A Formalization of Mediating Connectors: Towards on the fly Interoperability

Romina Spalazzese, Paola Inverardi, Valérie Issarny

dipartimentoinformatica

Università degli Studi dell'Aquila

A Formalization of Mediating Connectors: Towards on the fly Interoperability

Romina Spalazzese, Paola Inverardi, Valerie Issarny

Technical Report TRCS 004/2009

Dipartimento di Informatica
Università degli Studi dell'Aquila
Via Vetoio Loc. Coppito
I-67010 L'Aquila, Italy

http://www.di.univaq.it

Technical Report TRCS Series

A Formalization of Mediating Connectors: Towards on the fly Interoperability

Romina Spalazzese
Università degli Studi dell'Aquila
via Vetoio I-67100 L'Aquila, Italy
romina.spalazzese@di.univaq.it

Paola Inverardi
Università degli Studi dell'Aquila
via Vetoio I-67100 L'Aquila, Italy
paola.inverardi@di.univaq.it

Valérie Issarny
INRIA-Rocquencourt
Domaine de Voluceau
78153 Le Chesnay, France
valerie.issarny@inria.fr

Abstract

Mediators stand as a core architectural paradigm for today's and future systems that increasingly need be connected. The mediator concept has been used to cope with many heterogeneity dimensions spanning: terminology, representation format, transfer protocols, functionality, and application-layer protocols. Still, a key challenge for today's systems architectures is to embed the necessary support for automated mediation, i.e., the connector concept needs to evolve towards the one of mediating connectors. In this paper, we concentrate on the issue of enabling automated protocol mediation. Building upon tremendous research work in the area over the past few years we introduce a formalization of mediating connectors. The proposed formalization paves the way for automated reasoning about protocol matching and mapping, and thus for the dynamic synthesis of mediating connectors to enable eternal networked systems, which we investigate as part of the CONNECT *European project.*

1. Introduction

Our everyday activities are increasingly dependent upon the assistance of digital systems that pervade our living environment. Key technologies such as the Internet, the Web, and wireless computing devices and networks are now calm technologies in the sense of Marc Weiser's definition [1]. They can indeed be qualified as ubiquitous, even if converged computing and networking technologies have still not reached the maturity envisioned by the ubiquitous computing and subsequent pervasive computing and ambient intelligence paradigms.

However, the current ubiquity of digital systems is technology-dependent. The efficacy of integrating and composing networked systems is proportional to the level of interoperability of the systems' respective behaviors, from application- down to network-layers. This leads to a landscape of islands of networked systems, although interoperability bridges may possibly be deployed among them. Further, the fast pace at which technology evolves at all abstraction layers increasingly challenges the lifetime of networked systems in the digital environment.

Middleware positions itself as the architectural paradigm enabling to effectively network together heterogeneous systems, specifically providing upper layer interoperability. Middleware bridges 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. As a matter of fact, middleware is yet another technological block, which also creates islands of networked systems. A number of systems have then been introduced to provide middleware protocols interoperability [2]. These include middleware bridges implementing protocol translation [3], possibly based on intermediary reference protocol for greater flexibility as with Enterprise Service Buses (ESBs) [4]. Other approaches lie in protocol substitution at runtime, exploiting the reflection paradigm [5]. In general, various proven solutions exist for middleware protocol interoperability, each with respective drawbacks and advantages [6]. 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 [7].

Mediator then stands as a core architectural paradigm for today's and future systems that increasingly need be connected. The mediator concept was early introduced to cope with the integration of heterogeneous data sources [8], [9]. 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 [10]: 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 [11]. Protocol mediation is further concerned with behavioral mismatches that may occur during interactions.

A key challenge for today's systems architectures is to embed the necessary support for automated mediation, i.e., the connector concept needs to evolve towards the one of

mediating connectors. Indeed, the actual systems with which communications will take place cannot be anticipated at design time due to today's open networking and further continuous evolution of networked systems. As such, connectors not solely coordinate the interaction behaviors of connected systems but also mediate those behaviors to enable actual interactions. Automated mediation has deserved a great deal of attention in all the aforementioned heterogeneity dimensions. This especially holds in the context of Web services technologies that is certainly one of today's most popular and enabling architectures for networked resources. Considering today's state of the art, ontologies appear as the core concept to deal with data heterogeneity, logic-based formalisms stand as the natural paradigm for overcoming functional heterogeneity, and process algebras are obvious candidates for reasoning about protocol mediation. Still, enabling reasoning and further solving of semantic mismatches at runtime, while not over-constraining the ability to communicate, remain open research question. In this paper, we more specifically concentrate on the issue of enabling automated protocol mediation. Building upon tremendous research work in the area over the past few years (Section 2), we introduce a formalization of mediating connectors. The proposed formalization paves the way for automated reasoning about protocol matching and mapping (Section 3), and thus for the synthesis of mediating connectors (Section 4). It further provides a comprehensive framework to assess the various algorithmic and architectural solutions to protocol and process mediation that have been recently proposed. Overall, the proposed formalization of mediating connectors constitutes a first step towards the dynamic synthesis of mediating connectors to enable eternal networked systems, which we investigate as part of the CONNECT[1] European project (Section 5).

