Automated Mediator Synthesis: Combining Behavioural and Ontological Reasoning

Amel Bennaceur[1], Chris Chilton[2], Malte Isberner[3], and Bengt Jonsson[4]

[1] ARLES, Inria Paris - Rocquencourt, France
[2] Department of Computer Science, University of Oxford, UK
[3] Technical University of Dortmund, Germany
[4] Department of Information Technology, Uppsala University, Sweden

Abstract. Software systems are increasingly composed of independently developed heterogeneous components. To ensure interoperability, mediators are needed that coordinate actions and translate exchanged messages between the components. We present a technique for automated synthesis of mediators, by means of a quotient operator, that is based on behavioural models of the components and an ontological model of the data domain. By not requiring a specification of the composed system, the method supports both off-line and run-time synthesis. The obtained mediator is the most general component that ensures freedom of both communication mismatches and deadlock in the composition. Validation of the approach is given by implementation of a prototype tool, while applicability is illustrated on heterogeneous holiday booking components.

Keywords: mediator synthesis, quotient, ontology, deadlock-freeness.

1 Introduction

Modern software-intensive systems are increasingly composed of numerous independently developed and network-connected software components. These components often exhibit heterogeneous behaviour, which prevents them from interacting with one another according to a particular protocol. To circumvent this problem, *mediators* (or mediating adapters [YS97,CMS+09]) can be designed, which are intermediary software entities that allow heterogeneous software components to interact seamlessly, by coordinating their behaviours and translating the messages that they exchange. Due to the vast number of potential interaction patterns, it is not feasible to design a generic mediator that will allow an arbitrary collection of components to communicate. Instead, one approach towards facilitating communication involves the automated synthesis of a mediator, based on the behaviours of the components needing to interact.

Automatic mediator synthesis presupposes formal models of the participating components, each specifying the allowed sequences of interactions. Models can be directly specified by component developers, or can be automatically inferred (given their interfaces) by black-box inference [MHS+12]. Existing approaches for mediator synthesis also need specifications representing the composed behaviour of the components. For instance, synthesis of protocol converters

in [CL90,PdAHSV02] requires a specification of the service delivered by the composed system; similarly, in synthesising mediators for composed web services, an explicit specification of the relationships between the data parameters in different interface primitives is needed [BPT10]. Providing such specifications is an obstacle to automated synthesis, especially at run-time.

In this paper, we propose a rigorous methodology for the automated synthesis of mediators, based on models of the components and of the data domain, which does not require an explicit specification of the intended composition, making the techniques suitable for both off-line and run-time synthesis. Components may have incompatible behaviours and utilise different interaction vocabularies. To bridge this heterogeneity barrier, we rely on a domain ontology, which shows the relationships between data concepts of the interaction vocabularies. The domain ontology is generic to the application area of the components, and so no extra information need be supplied at synthesis time.

Our synthesis method is structured into two phases. First, the domain ontology is used to derive a correspondence between actions of different components, together with ordering constraints that must be respected between them. In the second phase, we synthesise a mediator using a quotient operator, by utilising the behavioural models of the participating components and the ordering constraints derived from the ontology. Our quotient operator extends existing definitions [BR08], in that it is sensitive to progress properties. Thus, mediators generated by our methodology are free from communication mismatches, ensure progress towards the goals of individual components, and respect the data constraints implicitly given by the domain ontology.

Outline. Section 2 introduces our component modelling formalism, along with the notions of parallel composition, refinement and quotient, which are essential for our synthesis methodology. Ontologies are presented in Section 3, where their role in modelling the semantics of component actions is explained. Section 4 describes the methodology for automatically synthesising mediators free of communication mismatches and premature deadlock, while Section 5 describes our prototype implementation and discusses its applicability. Section 6 examines related work, while Section 7 concludes and suggests future work.

2 Primer on the Compositional Specification Theory

In this section, we introduce the necessary parts of our compositional specification theory for modelling components [CCJK12]. The behaviour of a component specifies the sequences of allowed interactions between the component and its environment, which can be represented by a labelled transition system (LTS). The labels are partitioned into input and output actions, although internal actions can also be accommodated. In a state, the component is willing to receive (from the environment) any enabled input, and may emit any enabled output. If the environment supplies an input that is not enabled, an *inconsistency* arises, which can be understood as either underspecification, or an undesired situation corresponding to run-time error or bad behaviour.

To model deadlock and termination, a state can be designated as *quiescent*. The intuition is that a component must not block (i.e., must eventually emit an output if no input appears) in a non-quiescent state. The modelling formalism itself does not distinguish between undesirable deadlock and termination.

Our methodology is equally applicable to deterministic and non-deterministic models, using the theory in [CCJK12]. Some definitions can be simplified in the deterministic case, and for simplicity we will use these in this paper. Our specification theory can then be seen as interface automata extended with the capability to model deadlock and termination [dAH01].

Definition 1 (Behavioural model). *A* behavioural model of a component $\mathcal{P}$ *is a tuple* $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, S_\mathcal{P}, s^0_\mathcal{P}, \delta_\mathcal{P}, Q_\mathcal{P} \rangle$*, where* $\mathcal{A}^I_\mathcal{P}$ *and* $\mathcal{A}^O_\mathcal{P}$ *are disjoint sets referred to as the* inputs *and* outputs *(the union of which we denote by* $\mathcal{A}_\mathcal{P}$*),* $S_\mathcal{P}$ *is a finite set of* states *with* $s^0_\mathcal{P} \in S_\mathcal{P}$ *being the designated* initial state*,* $\delta_\mathcal{P} : S_\mathcal{P} \times \mathcal{A}_\mathcal{P} \rightharpoonup S_\mathcal{P}$ *is the partial* transition function*, and* $Q_\mathcal{P} \subseteq S_\mathcal{P}$ *are the* quiescent states*.*

We will not be fussy in distinguishing components from their models, and will often refer to "the component A" for "the behavioural model of A".

In a behavioural model of a component, we distinguish undesirable deadlocks from termination by introducing a designated ✓ action treated as an input. The convention is that ✓ can only be received when the component has successfully terminated.

