Achieving Interoperability through Semantics-Based Technologies: The Instant Messaging Case

Amel Bennaceur[1], Valérie Issarny[1], Romina Spalazzese[2], and Shashank Tyagi[3]

[1] Inria, Paris-Rocquencourt, France
[2] University of L'Aquila, L'Aquila, Italy
[3] Institute of Technology, Banaras Hindu University, India

Abstract. The success of pervasive computing depends on the ability to compose a multitude of networked applications dynamically in order to achieve user goals. However, applications from different providers are not able to interoperate due to incompatible interaction protocols or disparate data models. Instant messaging is a representative example of the current situation, where various competing applications keep emerging. To enforce interoperability at runtime and in a non-intrusive manner, *mediators* are used to perform the necessary translations and coordination between the heterogeneous applications. Nevertheless, the design of mediators requires considerable knowledge about each application as well as a substantial development effort. In this paper we present an approach based on ontology reasoning and model checking in order to generate correct-by-construction mediators automatically. We demonstrate the feasibility of our approach through a prototype tool and show that it synthesises mediators that achieve efficient interoperation of instant messaging applications.

Keywords: Interoperability, Composition, Ontology, Verification, Mediation, Universal Instant Messaging.

1 Introduction

Pervasive computing promises a future where a multitude of networked applications dynamically discover one another and seamlessly interconnect in order to achieve innovative services. However, this vision is hampered by a plethora of independently-developed applications with *compatible functionalities* but which are unable to interoperate as they realise them using disparate interfaces (data and operations) and protocols. Compatible functionalities means that at a high enough level of abstraction, the functionality provided by one application is semantically equivalent to that required by the other.

The evolution of instant messaging (IM) applications provides a valuable insight into the challenges facing interoperability between today's communicating applications. Indeed, the number of IM users is constantly growing – from around 1.2 billion in 2011 to a predicted 1.6 billion in 2014 [21] – with an increasing emphasis on mobility – 11% of desktop computers and 18% of smartphones have instant messaging applications installed [19]– and the scope of IM providers is expanding to include social networking such as Facebook that embeds native IM services onto their Web site. Consequently, different versions and competing standards continue to emerge. Although this situation may be frustrating from a user perspective, it seems unlikely to change. Therefore, many solutions that aggregate the disparate systems, without rewriting or modifying them, have been proposed [12]. These solutions use intermediary middleware entities, called *mediators* [24] – also called mediating adapters [25], or converters [4] – which perform the necessary coordination and translations to allow applications to interoperate despite the heterogeneity of their data models and interaction protocols.

Nevertheless, creating mediators requires a substantial development effort and thorough knowledge of the application-domain. Moreover, the increasing complexity of today's software systems, sometimes referred to as Systems of Systems [15], makes it almost impossible to manually develop 'correct' mediators, i.e., mediators guaranteeing deadlock-free interactions and the absence of unspecified receptions [25]. Starlink [3] assists developers in this task by providing a framework that performs the necessary mediation based on a domain-specific description of the translation logic. Although this approach facilitates the development of mediators, developers are still required to understand both systems to be bridged and to specify the translations.

Furthermore, in pervasive environments where there is no *a priori* knowledge about the concrete applications to be connected, it is essential to guarantee that the applications associate the same *meaning* to the data they exchange, i.e., semantic interoperability [10]. *Ontologies* support semantic interoperability by providing a machine-interpretable means to automatically reason about the meaning of data based on the shared understanding of the application domain [1]. Ontologies have been proposed for Instant Messaging although not for the sake of protocol interoperability but rather for semantic archiving and enhanced content management [8]. In a broader context, ontologies have also been widely used for the modelling of Semantic Web Services, and to achieve efficient service discovery and composition [17]. Semantic Markup for Web Services[1] (OWL-S) uses ontologies to model both the functionality and the behaviour of Web services. Besides semantic modelling, Web Service modelling Ontology (WSMO) supports runtime mediation based on pre-defined mediation patterns but without ensuring that such mediation does not lead to a deadlock [6]. Although ontologies have long been advocated as a key enabler in the context of service mediation, no principled approach has been proposed yet to the automated synthesis of mediators by systematically exploiting ontologies [2].

This paper focuses on distributed applications that exhibit compatible functionalities but are unable to interact successfully due to mismatching interfaces or protocols. We present an approach to synthesise mediators automatically to ensure the interoperation of heterogeneous applications based on the semantic compatibility of their data and operations. Specifically, we rely on a domain-specific

[1] `http://www.w3.org/Submission/OWL-S/`

ontology (e.g., an IM ontology) to infer one-to-one mappings between the operations of the applications' interfaces and exploit these mappings to generate a correct-by-construction mediator. Our contribution is threefold:

– *Formal modelling of interaction protocols.* We introduce an ontology-based process algebra, which we call Ontology-based Finite State Processes (OFSP), to describe the observable behaviour of applications. The rationale behind a formal specification is to make precise and rigorous the description and the automated analysis of the observable behaviour of applications.
– *Automated generation of mediators for distributed systems.* We reason about the semantics of data and operations of each application and use a domain ontology to establish, if they exist, one-to-one mappings between the operations of their interfaces. Then, we verify that these mappings guarantee the correct interaction of the two applications and we generate the corresponding mediator.
– *Framework for automated mediation.* We provide a framework that refines the synthesised mediator and deploys it in order to automatically translate and coordinate the messages of mediated applications.