2. Protocol mediation

The mediation paradigm encompasses a number of architectural paradigms like adapter, bridge or wrapper. The following section provides a brief definition of the concept as used in the paper. We then concentrate on protocol mediation that is our focus, surveying protocol mediation patterns that have been elicited in the literature together with proposed approaches to automatic mediation.

2.1. Mediation concept

Mediators are an essential means to cope with the inherent heterogeneity of today's open networking environments. Such a concern is in particular central to Web-based computing where many heterogeneity dimensions arise and need be mediated [12]:

- Mediation of data structures allows for data to be exchanged according to semantic matching, as opposed to requiring syntactic matching and further usage of identical data formats.
- Mediation of functionalities enables one to locate networked resources that provide a required functionality (in isolation and/or in combination) based on semantic matching and possible adaptation.
- Mediation of business logics enables networked resources that provide complementary functionalities to be connected together although they may execute interaction protocols whose respective behaviors do not match.
- Mediation of message exchange protocols supports the actual interaction among networked resources although they may use different middleware protocols for communication. Middleware heterogeneity ranges from heterogeneity of implementations to that of distributed computing models and related coordination models and extra-functional properties.

Facing this heterogeneity, mediation architectures embed a number of enablers [10]:

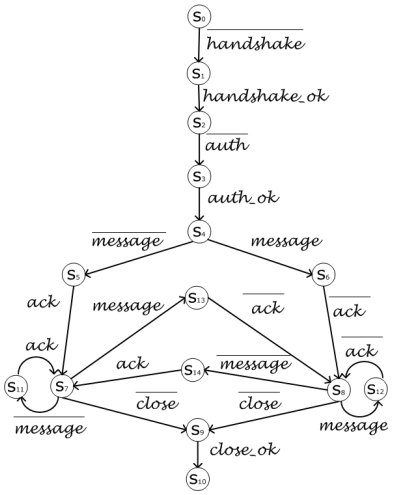
- Data level mediation primarily relies on techniques for ontology integration [13], dealing with the mapping, alignment, and merging of ontologies.
- Functional mediation may conveniently be based on logical relationships between functional descriptions of networked resources that are expressed in terms of pre- and post-conditions over the resources' states [14].
- Protocol mediation (*aka* process mediation) is concerned with the mediation of protocols from the application (possibly) down to the middleware layers. It strives techniques to solve behavioral mismatches among protocols run by interacting parties. As discussed in the next sections, proposed solutions introduce algorithms that establish a valid process for interaction given the respective processes run by the interacting parties. The challenge is then to promote flexibility by dynamically solving behavioral mismatches as far as the connected resources functionally match.

Other approaches that share the same formal settings have been proposed quite some time ago to solve mismatches in the field of supervisory control theory of discrete event systems [15] and, more recently, in the field of software architectures to address communication problems by proposing ad hoc wrappers [16].

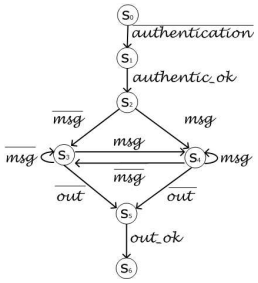
2.2. Protocol mediation patterns

To illustrate protocol mediation, let us consider the simple yet challenging example of instant messaging. Indeed, various instant messaging systems are now in use, facilitating communications among people. However, although

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



(a) Windows Messenger protocol



(b) Jabber protocol

Figure 1. Behavioral modeling of two instant messaging protocols

those systems implement matching functionalities, end-users need to use the very same system to communicate due to behavioral mismatches of respective protocols. For instance, let us envisage Windows Messenger (now called Windows Live Messenger) [17] and Jabber Messenger [18]. Figure 1 models the respective behaviors of associated protocols using Labelled Transition Systems (LTS) [19]. We use the usual convention that overlined actions denote output actions while non-overlined ones denote input actions. It is obvious that these systems should be able to interoperate since they both amount to supporting authentication of peers and then message exchanges among peers. It is also obvious that mediating their respective protocols to achieve the interoperability is far from trivial, especially if one wants to achieve this automatically.