Refinement of components is defined using the alternating simulation for interface automata extended to cope with quiescence. It guarantees safe-substitutivity of components and preservation of deadlock-freeness.

Definition 2 (Refinement). *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be components. Then* $\mathcal{Q}$ *is a refinement of* $\mathcal{P}$*, written* $\mathcal{Q} \sqsubseteq \mathcal{P}$*, if* $\mathcal{A}^I_\mathcal{P} \subseteq \mathcal{A}^I_\mathcal{Q}$ *and* $\mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^O_\mathcal{P}$*, and there is a relation* $\sqsubseteq \subseteq S_\mathcal{Q} \times S_\mathcal{P}$*, called an* alternating simulation*, such that whenever* $s_\mathcal{Q} \sqsubseteq s_\mathcal{P}$*:*

- *if* $i \in \mathcal{A}^I_\mathcal{P}$ *is enabled in* $s_\mathcal{P}$*, then* i *is enabled in* $s_\mathcal{Q}$ *and* $\delta_\mathcal{Q}(s_\mathcal{Q}, i) \sqsubseteq \delta_\mathcal{P}(s_\mathcal{P}, i)$*,*
- *if* $o \in \mathcal{A}^O_\mathcal{Q}$ *is enabled in* $s_\mathcal{Q}$*, then* o *is enabled in* $s_\mathcal{P}$ *and* $\delta_\mathcal{Q}(s_\mathcal{Q}, o) \sqsubseteq \delta_\mathcal{P}(s_\mathcal{P}, o)$*,*
- *if* $s_\mathcal{Q} \in Q_\mathcal{Q}$*, then* $s_\mathcal{P} \in Q_\mathcal{P}$*,*

and such that $s^0_\mathcal{Q} \sqsubseteq s^0_\mathcal{P}$*.*

The *parallel composition* of two components represents the combined effect of them running asynchronously, and synchronizing on actions that are common to their sets of inputs and outputs. To preserve the effect that a single output from a component can be received by multiple components in its environment, we must define the parallel composition to repeatedly broadcast an output: this means that an input a and output a combine to form an output a. As each output must be under the control of at most one component, the parallel composition is only defined when the composed components have disjoint sets of outputs. To obtain a modular definition of parallel composition, we first define a generic *product* of two transition functions.

Definition 3 (Product). *Let $\mathcal{P}$ and $\mathcal{Q}$ be components. The product of the transitions functions $\delta_{\mathcal{P}}$ of $\mathcal{P}$ and $\delta_{\mathcal{Q}}$ of $\mathcal{Q}$ is the partial function $\delta_{\mathcal{P}\otimes\mathcal{Q}} : (S_{\mathcal{P}} \times S_{\mathcal{Q}}) \times (\mathcal{A}_{\mathcal{P}} \cup \mathcal{A}_{\mathcal{Q}}) \rightharpoonup (S_{\mathcal{P}} \times S_{\mathcal{Q}})$, where $\delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ is defined as:*

- $\langle \delta_{\mathcal{P}}(s_{\mathcal{P}}, a), \delta_{\mathcal{Q}}(s_{\mathcal{Q}}, a)\rangle$ *when* $a \in \mathcal{A}_{\mathcal{P}} \cap \mathcal{A}_{\mathcal{Q}}$, *and both* $\delta_{\mathcal{P}}(s_{\mathcal{P}}, a)$ *and* $\delta_{\mathcal{Q}}(s_{\mathcal{Q}}, a)$ *are defined,*
- $\langle \delta_{\mathcal{P}}(s_{\mathcal{P}}, a), s_{\mathcal{Q}}\rangle$ *when* $a \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_{\mathcal{Q}}$ *and* $\delta_{\mathcal{P}}(s_{\mathcal{P}}, a)$ *is defined,*
- *symmetrically,* $\langle s_{\mathcal{P}}, \delta_{\mathcal{Q}}(s_{\mathcal{Q}}, a)\rangle$ *when* $a \in \mathcal{A}_{\mathcal{Q}} \setminus \mathcal{A}_{\mathcal{P}}$ *and* $\delta_{\mathcal{Q}}(s_{\mathcal{Q}}, a)$ *is defined,*

and $\delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ is undefined otherwise. □

A pair of states $\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle \in (S_{\mathcal{P}} \times S_{\mathcal{Q}})$ of $\mathcal{P}$ and $\mathcal{Q}$ is said to be *incompatible* if for some output action $a \in \mathcal{A}_{\mathcal{P}}^O \cup \mathcal{A}_{\mathcal{Q}}^O$, either $\delta_{\mathcal{P}}(s_{\mathcal{P}}, a)$ or $\delta_{\mathcal{Q}}(s_{\mathcal{Q}}, a)$ is defined, but $\delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ is undefined. Intuitively, in an incompatible pair of states, one component can perform an output that is not enabled as an input in the other component, thus creating a communication mismatch. A pair of states $\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle \in (S_{\mathcal{P}} \times S_{\mathcal{Q}})$ is said to be *potentially incompatible* if there is a (possibly empty) sequence of outputs $a_1 \cdots a_n$ in $\mathcal{A}_{\mathcal{P}}^O \cup \mathcal{A}_{\mathcal{Q}}^O$ that leads to an incompatible pair of states (i.e., $\delta_{\mathcal{P}\otimes\mathcal{Q}}(\ldots(\delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a_1), a_2)\ldots), a_n)$ is incompatible).

We use the product operation in the definition of parallel composition.

