A Theory of Mediators for Eternal Connectors*

Paola Inverardi[1], Valérie Issarny[2], and Romina Spalazzese[1]

[1] University of L'Aquila, Italy
[2] INRIA, CRI Paris-Rocquencourt, France
http://connect-forever.eu/

Abstract. On the fly synthesis of mediators is a revolutionary approach to the seamless networking of today's and future digital systems that increasingly need be connected. The resulting emergent mediators (or CONNECTors) adapt the interaction protocols run by the connected systems to let them communicate. However, although the mediator concept has been studied and used quite extensively to cope with many heterogeneity dimensions, a remaining key challenge is to support on-the-fly synthesis of mediators. Towards this end, this paper introduces a theory of mediators for the ubiquitous networking environment. The proposed formal model: (i) precisely characterizes the problem of interoperability between networked systems, and (ii) paves the way for automated reasoning about protocol matching (interoperability) and related mediator synthesis.

1 Introduction

Today's ubiquitous networked environments embed networked devices from a multitude of application domains, e.g., home automation, consumer electronics, mobile and personal computing domains to name a few. Middleware then positions itself as a core architectural paradigm to enable the heterogeneous networked systems to actually interact together. Middleware provides upper layer interoperability, bridging the gap between application programs and the lower-level hardware and software infrastructure in order to coordinate how application components are connected and how they interoperate, especially in the networked environment. However, ubiquitous networking has introduced new challenges for middleware. Devices need to dynamically detect services available in the ubiquitous networked environment and adapt their own communication protocols to interoperate with them, since networked applications are implemented on top of diverse middleware. As discussed in companion paper [5], a number of systems have been introduced to provide middleware protocols interoperability. However, these address only interoperability at the middleware-layer. Interoperability between networked software systems further concerns the systems' interfaces and behaviors at the application-layer, which calls for supporting *mediators*.

* The work is partly supported by the CONNECT European Project No 231167.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part II, LNCS 6416, pp. 236–250, 2010.

The mediator concept was initially introduced to cope with the integration of heterogeneous data sources [23,22], and as design pattern [4]. However, with the significant development of Web technologies and given abilities to communicate openly for networked systems, many heterogeneity dimensions shall now be mediated. Heterogeneity effectively spans [19]: terminology, representation format and transfer protocols, functionality, and application-layer protocols. The first heterogeneity dimension is addressed by data level mediation, while the second relies on a combination of data level and protocol mediations. Functional mediation then depends on the reasoning about logical relationships between the functional descriptions of networked systems, similar to the notion of behavioral subtyping [14]. Protocol mediation is further concerned with behavioral mismatches that may occur during interactions. Other approaches that share the same formal settings as protocol mediation have been proposed quite some time ago to solve mismatches in the field of supervisory control theory of discrete event systems [12] and, more recently, in the field of software architectures to address communication problems by proposing ad hoc wrappers [18]. However, while the concept of mediator has received a great deal of attention over the last two decades, on-the-fly synthesis of mediators, or *emergent mediation* for short, which is central to seamless interactions in the ubiquitous networked environment, remains a key challenge.

Towards enabling emergent mediation, this paper sets the underlying theory from which protocol matching (interoperability) and mediation may be formally reasoned upon. The work is part of the CONNECT[1] European research project, which investigates the development, from design to prototype implementation, of an overall framework for the seamless networking of heterogeneous networked systems [8,5]. The contribution of this paper is specifically a theory of mediators to precisely characterize:

(i) The interaction protocols that are functionally matching but behaviorally mismatching. Note that we assume given the specification of protocols, either as part of the advertisement of networked systems using some discovery protocol or based on some learning technique like the one discussed in companion paper [7].
(ii) The interoperability notion between protocols based on functional matching. Note that in a first step, we restrict ourselves to interoperability between pairs of protocols and we further do not address data-level heterogeneity, which is being extensively addressed elsewhere [16].
(iii) The behavior of mediators to achieve interoperability under functional matching despite behavioral mismatch.

The paper is organized as follows. We first set the principles of our approach, further describing the terminology we use and giving an illustrative scenario (Section 2). Then, we introduce a formalization for protocols, which paves the way for automated reasoning about protocols functional matching and for the automated synthesis of mediators (Section 3). Finally, we position our contribution

[1] http://connect-forever.eu/

with respect to related work (Section 4) and we conclude with perspectives for future work (Section 5).

2 Eternal Interoperability through Emergent Mediation

The focus of this paper is the *protocol interoperability problem* and our goal is to find an *automated solution* to solve it dynamically. In the following, we give the necessary definitions that set the context of the work.