Section 2 examines in more detail the challenges to interoperability using the IM case. Section 3 introduces the ontology-based model used to specify the interaction protocols of application. Section 4 presents our approach to the automated synthesis of mediators that overcome data and protocol mismatches of functionally compatible applications and illustrates it using heterogeneous instant messaging applications. Section 5 describes the tool implementation while Section 6 reports the experiments we conducted with the instant messaging applications and evaluate the approach. The results show that our solution significantly reduces the programming effort and ensures the correctness of the mediation while preserving efficient execution time. Section 7 examines related work. Finally, Section 8 concludes the paper and discusses future work.

2 The Instant Messaging Case

Instant messaging (IM) is a popular application for many Internet users and is now even embedded in many social networking systems such as Facebook. Moreover, since IM allows users to communicate in real-time and increases their collaboration, it is suitable for short-lived events and conferences such as Instant Communities for online interaction at the European Future Technologies Conference and Exhibition[2] (FET'11) that took place in May 2011.

Popular and widespread IM applications include Windows Live Messenger[3](commonly called MSN messenger), Yahoo! Messenger[4], and Google Talk[5]

[2] `http://www.fet11.eu/`
[3] `http://explore.live.com/windows-live-messenger/`
[4] `http://messenger.yahoo.com/`
[5] `http://www.google.com/talk/`

which is based on the Extensible Messaging and Presence Protocol[6] (XMPP) standard protocol. These IM applications offer similar functionalities such as managing a list of contacts or exchanging textual messages. However, a user of Yahoo! Messenger is unable to exchange instant messages with a user of Google Talk. Indeed, there is no common standard for IM. Thus, users have to maintain multiple accounts in order to interact with each other (see Figure 1). This situation, though cumbersome from a user perspective, unfortunately reflects the way IM – like many other existing applications – has developed.

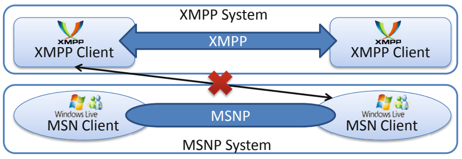

Fig. 1. Interoperability issue between heterogeneous IM systems

A solution that guarantees interoperability between heterogeneous IM applications has to cope with the following heterogeneity dimensions:

- *Data heterogeneity.* MSN Messenger protocol (MSNP), the protocol used by Windows Live Messenger, uses text-based messages whose structure includes several constants with predefined values. On the other hand, the Yahoo! Messenger Protocol (YMSG) defines binary messages that include a header and key-value pairs. As for XMPP messages, they are defined according to a given XML Schema.
- *Protocol heterogeneity.* Even though IM applications are simple and quite similar, each one communicates with its own proprietary application server used to authenticate and to relay the messages between instant messaging clients. Consequently, each application has its own interaction protocol.

Achieving interoperability between independently developed systems has been one of the fundamental goals of middleware research. Prior efforts have largely concentrated on solutions where conformance to the same standard is required e.g., XMPP. However, compliance to a unique standard is not always feasible given the competitive pressures in the marketplace.

Middleware-based approaches define a common abstraction interface (e.g., Adium[7]) or an intermediary protocol (e.g., J-EAI[8] and CrossTalk [18]) promote interoperability in a transparent manner. However, relying on a fixed intermediary interface or protocol might become restrictive over time as new functionalities and features emerge. By synthesising mediators automatically and rigorously we relieve developers from the burden of implementing or specifying such mediators and further ensures their correctness.

[6] `http://www.xmpp.org/`
[7] `http://adium.im/`
[8] `http://www.process-one.net`

Semantics-based solutions (e.g., SAM [8] and Nabu[9]) use ontologies to enhance the functionalities of IM applications by reasoning about the content of messages and overcoming mismatches at the data level but assume the use of the same underlying communication protocol. Hence, even though an enormous amount of work is being carried out on the development of concrete interoperability solutions that rely on ontologies to overcome application heterogeneity, none propose an approach to generate mediators able to overcome both data and protocol heterogeneity. In the next section, we introduce our ontology-based approach to interoperability that automatically synthesises mediators to transparently solve both data and protocol mismatches between functionally compatible applications at runtime.

3 Ontology-Based Modelling of Interaction Protocols

Automated mediation of heterogeneous applications requires the adequate modelling of their data and interaction protocols. In this section, we introduce OFSP (Ontology-based Finite State Processes), a semantically-annotated process algebra to model application behaviour.

3.1 Ontologies in a Nutshell

An ontology is *a shared, descriptive, structural model, representing reality by a set of concepts, their interrelations, and constraints under the open-world assumption* [1]. The Web Ontology Language[10] (OWL) is a W3C standard language to formally model ontologies in the Semantic Web. Concepts are defined as OWL classes. Relations between classes are called OWL properties. Ontology reasoners are used to support automatic inference on concepts in order to reveal new relations that may not have been recognised by the ontology designers. OWL is based on description logics (DL), which is a knowledge representation formalism with well-understood formal properties [1]. To verify the interaction of networked applications, we are in particular interested in specialisation/generalisation relations between their concepts. In this sense, DL resemble in many ways type systems with concept *subsumption* corresponding to type subsumption. Nevertheless, DL are by design and tradition well-suited for domain-specific services and further facilitate the definition and reasoning about composite concepts, e.g., concepts constructed as disjunction or conjunction of other concepts. Subsumption is the basic reasoning mechanism and can be used to implement other inferences, such as satisfiability and equivalence, using pre-defined reductions [1]:

Definition 1 ($\sqsubseteq$: Subsumption). *A concept C is subsumed by a concept D in a given ontology $\mathcal{O}$, written $C \sqsubseteq D$, if in every world consistent with the axioms of the ontology $\mathcal{O}$ the set denoted by C is a subset of the set denoted by D.*

[9] `http://nabu.opendfki.de/`

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

The subsumption relation is both transitive and reflexive and defines a hierarchy of concepts. This hierarchy always contains a built-in top concept *owl:Thing* and bottom concept *owl:Nothing*.

