Process Mediation: Requirements, Experiences and Challenges

Roman Vaculín
Institute of Computer Science, Academy of Sciences of the Czech Republic

Roman Neruda
Institute of Computer Science, Academy of Sciences of the Czech Republic

Katia Sycara
The Robotics Institute, Carnegie Mellon University

ABSTRACT
In this chapter we address the problem of interoperability of incompatible business processes consisting of semantically annotated Web services. We propose a (semi-) automated solution, called process mediation, that analyzes the potentially incompatible process models of service requesters and service providers, identifies all incompatibilities, and automatically synthesizes reconciliation plans that can be used at runtime for resolving the identified incompatibilities. Compared to the previous research, the main novelty of our work is its strong focus on dynamic scenarios common for process-based service-oriented systems operating in open or semi-open dynamic environments. The chapter provides an overview of our recent body of work that covers techniques for automated incompatibilities detection and resolution, service discovery in the process mediation context, techniques for monitoring, and techniques for fault handling and recovery. We discuss the lessons we have learned and we also provide an outlook for future research directions.

INTRODUCTION
Seamless interoperability of existing services and business process and, eventually, the ability of processes to adapt is crucial, especially with the advent of cloud computing, mobile computing, and the Software as a Service paradigm, where services, and Web services in particular, play the key integration role. As the application requirements, goals and needs change very rapidly in today business environments along with the rapidly changing operating environments, the need for automatic process adaptation, evolution and reconfiguration arises in all the mentioned areas.

Web services standards have a goal to enable and facilitate interoperability of heterogeneous applications and business processes. As such, the web services standards alone focus on facilitating interoperability in static sense, while the dynamic aspects of interoperability resulting for example from changes and evolutions of the existing processes, systems and their implementations are not in the center of attention. In situations when business needs and requirements change, business processes might need to get reconfigured, modified or completely replaced in order to meet the new needs and requirements. Subsequently, many existing business processes need to be updated as well only for the sake of maintaining interoperability with the
reconfigured processes. Naturally, such changes are rather laborious and costly. Similarly, in more dynamic scenarios where the service requesters and providers can be dynamically discovered by employing service search engines (registries), such dynamically matched processes will not typically be completely compatible and thus will not be able to interoperate directly. The main novelty the work presented in this chapter is in the focus on dynamic scenarios common for service-oriented process-based systems operating in open or semi-open dynamic environments of current corporations and of the Internet.

To address interoperability in dynamic situations there is a need for solutions which will enable process interoperability in an automated or a semi-automated fashion. Natural requirements are that such solution should achieve interoperability without needing to modify the implementation and interfaces of the to-be-integrated process, and it should allow interoperability to be achieved dynamically even at runtime. A possible solution is to apply a process mediation component that serves as a “mediator” between service requesters and providers. Such process mediator needs to be able to identify and resolve possible incompatibilities (e.g., by generating appropriate mappings between the incompatible processes) while making minimal assumptions about implementation details of service providers and requesters.

Recently, interoperability of Web services (WS) and of Semantic Web services (SWS) (McIlraith et al., 2001) has gained attention primarily because the existing open Web services standards provide a good shared integration basis, and the emerging SWS standards promote the explicit shared semantics necessary for achieving interoperability. Speaking of the current WS standards, the two probably most prominent standards are the Web Services Description Language (WSDL (Christensen et al., 2001)), and the Business Process Execution Language (WS-BPEL (Alves et al., 2007)). WSDL allows to declaratively describe operations, the format of messages, and the data structures that are used to communicate with a Web service. WS-BPEL allows to combine several Web services within a formally defined process model and to define an interaction protocol and possible control flows. However, none of the current standards goes beyond the syntactic descriptions.

The Semantic Web services languages such as SAWSDL (Farrell and Lausen, 2007), OWL-S (Martin et al., 2004) and WSMO (Roman et al., 2005b) enrich syntactic specifications of the current Web services standards with rich, machine processable semantic annotations to facilitate flexible dynamic Web services discovery, composition and invocation (Sycara et al., 2004). Research in Semantic Web services has its origins in various fields of artificial intelligence, such as knowledge representation, description logics reasoning, and automated planning. The AI methods proved to be very promising with respect to achieving a higher level of automation in interoperability of Web services.

The major problem of existing research is that typically rather static integration scenarios are considered which are usual for closed environments. Also, the existing standards do not provide reasoning methods for achieving interoperability of providers and requesters as application requirements change. It turns out that

1. the existing approaches are not suitable for achieving interoperability in dynamic open environments

2. in addition to developing reasoning methods for interoperability of providers and requesters, many problem areas need to be considered (such as data mediation, fault handling, monitoring, recovery, service discovery), in order to achieve the interoperability in dynamic open environments

3. the above mentioned problems cannot be addressed in isolation since they are tightly related to each other.
In the light of these observations, we will present an Abstract Process Mediation Framework that identifies the key functional areas that need to be addressed by process mediation components. The key functionalities include process mediation, data mediation, service invocation, monitoring, recovery, and discovery of external services. We will explain why all such functionalities are critical, and why it is impossible to deal with each of these area in a complete isolation. In essence, our research indicates that while some of the mentioned areas are seemingly very different and independent, if treated in isolation, various major problems implicated by process mediation cannot be resolved.

The chapter is structured as follows. We start with a brief review of related work, followed by an analysis of the interoperability problems and scenarios usual in various operating environments. We introduce the process mediation problem by presenting a set of easy to understand examples, and we also discuss the major challenges of process mediation. Next, we present our technical solution consisting of techniques for automated incompatibilities detection and resolution, service discovery in the process mediation context, techniques for monitoring, and techniques for recovery. Finally, we discuss the results, lessons learned and key challenges that need to be addressed in future research.

RELATED WORK

The problem of achieving interoperability of software systems and distributed systems in particular is as old as distributed systems themselves. Various approaches have been explored in areas such as protocols synthesis (Zafiropulo et al., 1980; Probert and Saleh, 1991), software adapters synthesis (Thatté, 1994; Yellin and Strom, 1997) and component-based software engineering (Plasil and Visnovsky, 2002). Web services present the most recent attempt of research and industry communities to address the hard problems of integration and interoperability in the context of the Service-Oriented Architecture (SOA).

An interesting work has been done in the context of the Semantic Web Services Challenge. The goal of this initiative is an automation of web services mediation, choreography and discovery. To mention a few examples, Cimpian and Mocan (2005) solve runtime mediation between two WSMO based processes. Besides structural transformations (e.g., change of message order) also data mediators can be plugged into the mediation process, however, recovery and discovery are not addressed at all. In a similar vein, Vitvar et al. (2007) present a semantic web services framework called WSMX based also on WSMO for integration of heterogeneous B2B services. Brambilla et al. (2006) apply a model-driven approach based on WebML language to design a mediator. Mediator is designed in the high-level modeling language which supports semi-automatic elicitation of semantic descriptions in WSMO.

