Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns

Paola Inverardi and Massimo Tivoli
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila, Italy
{paola.inverardi,massimo.tivoli}@univaq.it

***Abstract*—Ubiquitous and pervasive computing promotes the creation of an environment where Networked Systems (NSs) eternally provide connectivity and services without requiring explicit awareness of the underlying communications and computing technologies. In this context, achieving interoperability among heterogeneous NSs represents an important issue. In order to mediate the NSs interaction protocol and solve possible mismatches, connectors are often built. However, connector development is a never-ending and error-prone task and prevents the eternality of NSs. For this reason, in the literature, many approaches propose the automatic synthesis of connectors. However, solving the connector synthesis problem in general is hard and, when possible, it results in a monolithic connector hence preventing its evolution. In this paper, we define a method for the automatic synthesis of modular connectors, each of them expressed as the composition of independent mediators. A modular connector, as synthesized by our method, supports connector evolution and performs correct mediation.**

I. Introduction

The near future envisions an ubiquitous and pervasive computing environment that enables heterogeneous Networked Systems (NSs) to provide and access services without requiring an explicit awareness of the underlying communications and computing technologies [1]. In this scenario, a problem that arises when composing heterogeneous NSs is how to achieve their interoperability by solving protocol mismatches.

A widely used technique to cope with this problem is to build connectors [2], [3] that bridge the communication among heterogenous protocols and coordinate their interaction. However, due to the potentially infinite number of different available protocols, connector development is a never-ending and error-prone task and prevents the eternality of NSs. The efficacy of composing NSs is proportional to the level of interoperability of the respective underlying technologies. For this reason, starting from the pioneering work in [4], many approaches propose the automatic synthesis of connectors, see [5]–[8] just to cite a few.

As a matter of fact, the connector synthesis problem is hard in the sense that not all possible protocol mismatches are solvable. For instance, building a connector that reconciles the component interaction by reordering certain sequences of exchanged messages can lead to unbounded executions. As shown in [9], a suitable termination criterion can be defined with the aim of under-approximating unbounded interactions by means of bounded ones whenever a pattern of behaviour indicating potential infinity occurs. Thus, practical solutions can only deal with a combination of specific mediation patterns that correspond to tractable protocol mismatches [10]–[13]. However, these solutions eventually result in a monolithic connector hence preventing evolution, synthesis and maintenance.

In this paper, we define a method for the automatic synthesis of modular connectors. A modular connector is a composition of independent *mediators*. Each mediator is a primitive sub-connector that realizes a mediation pattern, which corresponds to the solution of a recurring protocol mismatch. The advantage of our connector decomposition is twofold: (i) it is *correct*, i.e., as for its monolithic version, a modular connector performs a mediation that is free from possible mismatches; and (ii) it promotes connector *evolution*, hence also easing code synthesis and maintenance. To show (i), we define the semantics of protocols (as well as of mediators and connectors) by using a revised version of the *Interface Automata* (IA) theory described in [14]. Then, we prove that a modular connector for two protocols P and R enjoys the same correctness properties of the monolithic connector obtained by expressing the synthesis problem as a *quotient* problem between P and R [15]. Concerning the set of considered mediation patterns and, hence, connector modularization, our synthesis method relies on a revised version of the connector algebra described in [16]. It is an algebra for reasoning about protocol mismatches where basic mismatches can be solved by suitably defined primitives, while complex mismatches can be settled by composition operators that build connectors out of simpler ones. We revise the original algebra by adding an iterator operator and by giving its semantics in terms of our revised IA theory. For (ii), we make use of a case study in the e-commerce domain to illustrate that relevant changes can be applied on a modular connector by acting on its constituent mediators, without entirely re-synthesizing its protocol.

The paper is organized as follows. Section II puts the bases for the definition of our synthesis method. Section III introduces the *purchase order mediation scenario* that we use as case study in the sequel of the paper. In Section IV, we formalize our method and illustrate it at work on the case study. In Section V, we state correctness and show how the method supports connector evolution. Section VI discusses related work, and Section VII provides final remarks and future research directions.

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA

II. Preamble on the Synthesis Method

We assume that a NS comes together with a IA-based specification of its Interaction Protocol (IP). Note that this assumption is supported by the increasing proliferation of techniques for software model elicitation (see [17]–[22] just to cite a few). The IP of a NS expresses the order in which *input* and *output* actions are performed while the NS interacts with the environment. Actions are used to abstract messages that can be sent (outputs) or received (inputs). Inputs are received from and controlled by the environment, whereas outputs are controlled and emitted by the NS. A NS can perform also *hidden* actions corresponding to internal computation. In this section, we instantiate some definitions from the IA theory in [14] to our context and, when needed for the purposes of connector synthesis, we also add new ones.

Definition 1 (*Interaction Protocol Specification*)
An Interaction Protocol Specification *(IPS)* P *is a tuple* $(A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$*, where* A_P^I*,* A_P^O*,* A_P^H *are disjoint sets referred to as* input*,* output*, and* hidden *actions (the union of which we denote by* A_P*),* S_P *is a finite set of* states *with* $s_P^0 \in S_P$ *being the designated* initial state*, and* $\delta_P : S_P \times A_P \to S_P$ *is the partial* transition function*.*

Intuitively, from a state, the NS may either emit any output that is enabled according to its IPS or perform internal computation. If the environment supplies an input that is enabled, the reaction of the NS is according to its IPS. If the input is not enabled, this causes an *inconsistency*.

Let a be an action, we denote with $\overline{a}$ its *complement*. If a is an input action then $\overline{a}$ is the corresponding output action, and vice versa. When a is a hidden action, $\overline{a}$ is hidden as well and its label is the one of a followed by ';'. Abusing notation, we extend the complement also to IPSs. That is, let P be an IPS, then $\overline{P}$ denotes its complement and it is P where all input, output, and hidden actions have been complemented. Furthermore, we consider a special kind of IPS denoted by I and called *identity*. It is defined as the IPS $(\emptyset, \emptyset, \emptyset, \{s_I^0\}, s_I^0, \emptyset)$. To give the possibility to express IPSs that take a message as input and forward the same message as output, given an action a, we consider also the action a' as semantically equivalent to a (yet syntactically different). We write $s \xrightarrow{a}_P s'$ to denote that $\delta_P(s, a) = s'$ (or, equivalently, that $(s, a, s') \in \delta_P$). An action a is *enabled* in s, if $\delta_P(s, a)$ is defined. $A_P(s)$ denotes the set of actions in P that are enabled in s. We denote with $s \overset{a}{\Longrightarrow}_P s'$ a sequence of internal actions starting from s, terminating to s', and with (an observable action) a in some point in the middle of the sequence. We write $s \overset{a}{\rightsquigarrow}_P$ to denote that, from s, P can perform a sequence of hidden actions terminating with a. Abusing notation, $\rightsquigarrow_P(s, a)$ denotes the set of states, in P, that are reachable from s by performing a sequence of hidden actions terminating with a.

Definition 2 (*Traces of an IPS*)
Let $P = (A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$ *be an IPS, a* trace *of* P *is a* $t_P \in ((A_P^I \cup A_P^O)^* \cup \{\epsilon\})$ *defined in such a way that* $t_P = \epsilon \vee \exists n > 0, s_0^P, \ldots, s_n^P \in S_P : t_P = a_1 a_2 \ldots a_n \wedge s_0^P \overset{a_1}{\Longrightarrow} \ldots \overset{a_n}{\Longrightarrow} s_n^P$*, where* ϵ *denotes the so called* empty trace*.*

We denote with A_{t_P} the *set of actions* in t_P; $|t_P|$ is the length of t_P. Furthermore, $t_P(a)$ is the first position of a in t_P. Finally, $Tr(P)$ denotes the *set of traces* of P.