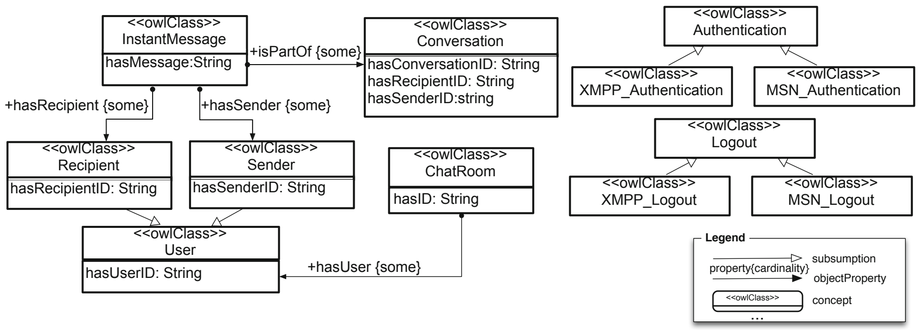

Fig. 2. The instant messaging ontology

Figure 2 depicts the instant messaging ontology. An InstantMessage class has at least one sender hasSender{some}, one recipient hasRecipient{some}, and one message hasMessage. hasSender{some} and hasRecipient{some} are object properties that relate an instant message to a sender or a recipient while hasMessage is a data property associated with the InstantMessage class. The Sender and Recipient classes are subsumed by the User class. Indeed, any instance of the two former classes is also an instance of the latter. A Conversation is performed between a sender (who initialises it) and a recipient, and the conversation has its own identifier. An instant message isPartOf a conversation. A ChatRoom represents a venue where multiple users can join and exchange messages.

3.2 Modelling Protocols Using Ontology-Based FSP

The interaction protocol of an application describes how the operations of its *interface* are coordinated in order to achieve a specified functionality. We build upon state-of-the-art approaches to formalise interaction protocols using process algebra, in particular Finite State Processes (FSP) [14]. FSP has proven to be a convenient formalism for specifying concurrent systems. Although another process algebra would have worked equally well, we choose FSP for convenience and to exploit the Labelled Transition System Analyser (LTSA) in order to automate reasoning and analysis of interaction protocols specified as finite processes.

Each process P is associated with an interface αP that defines the set of observable actions that the application requires from/provides to its running environment. We structure these actions and annotate them using a domain ontology $\mathcal{O}$ so as to specify their semantics, resulting in Ontology-based FSP (OFSP). An *input action* $a = <op, I, O>$ specifies a required operation $op \in \mathcal{O}$ for which the application produces a set of input data $I = \{in \in \mathcal{O}\}$ and consumes a set of output data $O = \{out \in \mathcal{O}\}$. The dual *output action*[11] $\overline{b} =<\overline{op}, I, O>$ refers to a provided operation *op* for which the application uses the inputs I and produces the corresponding outputs O. Note that all actions are annotated using the same domain ontology $\mathcal{O}$ describing the application-specific concepts and relations. The rationale behind this notation is to enable behavioural analysis based on the semantics of process actions. Indeed, only if both sides of communication assign the same semantics to their actions, can they interact correctly. In addition, τ is used to denote an internal action that cannot be observed by the environment. There are two types of processes: *primitive processes* and *composite processes*. Primitive processes are constructed through action prefix ($\rightarrow$), choice ($|$), and sequential composition (;). Composite processes are constructed using parallel composition ($\|$).

The semantics of OFSP builds upon the semantics of FSP, which is given in terms of Labelled Transition Systems (LTS) [13]. The LTS interpreting an OFSP process P is a directed graph whose nodes represent the process states and each edge is labelled with an action $a \in \alpha P$ representing the behaviour of P after it engages in an action a. $P \overset{a}{\rightarrow} P'$ denotes that P transits with action a into P'. $P \overset{s}{\Rightarrow} P'$ is a shorthand for $P \overset{a_1}{\rightarrow} P_1 ... \overset{a_n}{\rightarrow} P', s = \langle a_1, ..., a_n \rangle, a_i \in \alpha P \cup \tau$. There exists a start node from which the process begins its execution. The END state indicates a successful termination. $traces(P)$ denotes the set of all successfully-terminating traces of P. When composed in parallel, processes synchronise on dual actions while actions that are in the alphabet of only one of the two processes can occur independently of the other process.

$$
\begin{aligned}
\mathsf{MSNClient} &= (<MSN\_Authentication\_Request, \{UserID\}, \{Challenge\} > \\
&\rightarrow <\overline{MSN\_Authentication\_Response}, \{Response\}, \{Authentication\_ok\} > \\
&\rightarrow \mathsf{ExchangeMsgs}). \\
\mathsf{ExchangeMsgs} &= (<CreateChatRoom, \{UserID\}, \{ConversationID\} > \\
&\rightarrow <JoinChatRoom, \{UserID\}, \{Acceptance\} > \rightarrow \mathsf{P1} \\
&\mid < \overline{JoinChatRoom}, \{UserID\}, \{Acceptance\} > \\
&\rightarrow < \{ChatRoomInfo, \emptyset, \{ConversationID\} > \rightarrow \mathsf{P1}), \\
\mathsf{P1} &= (<InstantMessage, \{UserID, ConversationID, Message\}, \emptyset > \rightarrow \mathsf{P1} \\
&\mid <\overline{InstantMessage}, \{UserID, ConversationID, Message\}, \emptyset > \rightarrow \mathsf{P1} \\
&\mid <MSN\_Logout, \{UserID\}, \emptyset > \rightarrow \mathsf{END}).
\end{aligned}
$$