The most of the above mentioned efforts have focused on enterprise systems, where the problems of integration and interoperability rank among the most critical. Another body of work was done in the agent community. Aberg et al. (2005) describe an agent called sButler for mediation between organizations’ workflows and Semantic Web services. The mediation is more similar to the brokering, i.e., having a query or requirement specification, the sButler tries to discover services that can satisfy the query. The requester’s process model is not taken into considerations. The OWL-S broker (Paolucci et al., 2005) also assumes that the requester formulates its request as a query which is used to find appropriate providers and to translate between the requester and providers. Cabral et al. (2006) describe the IRS-III broker system based on the WSMO methodology. IRS-III requesters formulate their requests as goal instances and the broker mediates only with providers given their choreographies (explicit mediation services are used for mediation).
INTEROPERABILITY IN OPEN ENVIRONMENTS

Service-oriented business processes can be usually characterized as open in two senses. First, the most of the existing WS standards are open by their nature. Second, the WS processes are very often expected to operate in dynamic open environments with the integration scenarios often crossing the organizational boundaries. Such openness presents new challenges to interoperability.

Consider, for example, the public Web services search engine Seekda.com. Anybody is allowed to register his or her Web service into this registry, and, similarly, anybody can search for services that match some specific query. At the time of writing this chapter, the registry contained around 8000 service providers, who have registered around 29000 publicly available Web services. For example, when queried for the services somehow related to route planning (we used the “route planning” query string), the registry search engine found more than 60 Web services. At the first sight, 60 is not a very high number. However, the problem is that almost each service was developed by different provider with very different assumptions in mind. As an outcome very different specifications of services are provided at various levels of details, service interfaces are implemented very differently using different data structures, vocabularies, and different assumptions are expected about processes into which such services could be integrated. The results returned by such a discovery engine or a similar one are quite hard to interpret and integrate for a human programmer, not to speak about semi-automated integration into existing dynamic processes and workflows. Also, one should not be misguided by thinking that dynamic discovery and integration is just some artificial problem which is far away from a real life. On the contrary, problems of dynamic integration are becoming crucial for success of promising areas such as dynamic cloud computing or mobile computing. Our belief is that to enable a higher level of automation in processes interoperability, more machine processable semantics is needed, and more advanced methods for adaptive and interoperable processes have to be developed.

Shall We Care about More Semantics?

As demonstrated by the example with discovery of route planning services, one major issue of integration of services provided by independent vendors is the missing notion of the shared semantics. Essentially, this belief has motivated research efforts in the area of Semantic Web services (McIlraith et al., 2001). While some researchers and practitioners claim that the Semantic Web services effort did not find good use in practice, still there is an understanding shared by many researches that enforceable semantic descriptions is what matters, as recently expressed for example by Petrie (2009) who claims that “web services with semantic and enforceable descriptions that would enable dynamic interoperability over the Internet, would be more practical.”

Importantly, also many businesses believe that semantics is of an interest when it comes to the quest for interoperability and a higher value of services provided by the existing IT systems. According to the Forrester report from October 2009 while the “most application development professionals view it (semantic technology) with skepticism or outright distrust, believing that the dream of semantic technology is impractical in a time of stretched budgets and delayed technology road maps” there is also the following belief expressed:

“Semantic technology can genuinely enhance your organization’s ability to find, integrate, and store information. The time has come for application development professionals to build a pragmatic strategy for slowly integrating this new way of thinking into those projects where information provides the greatest value.”

Similarly, according to the Forrester report from March 2008, “… Standardized service semantics will become crucial for agility”. Furthermore, the same report claims that “the
business demands more agility ...“ and that “... standardized semantics can help next-generation architecture deliver”.

Thus, an enforceable, shared, machine processable semantics presents the corner stone for more interoperable, adaptive and agile business processes. However, as demonstrated by the Semantic Web Services Challenge, only shared semantics is not enough. The second part that requires the same attention is the techniques and methodologies for achieving more adaptive, interoperable processes.

**Bottom-up vs. Top-down Approach to Interoperability**

An important, realistic assumption about business processes operating in open environments is that by default such processes might not be interoperable directly due to existing incompatibilities. Subsequently some effort has to be invested in order to *make processes to interoperate*. Either some process has to be *adapted*, for example manually, to make it interoperate with some other process or a service, or the process has to be *adaptive*, so that it can adapt itself to its environments (i.e., to other processes). Speaking of achieving processes interoperability and/or adaptivity, two perspectives are possible: the bottom-up approach or the top-down approach. The *bottom-up approach* to adaptation can be characterized as an *adaptation of existing models and systems in order to achieve their interoperability*, while the top-down approach presents an alternative attitude to building adaptive systems which can be characterized as *building adaptive systems by design*. We focus on the bottom-up approach, but we briefly discuss the top-down approaches in the last section since it presents a very promising direction of future research.

In the bottom-up approach the goal is to achieve interoperability of systems and business process that already exist, and that were defined and developed by employing the currently available frameworks and languages for business processes specification. Typically, in the platform independent model business processes are modeled as so called abstract processes – e.g., in formalisms such as BPMN (Business Process Modeling Notation) (White, 2004; OMG, 2009), EPC (Event-driven process chain) (Keller et al., 1992), etc. In the platform specific model, abstract business processes are further detailed in the methodology specific for the given platform. These days, platform specific processes are usually modeled in the context of service-oriented architectures, and in particular Web services modeling languages, such as WS-BPEL (Alves et al., 2007), play a prominent role. Finally, when it comes to implementation, the platform specific models are grounded in systems, which often have form of Web services (e.g., presented as WSDL / SOAP services in the context of corporations, or REST-full services in the context of open Internet).

The existing business processes specification methodologies typically define complex processes and their logic in terms of control flows and information flows expressed by means of traditional constructs such as graph structures (Best and Fernandez, 1988; Leymann and Roller, 1999) or process algebras (Milner, 1999). While these traditional constructs proved to work very well for definition of static processes on all mentioned levels, it turns out that when the process definition changes or evolves often, traditional approaches are not suitable, primarily for their lack of flexibility. In particular, the workflow specifications defined in terms of graph based or process algebra based formalisms do not allow easy modifications, both at design time and at runtime, for the following reasons:

- lack of declarative information defined on the semantically appropriate level
- over-constrained specification of process flow and control which is not necessarily exactly in line with business goals, requirements and service level agreements
• a rather static nature of definitions which does not allow the processes to respond to unpredicted situations or to be evolved easily – methods such as dynamic service discovery, composition or mediation are hard to apply with existing process specifications

• a low or missing principled integration of processes with other modeling perspectives, in particular with conceptual data modeling – changes in business processes and in data modeling have to be usually performed independently, which leads to many overheads

• a low integration of process with the underlying data storage layer

These problems are mostly inherited also by existing Semantic Web services languages. In order to allow interoperability and adaptation of such existing systems and to overcome some of the mentioned shortcomings, the solution is to develop sophisticated techniques, typically based on the notion of software adapters and mediators. The focus of our work is essentially in the area of developing such methods, as we describe in subsequent sections.

Interoperability Scenarios

We focus primarily on the bottom-up approach to process adaptation with the goal of achieving interoperability of existing potentially partially incompatible processes. Figure 1 shows two generic interoperability scenarios that are of interest for us in this chapter. Specifically, Figure 1a considers a situation in which complex (composite) process models of a service requester and a service provider are designed to interoperate with each other, i.e., these two processes can interoperate since they were carefully designed to do so. This figure presents a very usual situation in which both the requester and the provider adhere to some relatively fixed process models. Symbolically, requester’s and provider’s process models are shown with small boxes standing for individual services (service calls, operations, atomic processes). We assume that both the requester’s and the provider’s process models are expressed declaratively.

