Quality-Oriented Execution and Optimization of Cooperative Processes: Model and Algorithms

Von der Fakultät für Mathematik und Informatik der Universität Leipzig angenommene

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR-INGENIEUR
(Dr.-Ing.)

im Fachgebiet
Informatik

vorgelegt
von Diplom-Informatikerin Ulrike Greiner
geboren am 18.01.1976 in Straubing

Die Annahme der Dissertation haben empfohlen:
1. Professor Dr. Stefan Jablonski (Universität Erlangen-Nürnberg)
2. Professor Dr. Erhard Rahm (Universität Leipzig)
3. Professor Dr. Manfred Reichert (Universität Twente, NL)

Die Verleihung des akademischen Grades erfolgt auf Beschluss des Rates der Fakultät für Mathematik und Informatik vom 23.01.2006 mit dem Gesamtprädikat magna cum laude.
Abstract

Cooperative business processes supporting the collaboration of companies and organizations are of increasing importance as they help to reduce the complexity of cooperation scenarios and optimize time and costs. More and more organizations use Internet technology such as Web services to support their cooperative business processes. Web services are an emerging technology to encapsulate services offered by different providers thus helping to preserve the autonomy of the service providers. Web services may realize a broad range of business functions ranging from database queries or calculations over complex applications to entire workflows. Furthermore, the applications encapsulated by Web services are heterogeneous, reaching from legacy applications over complex automatic applications to manual activities. To implement cooperative processes it is necessary to integrate a large number of heterogeneous services from autonomous providers.

To ensure that such processes reliably serve their purpose and meet the users’ expectations implies achieving a high execution quality of the processes. The execution quality of the process is based on quality characteristics imposed on the services executing the process tasks. The quality characteristics have to be ensured for heterogeneous services and have to be monitored and controlled without violating the autonomy of the service providers. Violation of quality characteristic should be handled dynamically to ensure the overall execution quality and to achieve high flexibility and robustness of the cooperative process. This also applies for other events that may hinder a successful and reliable process execution such as failed service executions.

In this thesis we propose the Web-Flow architecture that provides a new approach to support quality-oriented execution and optimization of processes in cooperative scenarios. The Web-Flow architecture provides a generic infrastructure that allows to monitor and ensure the quality-oriented execution of processes executed in different process engines and with different cooperation models.

Web-Flow provides a classification of quality characteristics describing the execution quality of services. The quality characteristics can be monitored automatically and violations can be detected automatically. To handle constraint violations and other events occurring during process execution Web-Flow offers a rule-based exception handling approach. This approach accomplishes any exception handling that is provided by the process engines executing the cooperative processes and thus contributes to providing a generic solution for quality support. The exception handling is also accomplished with a semi-automatic, log-based derivation of appropriate actions to relieve the administrator from having to define and maintain rules for all possible events. Furthermore, recommendations for the optimization of cooperative processes may be derived based on the analysis of log entries.

The Web-Flow architecture has been implemented in a runnable prototype which shows the usability and applicability of the approach. Also some first evaluations have been performed.
# Table of Contents

Table of Contents ....................................................... iii

CHAPTER 1 Introduction .................................................. 1

1.1 Web Services and Cooperative Processes ......................... 5
  1.1.1 Web Services .................................................. 5
  1.1.2 Workflows and Web-Service-based Cooperative Processes ...... 7
  1.1.3 Cooperation Models for Cooperative Processes ................. 8

1.2 Requirements for Quality-Oriented Process Execution .......... 10

1.3 Contributions of the Thesis ........................................ 11

1.4 Outline ............................................................ 13

CHAPTER 2 Related Work ................................................. 15

2.1 Quality of Service for Web Services and Workflows ............. 16
  2.1.1 Quality-oriented Web Service and Workflow Execution ........ 17
  2.1.2 Quality-Oriented Web Service Composition .................... 19
  2.1.3 Web Service Search ........................................... 20