A base approach to protocol mediation is to categorize the various types of protocol mismatches that may occur and that must be solved, according to the structure of the associated processes, and then to introduce corresponding mediation patterns. Five basic patterns have been introduced in the literature in the context of Web services [20], [21]. These concern: (i) stopping an unexpected message, (ii) inversing the order of messages, (iii) splitting a message, (iv) combining messages and (v) sending a dummy acknowledgment. Complementary mediation patterns are introduced in [22], focusing on middleware layer interoperability and more specifically that supported by the Synapse Enterprise Service Bus[2]. Patterns include: message transformation, protocol switching, message enrichment, message routing, message cloning and caching. Given mediation patterns, custom mediators may be designed through the assembly of relevant patterns according to the behavioral mismatches identified among the protocols to be made interoperable. Such an issue is in particular addressed in [23], which provides tools to developers to assist them to identify protocol mismatches and to compose mediators. However, this remains quite limited with respect to enabling interoperability in today's networking environments that are highly dynamic. Indeed, mediators need to be synthesized on the fly so as to allow interactions with networked systems that are not known in advance. Such a concern is in particular recognized by the Web service research community that has been studying solutions to the automatic mediation of business processes in the recent years.

2.3. Towards automatic mediation

Automated mediation of heterogeneous protocols basically relies on:

- The adequate modeling of processes abstracting the behavior of the protocols to be bridged where finite state machines is the modeling formalism of choice in most work.
- The definition of a matching relationship between the process models that sets the conditions under which protocol interoperability is supported, provided data and functional mediations.
- The elicitation of an algorithm that computes an appropriate mapping between matching process models.

A number of solutions to automated protocol mediation have recently emerged, leveraging the rich capabilities of Web services and Semantic Web technologies [24], [25], [26], [27]. They differ with respect to:

- A priori exposure of the process models associated with the protocols that are executed by networked resources, thus possibly requiring to learn the model on the fly.
- Knowledge assumed about the protocols run by the interacting parties, thus possibly enabling to synthesize off-line part of the mediator.
- Matching relationship that is enforced, possibly weakening flexibility to alleviate the complexity of mediation.

However, most solutions are discussed informally, making it difficult to assess their respective advantages and

2. http://synapse.apache.org/

drawbacks. They further remain rather vague on the definition of enforced matching relationship.

What is needed is a new and formal foundation for mediating connectors from which:

- Protocol matching and associated mapping relationships may be rigorously defined and assessed.
- The above 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 past although proposed algorithms for automated mediation manipulates formally grounded process models.

3. Protocols and mediation formalization

The need for a formalization of mediating connectors between mismatching protocols arise for different motivations among which: (i) to be able to better understand and convey the mediation problem, (ii) to dynamically discover an abstract mediator behavior (or its non existence), and (iii) to synthesize a concrete mediator behavior refining the abstract one and enabling actual protocol interoperability.

This section provides a formal definition of the needed concepts to model the interacting actors: the protocols that need to be coordinated and the mediating connector that enables actual communication. We then proceed with the rigorous description of protocol matching relationships, describing in particular the necessary conditions that must hold for protocols to be mediated. Then, the associated mappings are derived.

3.1. Protocols as LTS

We use Labelled Transition Systems (LTS)[19] to formally describe protocols implemented by networked system for interaction, and mediating connectors.

Let Act be the universal set of observable actions, we get the following definition for LTS.

*Definition 1 (**LTS**):* an LTS P is a quadruple (S, L, D, s_0) where:
- S is a finite set of states;
- $L \subseteq Act$ is a finite set of labels (that denote actions) called the *alphabet* of P;
- $D \subseteq S \times L \times S$ is a transition relation;
- $s_0 \in S$ is the initial state.

We then denote with L^* the set containing all words on the alphabet L.

We further consider an extended version of LTS, which includes the set of the LTS' final states, i.e., an **extended LTS** $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ is an LTS (S_E, L_E, D_E, s_{0E}) plus the set of its final states $F_E \subseteq S_E$.

We also make use of the usual following notation to denote transition relation:

$$s_i \xrightarrow{\alpha} s_j \Leftrightarrow (s_i, \alpha, s_j) \in D_E \quad (*)$$

The next concept that we need to describe is that of trace. Informally a trace is a sequence of actions of a given LTS.

*Definition 2 (**Trace**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$, and $s_k \in S_E$. A trace is $t = \alpha, \beta, \ldots, \sigma \in L_E^*$ such that $\exists\ CHAIN = (s_k \xrightarrow{\alpha} s_a \xrightarrow{\beta} s_b \ \ldots \ s_m \xrightarrow{\sigma} s_n)$ where $(s_k, s_a, s_b, \ldots, s_m, s_n) \in S_E \wedge \forall (s_i \xrightarrow{l} s_j) \in CHAIN$ (*) holds.

We also use the usual compact notation $s_k \overset{t}{\Rightarrow} s_n$ to denote a trace, where s_k, s_n, and t are initial state, final state, and sequence of actions of the trace, respectively.

Given the definition of enriched LTS associated with networked systems, what we are interested to formalize with respect to protocol mesitation is to identify whether two protocols share similar intent and if so to synthesize the mediating connector that enables them to interoperate despite mismatches. In more detail with "*systems with the same intent*" we mean that given two systems $S1$ and $S2$, with respective interaction protocols P_1 and P_2, *part of the behavior* of $P1$ and $P2$ is *complementary* thus showing an interaction potentiality. Thus, we expect to find, at a given level of abstraction, similarities in the structure of their protocol representation (P_1 and P_2). 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.