From hereon let $P = (A_P^I, A_P^O, A_P^H, S_P, s_P^0, \delta_P)$ and R=$(A_R^I, A_R^O, A_R^H, S_R, s_R^0, \delta_R)$ be two IPSs. P and R may only be composed if their action sets are compatible with each other. IPSs P and R are *composable* if $A_P^H \cap A_R = \emptyset$, $A_P \cap A_R^H = \emptyset$, $A_P^I \cap A_R^I = \emptyset$, and $A_P^O \cap A_R^O = \emptyset$. We denote with $common(P, R)$ the set $A_P \cap A_R$ of common actions. Note that if P and R are composable then $common(P, R) = (A_P^I \cap A_R^O) \cup (A_P^O \cap A_R^I)$. To define the parallel composition of composable IPSs, we use a *product* operation that accounts for possible semantically equivalent actions.

Definition 3 (*Product of two IPSs*)
The product of P *and* R *is an IPS* $P \otimes R = (A_{P\otimes R}^I, A_{P\otimes R}^O, A_{P\otimes R}^H, S_P \times S_R, (s_P^0, s_R^0), \delta_{P\otimes R})$*, where:*

- $A_{P\otimes R}^I = (A_P^I \cup A_R^I) \setminus common(P, R)$*;*
- $A_{P\otimes R}^O = (A_P^O \cup A_R^O) \setminus common(P, R)$*;*
- $A_{P\otimes R}^H = A_P^H \cup A_R^H \cup common(P, R)$*;*
- $(p, r) \xrightarrow{a}_{P\otimes R} (p', r') \Leftrightarrow$
  - $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge \delta_R(r, a')$ *(resp.,* $\delta_R(r, a)$*) is not defined* $\wedge\ r = r' \wedge a$ *(resp.,* a'*)* $\notin common(P, R)$*;*
  - $p = p' \wedge \delta_P(p, a')$ *(resp.,* $\delta_P(p, a)$*) is not defined* $\wedge$ $r \xrightarrow{a\ (resp.,\ a')}_R r' \wedge a$ *(resp.,* a'*)* $\notin common(P, R)$*;*
  - $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge r \xrightarrow{a\ (resp.,\ a')}_R r' \wedge$ a *(resp.,* a'*)* $\in common(P, R)$*;*
  - $p \xrightarrow{a\ (resp.,\ a')}_P p' \wedge r \xrightarrow{a'\ (resp.,\ a)}_R r'$*.*

The product can introduce a number of inconsistencies when one of the two protocols is willing to offer an output action in the common alphabet, but the second is not able to offer, possibly after a sequence of hidden actions, the corresponding input action (accounting also for possible semantically equivalent actions). $Inconsistencies(P, R)$ is the set of states in $P \otimes R$ from which inconsistencies can arise. The kernel of the inconsistencies in $P \otimes R$ is the set of states (p, r) for which: either (i) there is some $a \in common(P, R)$ (resp., $a' \in common(P, R)$) such that one of p and r can make an a-labelled (resp., a'-labelled) output transition, but the other cannot match it with the corresponding input transition; or (ii) one of p and r can make an a-labelled (resp., a'-labelled) output transition, but the other cannot match it with the semantically equivalent input transition. Thus, $Inconsistencies(P, R)$ is then the set of those states in the kernel, plus those that can reach a state in the kernel by a sequence of transitions labelled by either output or hidden actions.

Definition 4 (*Composition of two IPSs*)
The composition *of two composable IPSs* P *and* R*, written as* $P||R$*, is defined to be* $P \otimes R$ *after pruning all states in* $Inconsistencies(P, R)$*, providing the initial state* (s_0^P, s_0^R)

is contained within the remaining automaton. Otherwise, the composition is undefined.

As formally proven in [14], $||$ is a *compositional* operator meaning that, given three composable IPSs P, Q, and R, then $(P||Q)||R = P||(Q||R)$.

As it will be clear in Section V, to state correctness of our synthesis method, we use a notion of *refinement* based on a version of *alternating simulation* [23] that accounts for both hidden actions and semantically equivalent ones. Informally, R refines P if all input steps of P can be simulated by R, and all the output steps of R can be simulated by P, considering that internal steps of P and R are independent and an observable step can be simulated by a semantically equivalent one. We make use of a "semantic" inclusion operator, denoted by $\subseteq_{sem}$, between sets of actions. Its meaning is the same as $\subseteq$, but it accounts also for semantically equivalent actions. That is, given two sets of actions S and S', if $S \subseteq S'$ then $S \subseteq_{sem} S'$ and, given an action a, either $(\{a\} \cup S) \subseteq_{sem} (\{a'\} \cup S')$ or $(\{a'\} \cup S) \subseteq_{sem} (\{a\} \cup S')$.

Definition 5 (*Refinement between IPSs*)
R refines P, *i.e.,* $P \succeq R$, *if the following conditions hold:*

- $A^I_P \subseteq_{sem} A^I_R \wedge A^O_R \subseteq_{sem} A^O_P$;
- *there exists an* alternating simulation *as a binary relation* $\succeq \subseteq S_P \times S_R$ *such that for all states* $s \in S_P$ *and* $r \in S_R$, *with* $s \succeq r$, *the following conditions hold:*
  - (ii.1) $\{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A^I_P\} \subseteq_{sem} \{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A^I_R\}$;
  - (ii.2) $\{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A^O_R\} \subseteq_{sem} \{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A^O_P\}$;
  - (ii.3) $\forall z \in \{x | s \overset{x}{\rightsquigarrow}_P \wedge x \in A^I_P\} \cup \{y | r \overset{y}{\rightsquigarrow}_R \wedge y \in A^O_R\}$, $r' \in \rightsquigarrow_R(r, z) : \exists s' \in \rightsquigarrow_P(s, z) : s' \succeq r'$;
- $s^0_P \succeq s^0_R$.

Refinement between IPSs is a *preorder* (i.e., reflexive and transitive). Note that P and $\overline{P}$ are always composable and, under refinement, $P||\overline{P}$ and I are equivalent, i.e., $P||\overline{P} \succeq I$ and $I \succeq P||\overline{P}$. The same holds for $P||I$ and P, i.e., $P \succeq P||I$ and $P||I \succeq P$. Furthermore, refinement is *compositional* meaning that $P||P' \succeq R||R'$, if both $P \succeq R$ and $P' \succeq R'$.

As already mentioned in Section I, in order to reason about protocol mismatches, we consider a revised version of the connector algebra described in [16]. In the following, we report only the portion of the algebra that is relevant for the purposes of this work.

From hereon let $\mathcal{A}$ be the universal set of actions. To define the primitives, $\mathcal{AP}(\mathcal{A})$, of the connector algebra we exploit a categorisation of mediator patterns that represent possible solutions to recurring protocol mismatches. As described below, for each type of mismatch, a pattern can be defined as a solution to the interaction incompatibility.

1) **Extra send:** it concerns the possibility for a NS to generate either a redundant or an additional message. Such a mismatch can be solved by considering a pattern which represents mediators that *consume* messages.
2) **Missing send:** it occurs when a NS expects either a redundant message or a message that is not sent by another NS. It can be solved by a pattern which represents mediators that *produce* messages.
3) **Signature mismatch:** two messages of two NSs can be functionally compatible yet syntactically inconsistent. Mediators, compliant with a pattern that *translates* a message into another, can solve this mismatch.
4) **Split message mismatch:** a NS may expect to receive a message as a sequence of fragments of it. This mismatch can be solved by introducing mediators, compliant with a pattern that *splits* the message into an ordered sequence of its fragments.
5) **Merge message mismatch:** it is symmetric to the previous one. The mismatch can be solved by exploiting a pattern which represents mediators that *merge* an ordered sequence of message fragments into a composite message.
6) **Reordering mismatch:** a NS expects to receive messages in an order different from the order used by the sending NS. It can be solved by mediators, compliant with a pattern that *reorders* an ordered sequence of messages into another one that is ordered by applying a specific message permutation.