2.2 Dynamic Exception Handling for Cooperative Processes ........ 22
  2.2.1 Standards and Commercial Systems ............................. 23
    2.2.1.1 The BPEL4WS Standard Proposal .......................... 23
    2.2.1.2 Further Standards ........................................ 26
    2.2.1.3 Commercial Systems ...................................... 27
  2.2.2 Inter-organizational Workflows and Exception Handling ...... 28
  2.2.3 The AdaptFlow Prototype ...................................... 30
2.2.4 Web-Service-based Cooperative Processes ........................................ 32
2.2.5 Active Databases and Rule Engines .............................................. 33

2.3 Deriving Actions for Exception Handling ........................................... 34

2.4 Summary ......................................................................................... 35

CHAPTER 3 The Web-Flow Architecture ................................................. 37

3.1 Web-Flow Architecture: Concepts ................................................... 37

3.2 Web-Flow Architecture: Components .............................................. 40

3.3 Event Detection and Handling .......................................................... 44
  3.3.1 Example scenario ................................................................. 44
  3.3.2 Event Detection and Quality Constraint Monitoring ....................... 45
    3.3.2.1 Initialization Phase ....................................................... 46
    3.3.2.2 Monitoring Phase ....................................................... 47
  3.3.3 Web-Flow Event and Exception Handling ..................................... 48
    3.3.3.1 Event Handling Model .................................................. 48
    3.3.3.2 Selection of Actions ..................................................... 49
    3.3.3.3 Action Execution ....................................................... 50

3.4 Summary ......................................................................................... 52

CHAPTER 4 Web-Flow Metamodel and Quality Constraint Classification 55

4.1 Metamodel for Quality-Oriented Process Descriptions ....................... 56
  4.1.1 Overview ................................................................................. 56
  4.1.2 Semantic Description of Services, Operations, and Parameters ........ 59

4.2 Quality Constraints ........................................................................... 62
  4.2.1 Types and Scope of Quality Constraints ...................................... 62
  4.2.2 Comparison Operators .......................................................... 66
4.2.3 Definitions for Quality Constraints .............................................. 67
4.2.4 Constraint Expressions .......................................................... 72

4.3 Exception Handling Concepts ..................................................... 73
  4.3.1 Events .......................................................... 73
  4.3.2 Actions ......................................................... 76
  4.3.3 Rules .......................................................... 80

4.4 WXS: The Web-Flow quality and eXception Schema ......................... 81

4.5 Summary ........................................................................... 83

CHAPTER 5 Event Detection ................................................................. 87

5.1 Event Detection Overview .......................................................... 87

5.2 Detection and Evaluation of Events ................................................. 90
  5.2.1 Web Service Messages ......................................................... 91
    5.2.1.1 Identification of Process Instances ...................................... 92
    5.2.1.2 Identification of Activities ................................................ 93
  5.2.2 Interfaces for Manual Fault Messages, External Events, and Web Service Search Results ................................................. 94
    5.2.2.1 Interface for Manual Fault Messages and External Events .............. 95
    5.2.2.2 Web Service Search ......................................................... 97
  5.2.3 Timers ........................................................................... 97

5.3 Monitoring of Quality Constraints .................................................. 97
  5.3.1 Initialization .............................................................. 98
    5.3.1.1 Selection of Constraints .................................................. 101
    5.3.1.2 Initialization of Monitoring Tables ....................................... 103
    5.3.1.3 Construction of Constraint Evaluation Queries ......................... 104
  5.3.2 Constraint Evaluation ........................................................ 107
    5.3.2.1 Execution Constraints ...................................................... 109
    5.3.2.2 Input, Result, and Metadata Constraints ............................... 110
5.3.2.3 Updates of Monitoring Tables ........................................ 111

5.4 Summary ................................................................. 112

CHAPTER 6 Dynamic Event and Exception Handling .................. 113

6.1 Dynamic Exception Handling: Overview ............................. 114
   6.1.1 Basic Algorithm .................................................. 114
   6.1.2 Handling of Inter-Dependent Events ............................... 117

