.abort!* in the student protocol, even when ignoring "non-relevant" actions. Without the second *s.complete?* call, we want to consider the orchestration a refinement of the student protocol.

The initial problem indicated above (*s.complete?* not being possible after *e.cancel?*) is not detected by our refinement notion, which allows us to focus on finding protocol breaches. This problem can be dealt with using other analysis techniques such as interface theories building on weak refinement [2].

To capture our idea of application-level protocol breaches, we first define the notion of *action-weak transitions*. Given a MIO S, a set of actions $L \subseteq act_S$, and an action $a \in act_S$, we write $s \xrightarrow{a}{}^{\triangleleft(L)}_S s'$ iff either $s \xrightarrow{a}_S s'$ or there exist $n \geq 1$, states $s_1, \ldots, s_n \in states_S$, and actions $b_1, \ldots, b_n \in (act_S \setminus L)$ such that

$$s \xrightarrow{b_1}_S s_1 \ldots s_{n-1} \xrightarrow{b_n}_S s_n \xrightarrow{a}_S s'.$$

The intuition of an *action-weak transition* is that a single relevant action is performed, which may be preceded by irrelevant actions not in L. Moreover, we write $s \xrightarrow{a}{}^{\triangleleft}_S s'$ to abbreviate $s \xrightarrow{a}{}^{\triangleleft(ext_S)}_S s'$. The same notions are analogously used for may-transitions.

We now adapt weak modal refinement to satisfy our needs, i.e. we only want to consider relevant actions during refinement of MIOs. The basic idea for our refinement is to *skip* leading actions unrelated to the protocol under investigation. First, the refining MIOs may have more actions than the refined one, and second, in both directions in the definition we focus on the external actions of the more abstract MIO since these actions are the relevant ones.

Definition 3.2 [Strict-Observational Modal Refinement] Let S and T be MIOs such that $\alpha_{ext}(S) \supseteq \alpha_{ext}(T)$. S strict-observationally refines T, denoted by $S \leq_{so} T$, iff there exists a relation $R \subseteq states_S \times states_T$ containing $(start_S, start_T)$ such that for all $(s,t) \in R$, for all actions $a \in ext_T$,

(i) if $t \xrightarrow{a}{}^{\triangleleft}_{T} t'$ then there exists $s' \in states_S$ such that $s \xrightarrow{a}{}^{\triangleleft(ext_T)}_{S} s'$ and $(s',t') \in R$,

(ii) if $s \overset{a}{\dashrightarrow}{}^{\triangleleft(ext_T)}_{S} s'$ then there exists $t' \in states_T$ such that $t \overset{a}{\dashrightarrow}{}^{\triangleleft}_{T} t'$ and $(s',t') \in R$.

Note that strict-observational refinement only uses the modal aspects of MIOs and can thus also be defined for modal transition systems in their original form (cf. [21], not distinguishing between input/output/internal actions).

Strict-observational modal refinement can be proved to be a preorder on MIOs, i.e. it is reflexive and transitive.

Lemma 3.3 *Strict-observational modal refinement $\leq_{so}$ is reflexive and transitive.*

A proof of this lemma and all following lemmata and theorems can be found in [27].

Going back to our example, we had the problem that under weak refinement, the *s.complete?* call is reported as an error, as it is no longer accepted after *e.cancel?*. With strict-observational refinement, *e.cancel?* is no longer relevant for the refinement check between the student protocol and the orchestration. As a result, the only problem reported is the protocol breach due to the *s.complete?* call shown in dashed/dotted grey in the orchestration of Fig. 2.

3.2 Strict-Observational I/O Compatibility

Now we focus on our second aim which concerns compatibility checks. Consider Fig. 3, which is a replica of Fig. 2, however this time with complements of the protocols given before. The question is whether the protocols are compatible with their complements and whether the protocols are compatible with the orchestration.