The problems of the scenario with built-in interoperability start in the moment when some part of any participating process model changes, for example as an outcome of modified business needs. In cases like this, when processes may get reconfigured or additional process components and services can be added (as illustrated by big arrows in Figure 1a), the other process and its components need to be modified accordingly in order to become interoperable with the new updated process model. This can be accomplished by making manual changes to the existing workflow components and programming the new components in such a way as to maintain built-in interoperability. This is a rather laborious and inefficient process since it must be repeated every time workflow reconfiguration is needed.

Figure 1: Interoperability Scenarios
Moreover, since the Internet gives the opportunity to dynamically discover service providers, it is often not a priori known which service provider may best fit the application workflow changing needs. Figure 1b shows the scenario in which the service requester first dynamically discovers an appropriate service provider by using some discovery service, and later interoperates with such discovered provider. In the situation of dynamically discovered service providers, it is rather unrealistic to assume that process models of the service provider and the service requester will be completely compatible, simply because these processes might have been developed by independent vendors. Employing discovery mechanisms for finding suitable service providers is useful in various contexts where the very loose binding between requesters and providers is necessary, such as in mobile computing, as symbolically showed in Figure 1b.

**Interoperability With Process Mediation**

In both of the discussed scenarios we want to achieve interoperability of service requesters and providers that might have partially incompatible process models. An ideal solution should

1. achieve interoperability without actually modifying implementation and interfaces of participating process models,

2. make minimal assumptions about knowledge of service requester and service provider interfaces, implementation, etc.,

3. minimize the amount of manual work,

4. be flexible and should allow various concerns to be taken into account, such as security, privacy, scalability, etc.

A convenient solution that satisfies these requirements is to apply a *process mediation component*, which acts as a middle agent intercepting communication between the requester and the provider, and which generates appropriate mappings between different processes and potentially resolves all incompatibilities. Conceptually, this solution is based on a standard notion of software adapters known from various software engineering areas, and it is similar to the idea of brokering agents known from the AI agent research.

Figure 2 extends the two discussed interoperability scenarios with the process mediation component (or agent). Namely, Figure 2a employs the process mediation agent to the scenario of direct / built-in interoperability, whereas Figure 2b presents the case where the process mediation component is used to achieve interoperability in the scenario of dynamic discovery of a service provider.

![Figure 2: Interoperability with mediator in different scenarios](image-url)
In the context of process mediation the environment within which mediation takes place plays a significant role. We have identified visibility of involved process models as an important factor with a direct impact on design of effective process mediation architectures and algorithms. We propose mediation techniques for two major cases:

1. Complete visibility scenario: The mediation component has complete visibility of both the process models of the service provider and the service requester. This scenario, showed in Figure 2a, represents a typical situation for closed or semi-open environments of intranet or B2B applications.

2. Asymmetric scenario: The mediation component has visibility only of the process model of the service provider but not the service requester. This scenario, showed in Figure 2b, represents a mediation problem usual in open environments, e.g. in mobile computing, where clients are concerned about privacy and therefore do not wish to disclose their complete process models.

APPROACH ESSENTIALS

We concluded the previous section with an observation that when service requesters and providers use fixed, incompatible communication protocols interoperability can be achieved by applying a mediation component which resolves all incompatibilities, generates appropriate mappings between different processes and translates messages exchanged during runtime. In this section we will study how such mediation components can be designed and synthesized in a (semi-)automatic manner, given we know the specification of the process models of the requester and the provider.

Our approach is based on the following high-level principles:

• Conceptual abstraction from underlying systems: Although we primarily focus on the problem of interoperability of WS-based process models, in our approach we abstract from the specific WS- standards and/or implementation details. We specify the processes by means of Semantic Web languages (such as OWL) and Semantic Web services standards (OWL-S) which are implementation neutral. This way, we gain a unified view over various possible technologies, such as Web services, software agents, or distributed sensors.

• Formal representation of process related concepts: Rather than using the datatypes defined directly by services, we employ the semantic annotations, which present a conceptually higher perspective, and allow a more flexible reasoning about interoperability related problems, such as incompatibilities identification and resolution. In addition to specification of data exchanged by the services, we also employ the explicitly defined preconditions and effects of the services, which further extends reasoning options.

• Application of AI techniques: In order to analyze possible incompatibilities and to resolve them, we propose the use of AI techniques, such as automated planning, heuristic search, constraint optimization and satisfaction, and reasoning. In particular, flexible reasoning and heuristic planning allow us to find not only admissible solutions, but also high quality plans with minimal involvement of human programmers.

Our solution combines the AI planning to generate appropriate mappings between processes, and semantic reasoning (specifically, matching of services based on their semantically annotated specifications) with the discovery of appropriate external services (serving, e.g., as data mediators) and suitable recovery techniques.
We use the OWL-S ontology (Martin et al., 2004) for definition of process models because it provides support for description of individual services and also explicit constructs with clear semantics for describing process models. In OWL-S, the elementary unit of process models is an atomic process, which represents one indivisible operation that the client can perform by sending a particular message to the service and receiving a corresponding response. Processes are specified by means of their inputs, outputs, preconditions, and effects (IOPEs). Types of inputs and outputs are defined as concepts in domain ontologies (expressed in OWL) or as simple XSD data-types. Processes can be combined into composite processes by using control constructs such as sequence, any-order, if-then-else, split, loop, etc.

Process Mediation Problem Defined

Before describing the solution we have developed, let us first consider the notion of interoperability of processes in more detail. In the following text we use $R$ to stand for the requester’s process model and $P$ for the provider’s process model.

In order to introduce the problem of process mediation, we use the notion of the provider’s ability to satisfy the requester. We write $\text{satisfied}(R,P)$ if the provider’s process model $P$ is able to satisfy the requester’s process model $R$. Intuitively, $\text{satisfied}(R,P)$ holds if for each requester’s service call / request the provider can receive such request and return the appropriate outputs and effects required by the requester. Specifically, each requester’s operation (process) must be satisfied by the provider, i.e., the provider must produce required outputs, produce required effects, while consuming only available data (inputs), and while all preconditions of the provider are satisfied. Also we require that both the requester and provider execute their process models according to their execution semantics (Ankolekar et al., 2002) and that both initiated processes finish their execution in a proper final state.

**Figure 3:** Example of atomic processes interoperability: requester $R$ and provider $P$ can interoperate directly, i.e., $R$ can achieve its “goals” by “using/invoking” $P$

**Direct interoperability without process mediation:** Consider the example in Figure 3 which presents a simple integration scenario of a requester and a provider from the authentication domain. In this scenario, the client $R$ wants to perform login authentication by providing a user name and a password, and it wants to use the authentication service of the provider $P$. As showed in Figure 3, the client process $R$ consists only of a single atomic process called Login Request, which has two input parameters, the ?$\text{userID}$ with the User type, and the ?$\text{password}$ with the PWD type. The client further expects that after performing the login request a boolean output value ?$\text{logResult}$ will be returned which indicates whether the user name and password combination is correct or not. Also, the requester $R$ expects that after successful invocation of the authentication service the user ?$\text{userID}$ will be logged in, which is indicated by the requested
effect \textit{LoggedIn(?userID)}. Similarly, the provider’s process model \( P \) also consists of a single atomic process, called \textit{Login Provider}, which has a very similar structure of required inputs and produced outputs and effects. Specifically, the type definition and the number of inputs and outputs of the provider \( P \) exactly match those of the requester. This means that requester \( R \) can achieve its goals (in this case getting all requested outputs and effects) by directly “invoking” the provider \( P \). \( R \) simply sends values of its available inputs to the provider \( P \) which is able to process them, and the provider \( P \) responds by producing the \textit{?logResult} output and the corresponding effects (communication between \( R \) and \( P \) is symbolized by dashed arrows). Thus, in this case the predicate \( \textit{satisfied}(R,P) \) is true, since there are no incompatibilities between \( R \) and \( P \), and therefore there is no need for process mediation between \( R \) and \( P \).