Definitions that follow then allow reasoning about the appropriate structures of protocols. The first definition concerns states of the extended LTS from which at least two transitions start.

*Definition 3 (**Branch state**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ and $s_k \in S_E$. We say that s_k is a *branch state*, also written *branch(s_k)*, if it exists at least two transitions in D_E having s_k as starting state i.e., if $\exists B = \{d_i : d_i \in D_E \text{ and } d_i = (s_k, l, s_i)\}$ and $|B| \geq 2$.

The second definition refers to states that identify the entry point of some cycles. That is: (i) it exists a trace that starts from and ends into a state s_k and (ii) it exists a transition (s_i, l, s_k), where l is not included in any cycling trace, then s_k is an entry cycle state.

*Definition 4 (**Entry cycle state**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ and $s_k \in S_E$. We define s_k as *entry cycle state*, also written *entry_cycle(s_k)*, if there exists $(s_i, l, s_k) \in D_E$ and for any trace $t : s_k \stackrel{t}{\Rightarrow} s_k$ it holds that $l \notin t$.
Note that the length of a trace can also be 1 thus having a single transition in D_E having s_k as both source and target state, that is $d_i = (s_k, l, s_k) \in D_E$.

The third definition identifies states of the extended LTS in which two or more transitions converge.

*Definition 5 (**Convergence state**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ and $s_k \in S_E$. Then s_k is a *convergence state*, also written *convergence(s_k)*, if $\exists\ CNV = \{d_i : d_i \in D_E \text{ and } d_i = (s_i, l, s_k)\}$ and $|CNV| \geq 2$.

The fourth definition generically defines as rich state any of the above defined states or an initial or final state.

*Definition 6 (**Rich state**):* Let be $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ and $s_k \in S_E$. s_k is a *rich state*, also written *rich(s_k)*, if it is either *branch(s_k)* or *entry_cycle(s_k)*, or *convergence(s_k)*, or $s_k = s_{0E}$, or $s_k \in F_E$.

Referred to the previous definition is the notion of successive rich state. Given a rich state, the definition identifies the next immediately reachable rich state such that there is not any other rich state between them.

*Definition 7 (**Successive rich state**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$ and *rich(s_r)*$\in S_E$. *Successive rich state* of s_r, also written *succ_rich(s_i, s_r)*, is any $s_i \in S_E$ such that for any trace t such that $s_r \stackrel{t}{\Rightarrow} s_i$ and *rich(s_i)* it does not exist any other *rich(s_j)* between s_r and s_i.

The *structure* of an extended LTS P_E then follows from the previous definitions that introduce the building blocks for the structure. Given rich states, we define the structure of an LTS P_E as the LTS derived from P_E whose set of states is the set of P_E's rich states and whose transitions are the traces of P_E among those states.

*Definition 8 (**Structure**):* Let $P_E = (S_E, L_E, D_E, F_E, s_{0E})$. We define P'_E as the *structure* of P_E, also written *structure(P'_E, P_E)*, the extended LTS $P'_E = (S'_E, L'_E, D'_E, F'_E, s'_{0E})$ where $F'_E = F_E$ and $s'_{0E} = s_{0E}$, $S'_E = \{s_i \in S_E : rich(s_i)\}$, $L'_E = \{t : s_i \stackrel{t}{\Rightarrow} s_j$ and $succ\_rich(s_j, s_i)\}$, and $D'_E = \{d' = (s', t, s'')\}$ where $s', s'' \in S'_E$ and $t \in L'_E$.

We say that two structures $Q' = (S'_Q, L'_Q, D'_Q, F'_Q, s'_{0Q})$ and $R' = (S'_R, L'_R, D'_R, F'_R, s'_{0R})$ are ***equal***, also written *equal(structure(Q', Q), structure(R', R))*, if $S'_Q = S'_R$, $F'_Q = F'_R$, $s'_{0Q} = s'_{0R}$. Note that this definition does not take actions into account.

*Definition 9 (**Corresponding transition**):* Let $P_E = (S_P, L_P, D_P, F_P, s_{0P})$, $Q_E = (S_Q, L_Q, D_Q, F_Q, s_{0Q})$, such that P_E is ***equal*** Q_E. Let $d_i = (s_i, t_i, s'_i) \in D_P$, $d_j = (s_j, t_j, s'_j) \in D_Q$ and O be an ontology. d_j is *corresponding transition* of d_i, also written *corresponding(d_i, d_j)*, if $s_i = s_j$, $s'_i = s'_j$, and t_j has the same meaning of t_i through O only considering the action labels without taking into account if actions are complementary.