2.1 Definitions

With the term *protocols*, we refer to *application-layer interaction protocols* (or application-layer observable protocols). That is, a protocol is the behavior of a system in terms of the sequence of messages visible at the interface level, which it exchanges with other systems. We further focus on *compatible* or *functionally matching* application-layer interaction protocols. Functional matching means that protocols can *potentially communicate* by performing *complementary sequences of actions*. "Potentially" means that communication may not be achieved because: (i) the languages of the two protocols are different, although semantically equivalent, (ii) the sequence of actions performed by a protocol is different from the sequence of actions of the other one because of interleaved actions related to third parties communications (i.e., other systems, the environment). In the former case, (i), it is necessary to properly perform a translation of the two languages. In the latter case, (ii), it is necessary to provide an abstraction of the two sequences that results in sequences containing only relevant actions to the communication. Communication is then possible if the two abstracted sequences are complementary, i.e., are the same sequences of actions while having opposite send/receive "type" for all actions.

With *interoperability*, we mean the property referring to the ability of heterogeneous application-layer interaction protocols that *functionally match to coordinate* where the coordination is expressed as synchronization, i.e., two systems succeed in coordinating if they are able to synchronize.

As said before, we want to approach the protocol interoperability problem in an automated fashion. The solution we propose here, is to automatically synthesize *mediators* (also referred to as *mediating connectors* or CONNECTors in our work) that allow the protocols to interoperate by solving their behavioral mismatches. A mediator is then a protocol that is elicited according to the *automated mediation* paradigm.

2.2 Towards Emergent Mediators

The interoperability problem we specifically want to attack concerns automated and on-the-fly mediation between behaviorally mismatching, yet functionally matching application-layer interaction protocols. Starting from two protocols, the first condition we check is if they share some complementary sequence of actions.

Figure 1 depicts the main elements of our methodology:

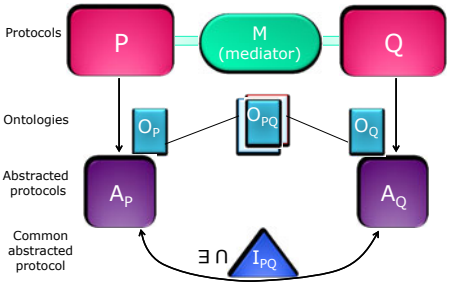


Fig. 1. An overview of our approach

(i) Two application-layer protocols P and Q whose representation is given in terms of *Labeled Transition Systems* (*LTS*s) [11], where the *initial* and *final states* on the LTSs define the *sequences of actions* (traces) that characterize the *coordination policies* of the protocols.
(ii) Two *ontologies* O_P and O_Q describing the meaning of P and Q's actions, respectively.
(iii) Two *ontology mapping functions* defined from O_P and from O_Q to a common ontology. The intersection O_{PQ} on the common ontology identifies the "common language" between P and Q. For simplicity, and without loss of generality, we consider protocols P and Q that have disjoint languages and that are minimal where we recall that every finite LTS has a unique minimal representative LTS.
(iv) Then, starting from P and Q, and based on the ontology mapping, we build two abstractions A_P and A_Q by the relabeling of P and Q respectively, where the actions not belonging to the common language O_{PQ} are hidden by means of silent actions (τs);
(v) After, we check the compatibility of the protocols by looking if there exist complementary traces in the set of traces T_P and T_Q generated by A_P and A_Q respectively. If this is the case, then we are able to synthesize a mediator that makes it possible for the protocols to coordinate.
(vi) Finally, given two protocols P and Q, and an environment E, the mediator M that we synthesize is such that when building the parallel composition $P||Q||E||M$, P and Q are able to coordinate by reaching their final states.

2.3 The Popcorn Scenario

To better illustrate protocol mediation, and to make the theory more concrete, we consider a scenario called Distributed Marketplace or Popcorn scenario that we describe in the following and that is detailed in [1]. Consider an event held within a stadium populated by heterogeneous authorized merchants and consumers. During the event, consumers (respectively merchants) may want to buy (respectively sell) some product by exploiting the applications on their devices. Consider further a consumer application implemented using Lime tuple space