**Interoperability of complex processes with process mediation:** Next, we need to consider the situation when there exist mismatches between \( R \) and \( P \) that do not allow \( \textit{satisfied}(R,P) \) to be true. Figure 4 depicts a somewhat more elaborate authentication scenario in which the requester \( R \) is exactly the same as in the previous example, but the provider’s process \( P \) consists of a sequence of two atomic processes, namely the \textit{LoginStep1} and \textit{LoginStep2}. This composite process \( P \) presents a two step authentication typical in many applications, such as in the Internet banking, where the service first validates a user ID in the first step (the \textit{?user} input parameter of \textit{LoginStep1}) and if this step succeeds the password is validated in the next step (the \textit{?pwd} input of \textit{LoginStep2}). The problem is that the requester \( R \) does not expect the authentication to be a two-step processes. Furthermore, there are some more discrepancies, such as the \textit{?sessionID} parameter used by the provider as a temporary variable and as a link between the first and the second step of its process model. For requester \( R \) the \textit{?sessionID} parameter is simply redundant and it presents another incompatibility between \( R \) and \( P \).

![Figure 4](image-url)

Figure 4: Example of complex processes interoperability: \( R \) can be satisfied by \( P \) with help of \( M \)

In order to achieve interoperability in this case, the process mediation component (labeled as \( M \)) can be used to resolve the incompatibilities as follows:

- \( M \) consumes the inputs provided by the requester \( R \) (the values of \textit{?userID} and \textit{?password} are sent by \( R \) as a single message)

- \( M \) splits the message and first invokes \textit{LoginStep1} of \( P \) by sending the value of \textit{?userID} and
by receiving the value of the \textit{sessionID} output

- next $M$ invokes \textit{LoginStep2} of $P$ by sending the \textit{password} and forwarding the \textit{sessionID}, and it receives the value of the \textit{logResult} output

- finally, $M$ forwards the value of the \textit{logResult} returned by $P$ to the requester $R$

As the example demonstrates, one can see the process mediator as a process model $M$ which has the goal of translating incompatibilities between $R$ and $P$. Thus, given the existence of mismatches between $R$ and $P$, the problem of process mediation can be seen as finding the process model $M$ of the mediator that translates the mismatches and realizes all the mappings between $R$ and $P$. The process $M$ has to be able to satisfy the requester $R$ by employing the services provided by $P$.

**Process mediation with help of external services:** While the previous example demonstrated some of the problems that might make processes incompatible because of their different structure, the situation might be much more tricky because of data incompatibilities (e.g., incompatible datatypes) or because some required information might be missing. The question is what the mediator component can do in such cases. In our approach we propose that the mediator can try to take advantage of external services that might serve for example as conversions between different formats or as translations capable of providing the missing pieces of data.

**Figure 5:** Example of mediation with external services: $R$ can be satisfied by $P$ with help of $M$ using translation services from $T$: \textit{satisfied}(R,P,M,T) services from $T$ can serve as data mediators, to get missing data or effects

Consider the example presented in Figure 5, in which the requester is a simple airlines client that wants to find a list of available flights between two places specified in the form of city codes (such as JFK or LAX), whereas the provider service expects the locations to be specified as the name of the city (e.g., Los Angeles) combined with a state in which this city is located (e.g., California for Los Angeles). These two partners cannot interoperate without the help of intermediary translations. However, let us assume that we have two external services available that can be used to provide translations between the city code and the city (the \textit{CodeToCity}}
service), and between the city and the state in which the city is located (the CityToState service). Given these two services can be used by the process mediator, it is possible to construct the process model \( M \) of the process mediator that takes care of the necessary translations, as symbolically showed in Figure 5.

As the example demonstrates, in this case the requester’s process \( R \) can be satisfied by \( P \) with help of \( M \) that is using translation services from the set \( T \). Services from \( T \) can serve as data mediators or translations, or can be used to get missing pieces of data or effects.

**Challenges of the Process Mediation**

Although we presented rather simple examples to demonstrate the substance of the process mediation problem, it is important to realize that typical interoperability problems are far more complex and creating a mediator component is a very challenging task. Here we present some of the major challenges which make the problem very hard.

**Process models are complex:** Usually, in a single instance of the process mediation problem both process models are complex, i.e., consist of several atomic processes arranged by means of various control constructs such as loops, branching, etc. This fact implies that the analysis of process models and the runtime mediation might be rather complicated, since the operational semantics of both all participating process models has to be respected.

**Various types of incompatibilities:** A usual instance of the process mediation problem typically combines several types of incompatibilities which all have to be resolved by the mediation component. Therefore creating a mediator component is very challenging since this component must be able to identify and resolve various types of incompatibilities on the

1. *data level*, e.g., different representation of exchanged data elements,

2. *service level*, e.g., some information required by the provider is not provided by the requester, and

3. *protocol level*, e.g., different control flows (different control constructs are used to define the process model logic).

**Incomplete knowledge of the environment:** In dynamic open environments an incomplete knowledge of the environment has to be assumed. For example, the set of transformation services \( T \), as demonstrated in example in Figure 5, might not be known to the process mediation component in advance. As an outcome, mechanisms such as service discovery need to be incorporated in process mediation solutions to be able to operate in open environments.

**Possibility of failures:** The mediation component has to be able to perform translations at runtime and also be able to respond to possible unexpected situations and failures, such as failures in communication.

**Abstract Process Mediation Framework**

In our recent body of work (Vaculín and Sycara, 2007b; Vaculín et al., 2008b, 2009; Vaculín and Sycara, 2009, 2008; Vaculín et al., 2008d,c), we have developed techniques that address the problem of automatic mediation of process models. Our solutions can be framed by the *abstract process mediation framework* (APMF). The goal of the APMF is a clear identification and separation of critical functional areas which need to be addressed by process mediation components in order to effectively solve the process mediation problem.

The three key functionalities are *process mediation*, *data mediation* and *service invocation*. 
The process mediation layer, realized by process mediators, is responsible for resolving service level and protocol level incompatibilities. The data mediation layer, realized by data mediators, is responsible for resolving data level mismatches. Typically, when trying to achieve interoperability, process mediators and data mediators are closely related. A natural approach is to use data mediators within the process mediation component to resolve data mismatches that were identified during the process mediation. The service invocation layer is responsible for interactions with actual Web services, which include the services of the requester, provider and possibly other external services.

To address runtime incompatibilities and possible service failures, mediation components make use of monitoring and recovery functionalities. These functionalities intercept the above mentioned key functional areas because monitoring and recovery are intertwined with the key mediation functionalities and both need to be performed on all different levels. Finally, in dynamic environments discovery of external services is closely related to the process mediation since external services (e.g. a translation service between inches and meters) might need to be discovered which are capable of delivering information for resolving mismatches identified between the requester and the provider.

