Recovery Mechanisms for Semantic Web Services

Kevin Wiesner, Roman Vaculín, Martin Kollingbaum, and Katia Sycara*

The Robotics Institute, Carnegie Mellon University
{kwiesner,rvaculin,mkolling,katia}@cs.cmu.edu

Abstract. Web service-based applications are widely used, which has inevitably led to the need for proper mechanisms for the web service paradigm that can provide sustainable and reliable execution flows. In this paper we revise recovery techniques in OWL-S and show how semantic annotations may ensure seamless web service provision in a sophisticated way, such as, exploiting the ontology-based description of processes in order to dynamically find alternative services as substitutes for failed services. We also discuss the consequences of these semantic-enabled approaches and point out required changes for integration in OWL-S.

Keywords: Semantic Web, Web Services, Recovery, OWL-S.

1 Introduction

The web services (WS) paradigm is widely used and many enterprises deploy their business processes as web services. Typically, web service-based processes tend to operate in rapidly changing environments where two main concerns need to be addressed. First, the business process has to fulfill the goals for which it was designed. Second, the process must respond to changes in its operating environment by adapting to them in order to guarantee long-term sustainability. These two concerns are orthogonal. Current WS and business process standards focus on the first issue. Constructs for control and data flow specifications are typically based on some form of process algebra and thus, allow an easy design of structured processes that are particularly suitable for stable environments. Exception and recovery mechanisms are used to deal with unusual situations and changes. Current web service recovery mechanisms are highly inflexible. BPEL [1], for instance, uses compensation handlers with explicitly defined compensation actions (i.e., service calls). This provides only one solution for recovery at a particular time. Other possible solutions that might exist are not taken into account. Since the environment is changing constantly, the availability and reachability of services may vary over time. With conventional approaches to recovery and exception handling, it is not possible to adapt to such changes.

* This research was supported in part by Darpa contract FA865006C7606, by AFOSR FA9550-07-1-0039, and by funding from France Telecom.

R. Meier and S. Terzis (Eds.): DAIS 2008, LNCS 5053, pp. 100–105, 2008.

A compensation containing a non-reachable service, for example, results in an inconsistent state of the system in the case of a failure, even though other services could be used as compensation for the failed process.

These shortcomings reveal a need for stronger and more flexible recovery mechanisms that allow a process to adapt while simultaneously respecting the design of the original process. Semantic Web Services (SWS) appear to be ideal for achieving this. The SWS standards introduce means for providing service specifications with rich semantic annotations that facilitate flexible dynamic discovery, invocation and composition of services. This paper focuses on how techniques such as dynamic discovery and composition can be exploited in the context of recovery and process adaptation. For example, SWS can take advantage of the dynamic discovery of either equivalent services (as a replacement) or other appropriate services, that may help to recover from a failure (e.g. for compensation), instead of relying on explicitly specified recovery solutions. In our previous work [2] OWL-S was extended with exception handling and basic recovery mechanisms. This paper revises recovery techniques introduced previously, and further presents new semantic-enabled mechanisms. We propose *ReplaceByEquivalent* and *Advanced Back & Forward Recovery* actions, which try to dynamically discover alternatives for erroneous tasks. Next, the *Automatic Compensation* technique exploits the semantic information to undo finished processes.

This paper is organized as follows: in Sect. 2 the existing exception handling and recovery in OWL-S is presented, followed by the introduction of new semantic-enabled mechanisms in Sect. 3. Next, these approaches and their consequences are discussed in Sect. 4 and related work is summarized in Sect. 5. In Sect. 6 we conclude and give an outlook on future work.

2 OWL-S and Recovery

OWL-S [3] is a description language for Semantic Web Services, based on OWL [4], that defines services through three kinds of information: the Service Profile describes what the service does in terms of its capabilities and is used for discovering and selecting suitable providers; the Process Model specifies how clients can interact with the service by defining the requester-provider interaction protocol; the Grounding links the Process Model to the specific execution infrastructure. For exception handling and recovery, the Process Model is of particular importance. Its elementary unit is an atomic process, which represents one indivisible operation. Processes are specified by means of their inputs, outputs, preconditions, and effects (IOPEs). Atomic processes can be combined into composite processes by using control constructs (e.g. sequence, split, etc.). All processes and control constructs must be strictly nested in order to ensure that every process or control construct has a defined parent.