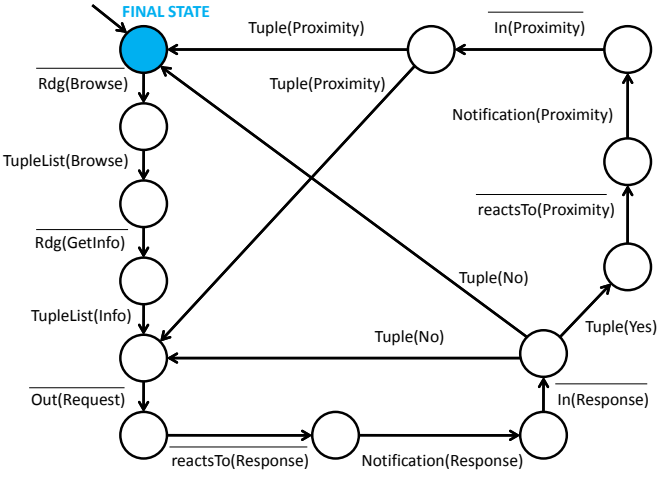


Fig. 2. The LTS of the tuple space consumer

and a merchant application implemented using UPnP. Figures 2 and 3 give their respective behavioral representation in terms of LTS.

Informally, the consumer application (Fig. 2) behaves as follows: first, it browses the tuple space to retrieve the list of all merchants. Once it gets it, it looks for details about the merchants that sell a specific product (popcorn in our case) with a certain price (for example, less than a threshold) within some distance (for example, within a given range). Then, the application writes into the tuple space a request of a given quantity of the product to the chosen merchant and waits for a response. If the request can be satisfied, the consumer receives a positive response and waits for a signal of proximity that the merchant application will send when close enough. Otherwise, the consumer receives a negative response (e.g, because the merchant has no sufficient quantity of product to satisfy the request). In both cases, the consumer can either send a new request or restart from the beginning, i.e., from browsing the tuple space.

The behavior of the merchant application (Fig. 3) can be described as follows: it gets authorization from the event organizers, it receives queries from consumers and sends answers to them advertising its information. Then, the application receives more requests of information from the consumers and answers them providing the required information. Further, it receives requests of ordering of products from consumers and answers a consumer either: positively sending a proximity message when it is physically close to the consumer, or negatively in case it is not able to satisfy the request.

Even though these two applications have some complementary behaviors (protocols are *functionally matching*), they are very different and they are not able to interoperate (because of protocol behavioral *mismatches*). Note that the merchant needs first to be authenticated by a third party. In addition, mediating the

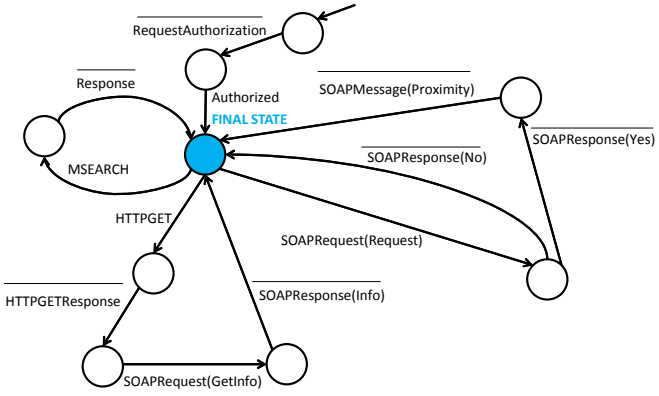


Fig. 3. The LTSs of the UPnP merchant

protocols of the consumer and merchant to achieve interoperability, is far from trivial, especially if one wants to achieve this automatically. This is such an automated support that we aim at in our work, where first results of our approach are presented in [17], which presents a high level description of the theory.

3 A Formalization of Protocols

As discussed in Section 2, the "application-layer interaction protocol" is the behavior of a system in terms of the actions it exchanges with its environment, i.e., other application-layer interaction protocols. We further exploit LTS to characterize such behavior.

3.1 Protocols as LTS

LTSs constitute a widely used model for concurrent computation and are often used as a semantic model for formal behavioral languages such as process algebras. Let Act be the set of observable actions (input/output actions), we get the following definition for LTS:

Definition 1 (*LTS*)
A LTS P is a quadruple (S, L, D, s_0) where:
- S is a finite set of states;
- $L \subseteq Act \bigcup \{\tau\}$ is a finite set of labels (that denote observable actions) called the alphabet of P. τ is the silent action. Labels with an overbar in L denote output actions while the ones without overbar denote input actions. We also use the usual convention that for all $l \in L, l = \overline{\overline{l}}$.
- $D \subseteq S \times L \times S$ is a transition relation;
- $s_0 \in S$ is the initial state.