According to the pattern categorisation discussed above, the syntax of a term t in $\mathcal{AP}(\mathcal{A})$ is given by:

$t ::= t \odot t \mid t^* \mid (t) \mid p$
$p ::= \mid Cons(a) \mid Prod(a) \mid Trans(a, b) \mid$
$Split(a, [a_1, \ldots, a_n]) \mid Merge([a_1, \ldots, a_n], a) \mid$
$Order([a_1, \ldots, a_n], \pi, [a'_1, \ldots, a'_n])$

where $a, a_i, a'_i, b \in \mathcal{A}$ and π is a permutation of $\{1, \ldots, n\}$. The symbol $\odot$ is a binary operator called *plugging*, and * is an unary operator called *iterator*. The semantics of $\mathcal{AP}(\mathcal{A})$ is given in terms of a function $[\![\cdot]\!] : \mathcal{AP}(A) \rightarrow \mathcal{IPS} \cup \{Err\}$, where $\mathcal{IPS}$ is the universal set of IPSs and Err represents the undefined IPS, i.e., it has no states. For any term t, the denotation $[\![t]\!]$ is defined inductively.

If t is a primitive, then $[\![t]\!]$ is the corresponding IPS[1], providing the parameters are well-defined (otherwise $[\![t]\!] = Err$).

If t is a compound term, such as $t = p \odot r$, then $[\![t]\!]$ is given by: if either $[\![p]\!]$ or $[\![r]\!]$ is equal to Err, then $[\![t]\!] = Err$. Alternatively, if $[\![p]\!]$ and $[\![r]\!]$ are not composable or $[\![p]\!] \,||\, [\![r]\!]$ is not defined, then $[\![t]\!] = Err$. Otherwise, $[\![t]\!] = [\![p]\!] \,||\, [\![r]\!]$.

If t is an iterative term, such as $t = p^*$, then $[\![t]\!]$ is given by the IPS $[\![p]\!]$ suitably adjusted, by means of (hidden) ϵ-transitions, for allowing cyclic behaviour [24]. Otherwise, if $[\![p]\!]$ is undefined, then $[\![t]\!] = Err$.

III. The Purchase Order Mediation Scenario

We borrow a case study from [25], slightly revised to better show the peculiarities of our method. The case study concerns the so called *Purchase Order Mediation scenario* from the Semantic Web Service (SWS) Challenge[2]. As stated in [25], "*it represents a typical real-world problem that is as close to industrial reality as practical... This scenario highlights*

[1]E.g., $Trans(a, b)$ is the IPS that takes a as input and gives b as output.
[2]http://sws-challenge.org/wiki/.

the various mismatches that can be encountered when making heterogeneous systems interoperable". The scenario considers two NSs implemented using different standards and protocols: the *Blue Service* (BS) and the *Moon Client* (MC).

Figures 1 and 2 show the IPSs of MC and BS, respectively. The box that encloses the transition system corresponds to the protocol interface. Labelled arrows pointing at the interface correspond to inputs, whereas departing arrows are outputs. Both MC and BS do not perform hidden actions.

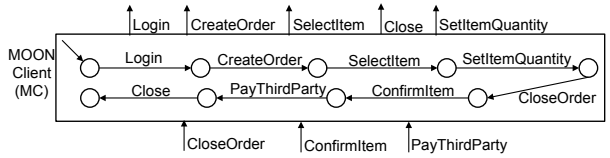

Fig. 1. MC: Moon Client Interaction Protocol

MC orders products by assuming to interact with a Moon Service according to the following protocol: MC performs authentication (Login) to prove that it represents an authorized customer; then an order can be created (CreateOrder) by starting from an empty cart, and individual items can be added to it. Thus, an item is selected (SelectItem) by also specifying the needed quantity (SetItemQuantity). MC is asked, from the Moon Service, to confirm the addition of the item to the order (CloseOrder followed by ConfirmItem). Finally, the order payment is asked by a third-party payment system (PayThirdParty) so as the order can be closed (Close).

BS is a service for purchasing orders. A client initiates an order (StartOrder) and adds items to it (AddItemtoOrder). For each added item, BS asks its clients to confirm the item addition (GetConfirmation) and, then, to place the order (PlaceOrder). Finally, a client can quit the session (Quit).

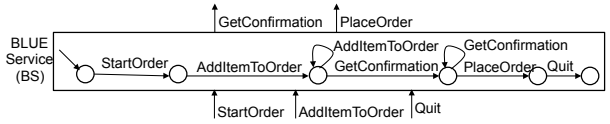

Fig. 2. BS: Blue Service Interaction Protocol

MC cannot communicate with BS due to the following protocol mismatches, of two different types.

Communication mismatches concern the semantics and granularity of the protocol actions. For instance, a client of BS provides its identifier while placing the order, whereas MC has to authenticate before performing any operation. Furthermore, BS provides a single operation to add an item, with the needed quantity, to the order, whereas MC expects to use two different operations, one for the addition and one for the quantity specification. To solve these kind of mismatches it is necessary to assume and use ontology knowledge in order to align the two protocols to the same concepts and language.

Coordination mismatches concern the control structure of the protocols and can be solved by means of the mediator that can mediate the conversation between the two protocols so that they can actually interact. For instance, BS requires its clients to confirm the ordered items and then place the order, whereas MC expects to confirm the ordered items only once the order is placed. Finally, BS allows the addition of several kinds of items in the same order, whereas MC performs the addition of only one kind of item per order.

As stated in [25], MC and BS "*are provided by the SWS-Challenge organizers and can not be altered (although their description may be semantically enriched)*". In particular, in [25], by exploiting an ontology [26] in the domain of *purchase processes*, the description of both MC and BS has been semantically enriched. This domain ontology is denoted by DO and shown in Figure 3. DO shows the relations holding between the various concepts used by MC and BS as purchase order systems. Typically, ontologies account for two fundamental relations between concepts: *subsumption* and *aggregation* [27]. A concept a is *subsumed by* a concept b, in a given ontology O, if in every model of O the set denoted by a is a subset of the set denoted by b. A concept a is *an aggregate of* concepts $b_1, \ldots, b_n$ if the latter are part of the former. It is worth to mention that our use of the ontology concept is specific of the CONNECT project (this work is part of). Thus, in the following, we will exploit these notions to our purposes. That is, concepts in DO correspond to NS input/output actions. The two relations between concepts are, then, used to account for the granularity of the data that define the structure of the messages exchanged by the respective input/output actions. For example, the message associated to the request of AddItemToOrder is an aggregate of the messages associated to the requests of SelectItem and SetItemQuantity.

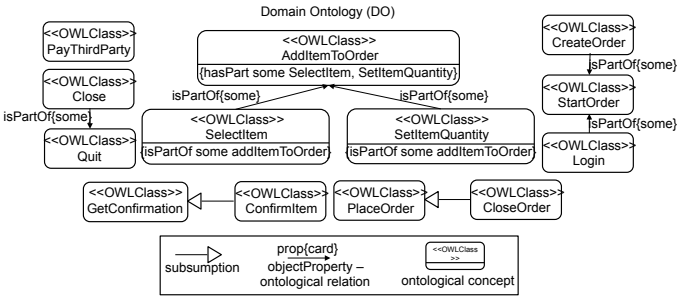

Fig. 3. DO: Domain Ontology

Indeed, in the current practice of ontology development, one cannot expect to find a highly specific (to the considered NSs) ontology as DO. The production of DO involves the extension of a more general ontology in the domain of e-commerce. This extension allows the definition of two specific ontologies that represent a semantic description for MC and BS, respectively. Then DO results from discovering mappings between these two ontologies. Note that nowadays there exist

several ontologies (e.g., for e-commerce domains, see at: http://www.heppnetz.de/projects/goodrelations/) that can serve as common descriptions of specific domains, which can be shared among different applications. Furthermore, they are expressed by using languages (e.g., OWL, DAML, OIL, RDF Schema, just to mention a few) that allow ontology extension and automated reasoning for ontology mapping discovery [28].