To support basic fault handling and recovery, the OWL-S Process Model was extended in [2]. In addition to the IOPEs, every process can define fault handlers, standard event handlers, constraint violation handlers (CV-handlers), and

compensation (FECCs). *Fault handlers* are used to respond to standard failures in the form of an exception event during the execution. The *CV-handlers* augment the basic fault handling by allowing a designer to define what situations during execution are supposed to trigger an erroneous state and how to recover from it. This is achieved by combining event expressions known from event algebras for specifying arbitrary event patterns [5] in the condition part of a CV-Handler, and *recovery actions* in the action part of a CV-Handler. The event conditions defined in CV-handlers are treated as hard constraints that lead to an abnormal execution state. In contrast, *event handlers* are used to express soft constraints. If the event condition of an event handler is met, its actions are processed without changing the execution state. In the *compensation* block, actions which are supposed to undo the effects of a process in case of a failure can be specified.

The following *recovery actions* were introduced in [2] to enable essential recovery in OWL-S:

Compensation: For every process, compensation can be defined. The actions specified in the compensation are used to undo the effects of the previously performed process. Compensation can be triggered either by *Compensate*, which invokes the compensation of the corresponding process, or by *CompensateProcess*, which enables to perform compensation for another process.

Retry: *Retry* simply restarts the same process again (which is especially useful for communication failures). It can be used in fault and CV-handlers to restore a normal execution flow after a failure has occurred. It either retries to execute the corresponding process n-times, or until the specified time expires (*timeout*).

Replace: Two replace operations are provided to replace a process by an alternative one which is supposed to achieve the same goal: *ReplaceBy* simply replaces the *failed* process with an alternative one explicitly specified in the process model. In contrast, *ReplaceProcessBy* replaces any *arbitrary* process with another one, which makes it possible to change the overall workflow.

Skip: The *Skip* action can be used in all FECC handlers to skip a process that has become dispensable as long as it has not been started yet.

Terminate: All running activities are stopped and performed tasks on the same level are undone. Subsequently, the same is done for parent levels. Termination is realized in two ways. *HardTerminate* terminates all running process without allowing compensation, whereas *SoftTerminate* compensates finished processes before terminating.

3 Semantic-Enabled Recovery

In this section, new recovery actions enabled by computer-interpretable descriptions of services are introduced in order to enhance and improve our existing mechanisms [2]. The following operations distinguish themselves in providing a flexible and adaptable way to recover from failures. We achieve this by exploiting mainly existing semantic annotations in OWL-S. The actions either replace or roll back the process with the help of dynamically discovered services.

ReplaceByEquivalent: The described *Replace* action recovers through a replacement, either specified in advance or selected by a human agent. We defined a more flexible operation, *ReplaceByEquivalent*, which dynamically adapts to the current situation, by using the information about the service capabilities. The OWL-S Process Model specifies inputs, outputs, preconditions, and effects, which we utilize to find an alternative service with the help of existing algorithms for automatic web service discovery (matchmaking) [6]. Since the replacement service is not selected in advance but discovered during run-time, the chances of a successful recovery and completion of the overall process are substantially increased.

Advanced Back & Forward Recovery: A possible operation for recovery is the *Back & Forward Recovery (BFR)*. After a failure, first a rollback is performed for all finished processes on the same level (of the process hierarchy) where the failure occured (*Back* phase). If the parent is non-vital for the overall outcome, the execution can continue in spite of the failure (*Forward* phase). If the parent is vital, the *Back* phase is repeated until the parent is non-vital, so that eventually *Forward* can be performed. In the worst case, i.e. when all tasks are vital, a complete rollback is performed. However, OWL-S does not support such an operation currently, since all processes are considered to be vital. A parameter indicating whether a process is *Vital/Non-Vital* can be easily added, and so recovery can be neglected for non-vital processes. Independently, we introduced a variation of the *BFR* operation, *Advanced Back & Forward Recovery (ABFR)*. The basic *BFR* goes back in the process hierarchy until a non-vital parent is found. The advanced variation of this makes use of the *ReplaceByEquivalent* in such a way that for each parent that is vital an alternative service is searched for. If an appropriate service exists, it is executed as a replacement and normal execution is resumed. If no replacement is found, *ABFR* continues just as the original approach and goes one step higher in the hierarchy. The following pseudo code demonstrates the ABFR algorithm:

if $hasParent(process) = false$ **then**
  $abortExecution$
else
  $compensateSiblings$
  if $parentOf(process) \neq vital$ **then**
    $continueExecution$
  else
    $successfulReplaced \Leftarrow ReplaceByEquivalent(process)$
    if $successfulReplaced \neq true$ **then**
      $ABFR(parentOf(process))$

Automatic Compensation: The definition of the Process in OWL-S contains information about the effects of a service. This can be exploited to discover an automatic compensation. If a service with the effects ε need to be compensated and no compensation has been specified, it is searched for a service with the effects ε^{-1} which undoes all changes. In particular, we assume that a service without effects does not need to be compensated. This simplifies the recovery.