We then denote with $\{L \bigcup \{\tau\}\}^*$ the set containing all words on the alphabet L. We also make use of the usual following notation to denote transitions:

$$s_i \xrightarrow{l} s_j \Leftrightarrow (s_i, l, s_j) \in D$$

We consider an extended version of LTS, where the set of the LTS' *final states* is explicit. An *extended LTS* is then a quintuple (S, L, D, F, s_0) where the quadruple (S, L, D, s_0) is a LTS and $F \subseteq S$. From now on, we use the terms LTS and extended LTS interchangeably, to denote the latter one. The initial state together with the final states, define the boundaries of the protocol's coordination policies. A *coordination policy* is indeed defined as any trace that starts from the initial state and ends into a final state. We get the following formal definition of traces/coordination policy:

Definition 2 (*Trace*)
Let $P = (S, L, D, F, s_0)$.
A trace $t = l_1, l_2, \ldots, l_n \in L^*$ *is such that:*
$\exists(s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \ \ldots \ s_m \xrightarrow{l_n} s_n)$ *where* $\{s_1, s_2, \ldots, s_m, s_n\} \in S \wedge s_n \in F$.
We also use the usual compact notation $s_0 \overset{t}{\Rightarrow} s_n$ *to denote a trace, where* t *is the concatenation of actions of the trace.*

We adopt the notion of parallel composition *à la* CSP [6]. We recall that the semantics of the parallel composition is that processes P and Q need to synchronize on common actions while can proceed independently when engaged in non common actions.

Definition 3 (*Parallel composition of protocols*)
Let $P = (S_P, L_P, D_P, F_P, s_{0_P})$ *and* $Q = (S_Q, L_Q, D_Q, F_Q, s_{0_Q})$.
The parallel composition between P *and* Q *is defined as:*
the LTS $P||Q = (S_P \times S_Q,\ L_P \cup L_Q,\ D,\ F_P \cup F_Q,\ (s_{0_P}, s_{0_Q}))$ *where the transition relation* D *is defined as follows:*

$$\frac{P \xrightarrow{m} P'}{P||Q \xrightarrow{m} P'||Q} \qquad m \notin L_Q$$

$$\frac{Q \xrightarrow{m} Q'}{P||Q \xrightarrow{m} P||Q'} \qquad m \notin L_P$$

$$\frac{P \xrightarrow{m} P'; Q \xrightarrow{\overline{m}} Q'}{P||Q \xrightarrow{\tau} P'||Q'} \qquad m \in L_P \bigcap L_Q$$

Note that when we build the parallel composition of protocols P and Q with the environment E and the mediator M, the composed protocol is restricted to the languages of P and Q thus forcing them to synchronize.

3.2 Abstract Protocol

Given the definition of enriched LTS associated with two interaction protocols run by networked systems, we want to identify whether such two protocols are functionally matching and, if so, to synthesize the mediator that enables them to interoperate, despite language differences, third parties communications, and behavioral mismatches. With *functional matching* we mean that given two systems with respective interaction protocols P and Q, ontologies O_P and O_Q describing their actions, ontology mapping functions of P and Q, and their common ontology O_{PQ}, there exist at least two complementary traces that allow P and Q to coordinate. That is, sequences of actions of one protocol can synchronize with sequences of actions in the other. This can happen by properly taking into account translation of the languages and communication with third parties. Thus, at a given level of abstraction, we expect to find a common protocol that represents their potential interactions. This leads us to formally analyze such alike protocols to find, if it exists, a suitable mediator that allows the interoperability that otherwise would not be possible.

In order to find the protocols' abstractions, we exploit the information contained in the ontology mapping to suitably relabel the protocols. The relabeling function allows us to substitute (sequences of) actions of the original language into action(s) on the common language thanks to the ontology mapping. After the relabeling operation on the LTSs, we obtain new LTSs labeled only by common actions and τs, that is more abstract than before, e.g., sequences of actions may have been compressed into single actions. In the following, we give the formal definitions concerning the abstract protocol. With respect to the popcorn scenario, Figure 4 summarizes the ontological information of the consumer (first column) and of the merchant (third column). The second column shows the common language mapping where Figures 2 and 3 illustrate the two protocols. Thanks to the ontology mapping, labels of consumer and merchant's protocols (expressed on two different ontologies) are *mapped* onto labels of the common ontology (more abstract). We specialize the usual ontology mapping definition [9,10] by considering pairs of elements made by more than one label. We use the specialized ontology mapping on protocols' ontology where the vocabularies are represented by the languages of the protocols. That is, we consider $P = (S_P, L_P, D_P, F_P, s_{0_P})$, $O_P = (L_P^*, A_P)$ the ontology of P, and $O = (L^*, A)$ another ontology. $maps : L_P^* \rightarrow L^*$ is the ontology mapping function that maps P's ontology into the ontology O.