Definition 4 (Parallel composition). *Let $\mathcal{P}$ and $\mathcal{Q}$ be components such that $\mathcal{A}_{\mathcal{P}}^O \cap \mathcal{A}_{\mathcal{Q}}^O = \emptyset$. If $\langle s_{\mathcal{Q}}^0, s_{\mathcal{P}}^0\rangle$ is not potentially incompatible, then the* parallel composition *of $\mathcal{P}$ and $\mathcal{Q}$ exists and is defined as the component $\mathcal{P} \,||\, \mathcal{Q} = \langle (\mathcal{A}_{\mathcal{P}}^I \cup \mathcal{A}_{\mathcal{Q}}^I) \setminus (\mathcal{A}_{\mathcal{P}}^O \cup \mathcal{A}_{\mathcal{Q}}^O), \mathcal{A}_{\mathcal{P}}^O \cup \mathcal{A}_{\mathcal{Q}}^O, S_{\mathcal{P}} \times S_{\mathcal{Q}}, (s_{\mathcal{P}}^0, s_{\mathcal{Q}}^0), \delta_{\mathcal{P}||\mathcal{Q}}, Q_{\mathcal{P}} \times Q_{\mathcal{Q}}\rangle$, where: $\delta_{\mathcal{P}||\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a) = \delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ whenever $\delta_{\mathcal{P}\otimes\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ is defined and not potentially incompatible, otherwise $\delta_{\mathcal{P}||\mathcal{Q}}(\langle s_{\mathcal{P}}, s_{\mathcal{Q}}\rangle, a)$ is undefined.*

Intuitively, the transition function of $\mathcal{P} \,||\, \mathcal{Q}$ is undefined for inputs that lead to potentially incompatible pairs of states. If the environment supplies such an input, then $\mathcal{P} \,||\, \mathcal{Q}$ can potentially reach an incompatible pair of states, and such a situation is regarded as inconsistent. The component $\mathcal{P} \,||\, \mathcal{Q}$ is quiescent if both $\mathcal{P}$ and $\mathcal{Q}$ are quiescent.

Travel agency example. To illustrate our synthesis methodology, we consider a simple yet challenging example of a componentised and heterogeneous travel agency system, initially presented in [BBG+11]. The first component, called *USClient*, is a client-side software entity that allows customers to search for a holiday package, which consists of a hotel, a flight, and a car, and to purchase one if they so desire. The second component, called *EUService*, is a server-side service that provides operations for selecting the constituent parts of a holiday package (i.e., a hotel, a flight, and a car) separately.

The behaviour of the *USClient* and *EUService* components is represented by the models in Figures 1 and 2. Component models are represented pictorially by enclosing the transition system within a box corresponding to the interface. Labelled arrows pointing at the interface correspond to inputs, whereas arrows

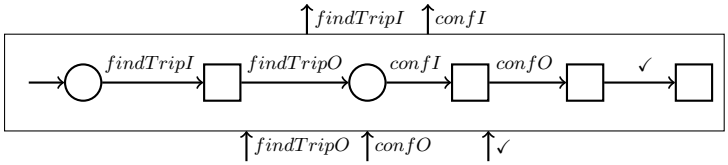

Fig. 1. Model of *USClient*

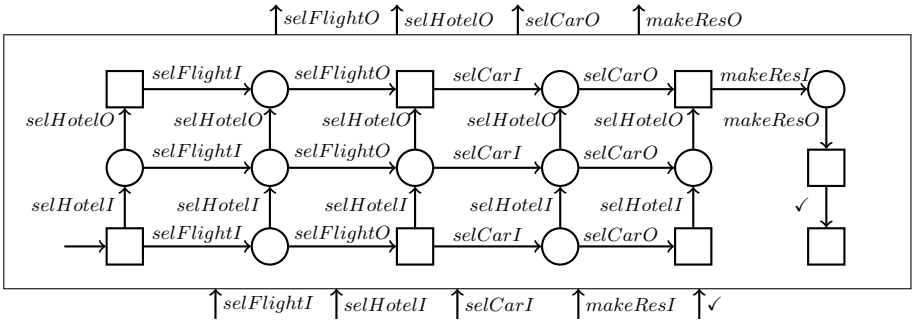

Fig. 2. Model of *EUService*

emanating from the interface correspond to outputs. Quiescent states are represented by squares, and other states by circles.

USClient sends $findTripI$, which is a request for a travel package that includes the travel preference of the customer (i.e., destination, departure and return dates). Then, *USClient* receives $findTripO$, which is a response including a trip proposition. *USClient* confirms the reservation by sending the $confI$ request and receiving the acknowledgement within the $confO$ response.

EUService receives the $selHotelI$, $selFlightI$, or $selCarI$ requests for finding a hotel, flight, or a car respectively given some customer preferences, which consists of a destination, departure and return dates. It then replies with one of the corresponding responses $selHotelO$, $selFlightO$, or $selCarO$, which include propositions for a hotel, flight, and car respectively. While requests for a hotel and a flight can be performed in any order, the request for a car can only be performed once the flight has been selected. Once all parts of a trip are validated, the *EUService* expects to receive a $makeResI$ request for completing the reservation, and sends the corresponding response $makeResO$.

The ✓ action, which may occur precisely at the end of the transaction, indicates that each component terminates only at the end of the transaction.

The two components have a functional discrepancy that prevents them from directly interacting to secure a holiday, even though, at a high-level of abstraction, *EUService* provides the functionality required by *USClient*. It is our intention to automatically synthesise a mediator allowing the two components to successfully interoperate. The synthesis approach is based on the behavioural models of each component, together with an ontology that represents knowledge about the domain in which the components belong.

Quotient. The final operation that we consider is that of quotient, which can be regarded as the adjoint (roughly "inverse") of parallel composition. Given a specification for a system $\mathcal{R}$, together with a component $\mathcal{P}$ implementing part of $\mathcal{R}$, the quotient, denoted $\mathcal{R}/\mathcal{P}$, yields the *weakest* specification for the remaining part of $\mathcal{R}$ to be implemented. Thus, $\mathcal{R}/\mathcal{P}$ is the weakest component such that $\mathcal{P} \,||\, (\mathcal{R}/\mathcal{P}) \sqsubseteq \mathcal{R}$. It is sufficient to understand quotient in this way without examining the formal definition below. Consequently, the remainder of this section may be skipped without losing the ability to understand our synthesis methodology.

Here, we define the quotient $\mathcal{R}/\mathcal{P}$ under the assumptions that $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$, reflecting that $\mathcal{P}$ is a sub-component of $\mathcal{R}$, and $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$, which is implied by $\mathcal{P} \,||\, (\mathcal{R}/\mathcal{P}) \sqsubseteq \mathcal{R}$. We postulate that $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_\mathcal{R} \setminus \mathcal{A}^O_\mathcal{P}$ and $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^I_\mathcal{R}$, which allows $\mathcal{R}/\mathcal{P}$ to monitor all the actions of $\mathcal{P}$ and $\mathcal{R}$.