Let us first formally define the notion of compatibility. As the closest match to our approach, we introduce weak compatibility [2], which moderates the usual notion of strong compatibility [19] by requiring that an output (issued by a may- or must-transition) must be accepted with a corresponding input (by a must-transition), which may, however, possibly be preceded by internal actions. [5]

[5] The definition of weak compatibility slightly differs from the one in [2] for technical reasons but is in fact equivalent.

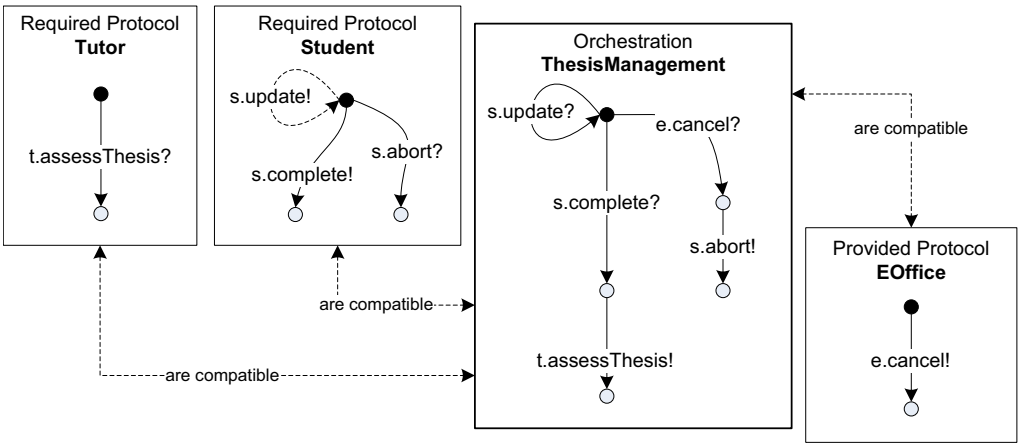

Figure 3. The eUniversity example (2)

Definition 3.4 [Weak Modal Compatibility, adapted from [2]] Let S and T be composable MIOs. S and T are called weakly modally compatible, written $S \sim_{wc} T$, iff there exists a relation $R \subseteq states_S \times states_T$ containing $(start_S, start_T)$ such that for all $(s,t) \in R$,

(i) for all $a \in (out_S \cap in_T)$, if $s \overset{a}{\dashrightarrow}_S s'$ then $\exists t' \in states_T . t \overset{a}{\longrightarrow}^{\triangleleft}_T t'$ and $(s',t') \in R$,

(ii) for all $a \in (out_T \cap in_S)$, if $t \overset{a}{\dashrightarrow}_T t'$ then $\exists s' \in states_S . s \overset{a}{\longrightarrow}^{\triangleleft}_S s'$ and $(s',t') \in R$,

(iii) for all $a \in (int_S \cup ext_S \setminus shared(S,T))$, if $s \overset{a}{\dashrightarrow}_S s'$ then $(s',t) \in R$,

(iv) for all $a \in (int_T \cup ext_T \setminus shared(S,T))$, if $t \overset{a}{\dashrightarrow}_T t'$ then $(s,t') \in R$.

Using weak modal compatibility, it is easy to see that the protocols and their complements in our example (Figs. 2 and 3) are compatible. However, the goal is to achieve preservation of compatibility under refinement – unfortunately, checking the `Student` protocol against the orchestration using weak modal compatibility yields a similar violation as observed before during our discussion of refinement: After having taken the *e.cancel?* transition, which is external to the orchestration but not shared with the protocol, the *s.update?* transition of the protocol can no longer be taken. This problem cannot be alleviated by hiding as hiding does not affect weak modal compatibility in any relevant way (moving external labels to internals, see Def. 3.4, (iii) and (iv)). Instead, the idea is to reduce the set of state pairs considered during compatibility checking.

Again, we want to consider all three protocols to be compatible with the orchestration, in each case only considering the protocol-observable actions as before. We define strict-observational I/O compatibility to reach this goal.