By applying this ontology mapping, we relabel protocols with words of their common language (ontology) and τs for the thirds parties languages. To identify which is the common language, we first map each protocol's ontology into another one, resulting from ontology mediation, and then by intersection we find their common language.

Figure 5 depicts the abstraction of protocols. We consider two protocols P and Q with respective ontologies O_P and O_Q and ontology mapping functions $maps_P$ and $maps_Q$. We first use the mapping functions to map O_P and O_Q into a target ontology where C_{OP} and C_{OQ} represent the codomain sets of $maps_P$ and $maps_Q$ respectively. The subsets of D_P and D_Q of O_P and O_Q, respectively, represent

Tuple Space Consumer	Common Language Mapping		UPnP+SOAP Merchant
Rdg(Browse). TupleList(Browse)	$\overline{\alpha_1}$ α_2	α_1 $\overline{\alpha_2}$	MSEARCH. Response
Rdg(GetInfo). TupleList(Info)	$\overline{\beta_1}$ β_2	 β_1 $\overline{\beta_2}$	HTTPGET. HTTPGETResponse. SOAPRequest(GetInfo). SOAPResponse(Info)
Out(Request). reactsTo(Response)	 $\overline{\gamma_1}$	γ_1	SOAPRequest(Request)
Notification(Response). In(Response). Tuple(No)	 δ_1	$\overline{\delta_1}$	SOAPResponse(No)
Notification(Response). In(Response). Tuple(Yes). reactsTo(Proximity). Notification(Proximity). In(Proximity). Tuple(Proximity)	 σ_1 σ_2 	$\overline{\sigma_1}$ $\overline{\sigma_2}$	SOAPResponse(Yes). SOAPMessage(Proximity)

Fig. 4. Ontology mapping between tuple space consumer and UPnP Merchant

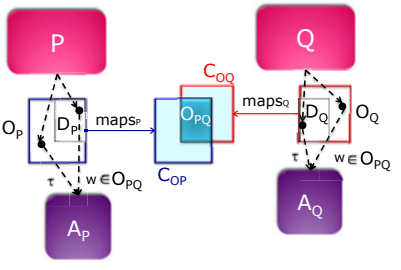
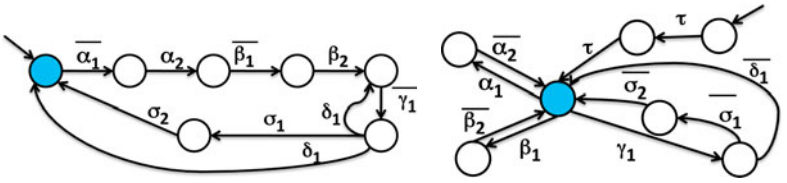


Fig. 5. The abstract protocol building

the portion of the domains of $maps_P$ and $maps_Q$ respectively corresponding to C_{OP} and C_{OQ}. Note that we consider protocols such that for each element of the codomain corresponds only one element of the domain. The common language between P and Q is defined as the intersection O_{PQ} of C_{OP} and C_{OQ}. The relabeled (abstracted) protocols, A_P and A_Q of P and Q respectively, are built as follows: (i) the chunks (states and transitions) of P and Q labeled by words of D_P and D_Q, respectively are substituted by building a single transition labeled with words of O_{PQ} ; (ii) all the other transitions labeled with actions belonging to the thirds parties language, are relabeled with τs . Having P, Q, O_P, O_Q, $maps_P$, and $maps_Q$, the relabeling function is applied after the computation of C_{OP}, C_{OQ}, D_P, D_Q, and O_{PQ}. The relabeling function on P takes as input $P, D_P, O_{PQ}, \{O_P \setminus D_P\}$ and gives as result an abstracted LTS A_P (it applies similarly for Q). More formally, having P, Q, O_P, O_Q, $maps_P$, $maps_Q$, C_{OP}, C_{OQ}, D_P, D_Q, and O_{PQ} the relabeling function is defined as follows:



(a) Consumer (b) Merchant

Fig. 6. Abstracted LTSs of consumer and merchant protocols

Definition 4 (*Relabeling function*)
Through the function relabels : $(P, D_P, O_{PQ}) \rightarrow A_P$, A_P *is built as follow: each chunk of* P *labeled with a trace* t *belonging to* D_P *is substituted by one transition with label* $w \in O_{PQ}$ *and each transition labeled with a word belonging to* $\{O_P \setminus D_P\}$ *is maintained and relabeled with a* τ.