In this chapter we discuss process mediation techniques, methods for discovery of external services suitable in the context of process mediation, and semantic monitoring, fault handling and error recovery, which are a necessary part of mediation components execution infrastructure. We do not address the problem of service invocation because it has already been successfully solved for classical Web services and for Semantic Web services as well (Paolucci et al., 2003a). We also do not address the data mediation problem in detail. We assume that data mediators have the form of external services which can be discovered and used by the process mediation component. In the following sections we introduce our proposed solution for each area covered.

**PROCESS MEDIATION ALGORITHM OVERVIEW**

The process mediation layer is primarily responsible for resolving service level and protocol level mismatches. The process mediation layer has to address three problems:

1. identify possible incompatibilities and information gaps between the process models that impede the mediation,

2. find mappings between the two processes, and

3. provide suitable mechanisms for runtime mediation and translations.

Various approaches can be applied to these problems ranging from manual to fully automatic techniques. Our goal is to facilitate an automation of these processes.

**Mechanisms for Process Mediation in Different Visibility Settings**

As we have already briefly discussed, visibility of participating process models has a significant impact on effective process mediation architectures and methods that can be employed in process mediation. We distinguish two possible degrees of visibility for a given process model:

1. The white box visibility means that the mediation component can see the complete process model, including all relevant operation definitions (atomic processes) and the complete interaction protocol (control and data flows).

2. The black box visibility means that the mediation component can see only definitions of available operations, while the process workflow and the interaction protocol are not
disclosed. Such a situation is motivated either by privacy concerns or by the fact that no formal specification of the interaction protocol exists.

We focus on 2 scenarios, namely the complete visibility scenario (Figure 2a) in which the mediation component has complete visibility of the process models of the service provider and the service requester, and on the asymmetric scenario (Figure 2b), in which the mediation process has visibility only of the process model of the service provider but not the service requester.

We distinguish two basic modes of process mediation: an offline analysis, in which some computations can be performed before runtime mediation starts, and a reactive mediation, in which all computations need to be performed strictly during runtime.

The offline analysis can be employed only when participating process models are known before the actual runtime interactions start. During the offline analysis, possible mismatches between the process models can be identified and corresponding mappings for bridging these mismatches can be computed either fully automatically or with a human assistance.

Reactive mediation addresses situations when offline analysis is not possible. Reactive mediation is suitable especially in open environments where cooperating partners are discovered at runtime. Since all computations need to be performed at runtime, there might be rather strict time constraints imposed by the requester’s expectations of a timely response. As an outcome of time constrained reasoning combined with incomplete or partial information about participating process models, in some cases imperfect decisions might be made. Therefore advanced recovery mechanisms need to be incorporated to enable recovery from such failures.

**Solution: Simulate + Plan to Compose + Discover**

In the following paragraphs we detail the technical solution for process mediation we have developed for the complete visibility (symmetric) scenario and for the partial visibility (asymmetric) scenario. Our solution consist of the following four principled techniques.

**I. Parallel simulation of both process models:** The idea of parallel simulation of process models is similar to the notion of product of automata which is often used for checking the partial compatibility and full compatibility of protocols (Benatallah et al., 2006). The major difference of our work is that instead of performing detection of weather two protocols are compatible, we assume that there might be incompatibilities that need to be detected, and a resolution of the identified incompatibilities should be proposed if possible. In the complete visibility scenario, our solution is based on an off-line analysis of possible execution sequences of the requester, while in the reactive case the analysis is constrained to the current atomic process (request) of the requester. As an outcome of this step all incompatibilities between the requester’s and the provider’s process models are detected.

**II. Reconciliation planning to find mappings:** The aim of reconciliation planning is to propose a solution for resolving the detected incompatibilities. We use the term reconciliation plan for a set of mappings / plans that resolve a detected incompatibility.

For example, the process model M presented in Figure 5 demonstrates a reconciliation plan consisting of a composition of external services that together resolve incompatibilities between the SearchFlight request and the SearchFlightUA provider.

Reconciliation planning algorithm (for technical details see (Vaculín et al., 2009)) finds a reconciliation plan that resolves one identified incompatibility. The reconciliation plan is a composition of primitive translations, such as external translation services and internal mediator
actions for receiving data and sending data. The reconciliation planner is given the following inputs:

- The provider’s and the requester’s process model
- Detected incompatibility in the form of
  - missing pieces of data and effects that need to be achieved (the goals of planning)
  - available data that can be used
  - the execution state of both the requester and the provider, which is the result of the simulated execution from the previous stage
- Set of available external translations / services, that can be used by the planner to produce missing pieces of data and to achieve missing effects

Reconciliation planning is based on the forward chaining composition algorithm using the $A^*$ algorithm as its core. The basic $A^*$ algorithm is extended in several directions. First, semantic reasoning is employed for matching the planning goals with known services based on their semantically annotated specifications. We employ ontologies using expressive DLs (OWL-DL) for semantic annotations and the Pellet reasoner to support working with ontologies. Second, the planning algorithm is designed so as to respects operational semantics of involved process models. Finally, the algorithm employs ranking of plans according to their length and quality, which is based on the degree of semantic match between the goals and the individual services that are part of the plan.

**III. Discover new data mediators and re-plan:** When no reconciliation plan can be found to resolve some identified incompatibility, process mediation layer uses the discovery layer in order to find appropriate external services (serving, e.g., as new data mediators). The incompatibility that cannot be resolved is reformulated into a query to the discovery service. Here the assumption is that the services registry or the discovery service has a more complete information about what external services might be provided by various providers. Assuming the discovery service finds some services that satisfy the query, the newly discovered services are added to the reconciliation planner, and the planner is used again to try to resolve the original incompatibility with new services.

For example, imagine that in the example in Figure 5 the set $T$ of external translation services is empty. In such case, the reconciliation planning algorithm would not be able to find any reconciliation plan resolving the incompatibilities between the $\text{SearchFlight}$ request and the $\text{SearchFlightUA}$ provider service. What we want to do in such case, is to reformulate the incompatibility into a query that can be sent to a discovery service, which eventually could be able to discover a set of services similar to the set $T$ presented in Figure 5, and to re-plan afterwards.

**IV. Runtime mediation (execution with recovery):** Finally, after reconciliation plans are computed they must be used later during runtime mediation to perform the necessary translations which enable interoperability of the provider and the requester. We developed generic algorithms and implemented them in the process mediation agent, that can interpret the reconciliation plans and that performs necessary translations at runtime. The agent exploits functionalities provided by the monitoring and the recovery layer to deal with service failures.
DISCOVERING SERVICE COMBINATIONS

The purpose of discovery in the context of the abstract process mediation framework is to provide mechanisms for finding external services which can deliver the missing pieces of information or can bridge the incompatibilities identified by the process mediation layer. Notice that in open dynamic environments discovery is a pure necessity because process mediation components simply cannot be assumed to know all suitable external services. While service discovery is an active research field, it turns out that traditional discovery approaches are not suitable for use in the context of process mediation.