*Definition 10 (**Protocol similarity**):* Let $P = (S_P, L_P, D_P, F_P, s_{0P})$, $Q = (S_Q, L_Q, D_Q, F_Q, s_{0Q})$. P is **similar** to Q, also written *similar(P, Q)*, if and only if equal(structure(P', P), structure(Q', Q)) and for any $d_i \in D'_Q \exists d_j \in D'_R$ such that corresponding(d_i, d_j) and viceversa.

The above definitions permit to formally describe, in the following section, the conditions that must hold in order to find (if they exist) a matching and then a mapping relationships between protocols.

3.2. Protocol matching and mapping

The problem of protocols mediation concerns the interoperability between two protocols, P_1 and P_2, having part of their behavior that is complementary behavior. Being (at least partly) complementary, we expect that the protocols will be (partly) similar in structure.

To single out the parts of the structures of the protocols that are expected to be similar, i.e., the parts of actual communication between them, we have to determine the common language between the two protocols modulo a given ontology.

Let P_1 and P_2 be two LTS and O an ontology. The ***common language*** between P_1 and P_2 is made by the set of traces that has the same meaning through O only considering the action labels without taking into account their complementarity (i.e., the label α is considered equal to $\overline{\alpha}$). Note that the ontology can map traces into action/traces and vice versa. The portions of the protocols that have not a common language are the parts of the protocols that synchronize with third entities.

We call ***common language structure***, the part of the structure of P_1 and P_2, labelled with traces belonging to their common language.

Thus we expect that the common language structure of P_1 and the one of P_2 are similar. We say similar instead of equal because of potential syntactical and semantic mismatches. To solve such mismatches a third protocol is needed to mediate the differences.

The problem of protocols mediation is thus transferred into finding, if it exists, a suitable mediating connector, that coordinates P_1 and P_2, thus making the interoperability possible.

Protocol matching. Let P_1 and P_2 be two LTS with alphabets L_1 and L_2 respectively and let the intersection between L_1 and L_2 be an empty set.

We assume that actions in the LTS are only the observable (external) actions of the protocol (i.e we do not consider its internal actions).

We adopt the usual convention that an action represented by the label α synchronizes with the action $\overline{\alpha}$.

Moreover we use the usual *progress property* that the parallel composition of two LTS evolves only by means of synchronization.

Let CLS_{P_1} (resp. CLS_{P_2}) be the common language structures of P_1 and of P_2 respectively. We describe the *mediated matching* $\mathcal{R}$ by the following formula:

$$P_1 \mathcal{R} P_2 \Leftrightarrow (similar(CLS_{P_1}, CLS_{P_2}) \ and \ (\exists M : M \ refines \ CLS_{P_1}, CLS_{P_2})$$

where M is an LTS and the refinement is defined as a splitting (if any is possible) of corresponding transitions of CLS_{P_1} and CLS_{P_2} into more than one in M based on the ontology mappings of the actions on the transition. The definition of refinement is more precisely defined at the end of Section 4.

The formula means that a mediated matching between two protocols P_1 and P_2 exists if and only if the common language structure of P_1 is similar to the common language structure of P_2 and there exists a mediator M which is a refinement of both the common language structures CLS_{P_1}, CLS_{P_2}.

In other words, P_1 matches P_2 if and only if: (i) the portions of the structures of P_1 and P_2, which represent the common language between them, are equal and (ii) for any transition in the common language structure of P_1, there exists the corresponding transition in the common language structure of P_2 and vice versa, and M is a refinement of both structures CLS_{P_1}, CLS_{P_2}.

Instantiating the formula, let A, B be two LTS representing protocols, $CLS_A =$ common language structure of A, and $CLS_B =$ common language structure of B.

$$A \ matches \ B \Leftrightarrow (equal(CLS_A, CLS_B) \ and$$
$$CLS_A = (S_A, L_A, D_A, F_A, s_{0A}) \ and$$
$$CLS_B = (S_B, L_B, D_B, F_B, s_{0B}) \ and$$
$$(\forall \ d_i \in D_A \ \exists \ d_j \in D_B : corresponding(d_i, d_j) \ and$$
$$viceversa)) \ and \ (\exists \ M : M \ refines \ CLS_A \ and$$
$$CLS_B))$$

Thus, suppose that P_1 and P_2 have equal common language structures (we recall that the definition of equality of structures do not take into account the labels but only looks at the control). The problem that remains still unsolved is to understand if it exists a protocol M that refines both the common language structures.

In order to solve it, the ontology has a central role because it maps one or more steps of a protocol (i.e., actions of its LTS) into one or more steps of the other one (i.e., actions of the other LTS).