IV. Synthesis of Modular Connectors

A mediator has an input-output behaviour (not necessarily strictly sequential, e.g., for allowing reordering of messages), and it is a "reactive" software entity harmonizing the interaction between heterogeneous NSs by intercepting output messages from one NS and eventually issuing to another NS the *co-related* input messages. Message co-relations can be inferred by taking into account ontological information, which is characterized as follows. We recall that the following definitions apply in the scope of our problem that is defined by two NSs enriched with the respective IPS and ontologies as discussed in the previous section.

Definition 6 (*Protocol Ontology*)
A protocol ontology O *is a triple* (C_O, S_O, A_O)*, where* C_O *is the set of* ontological concepts *and it is partitioned into the two disjoint sets* IC_O *and* OC_O *of* input *and* output *ontological concepts, respectively.* S_O *and* A_O *are the* subsumption *and* aggregation *relations among concepts, respectively. The following properties hold:*

- $(C_O \subseteq \mathcal{A}) \wedge (C_O = IC_O \cup OC_O) \wedge (IC_O \cap OC_O = \emptyset)$*;*
- $S_O \subseteq IC_O \times OC_O$*;*
- $A_O \subseteq IC_O \times OC_O^n \wedge n > 0$.

If $(a, b) \in S_O$ then we write that a *is subsumed by* b, and (a, b) is a *subsumption pair*. This means that the output b from a component C must precede the input a from a (different) component C'. In other words, since the set of data constituting a is a subset of those constituting b, to build the message associated to a, one needs to process the data contained in the message associated to b first. A mediator that makes C and C' able to interoperate would take b from C and send, as co-related message, a to C'. If $(a, b_1, \ldots, b_n) \in A_O$ then we write that $b_1, \ldots, b_n$ *are part of* a. Furthermore, $(a, b_1, \ldots, b_n)$ is an *aggregation tuple*, and a can be built by merging the messages associated to $b_1, \ldots, b_n$. Thus, a mediator would take $b_1, \ldots, b_n$ in any order, and send a as the merge of them, plus possible additional data explicitly specified in O for a.

We write that a protocol ontology $O{=}(IC_O{\cup}OC_O,S_O,A_O)$ is *valid* for protocols P and R if and only if $IC_O = A_P^I \cup A_R^I$ and $OC_O = A_P^O \cup A_R^O$.

Coming back to our case study, let $O{=}(IC_O{\cup}OC_O, S_O,A_O)$ be the protocol ontology shown in Figure 3 then:

$IC_O{=}\{ConfirmItem,PayThirdParty,CloseOrder,$
$\quad StartOrder,AddItemToOrder,Quit\}$;
$OC_O{=}\{GetConfirmation,Login,CreateOrder,SelectItem,$
$\quad SetItemQuantity,PlaceOrder,Close\}$;
$S_O{=}\{(ConfirmItem,GetConfirmation),$
$\quad (CloseOrder,PlaceOrder)\}$;
$A_O{=}\{(AddItemToOrder,SelectItem,SetItemQuantity),$
$\quad (Quit,Close),\ (StartOrder,Login,CreateOrder)\}$;
and it is a valid protocol ontology for MC and BS.

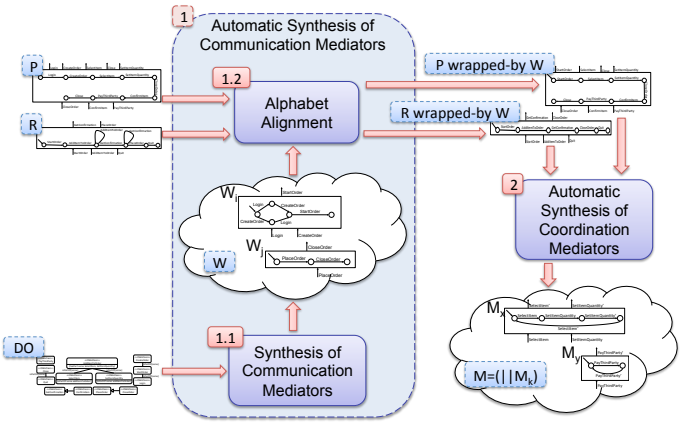

Fig. 4. Overview of the method

In Sections IV-A and IV-B, we formalize our synthesis method as organized into two phases, respectively. They are performed only if the protocol ontology is valid. Before going into the details of the two phases, we give an overview of them. Figure 4 pictorially shows the phases (as rounded-corner rectangles) with their related input/output artefacts. The numbers denote the order in which the phases are carried out. The first phase splits into two sub-phases (1.1 and 1.2); it takes as input a domain ontology DO, for IPSs P and R, and automatically synthesizes a set, W, of *Communication Mediators* (CMs). CMs are represented as terms in $\mathcal{AP}(\mathcal{A})$ and they are responsible for solving communication mismatches as defined in Section III. In particular, the CMs in W are used as wrappers for P and R so to "align" their different alphabets to the same alphabet. Roughly speaking, the goal of this phase is to make two heterogeneous protocols "speak" the same language. To this aim, the synthesized CMs translate an action from an alphabet into a certain sequence of actions from another alphabet. However, despite the achieved alphabet alignment, coordination mismatches (as defined in Section III) are still possible; the second phase is for solving such mismatches. The synthesis of *COordination Mediators* (COMs) is carried out by reasoning on the traces of the "wrapped" P and R. As detailed in Section IV-B, for all pairs of traces, if possible, a COM that makes the two traces interoperable is synthesized as a term in $\mathcal{AP}(\mathcal{A})$. The parallel composition of the synthesized COMs represents, under alphabet alignment, the correct modular connector for P and R.

A. Automatic Synthesis of Communication Mediators

Our method synthesizes CMs by defining the semantics of subsumption pairs and aggregation tuples in terms of IPSs. We do this by exploiting the primitives and operators of $\mathcal{AP}(\mathcal{A})$,

hence following a modular approach. CMs *align* different alphabets of two protocols according to the defined ontological relations (Figure 3). For instance, the CMs for the case study introduced in Section III are:

M_1=$[\![Split(GetConfirmation, [ConfirmItem, x_1]) \odot Cons(x_1)]\!]$;
M_2=$[\![Split(PlaceOrder, [CloseOrder, x_2]) \odot Cons(x_2)]\!]$;
M_3=$[\![Trans(SelectItem, x_3) \odot Trans(SetItemQuantity, x_4) \odot Prod(x_5) \odot Merge([x_3, x_4, x_5], AddItemToOrder)]\!]$;
M_4=$[\![Trans(Close, x_6) \odot Prod(x_7) \odot Merge([x_6, x_7], Quit)]\!]$;
M_5=$[\![Trans(Login, x_8) \odot Trans(CreateOrder, x_9) \odot Prod(x_{10}) \odot Merge([x_8, x_9, x_{10}], StartOrder)]\!]$.