Fig. 3. OFSP specification of MSNP

$$
\begin{aligned}
\mathsf{XMPPClient} &= (< XMPP\_Authentication\_Request, \{UserID\}, \{Challenge \} > \\
&\rightarrow <\overline{XMPP\_Authentication\_Response}, \{Response\}, \{Authentication\_ok\} > \\
&\rightarrow \mathsf{ExchangeMsgs}). \\
\mathsf{ExchangeMsgs} &= (<InstantMessage, \{SenderID, RecepientID, Message\}, \emptyset > \rightarrow \mathsf{ExchangeMsgs} \\
&\mid <\overline{InstantMessage}, \{SenderID, RecipientID, Message\}, emptyset > \\
&\rightarrow \mathsf{ExchangeMsgs} \\
&\mid < XMPP\_Logout, \{UserID\}, emptyset > \rightarrow \mathsf{END}).
\end{aligned}
$$

Fig. 4. OFSP specification of XMPP

The concepts and properties defined in the IM ontology are used to specify MSNP and XMPP clients using OFSP, as illustrated in Figures 3 and 4

[11] Note the use of an overline as a convenient shorthand notation to denote output actions

respectively, focusing on message exchange. Each IM application performs authentication and logout with the associated server. Before exchanging messages, the MSNP application has to configure a *chat room* where the MSN conversation can take place between the user that initiates this conversation (sender) and the user who accepts to participate in this conversation (recipient). In XMPP each message simply contains both the sender and the recipient identifiers.

4 Ontology-Based Approach to Mediator Synthesis

In this section we consider two functionally-compatible applications, described through OFSP processes P_1 and P_2, that are unable to interoperate due to differences in their interfaces or protocols. Functional compatibility means that their required/provided high-level functionalities are semantically equivalent [12]. Our aim is to enforce their interoperation by synthesising a mediator that addresses these differences and guarantees their *behavioural matching.* The notion of behavioural matching is formally captured through *refinement* [11]. A process Q *refines* a process P if every trace of Q is also a trace of P, i.e., $traces(Q) \subseteq traces(P)$. However, this notion of refinement analyses the dynamic behaviour of processes assuming close-world settings, i.e., the use of the same interface to define the actions of both processes. What is needed is a notion of compatibility that takes into account the semantics of actions while relying on a mediator process M to compensate for the syntactic differences between actions and guarantees that the processes communicate properly.

To this end, we first reason about the semantics of actions so as to infer the correspondences between the actions of the processes' interfaces and generate the mapping processes that perform the necessary translations between *semantically compatible actions.* Various mapping relations may be defined. They primarily differ according to their complexity and inversely proportional flexibility. In this paper we focus on one-to-one mappings, i.e., direct correspondences between actions. During the synthesis step, we explore the various possible mappings in order to produce a correct-by-construction mediator, i.e., a mediator M that guarantees that the composite process $P_1 \| M \| P_2$ reaches an END state, or determines that no such mediator exists.

In this section we introduce the semantic compatibility of actions, and use it to define behavioural matching. Then, we present the automated synthesis algorithm.

4.1 Semantic Compatibility of Actions

A *sine qua non* condition for two processes P_1 and P_2 to interact is to agree on the data they exchange. However, independently-developed applications often define different interfaces. The mediator can compensate for the differences between interfaces by mapping their actions if and only if they have the same semantics. We first define the notion of *action subsumption* and then, use it to define the *semantic compatibility* of actions.

Definition 2 ($\sqsubseteq_{\mathcal{O}}$: Action Subsumption). *An action $a_1 = <op_1, I_1, O_1>$ is subsumed by an action $\overline{a_2} = <\overline{op_2}, I_2, O_2>$ according to a given ontology $\mathcal{O}$, noted $a_1 \sqsubseteq_{\mathcal{O}} a_2$, iff: (i) $op_2 \sqsubseteq op_1$, (ii) $\forall i_2 \in I_2, \exists i_1 \in I_1$ such that $i_1 \sqsubseteq i_2$, and (iii) $\forall o_1 \in O_1, \exists o_2 \in O_2$ such that $o_2 \sqsubseteq o_1$.*

The idea behind this definition is that an input action can be mapped to an output one if the required operation is less demanding; it provides richer input data and needs less output data. This leads us to the following definition of semantic compatibility of actions:

Definition 3 ($\approx_{\mathcal{O}}$: Semantic Compatibility of Actions). *An action a_1 is semantically compatible with an action a_2, denoted $a_1 \approx_{\mathcal{O}} a_2$, iff a_1 is subsumed by a_2 (i.e., a_1 is required and a_2 provided) or a_2 is subsumed by a_1 (a_2 is required and a_1 provided) .*

The semantic compatibility between two actions allows us to generate an action mapping process as follows:

$$\mathsf{M}_{\mathcal{O}}(a_1, a_2) = \begin{cases} a_1 \overset{\mathcal{O}}{\longmapsto} \overline{a_2} \text{ if } a_1 \text{ is subsumed by } \overline{a_2} \\ a_2 \overset{\mathcal{O}}{\longmapsto} \overline{a_1} \text{ if } a_2 \text{ is subsumed by } \overline{a_1} \end{cases}$$

The process that maps action a_1 to action $\overline{a_2}$, written $a_1 \overset{\mathcal{O}}{\longmapsto} \overline{a_2}$ captures each input data from the input action, assigns it to the appropriate input of the output action ($i_2 \leftarrow i_1$), then takes each output data of the output action and assigns it to the expected output of the input action ($o_1 \leftarrow o_2$). This assignment is safe since an instance of i_1 (resp. o_2) is necessarily an instance i_2 of (resp. o_1).

