Process Mediation, Execution Monitoring and Recovery for Semantic Web Services

Katia Sycara
The Robotics Institute
Carnegie Mellon University
katia@cs.cmu.edu

Roman Vaculín
Institute of Computer Science
Academy of Sciences of the Czech Republic
vaculin@cs.cas.cz

1 Introduction

One of the main promises of web services standards is to enable and facilitate seamless interoperability of diverse applications and business processes implemented as components or services. A service can be part of a business workflow that prescribes control and data flows of complex applications. As business needs change, processes may need to get reconfigured or additional process components and services may need to be added. As a result of these changes, the previous components must become interoperable with the new one. This can be accomplished by making manual changes to the existing workflow components and programming the new components in such a way as to have interoperability built-in. This is a rather laborious and inefficient process since it must be repeated every time workflow reconfiguration is needed. Also, since many different elements of a business workflow may be under the control of third parties (e.g. subcontractors), additional costly coordination will be needed with these third parties to manually find interoperability solutions. 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. In other words, the new service component that must interoperate with old ones, is dynamically discovered. Therefore, (a) a more general solution is desired, namely the ability to achieve process interoperability (e.g. interoperability of existing processes with new ones) without actually modifying their implementation and interfaces, and (b) the mediation may need to be done dynamically even at runtime, which implies that only minimal assumptions about knowledge of service requester and service provider interfaces is allowed. One solution to this requirement is to apply a process mediation component which resolves all incompatibilities and generates appropriate mappings between different processes while making minimal assumptions about implementation details of service providers and requesters.

Creating such a process mediation component is a very challenging task. Service providers and requesters may not share basic standards for Web Service specification; they may not share domain ontologies; furthermore, they typically do not do share the same data models or interaction protocols. Moreover, the changing business needs may dictate that existing services are modified, thus rendering previous compatible interactions incompatible. As a result, a mediation module must deal with incompatibilities of multiple types and also be able to incorporate adaptive reasoning mechanisms to address dynamic environment changes.

Current web services standards provide a good basis for achieving at least some level of interoperability. WSDL allows to declaratively describe operations and format of messages and data structures that are used to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

communicate with the web service. BPEL4WS adds the possibility to define the interaction protocol and possible control flows and combine several web services within a formally defined process model. However, none of the current standards goes beyond the syntactic descriptions of web services. Newly emerging standards for semantic web services, such as SAWSDL [4], OWL-S [8] and WSMO [6], strive to enrich syntactic specifications with rich semantic annotations to further facilitate flexible dynamic web services discovery, composition and invocation [7]. However, the current standards do not provide reasoning methods for interoperability of providers and requesters as application requirements change. Various types of middle agents [14] – 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.

2 Process Mediation

In our recent body of work [11, 9], we address the problem of automatic mediation of process models consisting of semantically annotated web services. Processes can act as service providers, service requesters or communicate in peer-to-peer fashion. We are focusing on the situation where the interoperability of two components, one acting as the requester and the other as the provider, needs to be achieved. Usually, both the requester and provider adhere to some relatively fixed process models. The process models can either correspond to a particular existing implementation or they can be default (generic) process models that for example generalize a business processes of some specific problem domain (e.g., client or provider of flight booking service). In particular, our research focuses on mediation of process models of providers and requesters in open dynamic environments where new services could be dynamically discovered, thus necessitating runtime mediation. Additionally, we assume that both the requester and the provider interact according to specified process models that are fixed, are expressed declaratively and might be incompatible.

We use the OWL-S ontology [8] for semantic annotations 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 an ontology 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, loops, etc.

Creating a mediator component is very challenging since this component must resolve various types of mismatches, such as the following that we have identified:

A. Data level mismatches: e.g. data are represented as different lexical elements (numbers, dates format, local specifics, etc.); or ontological mismatches

B. Service level mismatches: e.g. a requester's service call is realized by several provider's services or a sequence of requester's calls is realized by one provider's call; some information required by the provider is not provided by the requester; information provided by one party is not needed by the other one

C. Protocol / structural level mismatches: e.g. control flow in the requester's process model can be realized in different ways in the provider's model (e.g., sequence can be realized as an unordered list of steps, etc.)

