Proc. of EAI-Workshop 2004 - Enterprise Application Integration, Oldenburg, GITO-Verlag, Feb. 2004

Quality-Oriented Handling of Exceptions in Web-Service-Based Cooperative Processes

Ulrike Greiner, Erhard Rahm

Department of Computer Science, University of Leipzig
{greiner, rahm}@informatik.uni-leipzig.de

Abstract: Web services are increasingly used to integrate heterogeneous and autonomous applications in cross-organizational cooperations. A key problem is to support a high execution quality of complex cooperative processes, e.g. in e-business or health care. One important aspect that has received little attention so far is the dynamic handling of exceptions during process execution. To address this problem, we propose a rule-based approach to automatically control and enforce quality constraints for web-service-based cooperative processes.

1 Introduction

The cooperation of companies and organizations leads to a rise of cooperative business processes integrating autonomous and heterogeneous applications of different organizations. Web services are increasingly used to facilitate such an application integration. Web services encapsulate applications and provide a machine-readable, XML-based interface for application calls [1], hence allowing the service providers to preserve their autonomy. Currently, web services typically realize only simple functions such as database queries. However, they can also be used to encapsulate complex legacy applications or entire workflows. To implement real-world cooperative processes a large number of web services of different complexity and from independent providers may have to be integrated. For instance, in a collaborative fulfillment process different partners cooperate to fulfill a customer's order (e.g. for assembling a PC) according to specified delivery time and other quality constraints. Other examples include treatment of a patient by different physicians and hospitals or a travel-booking process containing several services for booking a flight, hotel, etc. Ensuring that such processes reliably serve their purpose is challenging due to the high degree of autonomy and heterogeneity of the cooperation partners. It implies achieving a high quality of web service execution which is affected by various quality characteristics on services such as response time, cost, location, or constraints on service input and output parameters (e.g., price limits, product configurations or delivery deadlines).

Supporting quality of service for web services has found considerable interest recently. Several studies focus on the dynamic selection of the best provider for a particular web service (e.g. [2],[3]), others on dynamic resource management (e.g., load balancing) to support sufficiently fast web service execution (e.g. [2],[4]). [5] proposes the use of semantic service descriptions, e.g. to improve dynamic service selection. Several researchers studied cooperative business processes in the form of interorganization-

al workflows (e.g. [6],[7],[8],[9]) assuming a tight coupling between the cooperation partners. Newer studies address web-service-based cooperative processes (e.g. [10],[11],[12]) which offer more autonomy for the partners. However, most approaches lack flexible exception handling mechanisms to support the execution quality of cooperative processes. BPEL4WS (*Business Process Execution Language for Web Services*) is a proposed standard for defining business processes by integration of web services [12]. It supports a basic exception handling by checking whether predefined fault messages or time-outs occur for web services at particular process steps. Such exceptions are to be handled by calling compensating or alternative services. However, exceptions that are not covered by fault messages (e.g., violations of quality constraints) and which not only occur at predefined process steps cannot be handled adequately. Hence, a more flexible exception handling approach is needed to adequately support the execution quality and robustness of cooperative processes.

We present a new approach to dynamic exception handling in web-service-based processes that supports the specification of quality constraints for services, in addition to conditions a service may offer itself. A rule-based approach is used to handle exceptions such as the violation of constraints or other events (e.g. service faults) occurring during process execution. In section 2 we give an overview of the Web-Flow architecture implementing the new exception handling approach within a dedicated component that can be used together with different web service process engines. Section 3 presents a classification of the quality constraints to be supported. Dynamic exception handling is discussed in section 4; section 5 closes with an outlook on future work.

2 Web-Flow Architecture

The Web-Flow system aims at offering quality support for cooperative business processes integrating web services available locally or provided by external partners. It separates monitoring and exception handling functionality within a dedicated component, MaX (Monitoring and eXception handling). This is to allow a generic solution that can be used in combination with different process execution engines for definition and execution of cooperative processes (see Fig. 1). The process execution engine uses a web service environment to call local and external web services encapsulating simple application programs or entire workflows. Alternatively, external services may be called over the process engine of the partner. Services are assumed to provide a WSDL (Web Service Description Language, [13]) interface, if necessary through a wrapper (e.g. for workflows).