Definition 3.5 [Strict-Observational I/O Compatibility] Let S and T be composable MIOs, and let L be the set $shared(S,T)$ of shared labels of S and T. S and T are called strict-observationally I/O compatible, written $S \sim_{so} T$, iff there exists a relation $R \subseteq states_S \times states_T$ containing $(start_S, start_T)$ such that for all $(s,t) \in R$,

(i) for all $a \in (out_S \cap in_T)$, if $s \overset{a}{\dashrightarrow}_S^{\triangleleft(L)} s'$ then there exists $t' \in states_T$ such that $t \overset{a}{\longrightarrow}_T^{\triangleleft(L)} t'$ and $(s', t') \in R$,

(ii) for all $a \in (out_T \cap in_S)$, if $t \overset{a}{\dashrightarrow}_T^{\triangleleft(L)} t'$ then there exists $s' \in states_S$ such that $s \overset{a}{\longrightarrow}_S^{\triangleleft(L)} s'$ and $(s', t') \in R$.

Considering again the example of compatibility between the orchestration and the student protocol, strict-observational I/O compatibility treats *e.cancel?* and *t.assessThesis!* differently, as both actions are not defined in the student protocol. *e.cancel?* is only relevant as a prefix to *s.abort!*, while *t.assessThesis!* is not considered at all. We therefore get a positive compatibility result between `ThesisManagement` and `Student` as expected.

Lemma 3.6 *Strict-observational I/O compatibility $\sim_{so}$ is symmetric.*

3.3 Interface Theory

With strict-observational modal refinement and compatibility in place, we can now come back to our initial goal of defining a domain-specific interface theory targeted at checking protocol compliance in SOA systems.

In order to prove that MIOs together with strict-observational modal refinement and strict-observational I/O compatibility forms an interface theory, we need an appropriate notion of composition for MIOs, called strict-observational composition and denoted by $\otimes_{so}$, which is adapted to our strict-observational view. Then we show that indeed $\mathcal{I}_{so} = (\mathcal{MIO}, \leq_{so}, \sim_{so}, \otimes_{so})$, where $\mathcal{MIO}$ is the domain of all modal I/O automata, satisfies preservation of compatibility and compositionality.

Definition 3.7 [Strict-Observational Composition] Two composable MIOs S_1 and S_2 can be strict-observationally composed to a MIO $S_1 \otimes_{so} S_2$ defined by

- $states_{S_1 \otimes_{so} S_2} = states_{S_1} \times states_{S_2}$,
- $start_{S_1 \otimes_{so} S_2} = (start_{S_1}, start_{S_2})$,
- $in_{S_1 \otimes_{so} S_2} = (in_{S_1} \setminus out_{S_2}) \uplus (in_{S_2} \setminus out_{S_1})$,
- $out_{S_1 \otimes_{so} S_2} = (out_{S_1} \setminus in_{S_2}) \uplus (out_{S_2} \setminus in_{S_1})$,
- $int_{S_1 \otimes_{so} S_2} = \emptyset$.

The transition relations are given by

(i) for all $a_i \in shared(S_1, S_2)$, $1 \leq i \leq n$, if there exists $c \in (ext_{S_1} \setminus shared(S_1, S_2))$ and $s_1 \overset{a_1}{\longrightarrow}_{S_1}^{\triangleleft} \ldots \overset{a_n}{\longrightarrow}_{S_1}^{\triangleleft} s_1' \overset{c}{\longrightarrow}_{S_1}^{\triangleleft} s_1''$, and $s_2 \overset{a_1}{\longrightarrow}_{S_2}^{\triangleleft} \ldots \overset{a_n}{\longrightarrow}_{S_2}^{\triangleleft} s_2'$, then $(s_1, s_2) \overset{c}{\longrightarrow}_{S_1 \otimes_{so} S_2} (s_1'', s_2')$ (if $n = 0$ then $s_1 = s_1'$ and $s_2 = s_2'$),