We have developed an abstract process mediation framework (APFM) showed in Figure 1. The main goal of the APMF is a clear identification and separation of critical functional areas which need to be addressed by mediation components in order to effectively solve the process mediation problem. The three key functionalities, namely *process mediation*, *data mediation* and *service invocation*, are displayed as horizontal layers. The

process mediation layer, realized by process mediators, is responsible for resolving service level and protocol level mismatches (categories B and C). The data mediation layer, realized by data mediators, is responsible for resolving data level mismatches (category A). Typically, when trying to achieve interoperability, process mediators and data mediators are closely related. A natural way is to use data mediators within the process mediation component to resolve "lower" level 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, the mediation processes make use of *monitoring* and *recovery* functionalities, which are represented as vertical layers in Figure 1. 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 two processes.

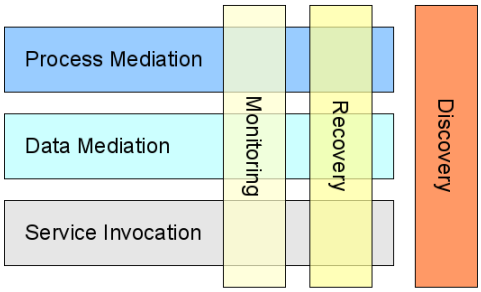

Figure 1: Abstract Process Mediation Framework

We have investigated and developed concrete architectures for the mediation components in the APMF framework for two cases: (a) when the mediation component has *complete visibility* of the process model of the service provider and the service requester [11] or (b) when the mediation component has *visibility only of the provider's process model* but not the requester's (we call this asymmetric visibility) [9].

In the complete visibility scenario, our solution is based on an off-line analysis of possible execution sequences of the requester. A planning algorithm is employed to identify mismatches between requester's execution sequences and the provider's process model, and to compute the appropriate mappings for bridging the identified mismatches. Such mappings are used during +runtime mediation to perform the necessary translations. In the case of asymmetric visibility, the off-line analysis cannot be employed because the requester's process model is not available. Therefore, the mediation must rely strictly on computing the mapping during runtime only. We have developed a process mediation agent that uses similar planning techniques as in the complete visibility scenario except that the planning is constrained by time due to the requirement of a timely response. Additionally, the process mediation agent incorporates advanced recovery techniques to deal both with service failures and with possible wrong choices made during the mediation.

3 Semantic Monitoring

We have developed an ontology [12] for specification of primitive events and a language for specification of composite event patterns [10] based on the event algebra developed originally in the context of active databases [3, 2]. Additionally, we have developed monitoring mechanisms combined with introspection mechanisms and error handling that we implemented as extensions of the OWL-S Virtual Machine [5] which is a component that controls interactions between the clients and semantic web services. Specifically, the OWL-S Virtual Machine (OVM) executes the process model of a given service by going through the process model while respecting the

OWL-S operational semantics [1] and invoking individual services represented by atomic processes. During the execution, the OVM processes inputs provided by the requester and outputs returned by the provider's services, realizes the control and data flow of the composite process model, and uses the grounding to invoke WSDL based web services when needed. The OVM is a generic execution engine which can be used to develop applications that need to interact with OWL-S web services. During the service execution, the execution engine (OVM) emits events specific to the state of process model execution. Emitted events (primitive events) are instances of generic event or fault types defined in the events ontology [12]. 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 semantically annotated by the same domain ontology concepts that are used in the service definition itself, which allows a more flexible events detection techniques than those derived from a simple syntactic keywords matching. Specifically, we employ semantic reasoning for detecting primitive events based on matching their event type and the content.