In the popcorn scenario, the only label that is not abstracted in the common language is the authorization that represents a third party coordination. The consumer and merchant's abstracted LTSs are shown in Figure 6. The subsequent step is to check whether the two abstracted protocols share a *complementary coordination policy*, i.e., whether the abstracted protocols may indeed synchronize, which we check over protocol traces as discussed next.

3.3 Towards Automated Matching and Mediator Synthesis

The formalization described so far is needed to: (1) characterize the protocols and (2) abstract them into protocols on the same alphabet. As illustrated previously, to establish whether two protocols P and Q can interoperate we have to check the existence of portions of their respective abstracted protocols (A_P and A_Q) that can interoperate. That is, A_P and A_Q have to share complementary policies. To establish this, we use the *functional matching relation* between A_P and A_Q. This relation succeeds if A_P and A_Q have complementary traces. More formally:

Definition 5 (*Functional matching*)
Let $P = (S_P, L_P, D_P, F_P, s_{0_P})$ *and* $Q = (S_Q, L_Q, D_Q, F_Q, s_{0_Q})$.
Let A_P *and* A_Q *be the abstracted protocols of* P *and* Q, *respectively.*
Let T_{AP} *and* T_{AQ} *be the set of all the traces of* A_P *and* A_Q, *respectively.*
Let $\mathcal{C}$ *be a coordination policy denoted by final state* $f_{\mathcal{C}}$
P *and* Q *have a functional matching under ontology mapping* $maps_P$ *and* $maps_Q$ *with respect to policy* $\mathcal{C}$ *iff:* $f_{\mathcal{C}} \in maps_P(F_P)$, $f_{\mathcal{C}} \in maps_Q(F_Q)$, *Let* $T_{\mathcal{C}_{AP}} = \{s_{0_P} \stackrel{t}{\Rightarrow} f_{\mathcal{C}} \in T_{AP}\}$ *and* $T_{\mathcal{C}_{AQ}} = \{s_{0_Q} \stackrel{t}{\Rightarrow} f_{\mathcal{C}} \in T_{AQ}\}$, *then* $T_{\mathcal{C}_{AP}} =_{s/r} T_{\mathcal{C}_{AQ}}$ *where the equality of sets of traces over send-receive (noted* s/r*) denotes equality modulo complementary send-receive actions.*

The *functional matching relation* defines necessary conditions that must hold in order for a set of networked systems to interoperate through a mediator. In

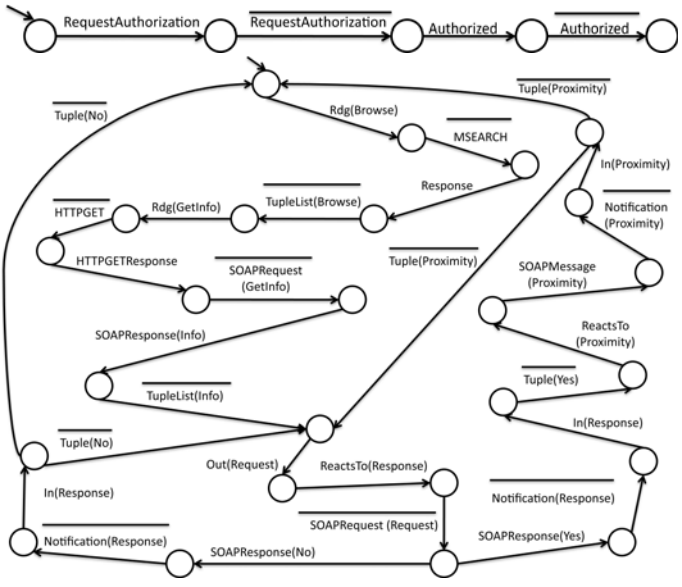


Fig. 7. Mediating connector for the popcorn scenario (M_T and M_C)

our case, till now, the set is made by two networked systems and the matching condition is that they have complementary traces regarding a given coordination policy. Note that these traces are computed on the abstracted protocols and might contain τ actions that represent third parties synchronization.

Then, given given two protocols P and Q that functionally match, we want to synthesize a mediator M such that the parallel composition $P||M||Q$, allows P and Q to evolve to their final states. An action of P and Q can belong either to the *common language* or the *third parties language*, i.e., the environment. We build the mediator in such a way such that it lets P and Q evolve independently for the portion of the behavior to be exchanged with the environment (denoted by τ action in the asbracted protocols) until they reach a "synchronization state" from which they can synchronize on complementary actions. Note that the synchronization cannot be direct since the mediator needs to perform a suitable translation according to the ontology mapping, e.g. $\alpha_1 = \overline{Rdg(Browse)}$ in one protocol and $\alpha_1 = MSEARCH$ in the other.