Definition 5 (Quotient). *Let $\mathcal{P}$ and $\mathcal{R}$ be components such that $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$ and $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$. The* quotient *of $\mathcal{P}$ from $\mathcal{R}$ is the component $\mathcal{R}/\mathcal{P} = \langle \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^I_\mathcal{R}, \mathcal{A}^O_\mathcal{R} \setminus \mathcal{A}^O_\mathcal{P}, S_{\mathcal{R}/\mathcal{P}}, (s^0_\mathcal{P}, s^0_\mathcal{R}), \delta_{\mathcal{R}/\mathcal{P}}, Q_{\mathcal{R}/\mathcal{P}} \rangle$, defined only when $(s^0_\mathcal{P}, s^0_\mathcal{R}) \in S_{\mathcal{R}/\mathcal{P}}$, where:*

- *$S_{\mathcal{R}/\mathcal{P}}$ is the largest subset of $(S_\mathcal{P} \times S_\mathcal{R})$ such that*
  - *if $\langle s_\mathcal{P}, s_\mathcal{R} \rangle \in S_{\mathcal{R}/\mathcal{P}}$ and either $a \in \mathcal{A}^O_\mathcal{P}$ is enabled in $s_\mathcal{P}$, or $a \in \mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^I_\mathcal{R}$ is enabled in $s_\mathcal{R}$, then $\delta_{\mathcal{P}\otimes\mathcal{R}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a)$ is defined and $\delta_{\mathcal{P}\otimes\mathcal{R}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a) \in S_{\mathcal{R}/\mathcal{P}}$*
  - *if $\langle s_\mathcal{P}, s_\mathcal{R} \rangle \in S_{\mathcal{R}/\mathcal{P}}$ and $s_\mathcal{P} \in Q_\mathcal{P}$ and $s_\mathcal{R} \notin Q_\mathcal{R}$, then there is some $a \in \mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\delta_{\mathcal{P}\otimes\mathcal{R}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a) \in S_{\mathcal{R}/\mathcal{P}}$*
- *$\delta_{\mathcal{R}/\mathcal{P}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a) = \delta_{\mathcal{P}\otimes\mathcal{R}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a)$ whenever $\langle s_\mathcal{P}, s_\mathcal{R} \rangle \in S_{\mathcal{R}/\mathcal{P}}$ and $\delta_{\mathcal{P}\otimes\mathcal{R}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a) \in S_{\mathcal{R}/\mathcal{P}}$, otherwise $\delta_{\mathcal{R}/\mathcal{P}}(\langle s_\mathcal{P}, s_\mathcal{R} \rangle, a)$ is undefined*
- *$Q_{\mathcal{R}/\mathcal{P}} = S_{\mathcal{R}/\mathcal{P}} \cap ((S_\mathcal{P} \times Q_\mathcal{R}) \cup (S_\mathcal{P} \setminus Q_\mathcal{P}) \times S_\mathcal{R})$.*

Intuitively, the quotient can be constructed in a manner similar to the parallel composition of $\mathcal{P}$ and $\mathcal{R}$, but avoiding situations where: $\mathcal{P}$ can produce an output not matched by $\mathcal{R}$; $\mathcal{R}$ can accept an input in $\mathcal{A}^I_\mathcal{R} \cap \mathcal{A}^I_\mathcal{P}$ that $\mathcal{P}$ cannot accept; and $\mathcal{P}$ is quiescent, $\mathcal{R}$ is not, and $\mathcal{R}/\mathcal{P}$ cannot enforce an output action.

Theorem 1. *Let $\mathcal{P}$ and $\mathcal{R}$ be components such that $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$ and $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$. If there exists a component $\mathcal{Q}$ with inputs $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and outputs $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$, such that $\mathcal{P} \,||\, \mathcal{Q} \sqsubseteq \mathcal{R}$, then $\mathcal{R}/\mathcal{P}$ is defined, $\mathcal{P} \,||\, (\mathcal{R}/\mathcal{P}) \sqsubseteq \mathcal{R}$ and $\mathcal{Q} \sqsubseteq (\mathcal{R}/\mathcal{P})$.*

3 Ontological Modelling and Reasoning

An ontology is *"a specification of a representational vocabulary for a shared domain of discourse"* [Gru93]. The goal of an ontology is to model and reason about domain knowledge. OWL DL[5] (Web Ontology Language), which is the W3C standard language to model ontologies, is based on a description logic (DL), which specifies the vocabulary of a domain using concepts, attributes of each concept, and relationships between these concepts.

[5] `http://www.w3.org/TR/owl2-overview/`

We provide an overview of the syntax and semantics of the basic DL constructs in Figure 3 and refer the interested reader to [BCM+03] for further details. Each concept is given a definition as a set of logical axioms, which can either be atomic or defined using different operators such as disjunction, conjunction, and quantifiers. The attributes of a concept are defined using an object property, which associates the concept with a built-in data type.

For example, consider an extract of the travel agency ontology depicted in Figure 4. The Flight concept is characterised by attributes hasDepartureDate and hasReturnDate of the built-in type DateTime, and hasFlightID of type String.

We describe the aggregation of concepts using the W3C recommendation for part-whole relations[6] (hasPart), where different concepts are composed together to build a whole. A concept E is an aggregation of concepts C and D, written $E = C \oplus D$, providing both C and D are parts of E, i.e., $E = \exists \mathsf{hasPart}.C \sqcap \exists \mathsf{hasPart}.D$. For example, the Trip concept is defined as the aggregation of the Flight, Hotel, and Car concepts, meaning that each trip instance $t \in \mathsf{Trip}$ encompasses a Flight instance ($\exists f \in \mathsf{Flight} \wedge (t, f) \in \mathsf{hasPart}$), as well as Hotel and Car instances. The rationale is that the mediator is able to generate a concept by concatenating the attributes of all its parts (while avoiding duplication of attributes). Dually, the mediator can create several concepts by distributing the attributes of the aggregated concept across its different parts. This corresponds to the merging and splitting of messages.