6.2 Selection of Actions ...................................................... 117

6.3 Handling of Pending Service Calls .................................... 120

6.4 Service Call Related Actions .......................................... 122
   6.4.1 Operator repeat-service-call .................................... 124
      6.4.1.1 Steps 1 & 2: Consideration of Quality Constraints .......... 125
      6.4.1.2 Step 3: Repetition of Service Call ............................ 126
      6.4.1.3 Monitoring of Quality Constraints During Action Execution .. 127
      6.4.1.4 Step 5: Decision about Repetition ............................ 128
   6.4.2 Operator suspend-service-call ................................... 129
      6.4.2.1 Suspend-Service-Call Before Operation Execution ............ 129
      6.4.2.2 Suspend-Service-Call After Operation Execution ............. 130
   6.4.3 Operator alternative-service-call ................................ 132
      6.4.3.1 Variant 1: Same Operation - Different Parameters ............ 133
      6.4.3.2 Variants 2 & 3: Different Service Operation .................. 135
      6.4.3.3 Web Service Search in Service Directories .................... 138
      6.4.3.4 Mapping of Input and Output Parameters ....................... 139
   6.4.4 Side Effects of Operator Execution .............................. 140
      6.4.4.1 Handling of Temporal Effects ................................ 141
      6.4.4.2 Missing Output Data ........................................ 143

6.5 Operator procedure-call ............................................. 143
   6.5.1 Operator Description ............................................ 143
6.5.2 Side Effects of Operator Execution ........................................ 145
6.6 Operator abort-process ......................................................... 146
  6.6.1 Operator Description ...................................................... 146
  6.6.2 Side Effects of Operator Execution .................................... 147
6.7 Processing of External Events .............................................. 148
6.8 Summary ............................................................................. 149

CHAPTER 7 Log-based Selection of Actions and Recommendations . . . 151
7.1 Gathering Log Data .............................................................. 152
  7.1.1 Structure of Web-Flow Log Entries .................................... 152
  7.1.2 Derivation of Execution Durations ..................................... 155
7.2 Selection of Actions for Event Handling ................................. 156
  7.2.1 Calculation of Similarity .................................................. 156
    7.2.1.1 Similarity of Process Definition Parts ........................... 157
    7.2.1.2 Similarity of Event Parts ......................................... 158
    7.2.1.3 Similarity of Operation Parts .................................... 161
    7.2.1.4 Syntactic Comparison of Operation Parts ..................... 162
    7.2.1.5 Semantic Comparison of Operation Parts ..................... 164
    7.2.1.6 Aggregation of Similarity Values ................................ 171
  7.2.2 Ranking of Actions ......................................................... 172
  7.2.3 Performance Improvements ............................................. 178
7.3 Generation of Recommendations and Rules ............................. 178
  7.3.1 Analysis of Log Data ..................................................... 179
  7.3.2 Generation of Rules ...................................................... 182
7.4 Summary ............................................................................. 185
1 Introduction

Companies and organizations concentrate on their core competencies to gain competitive advantage [PRAH1990]. They collaborate with several partners and thus become part of cooperative scenarios. One important example for cooperative scenarios are supply chains which aim at producing and delivering goods and services to customers. More and more companies use Internet technologies, such as e-mail, Web-EDI (Electronic Data Interchange) [UNEC2004], electronic marketplaces, and Web services, to reduce the complexity of their supply chains and to optimize time and costs. Thus, electronic cooperative business processes supporting cooperative scenarios are of increasing importance for companies and organizations [CORS2002, FÄHN2004].

Web services are an emerging technology to encapsulate services offered by different organizations that collaborate in cooperative scenarios. Web services provide a machine-readable, XML-based interface for automated service calls [CHAP2002], hence allowing the service providers to preserve their autonomy with respect to the internal realization of the services. Web services may realize a broad range of business functions ranging from rather simple functions such as database queries or calculations (for some examples see [XMET2004]) over applications to entire workflows. Web services may also encapsulate manually performed services or existing complex legacy applications which only provide a rather simple interface for calling the application and returning the results. To implement complex cooperative processes it is typically necessary to integrate a large number of Web services of different complexity and from independent providers. The increasing availability of Web services and Web service implementations leads to an increasing number of cooperative business processes using autonomous and heterogeneous services of different organizations [LEYM2002].