The mediator is made of two separate components: M_C and M_T. M_C speaks only the common language and M_T speaks only the third parties language. M_C is a LTS built starting from the common language between P and Q whose aim is to solve the protocol-level mismatches occurring among their dual interactions (complementary sequences of actions) by translating and coordinating between them. M_T, if it exists, is built starting from the third parties language of P and Q and represents the environment. The aim of M_T is to let the protocols evolve, from the initial state or from a state where a previous synchronization is ended, to the states where they can synchronize again. Formalization of mediator synthesis

given the specification of functional matching is part of our current work, while we summarize below the principles of our approach using the popcorn scenario.

In our illustration, we assume to have with the behavioral specification of consumer and merchant applications as LTSs (Figures 2 and 3), their coordination policies (thanks to the initial and final states on LTSs), their respective ontologies describing their actions, and the ontology mapping that defines the common language between consumer and merchant, i.e., represents their possible interactions (Figure 4). The first step is to *abstract* the protocols exploiting the ontology mapping. Following the theory, the abstracted protocols for the popcorn scenario are illustrated in Figure 6. The second step is to check whether they share some coordination policies. In this scenario we recall that the merchant is able to simulate the consumer. Then the coordination policies that they share are exactly the consumer's ones. Then, with the application of the theory to the scenario, we obtain the connector of Figure 7. In this case only the merchant have third parties language actions and then the mediator is made by the part that translates and coordinates regarding the common language and the part that simulates the environment.

4 Related Work

A number of solutions to automated protocol mediation have recently emerged, leveraging the rich capabilities of Web services and Semantic Web technologies [21,20,15,24]. They differ with respect to: (a) a priori exposure of the process models associated with the protocols that are executed by networked resources, (b) knowledge assumed about the protocols run by the interacting parties, (c) matching relationship that is enforced. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

What is needed is a new and formal foundation for mediating connectors from which protocol matching and associated mediation may be rigorously defined and assessed. These relationships may be automatically reasoned upon, thus paving the way for on the fly synthesis of mediating connectors. To the best of our knowledge, such an effort has not been addressed in the Web services and Semantic Web area although proposed algorithms for automated mediation manipulates formally grounded process models.

However a work very close to our is [25] that proposes a theory to characterize and solve the interoperability problem of augmented interfaces of applications. The authors formally defines the checks of applications compatibility and the concept of adapters. The latter can be used to bridge the differences discovered while checking the applications that have functional matching but are protocol incompatible. Furthermore they provide a theory for the automated generation of adapters based on interface mapping constraints. The main disadvantages of this work are that the approach is semi-automatic because of the interface mapping. Additionally, applications are assumed to agree on the ordering of messages, thus not solving ordering mismatches.

A recent work [3] addresses the interoperability problem between services and provide experimentation on real Web2.0 social applications. The paper deals with the integration of a new service implementation, to substitute a previous one with the same functionalities. The new implementation does not still guarantee behavioral compatibility despite complying with the same API of the previous one. They hence propose a technique to dynamically detect and fix interoperability problems based on a catalogue of inconsistencies and their respective adapters. This is similar to our proposal to use ontology mapping to discover mismatches and mediator to solve them. Our work differs with respect to theirs because we aim at automatically synthesizing the mediator. Instead, their approach is not fully automatic since although they discover and select mismatches dynamically, the identification of mismatches and of the opportune adapters is made by the engineer.

References [13,2] are related to our work since they identify and classify basic types of mismatches that can possibly occur when compatible but mismatching processes try to interoperate. Moreover, they provide support to the developers by assisting them while identifying protocol mismatches and composing mediators. In [13], the authors also take into consideration more complex mediators obtained by composition of basic ones. The main difference between these two works and ours is the semi-automation issue. Indeed, they require the developer intervention for detecting the mismatches, configuring the mediators, composing basic mediators while, thanks to formal methods, we are able to automatically derive the mediator under some conditions.