	DL Syntax	**DL Semantics**
Conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Universal quantifier	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
Existential quantifier	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Aggregation	$C \oplus D$	$(C \oplus D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in \mathsf{hasPart}^{\mathcal{I}} \wedge y \in C^{\mathcal{I}} \wedge \exists z.(x, z) \in \mathsf{hasPart}^{\mathcal{I}} \wedge z \in D^{\mathcal{I}}\}$

An interpretation $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic object property R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.
C and D are concepts and R is an object property.

Fig. 3. Overview of DL operators

DL is used to support automatic reasoning about concepts and their relationships, in order to infer new relations that may not have been recognised by the ontology designers. Traditionally, the basic reasoning mechanism is *subsumption*. Intuitively, if a concept C is subsumed by a concept D, written $C \leqslant_{\mathcal{O}} D$, then any instance of C also belongs to D. In addition, all the relationships in which D instances can be involved are applicable to C instances, i.e., all properties of D are also properties of C. Subsumption is a partial order relation, i.e., it is reflexive, antisymmetric, and transitive. As a result, the ontology can be represented as a hierarchy of concepts, which can be automatically inferred by

[6] `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`

ontology reasoners based on the axioms defining the ontological concepts. Subsumption allows for the replacement of inequivalent messages, provided all of the necessary data is conveyed. For instance, a mediator can generate a concept out of a more specific concept, since the latter includes all the necessary attributes of the former (cf subtyping in object-oriented systems).

Ontologies are used to represent the semantics of actions in components, by making explicit the meaning of the interaction primitives. Each action of a component refers to a concept in the ontology, which has an object property hasData specifying the semantics of the data embedded in the action sent or received by the component. For example, the input action $selFlightI \in \mathcal{A}^I_{EUService}$ is associated with the TravelPreferences concept (i.e., $selFlightI = \exists$hasData.TravelPreferences). The output action $selFlightO \in \mathcal{A}^O_{EUService}$ is associated with the Flight concept, that is $selFlightO = \exists$hasData.Flight. . The idea here is that *EUService* allows the selection of a flight by receiving a request message that contains the attributes of the TravelPreferences concept, that is a destination, departure, and return dates. Once the request is processed, *EUService* returns a response that includes the flight information, which consists of a hotel identifier and check in and check out dates. The output action $findTripI \in \mathcal{A}^O_{USClient}$ is also associated with the TravelPreferences concept, that is $findTripI = \exists$hasData.TravelPreferences. The input action $findTripO \in \mathcal{A}^I_{USClient}$ is associated with the Trip concept, that is $findTripO = \exists$hasData.Trip. *USClient* sends a request message that includes the travel preference of the customer, and receives a response with a trip, i.e., a holiday package consisting of a flight, hotel, and car.

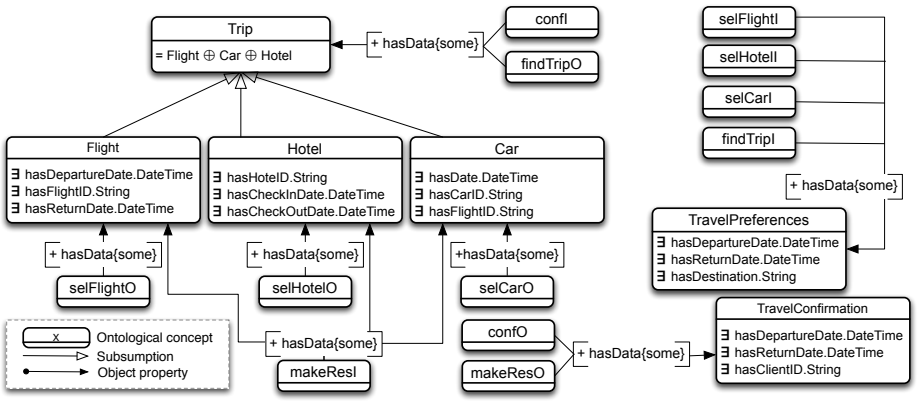

Fig. 4. The travel agency ontology

4 Automated Mediator Synthesis

In this section, we describe our synthesis methodology for generating mediators that allow components to successfully communicate with one another. Given components $\mathcal{P}$ and $\mathcal{Q}$, a component $\mathcal{M}$ is said to be a mediator if:

G1. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is defined, which implies that the composition will never exhibit any communication mismatches;
G2. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is guaranteed to progress until both $\mathcal{P}$ and $\mathcal{Q}$ have reached a successfully terminating state; and
G3. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ must satisfy constraints on correspondences between actions, and on data flow, that are implicitly imposed by the ontology.

Our methodology finds the most general mediator $\mathcal{M}$ satisfying the above requirements, meaning that it is least refined with respect to the refinement $\sqsubseteq$ of Definition 2. Due to the arbitrariness of the components needing to communicate (especially if they are functionally different), existence of a mediator is not guaranteed. This correlates with the fact that quotient is a partial operator. As part of our automated synthesis methodology, the following steps are performed:

1. Using the ontology, we first derive temporal constraints on the occurrences of actions in $\mathcal{P}$ and $\mathcal{Q}$ in order to respect the data flow on actions. These constraints are represented by a component $\mathcal{B}$ that *observes* the executions of $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ and generates an inconsistency when a constraint is violated.
2. After, we perform a quotient operation that automatically synthesises a mediator $\mathcal{M}$ such that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ satisfies requirements G1 and G2, along with the ordering constraints represented by the observer $\mathcal{B}$ (G3).

In the following two subsections, we explain these steps in more detail.

4.1 Inferring Ordering Constraints from Ontologies

By reasoning about the ontology-based semantics of actions, we can derive ordering constraints on the components' communication primitives that respect both the semantical meaning of actions.