It has to be noticed that we are considering a networked environment made by two mismatching protocols P_1 and P_2 that want to interoperate. But the environment can actually be populated by many protocols, and P_1 and P_2, during their evolution, can interact also with the remainder of the environment till they reach a suitable state in which they are ready to synchronize with each other. We manage this possibility by assuming that the mediator, besides forwarding the synchronization messages between the two protocols (i.e., the ones belonging to the common language), also provides the needed complementary behaviors to P_1 and P_2 to let them evolve. Hence we can consider as τ actions the actions that have to be exchanged with third parties. This may make it possible to relax the notion of equality with the notion of observational equivalence [28]; we leave this investigation for future work.

The two common language structures of P_1 and of P_2, represent a common abstraction for the two protocols and lay the bases to reach the communication.

The mediator M, while mediating between P_1 and P_2, implements a sort of hiding operation by providing the portions of P_1 and/or P_2 that allow them to evolve and reach the right state from which they can synchronize each other.

Thus the problem is shifted to understand what are the states of P_1 and P_2 from which they can synchronize. Note that the states of P_1 and P_2 from which they can synchronize can be elicited from the common language. If an action of P_1 (P_2) is a synchronization action with P_2 (P_1), its corresponding complementary action is in the common language. With corresponding complementary action we mean the action of a corresponding transition that if it is overlined in the first protocol, then is non-overlined in the second one and viceversa.

Due to the possibility for P_1 and P_2 to synchronize with third parties, both the ontology and the mediator are important. The first identifies the portions of the two protocols

for the mutual synchronization and identifies also the states that P_1 and P_2 have to reach in order to synchronize. The mediator simulates the complementary protocols of portions of P_1 and P_2 leading them to their mutual synchronization states. It also "tunnels" the messages between the protocols to let them interoperate.

Protocol mapping. Exploiting the provided definition of matching between two protocols, and related formalization, we describe the mapping algorithm for two given protocols.

Let P and Q be the LTS representing two protocols and let CLP_P, CLP_Q the common language structures of P and of Q respectively. The mapping algorithm checks the similarity of the common language structures of P and of Q. If the common language structures are similar, it constructs the LTS representing the abstract mediator protocol M for that portion of P and Q. The algorithm ends either with the mediator abstract specification (if it exists) or by demonstrating its non existence (and thus the impossibility for P and Q to interoperate). The protocol mapping, which is built by exploiting the matching notion, forms the basis for the subsequent synthesis of the mediator's actual behavior (the actual implementation of M). This latter will be constituted by: 1) the implementation of the mediator's abstract specification and 2) the implementation of portions of complementary protocols of P and Q (if any). The mapping algorithm is as follows:

1) Create an LTS M by copying the CLS_P (because in order to be able to interoperate the CLS_P and the CLS_Q must be equal);

2) WHILE (similar(M, CLS_Q) is TRUE) {
for any transition in M, update its label by concatenating to it the symbol / and the label of the corresponding transition in CLS_Q;
} IF (similar(M, CLS_Q) is FALSE)) THEN {
the mediating matching relation does not exists and P and Q are not able to interoperate
} ELSE return M

The abstract mediator M built by this algorithm (if it is possible), will be part of the actual mediator that has to be synthesized.

4. Towards mediating connector synthesis

The synthesis of a mediating connector between two protocols that want to interoperate follows from our definition of mediated matching and associated protocol mapping. Given two protocols A and B, one has to check if a mediated matching exists. If yes, then there exists the possibility for the two protocols to interoperate thanks to a mediator that has to be elicited and synthesized. The protocols mapping, if the matching succeeds, finds the LTS that represents the abstract behavior of the mediator while the synthesis produces the actual implementation of the mediator containing both the implementation of the mediator abstract specification and the implementation of portions of complementary protocols of A and B (if any).

Figure 2 illustrates the abstract mediator protocol M of the example sketched in Section 2.2 and also illustrated in Figure 1, which refers to two instant messaging protocols.

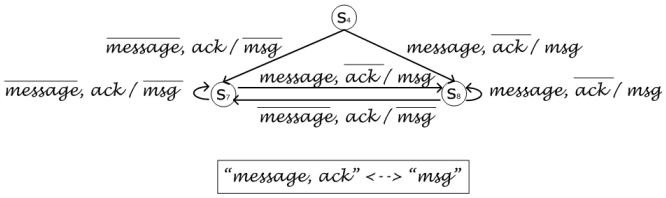
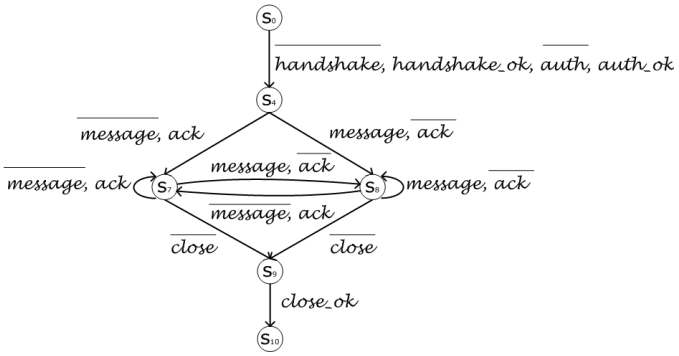
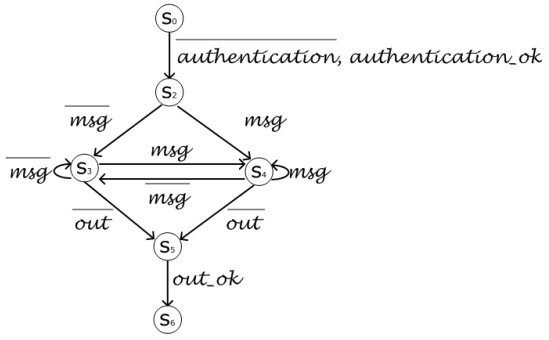