Cooperative processes may be specified according to two major cooperation models. In a *simple cooperation model*, a cooperative process corresponds to a workflow whose activities may refer to external web services. Thus, the providers of the integrated services do not know the overall process. In a *complex cooperation model*, the involved partners agree on a cooperative process (including the services used and the order in which they are called). The partners have an equivalent functionality and interact in a peer-to-peer fashion. This implies a reduced autonomy because changes on the cooperative process have to be coordinated between the partners. Web-Flow is to support both cooperation models: the first model can be supported by using the web service

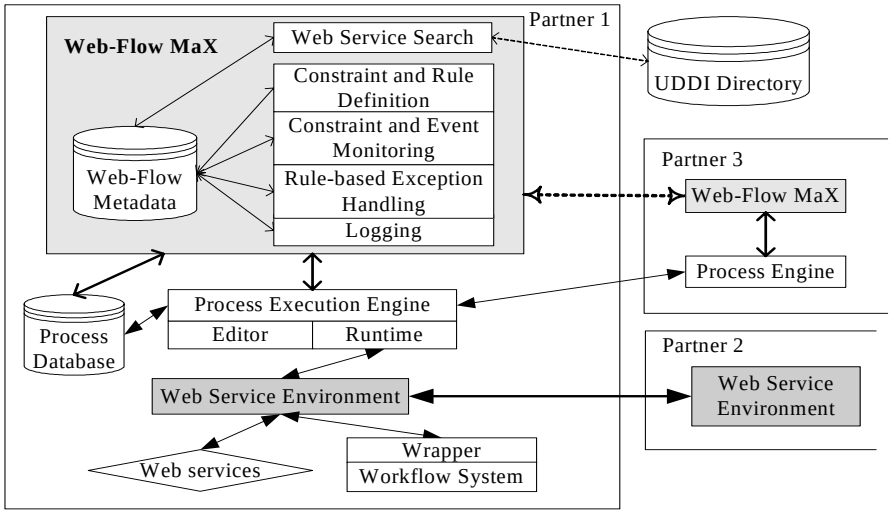

Fig. 1. Web-Flow architecture

functionality, the second by exploiting the direct communication between process engines (Fig. 1). Initially we focus on the simple cooperation model with one partner controlling the cooperative process. This approach supports maximal autonomy for service providers but implies that quality monitoring and exception handling have largely to be performed by the node controlling the cooperative process.

The Web-Flow MaX component has four core parts:

- The *constraint and rule definition* part is used to specify quality constraints for service calls and exception handling rules for events occurring during process execution. The specification has to be done manually, e.g. during process definition.
- The *constraint and event monitoring* checks whether called services violate any quality constraint or whether any other event (e.g., a fault message or a specific database update) occurs.
- The *exception handling* part uses the specified rules to determine whether an event constitutes an exception and how the exception should be handled. The main goal of the exception handling is to successfully continue the cooperative process so that quality constraints are met to the largest extent possible. This may require actions such as execution of additional services or adapting the process to compensate for the effects of the exception.
- A *logging* component records all events and exceptions together with the performed handling. The goal is to use this data for process optimization, in particular to provide recommendations for a manual exception handling and to eventually provide input for defining new exception handling rules.

Further components of the architecture are the *web service search* to use service directories for finding appropriate providers for a task to be executed, and the *Web-Flow metadata* repository maintaining metadata such as constraints and rules for quality monitoring and exception handling. Information about the cooperative processes and

the used services can be derived from the process database.

The Web-Flow MaX component uses a multilevel approach for exception handling. First, exception handling may take place at the site where a web service is executed, either by the web service itself or by the Web-Flow MaX component of the partner. For instance, a web service searching for offers in a particular price range may check the offers before returning them and perform a new search if no suitable results are found. Such mechanisms are typically invisible for the service user but require a sophisticated web service which cannot always be assumed. Therefore, additional levels of exception handling take place in the cooperative process and by the Web-Flow MaX component of the site invoking a remote web service. The Web-Flow MaX exception handling is largely independent of the respective web service implementations (i.e., can also be used for legacy applications wrapped as web services) and process specifications and thus enables enforcement of general exception handling policies.

A rule-based approach has the advantage that the standard process definition is clearly separated from the exception rules, therefore facilitating the readability and maintenance of both. Furthermore, necessary actions are not derived until an exception really occurs, so changes in business policies, laws, etc. can be taken into account without changes in the process definition.