Let us consider an input action b that is associated with data d_b in a domain ontology $\mathcal{O}$, i.e., $b = \exists \mathsf{hasData}.d_b$ where b and d_b are concepts belonging to $\mathcal{O}$. The data required for action b must be provided by one or several output actions; in the simplest case by an output action a that is associated with data d_a such that $d_a \leqslant_{\mathcal{O}} d_b$. Moreover, action a must precede action b, written a $\mathsf{precedes}$ b. The intuition behind this ordering constraint is that the action a needs to provide the data required to achieve action b; it essentially corresponds to a data-dependency in which each input action must be preceded by output actions that supply the data items required for the execution of this input action. For example, the travel agency ontology specifies that the input action $selFlightI \in \mathcal{A}^I_{EUService}$ is associated with the data concept $\mathsf{TravelPreference}$. Since the only output action associated with this data concept is $findTripI \in \mathcal{A}^I_{USClient}$, and $\mathsf{TravelPreference} \leqslant_{\mathcal{O}} \mathsf{TravelPreference}$ due to reflexivity of subsumption, we derive the ordering constraint $findTripI$ $\mathsf{precedes}$ $selFlightI$.

In the general case, a collection of input actions $\{b_1, \ldots, b_m\}$, which are associated with data concepts $\{d_{b_1}, \ldots, d_{b_m}\}$ respectively, must be preceded by

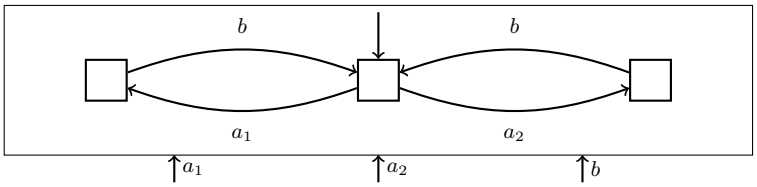

Fig. 5. Component $Seq(\{a_1, a_2\}, b)$ enforcing a_1 precedes b **or** a_2 precedes b

some collection of output actions $\{a_1, \ldots, a_n\}$, which are associated with data concepts $\{d_{a_1}, \ldots, d_{a_n}\}$ such that $d_{a_1} \oplus \cdots \oplus d_{a_n} \leqslant_{\mathcal{O}} d_{b_1} \oplus \cdots \oplus d_{b_m}$. If there are several such collections of output actions, it suffices that $\{b_1, \ldots, b_m\}$ is preceded by one of them. Clearly, we can restrict our consideration to minimal such collections of output actions. For example, the input action $\mathit{findTripO} \in \mathcal{A}^I_{USClient}$ must be preceded by the collection of output actions $\{\mathit{selFlightO}, \mathit{selHotelO}, \mathit{selCarO}\} \subseteq \mathcal{A}^O_{EUService}$, since it is the only minimal collection, whose aggregated data is subsumed by the data of $\mathit{findTripO}$, i.e., $\mathsf{Flight} \oplus \mathsf{Hotel} \oplus \mathsf{Car} \leqslant_{\mathcal{O}} \mathsf{Trip}$.

In order to extract such relations, we verify among possible combinations of input/output actions of both components those verifying the data flow conditions. Even though computing all possible preceding relations is NP-complete, we rely on efficient search algorithms, which are based on constraint programming, to make the computation effective in real-world settings [Con12a, pp. 49-57].

For each minimal disjunction of precedes relationships inferred from the ontology, we construct an observation component that has as interface the collection of primitives that appear in the relationship. For instance, the disjunction a_1 precedes b **or** a_2 precedes b is represented by a component whose behaviour forces either a_1 or a_2 to precede b, denoted by $Seq(\{a_1, a_2\}, b)$. Its behavior is shown in Figure 5. Note that all the actions are treated as inputs and all states are quiescent since the component is only observing the actions. If the mediator or a component violates a constraint, the corresponding observer will generate an inconsistency. The component $\mathcal{B}$ respecting the combined effect of all the ontological constraints is then defined as the parallel composition of the representations of the individual relationships. Note that this is always defined.

Considering the travel agency example, by reasoning about the semantics of the actions of *USClient* and *EUService* using the ontology depicted in Figure 4, we infer that $\mathit{findTripI}$ precedes $\mathit{selFlightI}$, $\mathit{findTripI}$ precedes $\mathit{selHotelI}$, $\mathit{findTripI}$ precedes $\mathit{selCarI}$, $\mathit{selFlightO}$ precedes $\mathit{findTripO}$, $\mathit{selHotelO}$ precedes $\mathit{findTripO}$, $\mathit{selCarO}$ precedes $\mathit{findTripO}$, confI precedes $\mathit{makeResI}$, and $\mathit{makeResO}$ precedes confO, which leads to the following observer component:

$$\begin{aligned}\mathcal{B} = (\ & Seq(\mathit{findTripI}, \mathit{selFlightI}) \ || \ Seq(\mathit{findTripI}, \mathit{selHotelI}) \\ & || \ Seq(\mathit{findTripI}, \mathit{selCarI}) \ || \ Seq(\mathit{selFlightO}, \mathit{findTripO}) \\ & || \ Seq(\mathit{selHotelO}, \mathit{findTripO}) \ || \ Seq(\mathit{selCarO}, \mathit{findTripO}) \\ & || \ Seq(\mathit{confI}, \mathit{makeResI}) \ || \ Seq(\mathit{makeResO}, \mathit{confO})).\end{aligned}$$

4.2 Synthesising a Mediator as a Quotient

Having derived the ordering constraints implicitly encoded in the ontology (represented by the observer component $\mathcal{B}$), we can formulate the synthesis problem as the problem of performing a quotient operation. We begin by constructing a *goal* component $\mathcal{G}$ that first performs any sequence of non-✓ actions of $\mathcal{P}$ and $\mathcal{Q}$, and thereafter perform a ✓ action before becoming quiescent. The goal $\mathcal{G}$, which can automatically be generated from the syntax of $\mathcal{P}$ and $\mathcal{Q}$, is shown in Figure 6. The synthesis problem involves finding a most general mediator $\mathcal{M}$