In the context of process mediation, new services need to be discovered based on needs identified by the process mediation component. During process mediation, incompatibilities (or "gaps") are identified which cannot be bridged by using any of the known services. Very often a specific gap cannot be bridged by any existing single service, and a combination of several services must be used. This introduces a problem because traditionally matchmaking algorithms (Paolucci et al., 2002; Klusch et al., 2006; Bellur and Kulkarni, 2007) used in service discovery components consider only one service as a suitable candidate satisfying a service request, while combinations or compositions of services are not considered. Therefore, when querying existing discovery components with a service request based on the identified gap, usually no match can be found, although some existing combination of services registered with the discovery component would be a suitable match. For example, in Figure 5 the whole set $T$ cannot be discovered by most of existing discovery methods, because very likely there exists no single service which would be able to reconcile the mismatch between the SearchFlight request and the SearchFlightUA provider service.

The reason why usually only one service is considered as a valid match is motivated by the requirements service discovery components need to fulfill: service registries are expected to store large numbers of services and at the same time the best matching set of services for a given query has to be retrieved in a timely manner (ideally, in order of milliseconds). Such a combination of requirements makes it difficult to employ full-fledged composition algorithms (Traverso and Pistore, 2004; Atif et al., 2006; McIlraith and San, 2002) during the discovery process because their time complexity is unacceptable for discovery purposes (composition algorithms usually assume a rather small number of services and operate in order of seconds or minutes).

We addressed this problem by enriching discovery algorithms with basic composition capabilities in a controlled manner that guarantees (1) the same flexibility as the one of classical matchmaking of individual services and (2) a modest time complexity increase compared to individual services matchmaking. Specifically, we extended the matchmaking conditions for individual services so as to allow a combination of several services as an acceptable match for a given service request. This has to be done carefully though, since allowing combinations of services can lead to efficiency problems, as identified in the work of Benatallah et al. (2003) which shows that finding an optimal combination of services covering the request can be NP-hard under certain conditions. We explored a similar direction by allowing the combination of services (a set) satisfying the request to be returned as a relevant match — we call it a combined match.

A combined match addresses the situation where a single service matching a given request does not exist, however it also introduces new problems. Since one combined match can consist of several services that together are able to satisfy the service request, various collisions between these matching services have to be taken into account. For example, one single effect (e.g., a booked plane ticket) might be delivered by more than one service in the combined match leading to an undesirable situation. Similarly, undesired side-effects can be produced by the combined match — e.g., if a flight reservation is provided only in a package with a hotel reservation, the hotel reservation might be an unwanted side-effect for a requester who needs a flight ticket only.
Finally, two services can produce contradictory effects (such as making and canceling a reservation). Depending on the requester’s needs each of these situations might cause a problem and thus rendering a combined match useless. Therefore, the discovery component has to be able to avoid collisions in matches.

Retrieving all matching service combinations can be computationally very expensive because of a possibly big size of such a set. To deal with this problem we devised an algorithm for retrieval of the best top \(k\) matching combinations with respect to an aggregate ranking function that computes the overall matching degree of service combinations (Vaculín and Sycara, 2009). The top \(k\) matchmaking algorithm is based on the best-first search approach. The basic idea of the algorithm is to retrieve advertisements for each requested output or effect separately and for each such an output/effect sort the retrieved advertisements based on the matching degree of the advertisement with respect to the requested output/effect. Having the basic relevant advertisements retrieved and sorted, in the next phase the search space of combined matches built out of these retrieved advertisements is traversed in such a way that guarantees the increasing order of generated combined matches. Additionally, to eliminate the combinations which contain some collision and to reduce the search space, we employed optimization techniques based on branch and bound pruning and constraints propagation known from constraint satisfaction problem solving.

We showed that if the overall ranking function is monotonic and monotonic in all its parameters, the retrieval of top \(k\) service combinations without undesired and contradictory effects can be performed with the time complexity \(O((m \log m) n)\) worse than the time complexity of the individual service retrieval for a request with \(n\) outputs or effects, with \(m\) being the maximum number of advertisements able to produce some output or effect in the request. We also showed that retrieving service combinations without effect duplications is an NP-complete problem. The experimental evaluations showed that although the problem is NP-hard in some cases our optimized algorithm performs very well and that it can be used as an any-time retrieval algorithm which improves the quality of the returned matches if more time for retrieval is available.

**SEMANTIC MONITORING**

While the advantages of semantically annotated Web services were recognized in the context of service discovery and composition, little effort has been invested into studying possibilities of Semantic Web services monitoring (semantic monitoring). The importance of powerful monitoring techniques increases as operating environments of Web services become more dynamic and Web services based information systems are expected to work in an autonomous or semi-autonomous fashion. Naturally, this is the case of the process mediation, since often neither the requester nor the provider are known in advance. The mediation component must be able to dynamically execute the process models and also be able to interpret the flow of execution and its results, including possible failures and their causes.

Execution monitoring mechanisms are needed to provide human or software agents with appropriate information about the execution course and results. Information provided by monitoring infrastructure can be used either during execution to support a dynamic response to the given execution course, or after execution is finished for purposes of analysis and auditing. In the broader context, monitoring is recognized as a natural part of the Service-oriented architectures. In SOA systems the main stress is usually put on monitoring of performance and availability metrics. The monitoring sub-system should be able to support tasks such as measuring and evaluation of Quality of Services (QoS) metrics, enforcement of Service Level Agreements (SLA), notifications and auditing of service performance, alert-based reporting on the level of adherence to the SLA, or sending automatic notifications and allow graceful
exception handling when the SLAs break down. Furthermore, since Web services are often used as part of complex processes models and workflows, the need for analyzing, diagnosing, simulating and optimizing of such processes models arises. The later scenario finds its applications mainly in areas such as (Semantic) Business Process Management (BPM) (Hepp et al., 2005) and Process Mining (Pedrinaci and Domingue, 2007). BPM is approaching management and execution of IT-supported business operations from a business expert’s view rather than from a technical perspective (Smith and Fingar, 2003). In the context of BPM and Business Process Analysis (BPA) (van der Aalst et al., 2003) monitoring is supposed to support mainly optimization, reengineering and fine-tuning of existing process models.

To address these challenges in the context of Semantic Web services, we have developed an ontology for specification of individual events (called primitive events) (Vaculin and Sycara, 2008). During service execution, the execution infrastructure emits semantically annotated events specific to the state of process model execution. The content of emitted event instances describes the execution context in the time when the event occurred and other information relevant to the given event type. The content is annotated by the same domain ontology concepts that are used in the service definition itself, which allows more flexible events detection techniques than those derived from a simple syntactic key-words matching. We employ semantic reasoning for detection of primitive events based on matching their event type and the content.

For many applications detection of individual events emitted by various components of the systems is a sufficient solution. However, often complex events patterns (called composite events) need to be detected. We have developed a language for specification of semantic composite events (Vaculín and Sycara, 2007a) based on the event algebra introduced originally in the context of active databases (Chakravarthy et al., 1994; Carlson and Lisper, 2004). Finally, we proposed monitoring mechanisms suitable for both efficient runtime monitoring and offline analysis. In earlier works, various mechanisms for composite event detection were proposed, including Petri nets (Gatziu and Dittrich, 1993), finite state automata (Pietzuch et al., 2004), and event detection graphs (Chakravarthy et al., 1994). We adopted an approach based on event detection trees proposed by Chakravarthy et al. (1994), however we modified the algorithm according to the work of Carlson and Lisper (2004) which presents a more efficient technique suitable especially for runtime detection. We extended the algorithm so as to allow detection of semantic events.