(ii) for all $a_i \in shared(S_1, S_2)$, $1 \leq i \leq n$, if there exists $c \in (ext_{S_2} \setminus shared(S_1, S_2))$ and $s_1 \overset{a_1}{\longrightarrow}_{S_1}^{\triangleleft} \ldots \overset{a_n}{\longrightarrow}_{S_1}^{\triangleleft} s_1'$, and $s_2 \overset{a_1}{\longrightarrow}_{S_2}^{\triangleleft} \ldots \overset{a_n}{\longrightarrow}_{S_2}^{\triangleleft} s_2' \overset{c}{\longrightarrow}_{S_2}^{\triangleleft} s_2''$, then $(s_1, s_2) \overset{c}{\longrightarrow}_{S_1 \otimes_{so} S_2} (s_1', s_2'')$ (if $n = 0$ then $s_1 = s_1'$ and $s_2 = s_2'$),

(iii) and (iv), two similar rules for may-transitions ($\dashrightarrow$).

The intuition of this composition operator is to capture non-shared external actions c, which may be prefixed by a number of shared actions $a_1 \dots a_n$ (or none at all): any path only involving transitions with shared labels do not appear as synchronised actions in the composition; in fact, compositions $S_1 \otimes_{so} S_2$ do not contain any internal actions at all. In particular, any communication failures of two composed MIOs which are not compatible (i.e. a shared output is not received) does not emerge in the composed MIO.

Strict-observational composition satisfies associativity and commutativity which is a basic requirement for any reasonable composition operator. [6]

Lemma 3.8 *Strict-observational composition $\otimes_{so}$ is commutative and associative.*

First, we show that strict-observational compatibility is preserved under strict-observational modal refinement.

Theorem 3.9 **[Preservation of Compatibility]** *Let S, T, T' be MIOs, and let S, T and S, T' be composable. If $S \sim_{so} T$, $T' \leq_{so} T$ and $shared(S,T) = shared(S,T')$, then it follows that $S \sim_{so} T'$.*

Compositionality is the prerequisite for independent implementability of services and their modular verification. [7]

Theorem 3.10 **[Compositionality]** *Let S, T, T' be MIOs, and let S, T and S, T' be composable. If $T' \leq_{so} T$ and $shared(S,T) = shared(S,T')$, then $S \otimes_{so} T' \leq_{so} S \otimes_{so} T$.*

Independent implementability is a direct consequence of preservation of compatibility under refinement and compositionality of refinement. Even more than in traditional software architectures, SOA systems benefit from this property due to the inherent distribution of services and orchestrations in the service-based application landscape.

Corollary 3.11 **[Independent Implementability]** *Let S, T, T' be MIOs, and let S, T and S', T' be composable. If $S \sim_{so} T$, $T' \leq_{so} T$, $S' \leq_{so} S$ and $shared(S,T) = shared(S',T')$, both $S' \sim_{so} T'$ and $S' \otimes_{so} T' \leq_{so} S \otimes_{so} T$ follow.*

Proof First, it can be easily shown that

$$shared(S,T) = shared(S,T') = shared(S',T').$$

By Thm. 3.9, it follows $S \sim_{so} T'$, and by Lem. 3.6, $T' \sim_{so} S$. Again by Thm. 3.9 and Lem. 3.6, it follows that $S' \sim_{so} T'$. Second, we show $S' \otimes_{so} T' \leq_{so} S \otimes_{so} T$. We apply Thm. 3.10 to get $S \otimes_{so} T' \leq_{so} S \otimes_{so} T$. Moreover, by Thm. 3.10, it follows $T' \otimes_{so} S' \leq_{so} T' \otimes_{so} S$. Since $\otimes_{so}$ is commutative (Lem. 3.8) and $\leq_{so}$ transitive (Lem. 3.3), it holds that $S' \otimes_{so} T' \leq_{so} S \otimes_{so} T$. □

[6] For commutativity to hold, we assume that two MIOs are considered equal if they have the same (internal and external) alphabet and there exists a bijection between the state spaces such that both may- and must-transition relations are preserved.

[7] Interestingly, note that if $\alpha_{ext}(T) = \alpha_{ext}(T')$, then $S \otimes_{so} T' \leq_{so} S \otimes_{so} T$ is equivalent to $S \otimes_{so} T' \leq_{m} S \otimes_{so} T$ where $\leq_m$ is strong modal refinement [21].