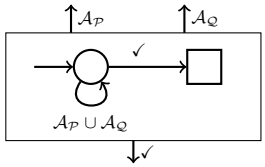

Fig. 6. Component representing the goal $\mathcal{G}$

such that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q} \parallel \mathcal{B} \sqsubseteq \mathcal{G}$. Note that the process $\mathcal{G}$ has all actions of $\mathcal{P}$ and $\mathcal{Q}$ as outputs. This means that each input action of either $\mathcal{P}$ or $\mathcal{Q}$ must be an output action of $\mathcal{M}$, implying that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ has no input actions (i.e., is a closed system). If such a mediator $\mathcal{M}$ exists, it is equal to the quotient $\mathcal{G}/(\mathcal{P} \parallel \mathcal{Q} \parallel \mathcal{B})$, for which it can be shown that requirements G1–G3 hold:

- G1 is guaranteed by the fact that $\mathcal{M}$ being defined implies that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q} \parallel \mathcal{B}$ is defined. Hence $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ cannot enter an incompatible state.
- G2 is satisfied since $\mathcal{G}$ can only become quiescent after having seen ✓, which means that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q} \parallel \mathcal{B}$ can only become quiescent after having seen ✓. Consequently, $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ can only deadlock when all components have terminated successfully.
- G3 is satisfied for the following reason. The data flow constraints are satisfied since $\mathcal{B}$ will generate an inconsistency whenever the sequence of actions does not satisfy them. This implies $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ will never produce any action that violates the constraints on occurrences of actions expressed by $\mathcal{B}$.

Remark. Our methodology has considered the case where $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is modeled as a closed system. In the case where $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is an open system and we have a model $\mathcal{E}$ of its environment, we can use the same technique by finding a mediator $\mathcal{M}$ such that $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q} \parallel \mathcal{B} \parallel \mathcal{E} \sqsubseteq \mathcal{G}$, where we assume that $\mathcal{E}$ is just another component.

Travel agency example. The mediator for the packaged holiday example is shown in Figure 7. Its inputs are the outputs of $EUService \parallel USClient$ and its outputs are the inputs of $EUService \parallel USClient$. The main idea is that the mediator first intercepts the output produced by *USClient* from the *findTripI* action, transforms it into the equivalent actions for *EUService* (*selHotelI*, *selFlightI* and *selCarI*), and then sends them to *EUService* (also respecting the constraint

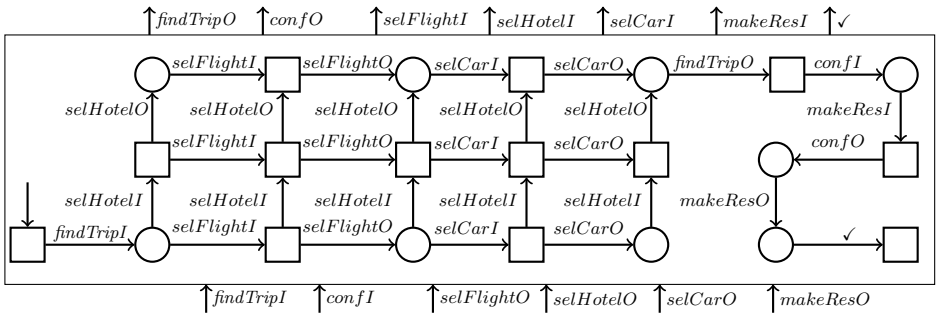

Fig. 7. The mediator for the travel agency example

that *selFlightO* must be sent before *selCarI*). Upon reception of the corresponding responses *selHotelO*, *selFlightO* and *selCarO* from *EUService*, the mediator forwards the expected output action *findTripO* to *USClient*. The process then evolves in a similar manner for confirmation of the reservation.

5 Implementation

We implemented the synthesis approach defined in this paper as a prototype tool available at `http://www.rocq.inria.fr/arles/software/onto-quotient/`. The Pellet reasoner[7] is used to build the hierarchy of concepts in the ontology and extract all the subsumption relations. The problem of inferring ordering constraints is formalised as a constraint satisfaction problem, which we solve efficiently using the Choco[8] constraint solver. In a second step, we generate the behavioural model of the observer $\mathcal{B}$ based on the generated ordering constraints, compute the goal specification as shown in Fig. 6, and calculate the quotient. It should be noted that even though $\mathcal{B}$ and $\mathcal{G}$ are always of a similar shape (since they are automatically generated), the implementation allows for arbitrary behavioural models to be used as observer and goal specifications.

In order to enable the components to interoperate, the synthesised behavioural model of the mediator needs to be concretised and deployed into a concrete artefact so as to realise the specified translations and coordination. This artefact is called *emergent middleware* [BBG+11]. The emergent middleware concretises the mediator model by incorporating information about underlying network layers. In particular, the emergent middleware: (i) intercepts the input messages, (ii) parses them so as to abstract from the communication details and represent them in terms of actions as expected by the mediator, (iii) performs the necessary data transformations, and (iv) uses the transformed data to construct an output message in the format expected by the interacting component. This is performed using specific parsers and composers, which are generated based on

[7] `http://clarkparsia.com/pellet/`
[8] `http://choco.emn.fr/`

existing libraries associated with the network communication protocols. In the case of the travel agency example, we used the `wsimport`[9] library to parse and compose the messages.

Apart from the travel agency example, we experimented with our approach on a number of case studies defined within the EU FP7 CONNECT project [Con12b]. These use cases mainly focus on enabling interoperability between heterogeneous systems from different countries in a cross-border emergency situation. The services are embedded within the GMES (Global Monitoring for Environment and Security) context, which gives rise to a common domain ontology comprising the concepts relevant in the considered emergency situations. For the sake of brevity, we refer to [Con12b] for a more elaborate description on the domain ontology and the systems involved.

In all the considered examples, we were able to generate a mediator automatically by means of quotient. The quotient could be computed virtually without any delay, even though in our prototype implementation we merely focused on functionality. We are thus confident that our approach will scale well even when applied to more complex case studies, which we are planning to evaluate as future work.