**FAULT HANDLING AND ERROR RECOVERY**

The ability to handle failures during execution and to recover from failures is important not only in the context of process mediation, but for Web services in general. Problems of fault handling, recovery and adaptation were extensively studied in areas such as programming languages, distributed systems and workflow systems (Müller et al., 2004). Current standards for Web services workflows (such as WS-BPEL (Alves et al., 2007)) and transactions (WS-Transaction, Business Transaction Protocol) solve fault handling and recovery problems partially, typically by providing support for some form of long running transactions (LRT). For example, WS-BPEL uses a known concept of compensation to achieve transaction properties and to allow execution recovery. However, fault handling in WS-BPEL is designed as a reverse work and its aim is to undo the partial and unsuccessful work only. Also WS-BPEL offers only a limited support for recovery and monitoring which makes it suitable rather for static scenarios. Currently, none of the Semantic Web services emerging standards (SA-WSDL, WSMO, OWL-S) provides any support for fault handling and recovery. We focus on providing the missing fault handling and recovery for Semantic Web services, which has a great potential for increasing the autonomy of Web services systems.

We have developed techniques for fault handling and recovery for Semantic Web services to
allow specification of reliable, adaptive process models (Vaculín et al., 2008d). The basic idea of our approach is to take advantage of powerful semantic monitoring techniques to define and detect possible erroneous states. To allow a controlled process recovery and gradual execution degradation standard fault handling must be augmented with mechanisms allowing a designer to define what situations are supposed to trigger an erroneous state. To achieve this, we augment the process model definition with constraint violation handlers (CV-handlers) for associating constraint violation conditions with appropriate explicit recovery actions that resolve the violations. Usually, the constraints can stem from applicable SLAs or from contractual requirements. Constraint violation conditions are treated as hard constraints that lead to an abnormal execution state. The recovery actions, such as retry, replaceBy, replaceByEquivalent, are supposed to fix problems manifested by the fault occurrence and thus present means of restoring the normal execution flow.

To express soft constraints that do not necessarily lead to an erroneous state, we use event handlers. A condition part of both event handlers and CV-handlers must be expressive and intuitive enough to allow encoding of SLAs and other constraints. We employ event algebra expressions combined with semantic filters developed as part of our work in semantic monitoring, which are suitable for describing complex event patterns and allow efficient event monitoring and detection. We use compensation statements for undoing effects of the partial work after a fault has occurred.

**Figure 6:** Mediation of process models by the Process Mediation Agent (PMA): the PMA execution architecture

**EXECUTION INFRASTRUCTURE: THE PROCESS MEDIATION AGENT**

We integrated the above described solutions into a concrete execution infrastructure for runtime mediation which we call the Process Mediation Agent (PMA). The architecture of the PMA is designed to support either the mediation in the complete visibility mode when it uses mappings computed by the offline reconciliation planning algorithms, or in the strictly reactive mode in which all mappings are computed only during runtime. Figure 6 shows the architecture of the PMA. The server port is used for interactions with the requester and the client port for interactions with the provider. The client port uses the OWL-S Virtual Machine (OVM) (Paolucci et al., 2003b) to interact with the provider. We extended the OVM, which is a generic OWL-S processor for execution of OWL-S services, with the advanced features for execution monitoring and recovery as described in the previous sections. Another instance of the OVM is used to
execute external data mediation services if necessary.

The **Execution Monitor** is the central part of the PMA. It executes the runtime mediation procedure and it integrates all the other components together. The **Execution Monitor** maintains the execution state and stores information received from the requester and provider in a **Knowledge Base**. When new data mediators need to be found during runtime, the **Execution Monitor** interacts with the **Service Discovery** component. The **Plans Library** is used to store reconciliation plans that were either provided as an output by the offline analysis, or that were used successfully in previous mediation sessions. The primary purpose of the plans library is to improve efficiency by avoiding redundant planning in future mediation sessions. Also information about discovered data mediators is cached in the **Plans Library**. When no historical or precomputed information is available, the PMA explores the search space by employing the reconciliation planning algorithm to find an appropriate reconciliation plans.

**RESULTS AND DISCUSSION**

Process mediation is an extremely challenging problem. Although Semantic Web services introduce an additional layer of semantics on top of classical Web services specifications, which makes the problem somewhat easier to conceptualize in a unified way amenable to various AI techniques such as automated planning, still the problem itself remains very hard. The above described set of solutions for process mediation of incompatible process models of services requesters and providers presents a rather comprehensive attempt to deal with interoperability in the context of Semantic Web services based business processes.

The developed process mediation algorithms are sound and complete for the complete visibility scenario. For the asymmetric scenario, the algorithms are sound and \( k \)-complete, where \( k \) is the maximal length of plans that are being searched. The algorithms guarantee a deadlock free communication between the provider and the requester. In terms of incompatibilities, our algorithms can resolve any combination of the following incompatibilities.

- **Step/process re-ordering**: Requester expects some operations to be executed in a different order than it is actually supported by the service provider. For example, the requester might request to first specify the payment method, and later to specify the delivery method, while the provider expects this to be performed in the opposite order. Such problem can be resolved by our algorithm, unless the step reordering leads to a deadlock situation.

- **Adding process**: A new service step, such as a call to an external service, can be added to resolve various incompatibilities.

- **Suppressing process**: If data or effects provided by some process of the requester or the provider are not instrumental for the other party of the interaction, such process and its data and effects are ignored.

- **Condensing several steps into one**: Calls of several processes (messages) can be translated into one service call if it is needed.

- **Decomposing one step into several**: Data provided by one step, i.e., message, can be decomposed into several individual messages.

- **Data transformation**: Data incompatibilities can be transformed by means of (chains) of several external services. In our work, we do not address data mediation explicitly. This means that only such data incompatibilities can be resolved for which the process mediator either knows some external service that can resolve the given data incompatibility, or it can discover such external service or a combination of external services.
• **Modification of data binding:** The original anticipated data-binding, as proposed by the participating process models, can be modified so as to satisfy requirements of the other party.

• **Reuse of data values:** One particular value can be reused as an input of several processes.

The distinguishing feature of our approach, besides enabling interoperability of requesters and providers, is the ability to operate in conditions where failures and changes of the environment can occur. Due to built-in recovery mechanisms, the Process Mediation Agent is either able to recover from failures completely so as to renew the normal execution flow, or its performance degrades gracefully in cases when a complete recovery is not possible. Furthermore, in all developed algorithms we assume that the external services and data mediators are not known to the process mediation component and that they need to be dynamically discovered based on identified incompatibilities. Combination of these assumptions together with employing ontologies for service specification and a flexible matchmaking makes our approach unique compared to other methods which assume that the environment is static, fully accessible and reliable.

In the process mediation part of our work we focused on developing methods for automated process mediation. To make the problem manageable we accepted several assumptions. First, we assumed that the mediation involves two partners, the service requester and the service provider. Such setting is the most usual, however, a question arises of how to deal with a situation when interoperability of more than two process models needs to be achieved. In our approach we do not address such situation, but our algorithms can be extended rather easily to deal with such situation, assuming that only one of the partners is the requester and the other partners are service providers. On the other hand, our method would require significant modifications in order to address a situation where every partner can behave as a peer, i.e., it can do both send requests and receive requests and answer to them.