Definition 7 (*Semantics of Subsumption Pairs*)
Let $O = (C_O, S_O, A_O)$ be a protocol ontology, the semantics *of $(a, b) \in S_O$ is given by $[\![Split(b, [a, x]) \odot Cons(x)]\!]$.*

Definition 8 (*Semantics of Aggregation Tuples*)
Let $O = (C_O, S_O, A_O)$ be a protocol ontology, the semantics *of $(a, b_1, \ldots, b_n)$ is given by $[\![Trans(b_1, x_1) \odot \ldots \odot Trans(b_n, x_n) \odot Prod(x_{n+1}) \odot Merge([x_1, \ldots, x_n, x_{n+1}], a)]\!]$.*

For the purposes of *alphabet alignment*, when synthesized out of a subsumption pair (a, b), a CM is used as a *wrapper for the output b of a protocol*. Instead, when synthesized out of an aggregation tuple $(a, b_1, \ldots, b_n)$, a CM is used as a *wrapper for the input a of a protocol*. Thus, we define a further derived composition operator called *wrapping*. P can be *wrapped* by R on action a if and only if P *and* R *are composable*, $a \in Common(P, R)$, and R is a CM.

Definition 9 (*Wrapping of an IPS*)
The wrapping of P by a CM R, on action a, is an IPS $P \triangleleft_a R = (A^I_{P \otimes R}, A^O_{P \otimes R}, A^H_{P \otimes R}, S_{P \triangleleft_a R}, s^0_P, \delta_{P \triangleleft_a R})$, where:

- $S_{P \triangleleft_a R} = (S_P \cup S_R)$;
- $\delta_{P \triangleleft_a R} = (\delta_P \cup \delta_R) \setminus \{(s, a, s') | s, s' \in (S_P \cup S_R)\} \cup \{(p, \epsilon_{s^0_R}; , s^0_R) | (p, a, p') \in \delta_P \wedge \delta_R(S^0_R, a)$ *is not defined*$\} \cup \{(r, \epsilon_{p'}; , p') | (p, a, p') \in \delta_P \wedge r \neq s^0_R \wedge \delta_R(r, a)$ *is defined*$\} \cup \{(p, \epsilon_{r'}; , r') | (p, a, p') \in \delta_P \wedge \delta_R(s^0_R, a)$ *is defined*$\}$.

Definition 10 (*Alphabet Alignment*)
Given P and R to be mediated, let $O = (C_O, S_O, A_O)$ be the valid protocol ontology specification for P and R. Let $S_1, \ldots, S_n$ be the CMs synthesized according to S_O, and let $A_1, \ldots, A_h$ be those synthesized according to A_O. The algorithm for the alphabet alignment *step is as follows:*

procedure *Alignment*
input: P, R, O, $S_1, \ldots, S_n$, $A_1, \ldots, A_h$
output: $AlignedProtocols$

1: $AlignedProtocols := \emptyset$
2: ***for* each** $j : (A^I_P \cap A^O_{A_j} \neq \emptyset) \wedge (A^O_{A_j} = \{a\})$ ***do***
3: $\quad P := P \triangleleft_a A_j$
4: ***end for***
5: ***for* each** $k : (A^O_P \cap A^I_{S_k} \neq \emptyset) \wedge (A^I_{S_k} = \{b\})$ ***do***
6: $\quad P := P \triangleleft_a S_k$
7: ***end for***
8: $AlignedProtocols := AlignedProtocols \cup \{P\}$
9: ***for* each** $j : (A^I_R \cap A^O_{A_j} \neq \emptyset) \wedge (A^O_{A_j} = \{a\})$ ***do***
10: $\quad R := R \triangleleft_a A_j$
11: ***end for***
12: ***for* each** $k : (A^O_R \cap A^I_{S_k} \neq \emptyset) \wedge (A^I_{S_k} = \{b\})$ ***do***
13: $\quad R := R \triangleleft_a S_k$
14: ***end for***
15: $AlignedProtocols := AlignedProtocols \cup \{R\}$
16: *return* $AlignedProtocols$

It is worth to note that after having performed the procedure Alignment, P and R have the same alphabet of actions, except for possible third-party and hidden actions. Hereafter, given a set W of CMs and a protocol P, we denote with $P_{\triangleleft W}$ the protocol P whose alphabet has been aligned by means of the CMs in W.

Coming back to our case study, after having performed the alphabet alignment step, the protocol of BS becomes the one shown in Figure 5. The protocol of MC does not change.

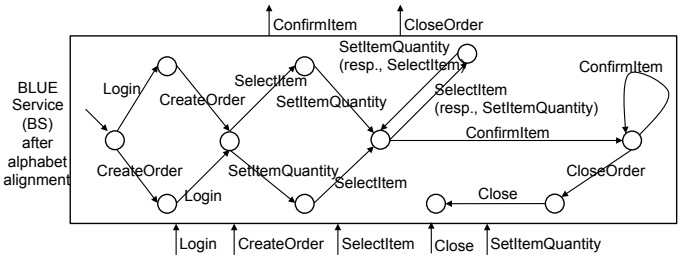

Fig. 5. BS after the alphabet alignment step

However, we recall that CMs are not able to solve all mismatches such as coordination mismatches. For instance, although the two protocols shown in Figures 1 and 5 share the same alphabet of actions, their interaction can still exhibit some mismatches. They are due to (i) messages sent/received in a different order (see the sequences made of $ConfirmItem$ and $CloseOrder$); (ii) third-party messages ($PayThirdParty$); and (iii) extra/missing sends corresponding to redundant messages (possibly also coming from looping/cyclic behavior, e.g., $SelectItem$ and $SetItemQuantity$). Thus, in general, the construction of COMs that can delegate/receive[3], consume, produce, and reorder messages is required.

B. Automatic Synthesis of Coordination Mediators

Given two protocols to be mediated, P and R, whose alphabets have been aligned, our method produces their respective sets of traces. P and R are prefix-closed and hence their sets of traces are finite. Furthermore, possible loops/cycles are considered k times (where k is a parameter of our synthesis algorithm). This means that our method produces finite sets of finite traces. We recall that, by Definition 2, possible hidden actions are not represented within a considered trace.

[3]To/from a third-party.

Hereafter, we denote with $\overline{Tr(P)}^k$ the *set of traces* of P where, for each trace, cycles/loops are considered k times.

As it will be clear later in this section, once $\overline{Tr(P)}^k$ and $\overline{Tr(R)}^k$ have been generated, for all $(t_P, t_R) \in \overline{Tr(P)}^k \times \overline{Tr(R)}^k$, our method tries to synthesize a COM that makes the protocols corresponding to t_P and t_R able to interoperate. If no mediator has been synthesized, then a modular connector for P and R does not exist. Otherwise, a non-empty set of COMs is produced. Indeed, considering all pairs in $\overline{Tr(P)}^k \times \overline{Tr(R)}^k$ is not needed. It is sufficient to consider only the subset of pairs of *semantically related* traces. Traces t_P and t_R are semantically related if every action that does not belong to their set of common actions is a third-party action.

Definition 11 (*Pairs of Semantically Related Traces*)
Given P and R to be mediated, let k be an integer such that $k > 0$, $(t_P, t_R) \in \overline{Tr(P)}^k \times \overline{Tr(R)}^k$ is a pair of semantically related traces *if and only if the following conditions hold:*

- $a \in A_{t_P} \setminus A_{t_R} \Rightarrow \nexists t \in \overline{Tr(R)}^k \setminus t_R : \overline{a} \in A_t$;
- $a \in A_{t_R} \setminus A_{t_P} \Rightarrow \nexists t \in \overline{Tr(P)}^k \setminus t_P : \overline{a} \in A_t$.

We denote with $\Pi(P, R, k)$ the k-bounded set of pairs of semantically related traces for P and R.

Before continuing our formalization, Definition 11 deserves some discussion. It formalizes a *strong* notion of *semantic co-relation* for traces. For instance, let us consider traces t_P and t_R as shown in Figure 6. They are not semantically related because both actions e and f are in the set of common actions of P and R and, hence, cannot be considered as third-party actions, although are not common for t_P and t_R.