3 Quality Constraints

Based on an analysis of different cooperative business process scenarios and on previous constraint classifications (e.g., [2]) we have identified several types of web-service-related quality constraints to be supported in Web-Flow:

- *Metadata constraints* refer to conditions on the description of web services, e.g. on the services' UDDI (Universal Description, Discovery and Integration, [14]) or WSDL metadata. Such constraints can restrict many different aspects, e.g. the provider of a service (specific companies, specific geographic locations,...) or a fee that may have to be paid for using a service (cost constraint).
- *Execution constraints* refer to conditions on the physical execution of web services, in particular response time limits or maximal number of re-trials for failed executions. Response time constraints specify the maximal waiting time for an answer from a service either as a fixed point in time (date) or as a time interval.
- *Input constraints* pose conditions on the input parameters of a web service call.
- *Result constraints* specify conditions on the result of a web service execution. These conditions typically refer to the XML documents returned by a service. They can be used to check whether the delivery time or price for a product is in an acceptable range or whether a returned product configuration matches the user requirements.

Metadata constraints refer to rather *static* information, which is specified when a service is registered. These constraints are primarily useful for *service selection*, i.e. to find a suitable service during process execution or after an exception has occurred. The other types of constraints are more *dynamic* in the sense that they refer to information related to the actual execution of web services (not only specification). They are of primary rel-

```
<service name="HotelSearchService">
 <operation operationName="hotelReservation">
  <resultConstraint name="RC1"
                   scope="HotelSearchService:hotelReservation"
                   strictness="high">
   <parameterCondition>
    <lessEqual>
     <leftOperand>
      <xPathQuery>/hotelDetails/maximumPrice</xPathQuery>
      <parameter name="hotelReservationRequest" type="Input"/>
     </leftOperand>
     <rightOperand>
      <xPathQuery> /reservation/reservationDetails/price</xPathQuery>
      <parameter name="hotelReservationResponse" type="Output"/>
     </rightOperand>
    </lessEqual>
   </parameterCondition>
  </resultConstraint>
 </operation>
...
 </service>
```

Fig. 2. Sample result constraint

evance for *dynamic exception handling*.

There are several additional properties which characterize quality constraints and their use:

- *Monitoring:* Dynamic quality constraints (response time, input, result constraints) may either be monitored at the *consumer side* or at the *provider side* of a service. The first alternative is typical for a simple cooperation model preserving a high autonomy of service providers. A complex cooperation model may support both approaches.
- *Strictness:* Constraints may be mandatory or only desirable (high vs. low strictness). A service violating a constraint of the latter type may thus still be useful.
- *Scope:* Quality constraints typically refer to a particular service or a service operation, but may also relate to a particular process or user. In addition, there may be *global constraints* applying to several or all services, e.g. to enforce company-wide regulations on eligible service providers / cooperation partners or to specify a default value for maximal response times. Moreover, a constraint may be *context-dependent*, i.e. it should only apply if a service is used in a process under certain conditions, e.g. depending on preceding service calls. For instance, in a medical context the allowed ranges for a blood value may depend on the drugs applied beforehand.

In Web-Flow, a declarative XML-based specification has been defined for each type of quality constraint. Constraints are specified as logical comparison predicates which can be combined to a more complex condition using boolean AND, OR and NOT operators. The XML fragment in Fig. 2 shows an operation-specific result constraint that specifies that the price for a reserved hotel room has to be less or equal than the maximal price specified by the user. To check this constraint the output document of the operation has

to be compared to the input document previously used to call the web service. Therefore the data of the input document has to be stored until the service returns a response and the constraint can be evaluated. The *xPathQuery* tag specifies which part of the document specified by the *parameter* tag has to be used for evaluation.

All quality constraints are defined in the constraint and rule definition component of the Web-Flow MaX and are maintained in the Web-Flow metadata repository. Service and operation specific quality constraints are saved in the Web-Flow extended service description (**.wesd*). Process specific constraints are part of the Web-Flow extended process description (**.wepd*) of a cooperative process. Global constraints are maintained separately.

4 Dynamic Exception Handling

Dynamic exception handling in Web-Flow is based on a rule-based approach using Event-Condition-Action (ECA) rules (as e.g. used in active database systems [15]). Events that may result in an exception include time-outs, web service calls, reception of web service output or fault messages, or manual notifications by e-mail or phone call. Furthermore, database updates (e.g., in a database with customer data or appointments) can also be events. Events are detected by the constraint and event monitoring of the Web-Flow MaX component. It observes all messages that are sent or received by the process engine during process execution.

The optional condition part of a rule is used to check the various quality constraints and to specify additional conditions, e.g. on Web-Flow logging information or metadata, that need to hold so that an event results in an exception. This can be useful to express context-dependent constraints. We use a query processor of a database engine to check metadata, input and result constraints. The violation of a response time constraint can be detected by the monitoring component if no response is received for a synchronous service call until the deadline has expired. An execution constraint specifying a maximal number of re-trials is checked using Web-Flow logging information.