Let us consider $a_1 = <InstantMessage, \{UserID, ConversationID, Message\}, \emptyset>$ associated to the MSN client and $\overline{a_2} = <\overline{InstantMessage}, \{SenderID, RecipientID, Message\}, \emptyset>$ associated to the XMPP client. The IM ontology indicates that (i) Sender is subsumed by User, and (ii) *ConversationID* identifies a unique Conversation, which includes a *RecipientID* attribute. Consequently, a_1 is subsumed by $\overline{a_2}$.

4.2 Behavioural Matching through Ontology-Based Model Checking

We aim at assessing behavioural matching of two processes P_1 and P_2 given the semantic compatibility of their actions according to an ontology $\mathcal{O}$. To this end, we first filter out communications with third party processes [20]. The communicating trace set of P_1 with P_2, noted $traces(P_1)\uparrow_{\mathcal{O}}P_2$ is the set of all successfully-terminating traces of P_1 restricted to the observable actions that have semantically compatible actions in αP_2.

Definition 4 ($\uparrow_{\mathcal{O}}$: Communicating Trace Set). $traces(P_1)\uparrow_{\mathcal{O}}P_2 \overset{\text{def}}{=}$ $\{s = \langle a_1, a_2, ..., a_n \rangle, a_i \in \alpha P_1 \mid P_1 \overset{s}{\Rightarrow} \mathsf{END}$ *such that* $\forall a_i, \exists b_i \in \alpha P_2 | a_i \approx_{\mathcal{O}} b_i\}$

As an illustration, both the MSNP and XMPP IM clients perform their authentication and logout with their respective servers. Additionally, MSNP also performs the actions related to the configuration of the chat room with its servers.

Consequently their communicating traces sets are restricted to instant message exchange.

Then, two traces $s_1 = \langle a_1 a_2 ... a_n \rangle$ and $s_2 = \langle b_1 b_2 ... b_n \rangle$ *semantically match*, written $s_1 \equiv_{\mathcal{O}} s_2$, iff their actions semantically match in sequence.

Definition 5 **($\equiv_{\mathcal{O}}$: Semantically Matching Traces)**
$s_1 \equiv_{\mathcal{O}} s_2 \overset{\text{def}}{=} a_i \approx_{\mathcal{O}} b_i \;\; 1 \leq i \leq n$

The associated mapping is then as follows:

$$\mathsf{Map}_{\mathcal{O}}(s_1, s_2) = \mathsf{M}_{\mathcal{O}}(a_1, b_1); ...; \mathsf{M}_{\mathcal{O}}(a_n, b_n)$$

Based on the semantic matching of traces, a process P_2 *ontologically refines* a process P_1 ($P_1 \models_{\mathcal{O}} P_2$) iff each trace of P_2 semantically matches a trace of P_1:

Definition 6 **($\models_{\mathcal{O}}$: Ontological Refinement)**

$$P_1 \models_{\mathcal{O}} P_2 \overset{\text{def}}{=} \forall s_2 \in traces(P_2){\uparrow_{\mathcal{O}}}P_1,\ \exists s_1 \in traces(P_1){\uparrow_{\mathcal{O}}}P_2 : s_2 {\equiv_{\mathcal{O}}} s_1$$

By checking ontological refinement between P_1 and P_2, we are able to determine the following *behavioural matching* relations:

– *Exact matching*–($P_1 \models_{\mathcal{O}} P_2$) $\wedge$ ($P_2 \models_{\mathcal{O}} P_1$): assesses compatibility for symmetric interactions such as peer-to-peer communication where both processes provide and require the similar functionality.
– *Plugin matching*–($P_1 \models_{\mathcal{O}} P_2$) $\wedge$ ($P_2 \not\models_{\mathcal{O}} P_1$): evaluates compatibility for asymmetric interactions such as client-server communication where P_1 is providing a functionality required by P_2.
– *No matching*–($P_1 \not\models_{\mathcal{O}} P_2$) $\wedge$ ($P_2 \not\models_{\mathcal{O}} P_1$): identifies behavioural mismatch.

Behavioural matching is automated through *ontology-based model checking*. Model checking is an attractive and appealing approach to ensure system correctness that proved to be a very sound technique to automatically verify concurrent systems. The gist of model checking approaches is the exhaustive state exploration. This exploration is performed by model checkers using efficient algorithms and techniques that make it possible to verify systems of up to 10^{1300} states in few seconds [7]. However, even if these techniques effectively handle very large systems, the actions of the models they consider are usually simple strings and the verification matches actions based on their syntactic equality. We build upon these model checking techniques but further match actions based on their semantic compatibility. The semantic compatibility of actions is defined based on the domain knowledge encoded within a given ontology.

Referring to the IM case, all the traces of MSNP and XMPP processes semantically match. Subsequently, these two processes are in exact matching relation, and a mediator can be synthesised to perform action translations and enable their correct interaction.