5 Conclusion

In this paper, we have formally investigated the interoperability of protocols that are observable at the interface level. Key issue is to solve behavioral mismatches among the protocols although they are functionally matching. We have specifically introduced a theory towards interoperability as a means to: (1) clearly define the problem, (2) show the feasibility of the automated reasoning about protocols, i.e., to check their functional matching and to detect their behavioral mismatches, (3) show the feasibility of the automated synthesis of abstract mediators under certain conditions to dynamically overcome behavioral mismatches of functionally matching protocols. Our theoretical framework is a first step towards the automatic synthesis of actual mediators and we believe that it is very important to devote investigation to this goal. Significant part of our current work is on leveraging practically the proposed theory in particular dealing with automated reasoning about protocol matching and further automated protocol mediation. Our current work is further concerned with the integration with complementary work ongoing within the CONNECT project so as to develop an overall framework enabling the dynamic synthesis of emergent connectors among networked systems. Relevant effort includes the study of: learning techniques to dynamically discover the protocols that are run in the environment, dependability assurance, data-level mediation, as well as algorithms and run-time techniques

towards efficient synthesis. Such effort is presented in the CONNECT companion papers of the Isola'2010 conference, and detail about overall CONNECT results may be found from the project Web site at http://connect-forever.eu/.

References

1. The popcorn scenario dry-run experiment's details, `http://www.connect-forever.eu/connect-dry-run/`
2. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
3. Denaro, G., Pezzè, M., Tosi, D.: Ensuring interoperable service-oriented systems through engineered self-healing. In: Proceedings of ESEC/FSE 2009. ACM Press, New York (2009)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Resusable Object-Oriented Software. Addison-Wesley, Reading (1995)
5. Grace, P., et al.: Towards an architecture for runtime interoperability. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 206–220. Springer, Heidelberg (2010)
6. Hoare, C.A.R.: Communicating sequential processes. ACM Commun. 26(1), 100–106 (1983)
7. Howar, F., Jonsson, B., Merten, M., Steffen, B., Cassel, S.: On handling data in automata learning: Considerations from the connect perspective. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 221–235. Springer, Heidelberg (2010)
8. Issarny, V., Steffen, B., Jonsson, B., Blair, G., Grace, P., Kwiatkowska, M., Calinescu, R., Inverardi, P., Tivoli, M., Bertolino, A., Sabetta, A.: CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In: 14th IEEE International Conference on Engineering of Complex Computer Systems, Postdam Germany (2009)
9. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. Knowl. Eng. Rev. 18(1), 1–31 (2003)
10. Kalfoglou, Y., Schorlemme, M.: Ontology mapping: The state of the art. In: Kalfoglou, Y., Schorlemmer, M., Sheth, A., Staab, S., Uschold, M. (eds.) Semantic Interoperability and Integration, Dagstuhl, Germany. Dagstuhl Seminar Proceedings, vol. 04391, IBFI, Schloss Dagstuhl (2005)
11. Keller, R.M.: Formal verification of parallel programs. ACM Commun. 19(7), 371–384 (1976)
12. Kumar, R., Nelvagal, S., Marcus, S.I.: A discrete event systems approach for protocol conversion. Discrete Event Dynamic Systems 7(3) (1997)
13. Li, X., Fan, Y., Wang, J., Wang, L., Jiang, F.: A pattern-based approach to development of service mediators for protocol mediation. In: Proceedings of WICSA 2008, pp. 137–146. IEEE Computer Society, Los Alamitos (2008)
14. Liskov, B.H., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. 16(6) (1994)
15. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007: Proceedings of the 16th international conference on World Wide Web, pp. 993–1002. ACM, New York (2007)

16. Noy, N.F.: Semantic integration: a survey of ontology-based approaches. SIGMOD Rec. 33(4) (2004)
17. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2009 (2009)
18. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering (2003)
19. Stollberg, M., Cimpian, E., Mocan, A., Fensel, D.: A semantic web mediation architecture. In: Proceedings of the 1st Canadian Semantic Web Working Symposium (CSWWS 2006). Springer, Heidelberg (2006)
20. Vaculín, R., Neruda, R., Sycara, K.P.: An Agent for Asymmetric Process Mediation in Open Environments. In: Kowalczyk, R., Huhns, M.N., Klusch, M., Maamar, Z., Vo, Q.B. (eds.) SOCASE 2008. LNCS, vol. 5006, pp. 104–117. Springer, Heidelberg (2008)
21. Vaculín, R., Sycara, K.: Towards automatic mediation of OWL-S process models. In: IEEE International Conference on Web Services, pp. 1032–1039 (2007)
22. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25, 38–49 (1992)
23. Wiederhold, G., Genesereth, M.: The conceptual basis for mediation services. IEEE Expert: Intelligent Systems and Their Applications 12(5), 38–47 (1997)
24. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol mediation for adaptation in semantic web services. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 635–649. Springer, Heidelberg (2006)
25. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. 19(2), 292–333 (1997)