The implemented monitoring extensions allow to perform different monitoring tasks such as logging, performance measuring, execution progress tracking, execution debugging or evaluations of security parameters. For many applications simple detection of individual events (called primitive events) emitted by various components of the systems is a sufficient solution. However, often complex events patterns (called composite events) such as co-occurrence of different events or sequence of events need to be detected.

4 Fault Handling and Error Recovery

Currently, neither WSMO, nor OWL-S provide any support for fault handling and recovery. The ability to handle failures correctly and to possibly be able to recover from failures is important not only in the context of process mediation, but for web services in general. We have developed techniques for fault handling and recovery for semantic web services [13] to allow specification of reliable, possibly adaptive process models and so to increase the autonomy of web services systems. Again, we focus primarily on dynamic environments where cooperating services might need to be discovered during runtime. Our approach to fault handling and recovery shares similarities with fault handling in WS-BPEL. However, WS-BPEL offers only a limited support for recovery and the monitoring which makes it suitable rather for static scenarios.

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. Such 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. 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 have employed event algebra expressions [2] combined with semantic filters [10], which are suitable for describing complex event patterns and allow an efficient events monitoring and detection (described briefly in the previous section). Similarly to WS-BPEL, we use *compensation* for undoing effects of the partial work after a fault has occurred. Finally, we introduced *explicit recovery actions* (such as *retry*, *replaceBy*, *replaceByEquivalent*) as means of fixing problems manifested by the fault occurrence. Recovery actions present means of restoring the normal execution flow.

5 Acknowledgments

This research was supported in part by Darpa contract FA865006C7606 and in part by funding from France Telecom. Research partially supported by the Czech Science Foundation project 201/05/H014 and the Czech Ministry of Education project ME08095.

References

[1] Anupriya Ankolekar, Frank Huch, and Katia P. Sycara. Concurrent semantics for the web services specification language DAML-S. In Farhad Arbab and Carolyn L. Talcott, editors, *COORDINATION*, volume 2315 of *Lecture Notes in Computer Science*, pages 14–21. Springer, 2002.

[2] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In Giorgio C. Buttazzo, editor, *EMSOFT*, pages 147–154. ACM, 2004.

[3] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 606–617, Santiago, Chile, 1994.

[4] Joel Farrell and Holger Lausen. Semantic annotations for WSDL and XML schema, 2007. `http://www.w3.org/TR/sawsdl/`.

[5] Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia P. Sycara. The DAML-S virtual machine. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2003.

[6] Dumitru Roman et al. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77 – 106, 2005.

[7] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1 (1):27–46, 2004.

[8] The OWL Services Coalition. *Semantic Markup for Web Services (OWL-S)*. `http://www.daml.org/services/owl-s/1.1/`.

[9] Roman Vaculín, Roman Neruda, and Katia Sycara. An agent for asymmetric process mediation in open environments. In *Service Oriented Computing: Agents, Semantics and Engineering*, pages 1032–1039. Springer Verlag, May 12-16 2008.

[10] Roman Vaculín and Katia Sycara. Specifying and monitoring composite events for semantic web services. In *The 5th IEEE European Conference on Web Services*. IEEE Computer Society, November 26-28 2007.

[11] Roman Vaculín and Katia Sycara. Towards automatic mediation of OWL-S process models. In *2007 IEEE International Conference on Web Services*, pages 1032–1039. IEEE Computer Society, July 9-13 2007.

[12] Roman Vaculín and Katia Sycara. Semantic web services monitoring: An OWL-S based approach. In *41st Hawaii International Conference on System Sciences*. IEEE Computer Society Press, January 7-10 2008.

[13] Roman Vaculín, Kevin Wiesner, and Katia Sycara. Exception handling and recovery of semantic web services. In *Fourth International Conference on Networking and Services*. IEEE Computer Society Press, 2008.

[14] H. Chi Wong and Katia P. Sycara. A taxonomy of middle-agents for the internet. In *ICMAS*, pages 465–466. IEEE Computer Society, 2000.