4.3 Automated Mediator Synthesis

In the case where P_1 and P_2 match, that is exact matching in the case of peer-to-peer communication or plugin matching in the case of client/server communication, we synthesise the mediator that makes them properly interact. The algorithm incrementally builds a mediator M by forcing the two protocols to progress synchronously so that if one requires an action a, the other must provide a semantically compatible action $\overline{b}$. The mediator compensates for the syntactic differences between their actions by performing the necessary transformations, which is formalised as follows:

$$\begin{gathered}\mathsf{Mediator}_{\mathcal{O}}(P_1, P_2) = \|\mathsf{Map}(s_1, s_2) \text{ such that} \\ s_2 \in traces(P_2)\uparrow_{\mathcal{O}} P_1, s_1 \in traces(P_1)\uparrow_{\mathcal{O}} P_2 : s_2 \equiv_{\mathcal{O}} s_1\end{gathered}$$

In the IM case, we are able to produce the mediator for the MSNP and XMPP processes as illustrated in Figure 5. The mediator intercepts an instant message sent by an MSNP user and forwards it to the appropriate XMPP user. Similarly, each instant message sent by an XMPP user, is forwarded by the mediator to the corresponding MSNP user.

$$\boxed{\begin{aligned}\mathsf{Map}_1 &= (<InstantMessage, \{SenderID, RecepientID, Message\}, \emptyset > \\ &\quad \rightarrow <\overline{InstantMessage}, \{UserID, ConversationID, Message\}, \emptyset > \rightarrow \mathsf{END}). \\ \mathsf{Map}_2 &= (< InstantMessage, \{UserID, ConversationID, Message\}, \emptyset > \\ &\quad \rightarrow <\overline{InstantMessage}, \{SenderID, RecepientID, Message\}, \emptyset > \rightarrow \mathsf{END}). \\ \|\mathsf{Mediator} &= (\mathsf{Map}_1 \| \mathsf{Map}_2).\end{aligned}}$$

Fig. 5. OFSP specification of the Mediator between MSNP and XMPP

5 Implementation

In order to validate our approach, we have combined the LTSA[12] model checker with an OWL-based reasoner to achieve ontological refinement leading to the OLTSA tool (Figure 6-❶). LTSA is a free Java-based verification tool that automatically composes, analyses, graphically animates FSP processes and checks safety and liveness properties against them.

In the case where the processes match, a concrete mediator that implements the actual message translation is deployed atop of the Starlink framework [3], see Figure 6-❸. Starlink interprets the specification of mediators given in a domain-specific language called Message Translation Logic (MTL). An MTL specification describes a set of *assignment*s between message *field*s. The messages correspond to action names and the fields to the name of input/output data. Note that the OFSP description focuses on the ontological annotations and not the the actual name. Therefore, we refine the OFSP specification of the mediator so as to generate the associated MTL before deploying the mediator atop of Starlink, see Figure 6-❷.

[12] `http://www.doc.ic.ac.uk/ltsa/`

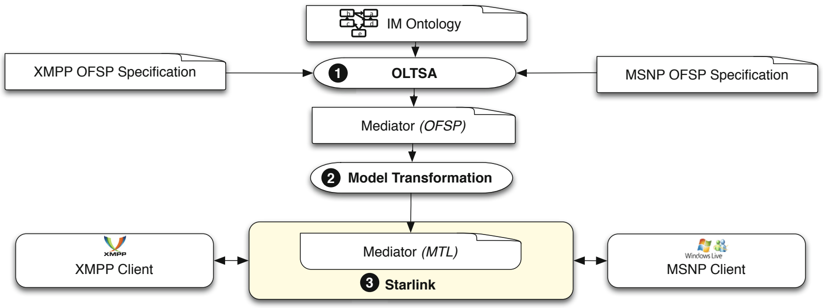

Fig. 6. Mediation Architecture

Let us consider the mapping Map_1 (see Figure 5), which transforms an XMPP input action to the associated MSNP action. Figure 7 shows a small fragment of the associated translation logic described in MTL and which corresponds to the assignment of the UserID field of the XMPP message ($\mathit{ReceivedInstantMessage}$) to the $\mathit{SenderID}$ field of of the MSNP message (SDG) with the mediator transiting from state $\mathit{XS1}$ to state $\mathit{MR1}$.

The tool, the IM ontology, and a video demonstration are available at `http://www-roc.inria.fr/arles/download/imInteroperability/`.

```
<translationlogic>
 <assignment>
  <field>
   <statelabel>MR1</statelabel><message>SDG</message>
   <xpath>/field/primitiveField[label='UserID']/value</xpath>
  </field>
  <field>
   <statelabel>XS1</statelabel><message>ReceivedInstantMessage</message>
   <xpath>/field/primitiveField[label='SenderID']/value</xpath>
  </field></assignment> ....
</translationlogic>
```

Fig. 7. Translation logic to map MSNP and XMPP instant messages

6 Assessment

In this section we first report a set of experiments we conducted to evaluate the effectiveness of our approach when applying it to the instant messaging case. Then, we discuss some of its quality properties.

6.1 Experimental Results

We have evaluated the time for translating one protocol to the other by the synthesised mediator and the effort required by the developer to enable mediation. We have hand-coded a mediator that makes MSNP, YMSG, XMPP interoperable in order to gauge the complexity of the mediation. We considered the

Windows Live Messenger for MSNP, Yahoo! Messenger for YMSG, and Pidgin[13] for XMPP. We run the OLTSA tool and the Starlink framework on a Mac computer with a 2,7 GHz processor and 8 GB of memory.