This concludes our work to form an appropriate interface theory for protocol compliance checking of service orchestrations. As we have illustrated with our practical example, $\mathcal{I}_{so} = (\mathcal{MIO}, \leq_{so}, \sim_{so}, \otimes_{so})$ is indeed a suitable semantic framework for checking refinement and compatibility of service protocols and orchestrations. The strict-observational interface theory provides SOA developers with a tool for checking protocol implementation and compatibility on an application level, which can be used in addition to existing interface theories, e.g. building on weak refinement and compatibility.

We have implemented strict-observational refinement, compatibility, and composition in the MIO Workbench [2], an Eclipse-based verification tool which includes an editor for MIOs and is able to depict relations and problematic paths directly on the graphical MIO representation. The workbench also implements the standard notions of refinement and compatibility (strong, weak, and may-weak refinement as well as strong, weak, and friendly environment compatibility, cf. [2,19]), which allows a direct comparison between the individual interface theories. The MIO workbench is available for download from `http://www.miowb.net/`.

4 Related Work

The general study of interface theories was started by de Alfaro and Henzinger in [5,9]. Their well-known interface theory called interface automata essentially builds on the formalism of input/output automata [24,25,14] accompanied with notions of refinement (defined by alternating simulation) and compatibility [4]. This theory has recently been generalized to an interface theory based on modal input/output transition systems [19,20] which uses modal automata [21,17] for modelling interface behaviour. However, less attention has been paid to refinement between interfaces with different alphabets. Recently, the approach proposed in [29,30] deals with alphabet extension by adding self-loops for the new actions to all states of the automaton. It is easy to see (same argument as given in Sect. 3.1) that this solution is not adequate to handle our situations. In [11] Fischbein et al. propose branching alphabet refinement of modal transition systems to cope with the problem of unintuitive implementations allowed by weak modal refinement. However, their refinement is classic in the sense that it considers single transitions in the preconditions, which is too strict for our application area.

Action refinement [15] is a flexible notion of refinement which is based on refining actions when changing the abstraction level: An action of an abstract specification can be decomposed into a sequence of low-level actions specifying the system in more detail. We differ from this notion as all of our actions reside on the same abstraction level, yet we only consider some of them depending on the current viewpoint.

There is also an extensive body of knowledge on semantics and analysis of Web service orchestrations based on industry standards like BPEL; [31] provides a decent overview. However, to the best of our knowledge, no approach so far has considered early application-level verification as a precursor to existing approaches. Instead, the focus lies on analysis of specific aspects of service orchestrations. Both [22] and

[26] analyse BPEL compositions through transformations to petri-nets. Their composition analysis assumes a friendly environment in the sense of [4], but is not geared towards application-level analysis of service orchestrations. Fu et al. [13] present a translation of a composition of BPEL processes to Promela, the input language of the SPIN model checker [16]. However, due to the interaction semantics of the translation, application-level verification is not feasible. Two further approaches that are closer to ours use calculi to specify the underlying labelled transition systems: [3] explicitly focusses on strong notions of compliance and compatibility. For calculi-based model-checking approaches like [10], the same reasoning as for the approach of [13] applies: application-level verification is prohibited by the composition semantics of the language.

Regarding tool support, the MIO Workbench differs from existing tools such as MTSA [7,8], TICC [1], or Tempo [23] by explicitly considering both modality and input/output aspects of MIOs. Compared to SOA verification tools such as WS-Engineer [12], the MIO Workbench is more experimental: Based on input given directly as MIOs, it allows different (pluggable) interface theories to be applied to investigate both the model and the theories themselves.

5 Conclusion

The development of large, distributed software systems is increasingly carried out by using service-oriented architectures, in which services form the basic building blocks of the system to be composed to form new services, a process which is known as orchestration. By using a model-driven approach to SOA development, models of services and service orchestrations are available early in the development process, where they can be analysed and verified to increase overall system quality.