4 Discussion

Although replacing an erroneous service with a dynamically discovered alternative by using an operation like *ReplaceByEquivalent* can significantly increase the robustness of workflows, it also adds some degree of non-determinism to it. Unbeknownst to the user, malicious services could be executed. In contrast, searching for an alternative only in a local, controlled environment (e.g. within a company's intranet) would not really exploit the potential of such recovery operations; consequently, a central pool with trusted services is a possible solution. The same applies for ABFR as well. Furthermore, the latter poses another problem. ABFR might go back several steps in the hierarchy before finding a replacement, which would lead to a substantial change of the original service. In this case, the user may perform a largely different service than he wanted or the process designer intended him to run. This, however, can be bypassed by specifying tasks as *replaceable/non-replaceable*. As a result, the process designer is able to ensure that some core processes cannot be replaced. This also facilitates a way to specify recovery operations in a more general way. Instead of specifying recovery actions for each process separately, the expressivity of FECC handlers can be exploited. A CV-handler might associate the *ReplaceByEquivalent* with an event expression like $ServiceInvocationException \wedge replaceable$, so that each service that cannot be invoked is automatically replaced.

5 Related Work

The approach introduced in [7], is based on *long-lived transactions* (LLTs). A LLT can be broken up into several sub-transactions, which are then executed as an atomic unit and can be compared to traditional transactions. Only complete executions are accepted, so some sub-transactions have to be undone in the case of failures. This is solved by providing a compensation for each of them and executing those in the reverse order. E.g. WS-BPEL [1] exploits this method for its error handling. In [8], Greenfield et al. discuss this approach in detail.

Intended especially for distributed transactions, several mechanisms for *Web service transactions* have evolved, as Business Transaction Protocol (BTP), WS-Tx[1]. These approaches provide protocols enabling a two-phase commit as in traditional transactions, but also leaving applications in full control of single steps. A failure in a sub-transaction does not inevitably lead to an abortion of the actual transaction. In [9], Papazoglou gives an extensive overview. Curbera et al. [10] suggest a combination of WS-BPEL and WS-Tx. The incorporation of both is discussed in [11].

Workflow transaction approaches (such as in [12]) primarily focus on ensuring a consistent state from a business point of view, i.e. achieving the business goal. In case of a failure, the execution returns to the most recent consistent state and tries to continue the execution in order to complete the workflow.

[1] WS-Tx (WS-Transaction) includes WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity.

6 Conclusions and Future Work

In this paper we revised current recovery techniques in OWL-S and presented new mechanisms enabled by the semantic layer. We introduced a new kind of recovery actions that exploit the semantic annotation of SWS to facilitate dynamic discovery of alternative or auxiliary web services. We proposed that semantic web services can be a key technology in achieving reliable and adaptable service executions. We are curently working on the implementation of semantic-enabled recovery actions mechanisms in the OWL-S Virtual Machine [13]. We plan to extend the process specification to support features like *(non)-vital* as well as *(non)-replaceable*. Additionally, we will investigate further possibilities of semantic-enabled recovery actions.

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0 (April 2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
2. Vaculín, R., Wiesner, K., Sycara, K.: Exception handling and recovery of semantic web services. In: The Fourth International Conference on Networking and Services, IEEE Computer Society, Los Alamitos (2008)
3. The OWL Services Coaltion: Semantic Markup for Web Services (OWL-S), `http://www.daml.org/services/owl-s/1.1/`
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: OWL Web Ontology Language Reference. W3C Recommendation (February 2004), `http://www.w3.org/TR/owl-ref/`
5. Vaculín, R., Sycara, K.: Specifying and Monitoring Composite Events for Semantic Web Services. In: The 5th IEEE European Conference on Web Services, IEEE Computer Society, Los Alamitos (2007)
6. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web 1(1), 27–46 (2003)
7. Garcia-Molina, H., Salem, K.: Sagas. SIGMOD Rec. 16(3), 249–259 (1987)
8. Greenfield, P., Fekete, A., Jang, J., Kuo, D.: Compensation is Not Enough. In: Proceedings of the 7th International Enterprise Distributed Object Computing Conference (EDOC) (2003)
9. Papazoglou, M.: Web Services and Business Transactions. World Wide Web 6(1), 49–91 (2003)
10. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The Next Step in Web Services. Communications of the ACM 46(10), 29–34 (2003)
11. Sauter, P., Melzer, I.: A Comparison of WS-BusinessActivity and BPEL4WS Long-Running Transaction. In: Kommunikation in Verteilten Systemen (KiVS), ser. Informatik Aktuell, pp. 115–125. Springer, Heidelberg (2005)
12. Eder, J., Liebhart, W.: Workflow recovery. In: Proceedings of the First IFCIS International Conference on Cooperative Information Systems, pp. 124–134 (1996)
13. Paolucci, M., Ankolekar, A., Srinivasan, N., Sycara, K.P.: The DAML-S virtual machine. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 290–305. Springer, Heidelberg (2003)