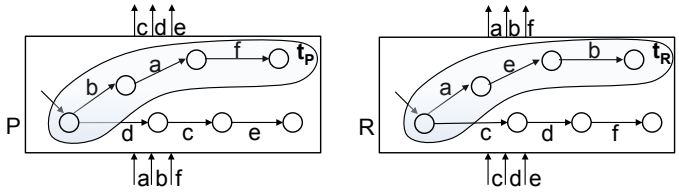

Fig. 6. t_P and t_R: two traces that are not semantically related

However, note that a way of mediating t_P and t_R exists, e.g., by re-ordering a and b and producing e and, afterwards, f. This mediation logic is correct only if e and f are independent from a and b, and a *causality dependency* is defined for f and e, e.g., the production of f depends on the production of e. Otherwise (e.g., the production of e depends on the production of f), it could result in an inconsistent mediation. Note that the relations defined in the protocol ontology do not necessarily represent all the possible causality dependencies. They are those related only to the semantics and structure of the exchanged data. That is, causality dependencies related to the control structure, and hence depending on the protocol semantics, could not be represented. Thus, despite the existence of a protocol ontology, without further information about protocol actions and their causality dependencies, one cannot in general know how to correctly mediate t_P and t_R. The safest choice that one can take, while synthesizing COMs, is to discard this pair of traces. However, to avoid inconsistencies, discarding a pair of traces means that the COM being synthesized has to consume all the outputs from the two traces. In other words, although this strong notion of semantic co-relation leads to synthesize correct modular connectors (i.e., that do not introduce inconsistencies), the synthesized connector might constrain too much the set of interactions on which the two protocols are made able to interoperate, hence possibly under-using them.

To avoid this problem we can consider a *weak* notion of semantic co-relation. This notion assumes the existence of a specification of the desired *coordination protocol* that the COM to be synthesized should satisfy. For instance, by referring to the example depicted in Figure 6, the coordination protocol specification could express that *e must always precedes f* or, equivalently, that *f must be always eventually performed after e*. For space reasons, we do not formalize this weak notion of semantic co-relation and, hereafter, we consider only the notion formalized by Definition 11.

Continuing the formalization of our method, for each $(t_P, t_R) \in \Pi(P, R, k)$, the method computes the so called *difference pair* (t'_P, t'_R). t'_P (resp., t'_R) is a sub-trace of t_P (resp., t_R) representing, in a single sequence, the sequences of actions in which t_P (resp., t_R) differs from t_R (resp., t_P). Due to the alphabet alignment, finding a COM for t_P and t_R means finding a COM for t'_P and t'_R. Since t_P and t_R are semantically related, their possible hidden actions have been removed, and their loops/cycles are considered k times, t'_P and t'_R can be different for three reasons only: (i) they have unshared actions, i.e., third-party inputs/outputs; (ii) they exhibit extra/missing sends, i.e., redundant messages, possibly also coming from looping/cyclic behavior; and (iii) they have complementary shared actions that appear in a different order. By means of the COMs to be synthesized, the first ones should be received by a third-party (resp., an NS) and delegated to the receiving NS (resp., third-party), the second ones should be produced/consumed, and the third ones reordered.

Definition 12 (*Coordination Mismatches Resolution*)
Let W be the set of synthesized CMs for protocols P and R, and valid protocol ontology O. Let k be the bound considered for the length of possible loops/cycles in $P_{\triangleleft W}$ and $R_{\triangleleft W}$. The algorithm for the coordination mismatches resolution *step (i.e., the automatic synthesis of COMs) is as follows:*

procedure *CoordinationMismatchesResolution*
input: $P_{\triangleleft W}$, $R_{\triangleleft W}$, k
output: $CoordMediators$

1: $CoordMediators := \emptyset$
2: ***for* each** $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}) \in \Pi(P_{\triangleleft W}, R_{\triangleleft W}, k)$ ***do***
3: *computes the difference pair* $(t'_{P_{\triangleleft W}}, t'_{R_{\triangleleft W}})$ *of* $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}})$
4: ***for* each** $a \in (A_{t'_{P_{\triangleleft W}}} \setminus A_{t'_{R_{\triangleleft W}}}) \cup (A_{t'_{R_{\triangleleft W}}} \setminus A_{t'_{P_{\triangleleft W}}})$ ***do***
5: $CoordMediators := CoordMediators \cup \{ThirdParty(a)\}$

remove a from either $t'_{P_{\triangleleft W}}$ or $t'_{R_{\triangleleft W}}$
end for
if $|t'_{P_{\triangleleft W}}| \neq |t'_{R_{\triangleleft W}}|$ **then**
for each $a \in A_{t'_{P_{\triangleleft W}}} \cup A_{t'_{R_{\triangleleft W}}}$ *that appears more than once in either* $t'_{P_{\triangleleft W}}$ *or* $t'_{R_{\triangleleft W}}$ **do**
$CoordMediators := CoordMediators \cup \{ExtraOrMissing(a)\}$
remove a from either $t'_{P_{\triangleleft W}}$ or $t'_{R_{\triangleleft W}}$
end for
end if
if $t'_{P_{\triangleleft W}} \neq t'_{R_{\triangleleft W}}$ **then**
$CoordMediators := CoordMediators \cup \{Reorder(a, t'_{P_{\triangleleft W}}, t'_{R_{\triangleleft W}})\}$
$CoordMediators := CoordMediators \cup \{Reorder(a, t'_{R_{\triangleleft W}}, t'_{P_{\triangleleft W}})\}$
end if
end for
for each $(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}) \notin \Pi(P_{\triangleleft W}, R_{\triangleleft W}, k)$ **do**
$CoordMediators := CoordMediators \cup Discard(t_{P_{\triangleleft W}}, t_{R_{\triangleleft W}}, P_{\triangleleft W}, R_{\triangleleft W})$
end for
return $CoordMediators$

where the $ThirdParty$, $ExtraOrMissing$, $Reorder$, and $Discard$ (sub-)procedures are defined as follows:

procedure *ThirdParty*
input: a
output: $Mediator$
$Mediator := [\![Trans(a, a')^*]\!]$
return $Mediator$

procedure *ExtraOrMissing*
input: a
output: $Mediator$
if a *is an input action* **then**
$Mediator := [\![Prod(a)^*]\!]$
else
$Mediator := [\![Cons(a)^*]\!]$
end if
return $Mediator$

procedure *Reorder*
input: a, $t'_{P_{\triangleleft W}}$, $t'_{R_{\triangleleft W}}$
output: $Mediator$
$aList := []$
$pTuple := ()$
for each $a \in A_{t'_{P_{\triangleleft W}}}$ **do**
if a *is an output action* **then**
add a as last element of $aList$
add $t'_{R_{\triangleleft W}}(a)$ as last element of $pTuple$
end if
end for
$Mediator := Order(aList, pTuple, aList')$, *where* $aList' = [a'_1, \ldots, a'_n]$ *and* $aList = [a_1, \ldots, a_n]$
$Mediator := [\![Mediator^*]\!]$
return $Mediator$

procedure *Discard*
input: $t_{P_{\triangleleft W}}$, $t_{R_{\triangleleft W}}$, $P_{\triangleleft W}$, $R_{\triangleleft W}$
output: $Mediators$
for each $a \in A_{t_{P_{\triangleleft W}}} \cup A_{t_{R_{\triangleleft W}}} \wedge a \in A^O_{P_{\triangleleft W}} \cup A^O_{R_{\triangleleft W}}$ **do**
$Mediators := Mediators \cup \{[\![Cons(a)^*]\!]\}$
end for
return $Mediators$

Coming back to our case study, the synthesized COMs are:
M_6=$[\![Trans(PayThirdParty, PayThirdParty')^*]\!]$;
M_7=$[\![Order([ConfirmItem, CloseOrder], (2,1), [ConfirmItem', CloseOrder'])^*]\!]$;
M_8=$[\![Order([SelectItem, SetItemQuantity], (2,1), [SelectItem', SetItemQuantity'])^*]\!]$;
M_9=$[\![(Prod(SelectItem))^*]\!]$;
M_{10}=$[\![(Prod(SetItemQuantity))^*]\!]$;
M_{11}=$[\![(Cons(ConfirmItem))^*]\!]$.