In the first experiment, we measured the time taken to translate from one protocol to another. We repeated the experiments 50 times and reported the mean time for each case in Table 1. The hand-coded mediator is approximately 3 times faster than the synthesised one. This is mainly due to the fact that the models are first interpreted then executed by Starlink at runtime whereas the hand-coded mediator is already compiled and hence more efficient.

In the second experiment, we measured the time for synthesising the mediator (see Table 2). One can note that action mapping is the most time consuming step as it necessitates ontology reasoning in order to infer semantically matching actions while the behavioural matching is performed is less than 1 ms. Nevertheless, this step needs to be performed once only and is definitely faster than hand-coding the mediator or even specifying it. Moreover, for each new version of one of the protocols, the hand-coded mediator has to be re-implemented and re-compiled, Starlink requires the specification of the translation logic to be re-specified whereas the automated synthesis requires only the specification of the protocol to be re-loaded.

Table 1. Translation time (ms)

	Hand-coded	Mediator atop Starlink
YMSG ↔ MSNP	22	69
MSNP ↔ XMPP	52	131
YMSG ↔ XMPP	44	126

Table 2. Time for Synthesis (ms)

	Act. mapping	Beh. match.
YMSG ↔ MSNP	306	<1
MSNP ↔ XMPP	252	<1
YMSG ↔ XMPP	244	<1

The third experiment measures the effort demanded from the developer to produce mediators between different IM applications. We calculate the number of Java code lines of the hand-coded mediator, the number of lines of DSL specification that need to be specified for Starlink and those needed to specify the individual applications for the automated synthesis.

Table 3. Development effort

	Hand-Coded	Starlink	Automated
YMSG ↔ MSNP	1172	258	96
MSNP ↔ XMPP	750	198	84
YMSG ↔ XMPP	945	168	76

The results are given in Table 3. One can notice that although Starlink reduces considerably (around 4 times) the lines of code that need to be written,

[13] `http://www.pidgin.im/`

the automated approach requires the OFSP specifications only and decreases this number drastically (around 10 times). This is mainly due to (i) the use of OFSP to model the interaction protocols, which introduces an ontology-based domain-specific language grounded in process algebra and especially targeted for concurrent systems. For example, the MSNP behaviour is described in Starlink using 30 XML lines and only 6 lines with our approach (ii) Further, the translation code need not be specified. More importantly, unlike the hand-coded or the Starlink versions where the developer is required to know both protocols and define the translation manually, the protocols are specified separately in the automated version. Thus, each IM provider can independently specify its own protocol. Finally, we are investigating within the CONNECT[14] project learning-based techniques to infer such a specification automatically [2].

To sum up, our automated approach to interoperability significantly reduces the programming effort and ensures the correctness of the translation while requiring a negligible time for synthesising the mediator and guaranteeing good performances at translation time.

6.2 Qualitative Assessment

In addition to the above-mentioned performances, our approach satisfies the following properties:

- *Correctness by construction.* The correctness of the mediation, i.e., the absence of deadlock and unspecified receptions [25], is guaranteed by construction. Indeed, if there is an exact match between P_1 and P_2 then the parallel composition $P_1 \| M \| P_2$ is deadlock free. Exact matching means that each trace of P_1 (P_2) has a corresponding semantically-matching trace in P_2 (P_1), which amounts to setting P_1 (P_2) as a safety property that needs to be verified by P_2 (P_1). This verification is performed by exhaustively exploring the state space. Note though that efficient model checkers use optimisation techniques to reduce the space if possible. The reduction techniques are even more efficient in the case of process algebra.
- *Formal yet tractable DSL specification.* OFSP introduces an ontology-based domain-specific language grounded in process algebra. Process algebra constitute a very expressive behavioural specification language for complex concurrent systems while ontologies are the model of choice to describe data semantics. Furthermore, standard modelling languages that developers are familiar with (e.g., BPEL or CDL) can be used to specify the interaction protocols and then automatically translate them to FSP using existing tools[15].
- *Dealing with encryption.* When encryption is enforced (e.g., Google Talk encrypts XMPP messages), the mediator cannot parse or modify these messages all the way between the initial sender and the ultimate receiver. Transparency cannot be ensured anymore. Instead, the user get involved and handles some of the translation tasks [23]. In the Google Talk case, the mediator

[14] `http://connect-forever.eu/`

[15] `http://www.doc.ic.ac.uk/ltsa/bpel4ws/`

uses a robot (bot) that the user adds to its contact list. The robot manages a set of commands, e.g., `IM <destinationID> <message>` to send a message `message` to user `destinationID`.