6 Related Work

Pioneering work on mediator synthesis involves the use of formal methods for protocol conversion, given behavioural models of the participating components. Existing approaches for the synthesis of protocol converters require a specification of the service delivered by the composed system. Lam [Lam88] assumes a declarative specification of a common protocol to which both protocols can be abstracted, which presupposes an intuitive understanding of the protocols to be mediated. Calvert and Lam [CL90] propose a quotient operation for mediating communication protocols, which is related to, but different from, our quotient operation. They require a global specification of the composed system, which makes it difficult to apply automated mediator synthesis at run-time. A quotient operation for deterministic interface automata has been presented by Bhaduri and Ramesh [BR08]. Our quotient extends this definition by considering also quiescence (deadlock) properties.

To improve the automation of mediator synthesis, Yellin and Strom [YS97] define an algorithm to generate mediators automatically assuming that there exist one-to-one correspondences between their actions, which have to be provided by the developer. Bertoli *et al.* [BPT10] also assume the correspondence to be given and use a planning-based algorithm to generate the mediator. Bersani *et al.* [BCF+10] define an approach based on SMT-based model checking but assume that the protocols have the same alphabet. Finally, Inverardi and Tivoli [IT13] adopt a compositional approach where the mediator is generated based on pre-defined patterns of translations, which have to be given by the developer. Common to all the aforementioned methods is the assumption of

[9] `http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html`

a priori knowledge about the components to be mediated and hence the correspondence between their actions must be provided beforehand by developers using their intuitive understanding of the application domain.

The emergence of the Semantic Web has led to the use of ontologies in system integration as a means of interpreting the meaning of data or associated services as they are dynamically encountered in the World-Wide Web. The Web Service Execution Environment (WSMX) [CM05] provides a framework to mediate interactions between heterogeneous Web Services by inspecting their individual protocols and performing the necessary translation using predefined mediation patterns. However, the composition of these patterns is not considered, and there is no guarantee of deadlock-freeness. Vaculín *et al.* [VNS09] synthesise mediators between a client and a service specified as OWL-S processes by generating all the traces of the client protocol and finding the appropriate mapping for each trace by simulating the service protocol.

In this paper, we define an approach that extends and improves existing work on mediator synthesis by using ontologies to reason about the domain and automatically infer the correspondence between the actions of the components involved. This removes the need for a declarative specification of the global system or the assumption that the components share the same alphabet.

7 Discussion and Evaluation

We have devised a methodology for synthesising mediators to support the interoperability of components. Unlike existing techniques, we make use of an ontology that relates the functional behaviour of the components, meaning that the components do not have to share similar communication alphabets. The synthesis technique is automated in the sense that the user does not need to specify what the mediator should do, as this can be inferred using behavioural and ontological reasoning. The synthesis is performed by means of a quotient operation, which has received renewed interest in the literature recently. The synthesised mediators are free of communication mismatches, and by consideration of quiescence are guaranteed not to deadlock prematurely or inopportunely.

As a matter of simplicity, we have shunned away from components exhibiting non-determinism and hidden transitions, although our theory can support these by using the definitions of parallel composition and quotient in [CCJK12]. Future work includes incremental re-synthesis of mediators so as to respond efficiently to changes in the individual components or in the ontology.

Acknowledgements. This work is carried out as part of the European FP7 ICT FET Connect project (http://connect-forever.eu/). The last author was supported in part by the UPMARC centre of excellence.

References

BBG+11. G. S. Blair, A. Bennaceur, N. Georgantas, P. Grace, V. Issarny, V. Nundloll, and Massimo Paolucci. The role of ontologies in emergent mid-

dleware: Supporting interoperability in complex distributed systems. In *Proc. Middleware*, pages 410–430, 2011.

BCF+10. M.M. Bersani, L. Cavallaro, A. Frigeri, M. Pradella, and M. Rossi. SMT-based verification of ltl specification with integer constraints and its application to runtime checking of service substitutability. In *Proc. SEFM*, pages 244–254. IEEE, 2010.

BCM+03. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

BPT10. P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3-4):316–361, 2010.

BR08. Purandar Bhaduri and S. Ramesh. Interface synthesis and protocol conversion. *Form. Asp. Comput.*, 20(2):205–224, March 2008.

CCJK12. T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska. A compositional specification theory for component behaviours. In *ESOP*, volume 7211 of *LNCS*, pages 148–168, 2012.

CL90. K. L. Calvert and S. S. Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas in Communications*, 8(1):127–142, 1990.

CM05. E. Cimpian and A. Mocan. WSMX process mediation based on choreographies. In *Proc. of Business Process Management Workshop*, pages 130–143, 2005.

CMS+09. J. Cámara, J. Martín, G. Salaün, J. Cubo, M. Ouederni, C. Canal, and E. Pimentel. ITACA: An integrated toolbox for the automatic composition and adaptation of web services. In *ICSE*, pages 627–630, 2009.

Con12a. Connect Consortium. Deliverable D3.4: Dynamic Connector Synthesis: Principles, Methods, Tools and Assessment. FET IP Connect EU project., 2012. http://hal.inria.fr/hal-00805618.

Con12b. Connect Consortium. Deliverable D6.4: Assessment report: Experimenting with CONNECT in Systems of Systems, and Mobile Environments. FET IP Connect EU project., 2012. http://hal.inria.fr/hal-00793920.

dAH01. L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.

Gru93. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

IT13. P. Inverardi and M. Tivoli. Automatic synthesis of modular connectors via composition of protocol mediation patterns. In *ICSE*, 2013. to appear.

Lam88. S. Lam. Protocol conversion. *IEEE Transaction Software Engineering*, 14(3):353–362, 1988.

MHS+12. M. Merten, F. Howar, B. Steffen, P. Pellicione, and M. Tivoli. Automated inference of models for black box systems based on interface descriptions. In *ISOLA*, volume 7609 of *LNCS*, pages 79–96, 2012.

PdAHSV02. R. Passerone, L. de Alfaro, T. A. Henzinger, and A. L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: two faces of the same coin. In *Proc. IEEE/ACM Int. Conf. on Computer-aided Design*, pages 132–139. ACM, 2002.

VNS09. R. Vaculín, R. Neruda, and K. P. Sycara. The process mediation framework for semantic web services. *International Journal of Agent-Oriented Software Engineering, IJAOSE*, 3(1):27–58, 2009.

YS97. D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.