By referring to Definition 4, the modular connector for our case study is given by the following composition of COMs: $M = M_6||\ldots||M_{11}$; plus the set $W = \{M_1, \ldots, M_5\}$ of CMs used for the alphabet alignment. As formally shown in the next section, under alphabet alignment, M is a correct connector, i.e., both $\overline{P_{\triangleleft W}} \succeq M||(R_{\triangleleft W})$ and $\overline{R_{\triangleleft W}} \succeq (P_{\triangleleft W})||M$ hold.

V. Correctness and Connector Evolution

By taking into account hidden actions and denying broadcast communication (as done in [14]), the IA theory described in [15] can be used to synthesize, via a *quotient* operator $/$, a monolithic connector M such that $G \succeq P||M||R$, i.e., $M = G/(P||R)$. The formal definition of G is out of the scope of this work. For the purposes of this section, it is sufficient to say that G is an IPS, representing the *connected* system goal, which explicitly models three crucial conditions for correct communication and coordination: (*c1*) $P||M||R$ is not permitted to generate any inconsistencies; (*c2*) $P||M||R$ is only permitted to deadlock when all P, M, and R deadlock; and (*c3*) $P||M||R$ must satisfy the constraints imposed by the given protocol ontology.

Stating correctness of our synthesis method means showing that a modular connector M synthesized for protocols P and R is such that *c1*, *c2*, and *c3* hold, under alphabet alignment. However, note that *c2* and *c3* trivially hold by construction. In fact, when composing in parallel protocols, the only possibility to have "sink" states concerns scenarios in which none of the protocols is willing to perform any action (*c2*); and CMs ensure alphabet alignment (*c3*). Thus, in this section, by considering W as the set of synthesized CMs, we focus on proving that $P_{\triangleleft W}||M||R_{\triangleleft W}$ is free from inconsistencies, i.e., it is defined (*c1*). To do this, we can exploit Definition 5, hence checking both $\overline{P_{\triangleleft W}} \succeq M||R_{\triangleleft W}$ and $\overline{R_{\triangleleft W}} \succeq P||M_{\triangleleft W}$.

Theorem 1 (*Correctness under alphabet alignment*)
Let M be a modular connector synthesized for aligned protocols $P_{\triangleleft W}$ and $R_{\triangleleft W}$, then the following properties hold: (1) $\overline{P_{\triangleleft W}} \succeq M||R_{\triangleleft W}$, *and* (2) $\overline{R_{\triangleleft W}} \succeq P_{\triangleleft W}||M$.

Proof*: To prove* (1)*, we must prove that* (i) $A^I_{\overline{P_{\triangleleft W}}} \subseteq_{sem} A^I_{M||R_{\triangleleft W}} \wedge A^O_{M||R_{\triangleleft W}} \subseteq_{sem} A^O_{\overline{P_{\triangleleft W}}}$*, and* (ii) *there is an alternating simulation from $M||R_{\triangleleft W}$ to $\overline{P_{\triangleleft W}}$, with $s^0_{\overline{P_{\triangleleft W}}} \succeq s^0_{M||R_{\triangleleft W}}$. By construction, $A^I_M = A^O_{P_{\triangleleft W}} \cup A^O_{R_{\triangleleft W}}$ and $A^O_M \subseteq A^I_P \cup A^I_R$. Furthermore, $common(M, R_{\triangleleft W})$ is given by the union set of $A^O_{R_{\triangleleft W}}$ and a subset of $A^I_{R_{\triangleleft W}}$. Thus, by definition of complement operator for IPSs, it follows that both $A^I_{\overline{P_{\triangleleft W}}} \subseteq_{sem} A^I_{M||R_{\triangleleft W}}$ and $A^O_{M||R_{\triangleleft W}} \subseteq_{sem} A^O_{\overline{P_{\triangleleft W}}}$ hold. To prove (ii), we have to show that ii.1, ii.2, and ii.3 of*

Definition 5 hold, where P and R have been replaced with $\overline{P_{\triangleleft W}}$ and $M||R_{\triangleleft W}$, respectively. Let us assume that ii.1 does not hold. This means that $P \triangleleft W$ would perform an output action x that is not consumed by M. This is a contradiction since by construction M always consumes output actions from both $P \triangleleft W$ and $R \triangleleft W$. Analogously, if ii.2 would not hold, then $M||R_{\triangleleft W}$ would produce an output action that does not match any input action of $P \triangleleft W$. Again, this is a contradiction because $M||R_{\triangleleft W}$ produces output actions only when there is the need to match an input from $R_{\triangleleft W}$ with an input from $P_{\triangleleft W}$. ii.3 directly follows from the previous considerations hence recursively propagating the alternating simulation relation from $M||R_{\triangleleft W}$ to $\overline{P_{\triangleleft W}}$. The proof of (2) *is analogous and hence, for space reasons, we omit it.* ■

Concerning the ability, for modular connectors, to evolve in response of changes, the most interesting scenario is related to changes at the level of the protocol ontology. In fact, syntactic changes at the level of the NSs' interface directly correspond to a relabeling of mediator inputs/outputs, and related concepts in the ontology. We recall that the synthesis of COMs deals with sets of traces. Thus, changes at the protocol level imply to re-iter the synthesis step on the affected traces only, hence accordingly changing the corresponding mediators. However, in the worst case, i.e., all the traces of a protocol share at least one action, the entire synthesis step must be repeated.

As an example of a possible change at the level of the protocol ontology, let us go back to our case study and apply the following modification to the ontology O shown in Figure 3: $S_O = S_O \cup \{(AddItemToOrder,\ SelectItem),\ (AddItemToOrder,\ SetItemQuantity)\}$ and $A_O = A_O \setminus \{(AddItemToOrder,\ SelectItem,\ SetItemQuantity)\}$. Although simple, this change highlights the effectiveness of our decomposition with respect to supporting connector evolution. In fact, to address the applied change, it is sufficient to reason compositionally at the level of the algebra-based description of the modular connector M and related set W of CMs, instead of reasoning in terms of its underlying IA-based monolithic representation. In particular, by just looking at the mediators' interface, one can easily recognize that the CM affected by the proposed change is M_3, while M_8, M_9, and M_{10} are the affected COMs. Due to the fact that the aggregation tuple $(AddItemToOrder,\ SelectItem,\ SetItemQuantity)$ has been removed by A_O, M_3 is removed as well. In place of it two CMs, $M_{3.1} = [\![(Split(SelectItem, [AddItemToOrder, z]) \odot Cons(z))^*]\!]$ and $M_{3.2} = [\![(Split(SetItemQuantity, [AddItemToOrder,\ k]) \odot Cons(k))^*]\!]$, are synthesized due to the addition to S_O of the above considered subsumption tuples. Furthermore, we recall that the IPS of BS has been modified in order to align its alphabet to the one of MC. To reflect the change on the performed alphabet alignment, a trace in $\overline{Tr(MC)}^k$ that contains $SelectItem$ and/or $SetItemQuantity$ is modified by considering the following substitution: $\{AddItemToOrder/SelectItem,\ AddItemToOrder/SetItemQuantity\}$. Analogously, a trace in $\overline{Tr(BS)}^k$ that contains either the sequence $\langle SelectItem\ \ SetItemQuantity\rangle$ or $\langle SetItemQuantity\ SelectItem\rangle$ is modified by replacing any of these sequences with $AddItemToOrder$. According to the new alphabet alignment, M_8 is removed and in place of both M_9 and M_{10} the COM $[\![(Prod(AddItemToOrder))^*]\!]$ is synthesized. Note that, in the monolithic connector, $SelectItem$ and $SetItemQuantity$ would always appear one after the other and modifying the connector according to the applied change would mean to solve again the entire quotient problem.