7 Related Work

The problem of mediating applications has been studied in different domains. Middleware solutions focus on providing abstraction and execution environments that enable interoperation by providing an abstract interface and exploiting reflection [9], by translating into a common intermediary protocol such as in the case of Enterprise Service Buses [16] or by proposing a domain-specific language to describe the translation logic and automatically generate the corresponding gateways [3]. However, these solutions require the developer to specify the translation to be made and hence to know both protocols in advance whereas in our approach, each protocol is independently specified and the translation is produced automatically. The Web Service Execution Environment (WSMX) performs the necessary translation on the basis of pre-defined mediation patterns. However, the composition of these patterns is not considered, and there is no guarantee that it will not lead to a deadlock. Vaculín *et al.* [22] devise a mediation approach for OWL-S processes. They first generate all requester paths, then find the appropriate mapping for each path by simulating the provider process. This approach deals only with client/server Web service interactions. It is not able to deal with the heterogeneity of instant messaging applications for example. Calvert and Lam [4] propose an approach to reason about the existence of a mediator by projecting both systems into a common sub-protocol. However, this common sub-protocol needs to be specified using an intuitive understanding of the protocols. In their seminal paper, Yellin and Strom [25] propose an algorithm for the automated synthesis of mediators based on predefined correspondences between messages. By considering the semantics of actions, we are able to infer the correspondences between messages automatically. Finally, Cavallaro *et al.* [5] also consider the semantics of data and relies on model checking to identify mapping scripts between interaction protocols automatically. However, they do not take into account the actual semantics of the operations. Moreover, they propose to perform the interface mapping beforehand so as to align the vocabulary of the processes, but many mappings may exist and should be considered during the generation of the mediator. Hence, even though there exists a significant amount of work to achieve interoperability, none of the existing approaches proposes to generate automatically mediators that are able to deal with both data and protocol mismatches.

8 Conclusion

Achieving interoperability between heterogeneous distributed applications without actually modifying their interfaces or behaviour is desirable and often necessary in today's pervasive systems. Mediators promote the seamless interconnection of distributed applications by performing the necessary translations between their messages and coordinating their behaviour. In this paper, we have presented a principled approach to the automated synthesis of mediators at runtime. We first infer mappings between application interfaces by reasoning about the semantics of their data and operations annotated using a domain-specific ontology. We then use these mappings to automatically synthesise a correct-by-construction mediator. This principled approach to generating mediators removes the need to develop *ad hoc* bridging solutions and fosters future-proof interoperability. We evaluated the approach using a case study involving heterogeneous instant messaging applications and showed that it can successfully ensure their interoperation.

Work in progress includes the definition of many-to-many operation mappings to manage a broader set of heterogeneous systems. We are also investigating the synthesis of mediators between more than a pair of networked applications. This is for example the case when IM conversations involve multiple users. Our work further integrates with complementary work ongoing within the Connect European project so as to develop a framework to support the interoperability lifecycle by using semantic technologies to synthesise mediators dynamically and ensure their evolution to respond efficiently to changes in the individual systems or in the ontology. A further direction is to consider improved modelling capabilities that take into account the probabilistic nature of systems and the uncertainties in the ontology. This would facilitate the construction of mediators where we have only partial knowledge about the system.

Acknowledgment. This work is carried out as part of the European Connect and EternalS projects.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press (2003)
2. Blair, G.S., Bennaceur, A., Georgantas, N., Grace, P., Issarny, V., Nundloll, V., Paolucci, M.: The role of ontologies in emergent middleware: Supporting interoperability in complex distributed systems. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 410–430. Springer, Heidelberg (2011)
3. Bromberg, Y.D., Grace, P., Réveillère, L.: Starlink: Runtime interoperability between heterogeneous middleware protocols. In: Proc. ICDCS (2011)
4. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Comm. (1990)
5. Cavallaro, L., Di Nitto, E., Pradella, M.: An Automatic Approach to Enable Replacement of Conversational Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 159–174. Springer, Heidelberg (2009)
6. Cimpian, E., Mocan, A.: WSMX process mediation based on choreographies. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 130–143. Springer, Heidelberg (2006)

7. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. (1994)
8. Franz, T., Staab, S.: SAM: Semantics aware instant messaging for the networked semantic desktop. In: Proc. International Sem. Web Conf. Workshops (2005)
9. Grace, P., Blair, G.S., Samuel, S.C.: ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability. In: Meersman, R., Schmidt, D.C. (eds.) CoopIS/DOA/ODBASE 2003. LNCS, vol. 2888, pp. 1170–1187. Springer, Heidelberg (2003)
10. Heiler, S.: Semantic interoperability. ACM Surv. (1995)
11. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
12. Issarny, V., Bennaceur, A., Bromberg, Y.-D.: Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 217–255. Springer, Heidelberg (2011)
13. Keller, R.M.: Formal verification of parallel programs. Commun. ACM (1976)
14. Magee, J., Kramer, J.: Concurrency: State models and Java prog. Wiley (2006)
15. Maier, M.W.: Integrated modeling: A unified approach to system engineering. Journal of Syst. and Softw. (1996)
16. Menge, F.: Enterprise Service Bus. In: Proc. Free and Open Source Soft. Conf. (2007)
17. Ben Mokhtar, S., Kaul, A., Georgantas, N., Issarny, V.: Efficient Semantic Service Discovery in Pervasive Computing Environments. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, pp. 240–259. Springer, Heidelberg (2006)
18. Motoyama, M.A., Varghese, G.: CrossTalk: scalably interconnecting instant messaging networks. In: Proc. ACM Workshop on Online Social Networks (2009)
19. Nielsen: Games Dominate America's Growing Appetite for Mobile Apps (2010)
20. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: WICSA/ECSA (2009)
21. The Radicati Group: Instant Messaging Market 10-14 (2010)
22. Vaculín, R., Neruda, R., Sycara, K.P.: The process mediation framework for semantic web services. Journal of Agent-Oriented Softw. Eng. (2009)
23. Vassilakis, C., Kareliotis, C.: A framework for adaptation in secure web services. In: Medi. Conf. on Info. Syst. (2009)
24. Wiederhold, G.: Mediators in the architecture of future info. syst. Computer (1992)
25. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Prog. Lang. Syst. (1997)