Next, in all parts of our work we assumed that services are described by using a shared ontology. This assumption was necessary in order to isolate problems of mediation, discovery and monitoring from a very different problem of semantic integration of ontologies (Noy, 2004; Euzenat and Shvaiko, 2007) (also known as ontology alignment or ontology matching). In the real world, however, it is impossible to avoid ontology mismatches. A possible solution to this problem in context of process mediation could be to address the ontology mismatches in a standalone component or a service that would encapsulate some approach to managing or finding ontology mappings.

Important research questions are related to data mediation, which we did not address in this work. While data mediators have a very clear role and position in the abstract process mediation framework and in our mediation algorithms, we did not discuss any details about how data mediators can be implemented (which is an independent research field on its own). Also, data mediators or services specialized in providing access to data or to data transformations have specific needs with respect to discovery. We started to explore the problem of modeling and recovery of data providing services in our recent work (Vaculín et al., 2008a) since this problem is very closely related to data mediation and process mediation as well.

**LESSONS LEARNED AND FUTURE DIRECTIONS**

We conclude this chapter with some lessons that we have learned throughout our research. In particular these include the following:

• Current Web services standards such as WSDL or WS-BPEL provide a good basis for achieving at least some level of interoperability, however, they provide only a starting point
- e.g., shared semantics is completely missing.

- Newly emerging standards for Semantic Web services, such as SAWSDL (Farrell and Lausen, 2007), OWL-S (Martin et al., 2004) and WSMO (Roman et al., 2005a), strive to enrich syntactic specifications with rich semantic annotations to further facilitate flexible dynamic web services discovery, composition and invocation (Sycara et al., 2004). However, the current standards do not provide reasoning methods for interoperability of providers and requesters as application requirements change. In any case, some form of shared, formalized, machine processable semantics is a crucial part for achieving interoperability.

- Various types of middle agents (Wong and Sycara, 2000) – employing techniques such as reasoning and planning combined with approaches like dynamic discovery and recovery from failure – present a possible solution for bridging the gap between service requesters and providers with incompatible interaction protocols (process models) and possibly incompatible data models.

- The AI methods (e.g., planning, reasoning) can help substantially in achieving interoperability in open environments, however, such methods (mostly due to their possibly heuristic nature) need to be complemented either with human based verification mechanisms or with some advanced automated verification techniques.

- It turned out that the existing mechanisms for business process specification (BPMN, WS-BPEL, YAWL, etc.), fault handling, monitoring, and recovery are very static in their nature and as such they are simply not suitable for dealing with dynamic scenarios and adaptation (Russell and ter Hofstede, 2009).

- Traditional approaches focus on very static, rather deterministic interoperability scenarios, while the reality is much more dynamic and often quite non-deterministic. This is going the be more and more the case especially with the advent of mobile computing, cloud computing, and software as a service paradigm. Thus, the current methods will have to focus more on such dynamic scenarios.

With respect to these observations we have identified several important areas for future research directions.

Adaptation by Design

A very promising area of research, which we already mentioned earlier as the so called top-down approach to achieving interoperability, revolves around the question how to design business processes in such a way so as to make them more adaptive by design. Such processes would be much easier to modify, evolve, adapt to changing needs, and subsequently also easier to achieve interoperability among potentially incompatible processes. In particular, some interesting research questions are related to possibility of connecting the developed process mediation techniques with higher-level modeling of business processes, e.g., based on higher level modeling concepts, such as business & organizational goals, commitments, requirements, and data artifacts. A lot of research needs to be done in the area of formalisms for specification of adaptive processes, adaptation methods and suitable methods for design time and runtime verification.

Adaptation & Interoperability in the Web of Linked Data

With the ever increasing importance of services provided over the Internet, a very important
problem is interoperability of business processes using the Internet-based services. As opposed to traditional WS-based services, most of services available on the web these days are light-weight services, such as REST-full or Ajax based services. Currently, such services have a rather ad hoc form, however, as their importance will grow, we can expect a stronger push on standardization and interoperability aspects. Furthermore, the web based services mostly operate in the very data centric environment, where data are oftentimes interlinked. Thus the challenges are centered around development of suitable interoperability and adaptation methods for such environment.

Interoperability of Data Intensive Business Processes

Similarly to web based business processes, even in business environments many business processes are very data centric. Recently, data-centric business processes have gained a substantial attention (Cohn and Hull, 2009). Compared to traditional approaches, such as BPMN, the data centric process specifications put a higher stress on importance of data in the process life-cycle. One research question is how to adapt the process mediation techniques to data centric business processes. More general question is how to organically incorporate information and data centric services into SOA, into composition techniques, discovery, and mediation. We have presented our first attempt focused on representation of data centric services with the main focus on discovery of such services in (Vaculín et al., 2008a).

ACKNOWLEDGEMENTS

This research was partially supported by the Czech Science Foundation project P202/10/P604 and the Czech Ministry of Education project ME08095. Katia Sycara was supported in part by Darpa contract FA865006C7606 and in part by funding from France Telecom.

REFERENCES


**KEY TERMS & DEFINITIONS**

**Process mediation**
Solves the problem of interoperability of (partially) incompatible business processes by identifying and resolving all incompatibilities, generating appropriate mappings between different processes, and by mediating between service requesters and providers, while making minimal assumptions about implementation details of service providers and requesters, and without requiring any modifications of implementation of participating business processes, web services and their interfaces.

**Web services**
Web services are distributed information systems, which can be seen as an evolution of conventional distributed systems, such as middleware and workflow systems. According to W3C a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. In the traditional sense a Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

**Semantic Web services**
Research in the Semantic Web services area is addressing the problems of automated service discovery, composition, execution and similar problems by augmenting the existing Web services standards by additional semantic layer(s). The additional semantic focuses on enabling “... semantically transparent services which will make it possible for clients to successfully use services that are dynamically discovered without prior negotiations between client and service developers” (Burstein et al., 2005). Most of the proposed mechanisms rely on extending the current WS descriptions with descriptions in some logical formalism more suitable for capturing the explicit meaning of provided Web services. Such formalisms typically rely on formal ontologies and research conducted in the Semantic Web area.

**Web services discovery**
Web services discovery addresses a problem of finding a suitable service or a set of services that are able to match specified requirements or objectives. Service discovery and selection can be performed manually, in which case the user is responsible for selecting a matching service. In case of automated discovery, matchmaking algorithms need to be applied to decide about a degree of match between a service request and a service advertisement.

**Web service composition**
Service composition focuses on models and methods supporting composition of several
existing Web services into more complex processes that provide new functionalities. The main focus of research has been in exploring methods of automated or semi-automated service composition. A usual formulation of the automated composition problem assumes that there is a given set of existing services and a set of requirements, and the goal is to find a composition (possibly the best) that satisfies the specified requirements.

**Automated planning**

An automated planning problem can be formulated as follows: given a description of the initial state of the world, a description of the desired goal, and a set of possible actions (operators) that can be used to change the state of the world, the question is if there exists a sequence of actions that lead from the initial state to a state meeting the goal. Research in automated planning focuses on devising efficient methods for automatically solving the planning problem.

---

2 [http://www.seekda.com](http://www.seekda.com)
3 Forrester Report, October 2, 2009: “What Semantic Technology Means To Application Development Professionals” (Semantic Technology Is Now Ready For Increased But Focused Use)