VI. Related Work

Interoperability and mediation have been investigated in several contexts, among which integration of heterogeneous data sources [13], architectural patterns [29], patterns of connectors [12], Web services [10], [11], and algebra to solve mismatches [30]. For space reasons, we discuss only the works, from the different contexts, closest to our method.

The interoperability/mediation of protocols have received attention since the early days of networking. Indeed many efforts have been done in several directions including for example formal approaches to protocol conversion, like in [31], [32].

The seminal work in [4] is strictly related to the notions of mediator presented in this paper. Compared to our connector synthesis, this work does not allow to deal with ordering mismatches and different granularity of the languages (solvable by the split and merge primitives).

Recently, with the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [33], [34]. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

In [12] the authors present an approach for formally specifying connector wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches. In [35] the authors present an algebra for five basic stateless connectors that are symmetry, synchronization, mutual exclusion, hiding and inaction. They also give the operational, observational and denotational semantics and a complete normal-form axiomatization. The presented connectors can be composed in series and in parallel. Although these formalizations supports connector modularization, automated synthesis is not treated at all hence keeping the focus only on connector design and specification.

In [8], the authors use a game theoretic approach for checking whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements. This approach is able to automatically synthesize the converter. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adaptor, by delegating to the user the non-trivial task of specifying that.

In other work in the area of component adaptation [7], it is shown how to automatically generate a concrete adaptor from: (i) a specification of component interfaces, (ii) a partial

specification of the components interaction behavior, (iii) a specification of the adaptation in terms of a set of correspondences between actions of different components and (iv) a partial specification of the adaptor. The key result is the setting of a formal foundation for the adaptation of heterogeneous components that may present mismatching interaction behavior. Assuming a specification of the adaptation in terms of a set of correspondences between methods (and their parameters) of two components requires to know many implementation details (about the adaptation) that we do not want to consider in order to synthesize a connector.

VII. Conclusions and Future Work

In this paper, we formalized a method for the automated synthesis of modular connectors. A modular connector is structured as a composition of independent mediators, each of them corresponding to the solution of a recurring protocol mismatch. We have proven that our connector decomposition is correct and, by means of a case study, we have shown how it promotes connector evolution. An overall advantage of our approach with respect to the work in the state of the art (Section VI) is that our connectors have a modular software architecture organized as a composition of fundamentals mediation primitives. This supports connector evolution and automated generation of the connector's implementation code. In particular, we have recently released a first implementation (http://code.google.com/p/otf-connector/) of both the algebra primitives and the plugging operator. This implementation is based on the use of *Enterprise Integration Patterns* (http://www.eaipatterns.com/) and is developed through the *Apache Camel* framework (http://camel.apache.org/). Because of the way a modular connector is structured, the automatic generation of its actual code written in terms of our algebra implementation is viable and can be achieved with little effort. We have started to show, through its application to the real world case study presented in this paper, that our method supports connector evolution. As future work, we intend to carry out a rigorous empirical investigation to confirm the results reported in this paper. Another future research direction concerns the ability to infer the needed ontological information, out of the interface description of the two protocols, rather than assuming it as given.

Acknowledgments

This work has been partially supported by the FET Connect No 231167.

References

[1] M. Weiser, "The computer for the twenty-first century," *Scientific American*, 1991.

[2] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, 1992.

[3] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 3, 1997.

[4] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, 1997.

[5] P. Inverardi, R. Spalazzese, and M. Tivoli, "Application-layer connector synthesis," in *SFM*, 2011.

[6] V. Issarny, A. Bennaceur, and Y.-D. Bromberg, "Middleware-layer connector synthesis: Beyond state of the art in middleware interoperability," in *SFM*, 2011.

[7] C. Canal, P. Poizat, and G. Salaün, "Model-based adaptation of behavioral mismatching components," *IEEE Trans. Software Eng.*, 2008.

[8] 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 *ICCAD*, 2002.

[9] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe, "Quasi-static scheduling of independent tasksfor reactive systems," in *ICATPN*, 2002.

[10] F. Jiang, Y. Fan, and X. Zhang, "Rule-based automatic generation of mediator patterns for service composition mismatches," in *GPC*, 2008.

[11] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang, "A pattern-based approach to development of service mediators for protocol mediation," in *WICSA*, 2008.

[12] B. Spitznagel and D. Garlan, "A compositional formalization of connector wrappers," in *ICSE*, 2003.

[13] G. Wiederhold and M. Genesereth, "The conceptual basis for mediation services," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 12, no. 5, 1997.

[14] L. de Alfaro and T. A. Henzinger, "Interface automata," in *ESEC/FSE*, 2001.

[15] T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska, "A compositional specification theory for component behaviours," in *ESOP*, 2012.

[16] M. Autili, C. Chilton, P. Inverardi, M. Kwiatkowska, and M. Tivoli, "Towards a connector algebra," in *ISoLA*, 2010.

[17] D. Lo, L. Mariani, and M. Santoro, "Learning extended fsa from software: An empirical assessment," *J. Syst. Softw.*, vol. 85, no. 9, 2012.

[18] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *Proc. of ICSE08*, 2008.

[19] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *IEEE Trans. Software Eng.*, vol. 27, no. 2, 2001.

[20] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller, "Automatically generating test cases for specification mining," *IEEE Transactions on Software Engineering*, vol. 38, no. 2.

[21] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in *Proc. of the ESEC/FSE09*, 2009.

[22] H. Raffelt, B. Steffen, T. Berg, and T. Margaria, "Learnlib: a framework for extrapolating behavioral models," *Int. J. Softw. Tools Technol. Transf.*, vol. 11, no. 5, 2009.

[23] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi, "Alternating refinement relations," in *CONCUR'98*, 1998.

[24] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[25] "CONNECT consortium. CONNECT Deliverable D3.3: Dynamic connector synthesis: revised prototype implementation. FET IP CONNECT EU project, FP7 GA n.231167, http://connect-forever.eu/."

[26] U. Aßmann, S. Zschaler, and G. Wagner, *Ontologies, Meta-Models, and the Model-Driven Paradigm*. Springer, 2006.

[27] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[28] Y. Kalfoglou and M. Schorlemmer, "Ontology mapping: the state of the art," *Knowl. Eng. Rev.*, vol. 18, no. 1, 2003.

[29] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Chichester, UK: Wiley, 1996.

[30] M. Dumas, M. Spork, and K. Wang, "Adapt or perish: Algebra and visual notation for service interface adaptation," in *Business Process Management*, 2006.

[31] K. L. Calvert and S. S. Lam, "Formal methods for protocol conversion," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 1, 1990.

[32] S. S. Lam, "Correction to "protocol conversion"," *IEEE Trans. Software Eng.*, vol. 14, no. 9, 1988.

[33] R. Vaculín and K. Sycara, "Towards automatic mediation of OWL-S process models," *Web Services, IEEE International Conference on*, 2007.

[34] R. Vaculín, R. Neruda, and K. P. Sycara, "An agent for asymmetric process mediation in open environments." in *SOCASE*, 2008.

[35] R. Bruni, I. Lanese, and U. Montanari, "A basic algebra of stateless connectors," *Theor. Comput. Sci.*, vol. 366, no. 1, 2006.