The action part specifies how an exception should be handled. Possibilities include delegating exception handling to the respective process, manual reaction, search for an alternative web service, invocation of a particular web service, process abort, and dynamic adaptation of the calling process, e.g. by adding/deleting activities. In Web-Flow, a default rule is created for each dynamic quality constraint specifying that a user has to be notified if the constraint is violated. These rules can be manually edited to specify alternative automatic reactions.

Fig. 3 shows an example rule specifying that an alternative service operation (operation B of service 2) should be called if the result constraint named RC1 (Fig. 2) is violated and the service has already been called two times. Rule priorities (PRIORITY part) are used to determine the order of rule execution if several rules apply per event (1 = highest priority).

Event processing includes logging the event and checking all relevant rules. This is performed as soon as an event is detected by the constraint and event monitoring component of the Web-Flow MaX. If a rule is found for an event specifying an automatically executable action Web-Flow will execute this action. If execution fails a manual

EVENT	*reception of output message o*
CONDITION	*violation of constraint RC1 AND # iteration > 2*
ACTION	*callService(service2.B)*
PRIORITY	*3*

Fig. 3. Sample exception handling rule

reaction is needed.

To reduce the dependence on manual exception handling we intend to evaluate all logged successful exception handling steps. When an exception occurs asking for a manual reaction (e.g. due to the default rule for quality constraints), the logged information should be searched to present the actions taken in similar exception cases in the past to the user. One difficulty with this step is to find appropriate metrics for similarity of exceptions, e.g. taking into account the service called, parameter values etc.

5 Conclusion and Future Work

To support a high execution quality of web-service-based cooperative processes dynamic handling of exceptions is needed. The Web-Flow approach presented in this paper is based on the specification of a variety of quality constraints for heterogeneous and autonomous web services. A rule-based monitoring and exception handling component automatically deals with many exceptions such as the violation of quality constraints or service faults.

We are currently implementing the monitoring, exception handling and logging algorithms of the Web-Flow MaX component. We intend to evaluate the approach not only in cross-organizational processes in the e-business domain but also in distributed health care scenarios.

Acknowledgements

This work has been supported by the German Research Association under grant Ra 497-12.

References

1. Chappell, D.A., Jewell, T.: Java Web Services. O'Reilly & Associates, Inc. (2002)
2. Keidl, M., Seltzsam, S., Kemper, A.: Reliable Web Service Execution and Deployment in Dynamic Environments. In: Benatallah, B., Shan, M-C. (eds.): TES 2003. Springer-Verlag, Berlin Heidelberg (2003) 104-118
3. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Peer-to-Peer Process Execution with Osiris. In: Proc. of First International Conference on Service-Oriented Computing ICSOC (2003)
4. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. Electr. Commerce Research **3** (1&2) (2003)
5. DAML-S Coalition: DAML-S: Web Service Description for the Semantic Web. In: Proc. of the First International Semantic Web Conference (ISWC), Italy, June 2002 (2002)

6. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In: Proc. of CAiSE 2001 (2001) 140-156
7. Grefen, P., Aberer, K., Ludwig, H., Hoffner, Y.: CrossFlow: Cross-Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. In: IEEE Bull. of the Tech. Comm. on Data Engineering **24** (1) (2001) 52-57
8. Reichert, M., Bauer, T., Fries, T., Dadam, P.: Realisierung flexibler, unternehmensweiter Workflow-Anwendungen mit ADEPT. In: Horster P. (ed.): Proc. Elektr. Geschäftsprozesse - Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen. Klagenfurt (2001) 217-228
9. Luo, Z., Sheth, A., Kochut, K., Arpinar, B.: Exception Handling for Conflict Resolution in Cross-Organizational Workflows. J. of Distributed and Parallel Databases **13** (2003) 271-306
10. Casati, F., Shan, M.-C.: Dynamic and adaptive composition of e-services. Inf. Systems **26** (2001) 143-163
11. Chiu, D.K.W., Cheung, S-C., Karlapalem, K., Li, Q., Till, S.: Workflow View Driven Cross-Organizational Interoperability in a Web-Service Environment. In: Proc. of Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES 2002, Canada (2002) 41-56
12. Andrews, T. et al.: Business Process Execution Language for Web Services, Version 1.1, May 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/
13. Christensen, E. et al.: Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl (2001)
14. Bellwood, T. et al.: UDDI Version 3.0, July 2002. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm (2002)
15. Paton, N. (ed.): Active Rules in Database Systems. Springer-Verlag, New York (1999)