Figure 2. Abstract behavior of the mediator.

The abstract mediator is built, as described in Section 3.2, by first computing the structures of MSN and of Jabber protocols that are shown in Figure 3.



(a) Windows Messenger structure

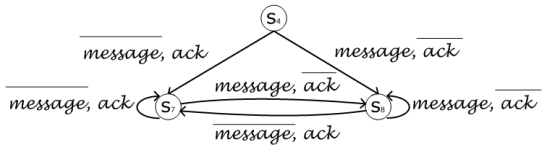


(b) Jabber Messenger structure

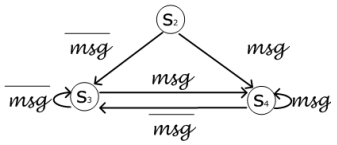
Figure 3. Structures of the two instant messaging protocols

Then one has to compute the common language structures of the two protocols, illustrated in Figure 4, through a given

ontology (not completely shown here) and finally, while checking their similarity, one has to update the labels of the mediator protocol previously created and initialized to the MSN protocol (or the Jabber protocol).



(a) Common language structure of Windows Messenger



(b) Common language structure of Jabber

Figure 4. Common language structures of the MSN and of the Jabber protocols

In addition to the abstract mediator, in order to synthesize the actual mediator behavior, we need also portions of complementary protocols for both MSN and Jabber for the part of their structures that is not the common language structures. These portions of protocols allow the two protocols to evolve till the state where they synchronize with each other. Referring to the concrete example, the portions of complementary protocols for MSN and Jabber are the complement of the parts of their structures that remain when eliminating the ones in Figure 4.

The other issue that still needs explanation is that our definition of refinement is slightly different from that of [29], [30]. In the mediated matching we require that the mediator refines the common language structure of the two protocols that want to interoperate and this is achieved through the following algorithm.

Consider the abstract mediator behavior M of Figure 5. Any transition $t = \alpha \ / \ \beta$ is labelled with two traces divided by a slash symbol where α belongs to a given LTS (protocol) P_1, and β belongs to another given LTS (protocol) P_2.

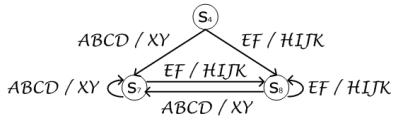


Figure 5. Abstract behavior of a mediator.

Then the refinement algorithm behaves as follows:

For any transition $s_i \stackrel{t}{\Rightarrow} s_j$ of M

IF either α and/or β contain only one action, THEN mark the transition as visited;

ELSE (α and β contain n actions with $n > 1$) finds a more refined correspondence (if any) between the actions of α and β thanks to the ontology (also used to highlight the common language structure) and refines M accordingly by adding new states and transitions between s_i and s_j.

Putting this algorithm in practice, and referring to the abstract mediator of Figure 5, suppose that the ontology maps "ABC" into "X" and "D" into "Y". Then the refined mediator is shown in Figure 6.

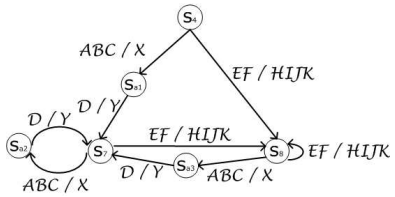


Figure 6. Refinement of the abstract mediator of Figure 5.

5. Conclusion

With the networked environment being increasingly open and dynamics, the seamless composition and related connection of systems become a prime requirement. However, the openness of the environment comes along with great heterogeneity in the networked systems, which challenges their connection. Mediating connector then appears as a paradigm of choice to effectively overcome behavioral mismatches among the protocols run by the networked systems that need to coordinate. However, while the mediator paradigm has deserved much attention over the last few years, including research towards supported automated mediation, key principles remain loosely defined.

This paper proposes a first formal characterization of the mediating connector concept. The approach is a first attempt and cover some of the mismatches listed in Sections 1 and 2 but needs to be extended to cover a larger set. Future work concerns several investigation areas among which an assessment of the generality of the proposed formalization and the realization of automatic support to the elicitation and the subsequent synthesis of the mediator.