In this paper, we have investigated protocol compliance checking for service orchestrations. We have placed particular focus on detecting protocol breaches, allowing service engineers to directly work with their models on the application-level. Our approach is based on the semantic domain of modal I/O automata and supported by a new interface theory we call *strict-observational* as it focusses on externally visible actions from the viewpoint of a certain protocol. We have given formal definitions of this theory, and shown that it indeed constitutes a valid interface theory for domain-specific formal analysis of service-oriented systems.

We have successfully used strict-observational refinement and compatibility for checking the case studies of the EU project SENSORIA. A set of examples as well as full tool support is available through the MIO Workbench available from `http://www.miowb.net`.

As future work, we intend to include the strict-observational theory into our larger framework of interface theories already discussed in [2].

Acknowledgement

This research has been partially supported by the GLOWA-Danube project 01LW0602A2 sponsored by the German Federal Ministry of Education and Research, and the EC project SENSORIA, IST-2005-016004.

We are indebted to Rolf Hennicker and Stephan Janisch for their valuable hints and fruitful discussions. Furthermore, we also thank the anonymous reviewers for their helpful comments.

References

[1] B. Thomas Adler, Luca de Alfaro, Leandro Dias da Silva, Marco Faella, Axel Legay, Vishwanath Raman, and Pritam Roy. Ticc: A Tool for Interface Compatibility and Composition. In Thomas Ball and Robert B. Jones, editors, *18th Int. Conf. Computer Aided Verification, CAV 2006*, volume 4144 of *LNCS*, pages 59–62. Springer, 2006.

[2] Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On Weak Modal Compatibility, Refinement, and the MIO Workbench. In *16th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, 2010. To Appear.

[3] Mario Bravetti and Gianluigi Zavattaro. A Theory for Strong Service Compliance. In Amy L. Murphy and Jan Vitek, editors, *9th Int. Conf. Coordination Models and Languages, COORDINATION 2007*, volume 4467 of *LNCS*, pages 96–112. Springer, 2007.

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

[5] Luca de Alfaro and Thomas A. Henzinger. Interface Theories for Component-Based Design. In Thomas A. Henzinger and Christoph M. Kirsch, editors, *First Int. Workshop Embedded Software, EMSOFT 2001*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.

[6] Luca de Alfaro and Thomas A. Henzinger. Interface-based Design. In Manfred Broy, Johannes Grünbauer, David Harel, and C. A. R. Hoare, editors, *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.

[7] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. MTSA: The Modal Transition System Analyser. In *23rd Int. Conf. Automated Software Engineering, ASE 2008*, pages 475–476. IEEE Computer Societey, 2008.

[8] Nicolás D'Ippolito, Dario Fischbein, Howard Foster, and Sebastián Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In Li-Te Cheng, Alessandro Orso, and Martin P. Robillard, editors, *OOPSLA Workshop Eclipse Technology eXchange, ETX 2007*, pages 6–10. ACM Press, 2007.

[9] Laurent Doyen, Thomas A. Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface theories with component reuse. In Luca de Alfaro and Jens Palsberg, editors, *8th Int. Conf. Embedded software, EMSOFT 2008*, pages 79–88. ACM Press, 2008.

[10] Alessandro Fantechi, Stefania Gnesi, Alessandro Lapadula, Franco Mazzanti, Rosario Pugliese, and Francesco Tiezzi. A model checking approach for verifying COWS specifications. In J. L. Fiadeiro and P. Inverardi, editors, *11th Int. Conf. Fundamental Approaches to Software Engineering, FASE 2008*, volume 4961 of *LNCS*, pages 230–245. Springer, 2008.

[11] Dario Fischbein, Víctor A. Braberman, and Sebastián Uchitel. A Sound Observational Semantics for Modal Transition Systems. In Martin Leucker and Carroll Morgan, editors, *6th Int. Colloquium Theoretical Aspects of Computing, ICTAC 2009*, volume 5684 of *LNCS*, pages 215–230. Springer, 2009.