Acknowledgment

The work is partly supported by: the project CONNECT No 231167 of the Future and Emerging Technologies (FET) programme within the ICT theme of the Seventh Framework Programme for Research of the European Commission, the Italian PRIN D-ASAP, and French project INRIA associated

team ASA. Romina Spalazzese would also like to thank ISTI CNR for hosting her.

References

[1] M. Weiser, "The computer for the 21^{st} century," *Scientific American*, Sep. 1991.

[2] J. Nakazawa, H. Tokuda, W. K. Edwards, and U. Ramachandran, "A bridging framework for universal interoperability in pervasive systems," in *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2006, p. 3.

[3] Y.-D. Bromberg and V. Issarny, "Indiss: interoperable discovery system for networked services," in *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2005, pp. 164–183.

[4] D. Chappell., *Enterprise Service Bus*. O'Reilly, 2004.

[5] P. Grace and G. B. S. Samuel, "A reflective framework for discovery and interaction in heterogeneous mobile environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, 2005.

[6] Y.-D. Bromberg, V. Issarny, and P.-G. Raverdy, "Interoperability of service discovery protocols: Transparent versus explicit approaches," in *Proceedings of the 15th IST Mobile & Wireless Communications Summit*, Mykonos, June, Jun. 2006.

[7] V. Issarny, M. Caporuscio, and N. Georgantas, "A perspective on the future of middleware-based software engineering," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 244–258.

[8] G. Wiederhold and M. Genesereth, "The conceptual basis for mediation services," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 12, no. 5, pp. 38–47, 1997.

[9] G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Computer*, vol. 25, pp. 38–49, 1992.

[10] M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel, "A semantic web mediation architecture," in *In Proceedings of the 1st Canadian Semantic Web Working Symposium (CSWWS 2006*. Springer, 2006.

[11] B. Liskov and J. Wing, "Behavioral subtyping using invariants and constraints," 1999. [Online]. Available: citeseer.ist.psu.edu/article/liskov99behavioral.html

[12] D. Fensel and C. Bussler, "The web service modeling framework wsm," *Journal of Electronic Commerce Research and Application*, vol. 1, no. 1, pp. 113–137, 2002.

[13] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," *SIGMOD Rec.*, vol. 33, no. 4, pp. 65–70, 2004.

[14] M. Stollberg, E. Cimpian, and D. Fensel, "Mediating capabilities with deltarelations," in *In Proceedings of the First International Workshop on Mediation in Semantic Web Services, co-located with the Third International Conference on Service Oriented Computing (ICSOC 2005*, 2005.

[15] R. Kumar, S. Nelvagal, and S. I. Marcus, "A discrete event systems approach for protocol conversion," *Discrete Event Dynamic Systems*, vol. 7, no. 3, pp. 295–315, 1997.

[16] B. Spitznagel and D. Garlan, "A compositional formalization of connector wrappers," in *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 374–384.

[17] Windows Live Messenger, http://www.messenger.it/.

[18] Jabber Software Foundation, http://www.jabber.org/.

[19] R. M. Keller, "Formal verification of parallel programs," *Commun. ACM*, vol. 19, no. 7, pp. 371–384, 1976.

[20] E. Cimpian and A. Mocan, "Wsmx process mediation based on choreographies." in *Business Process Management Workshops*, C. Bussler and A. Haller, Eds., vol. 3812, 2005, pp. 130–143.

[21] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing adapters for web services integration," in *In Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE), Porto, Portugal*. Springer Verlag, 2005, pp. 415–429.

[22] C. Wu and E. Chang, "An analysis of web services mediation architecture and pattern in synapse," in *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1001–1006.

[23] 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 '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 137–146.

[24] R. Vaculín and K. Sycara, "Towards automatic mediation of owl-s process models," *Web Services, IEEE International Conference on*, vol. 0, pp. 1032–1039, 2007.

[25] R. Vaculín, R. Neruda, and K. P. Sycara, "An agent for asymmetric process mediation in open environments." in *SOCASE*, ser. Lecture Notes in Computer Science, R. Kowalczyk, M. N. Huhns, M. Klusch, Z. Maamar, and Q. B. Vo, Eds., vol. 5006. Springer, 2008, pp. 104–117.

[26] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati, "Semi-automated adaptation of service interactions," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 993–1002.

[27] S. K. Williams, S. A. Battle, and J. E. Cuadrado, "Protocol mediation for adaptation in semantic web services," in *ESWC*, 2006, pp. 635–649.

[28] R. Milner, *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[29] R. V. Glabbeek and U. Goltz, "Refinement of actions and equivalence notions for concurrent systems," *Acta Informatica*, vol. 37, pp. 229–327, 1998.

[30] R. Gorrieri and A. Rensink, "Action refinement," in *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds. Elsevier, 2001, ch. 16, pp. 1047–1147.