[12] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. WS-Engineer: A Model-Based Approach to Engineering Web Service Compositions and Choreography. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 87–119. Springer, 2007.

[13] Xian Fu, Tevfik Bultan, and Jianwen Su. Analysis of Interacting BPEL Web Services. In *3rd Int. Conf. on Web Services, ICWS 2004*, pages 621–630. IEEE Computer Society, 2004.

[14] Stephen J. Garland and Nancy Lynch. Using I/O Automata for Developing Distributed Systems. In *In Gary T. Leavens and Murali Sitaraman, editors, Foundations of Component-Based Systems*, pages 285–312. Cambridge University Press, 2000.

[15] Roberto Gorrieri and Arend Rensink. Action refinement. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 16, pages 1047–1147. Elsevier, 2001.

[16] Gerard J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[17] Hans Hüttel and Kim Guldstrand Larsen. The Use of Static Constructs in A Modal Process Logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Symp. Logical Foundations of Computer Science, Logic at Botik 1989*, volume 363 of *LNCS*, pages 163–180. Springer, 1989.

[18] Alexander Knapp, Stephan Janisch, Rolf Hennicker, Allan Clark, Stephen Gilmore, Florian Hacklinger, Hubert Baumeister, and Martin Wirsing. Modelling the CoCoME with the Java/A Component Model. In Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Frantisek Plasil, editors, *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*, pages 207–237. Springer, 2007.

[19] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O Automata for Interface and Product Line Theories. In Rocco De Nicola, editor, *16th Eur. Symp. Programming, Programming Languages and Systems, ESOP 2007*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.

[20] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. On Modal Refinement and Consistency. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *18th Int. Conf. Concurrency Theory, CONCUR 2007*, volume 4703 of *LNCS*, pages 105–119. Springer, 2007.

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

[22] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting BPEL processes. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *4th Int. Conf. Business Process Management, BPM 2006*, volume 4102 of *LNCS*, pages 17–32. Springer, 2006.

[23] Nancy A. Lynch, Laurent Michel, and Alexander Shvartsman. Tempo: A Toolkit for The Timed Input/Output Automata Formalism. In Sándor Molnár, John Heath, Olivier Dalle, and Gabriel A. Wainer, editors, *1st Int. Conf. Simulation Tools and Techniques for Communications, Networks, and Systems, SimuTools 2008*. ICST, 2008.

[24] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th Annual Symp. Principles of Distributed Computing, PODC 1987*, pages 137–151. ACM Press, 1987.

[25] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[26] Axel Martens. Analyzing web service based business processes. In Maura Cerioli, editor, *8th Int. Conf. on Fundamental Approaches to Software Engineering, FASE 2005*, volume 3442 of *LNCS*, pages 19–33. Springer, 2005.

[27] Philip Mayer, Andreas Schroeder, and Sebastian S. Bauer. A strict-observational interface theory for analysing service orchestrations. Technical Report 1003, Ludwig-Maximilians-Universität München, Germany, 2010.

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

[29] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal interfaces: unifying interface automata and modal specifications. In Samarjit Chakraborty and Nicolas Halbwachs, editors, *9th Int. Conf. Embedded software, EMSOFT 2009*, pages 87–96. ACM Press, 2009.

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

[31] Maurice H. ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web Service Composition Approaches: From Industrial Standards to Formal Methods. In *2nd Int. Conf. Internet and Web Applications and Services, ICIW 2007*. IEEE Computer Society, 2007.

[32] Martin Wirsing, Laura Bocchi, Alan Clark, Jose Fiadeiro, Stephen Gilmore, Matthias Hölzl, Nora Koch, Philip Mayer, Rosario Pugliese, and Andreas Schroeder. Sensoria: Engineering for Service-Oriented Overlay Computers. In Elisabetta Di Nitto, Anne-Marie Sassen, Paolo Traverso, and Arian Zwegers, editors, *At Your Service: Service-Oriented Computing from an EU Perspective*, pages 159–182. MIT Press, 2009.