Fig. 1-1 shows a collaborative fulfillment process from the perspective of a personal computer (PC) manufacturer to illustrate the complexity of Web-service-based cooperative processes and the number of services and operations involved. The different lanes group the services of the participating providers. In particular, these are the PC manufacturer providing customer related services as well as internal manufacturing and order processing services, the monitor supplier, the printer supplier, and the distribution center. Other examples of cooperative processes include
Introduction

Collaborative fulfillment describes the process where different suppliers work together in a supply network to produce and deliver products to customers in time. In the example a customer enters an order for an individually configured personal computer (PC) with monitor and printer at the PC manufacturer. The grey activities denote customer related services of the PC manufacturer in contrast to manufacturing and order processing services.

Fig. 1-1: Collaborative fulfillment (swimlane presentation).

The treatment of a patient by different physicians and hospitals, a travel-booking process containing several services for booking a flight, hotel, etc., or procurement processes [GRAF2002].
manufacturer’s website. After entering the configuration different available-to-promise (ATP) checks are performed to either confirm delivery dates demanded by the customer or to suggest a possible delivery date for the order. The customer places the order if he is satisfied with the configuration of the PC and the delivery date. Then the system performs a credit check. If the customer passes the check the assembly of the order is scheduled and initiated. In particular, the PC has to be assembled, the monitor has to be ordered from an OEM (Original Equipment Manufacturer) and both components have to be sent to a distribution center (DC). The printer has to be procured from an external supplier which has been chosen during order entry from several suppliers based on the printer configuration and the ATP check. The printer is sent directly to the distribution center. Finally, the components are packed at the DC and transported to the customer.

Cooperative processes are typically long running and use messages to communicate with the Web services used. To ensure that such processes reliably serve their purpose and meet the users’ expectations implies achieving a high execution quality of the processes. The execution quality of a cooperative process is based on quality characteristics imposed on the services executing the process tasks. However, the applications behind the services can be heterogeneous, reaching from legacy applications over complex automatic applications to manual activities. To achieve a high execution quality of a cooperative process a high execution quality of all these different types of applications has to be ensured. However, depending on its complexity a service may only provide limited / no features to control execution quality. For example, some exception handling may be performed inside the service operation. But process execution quality typically also comprises quality characteristics that are not covered by the service internal exception handling. For instance, quality aspects may depend on the concrete process context in which a service operation is used. Thus, an automated quality monitoring on the process level is necessary. Furthermore, the organizations participating in a cooperative process use Web services to encapsulate the internal realization of their services and to preserve their autonomy. Thus, it is not desired to influence the service execution quality from the outside. This makes the task of ensuring the service execution quality in the context of cooperative processes still more challenging.

Violation of quality characteristic imposed on a service execution should be handled dynamically to ensure the overall execution quality and to achieve high flexibility and robustness of the cooperative process. This also applies for other events that may hinder a successful and reliable process execution such as fault messages notifying a service consumer about faults that occurred during service execution. A manual monitoring of quality characteristics and events is inefficient and may lead to events that are detected too late or not at all. Thus, a semi-automatic approach should be used to relieve administrators from manual event / exception handling. Furthermore, cooperative processes may be controlled not only by one organization but be distributed between several organizations. All organizations should have a common notion of process execution quality and service quality characteristics to be able to monitor and ensure the execution quality of the cooperative process in a coherent way.

To support process execution quality four different categories of approaches may be considered.
A classification of the categories is given in Fig. 1-2. The categories can be distinguished along two dimension: first, whether they focus on exception handling or exception prevention, second whether the approach considers the context generated by a concrete execution of a process instance for exception handling or not. The latter reflects the fact that service operations or exception handling steps that seemed appropriate at definition time may not be appropriate any more during runtime. The different categories provide the following features for ensuring the execution quality:

1. **Selection of the best fitting service operation during process definition:**
   That service operation is chosen to execute a particular activity in a cooperative process which promises to fulfill the task best. The selection is based on descriptions and promises given by the service provider and experiences from earlier service calls.

2. **Selection of the best fitting service provider during process execution:**
   In this category the service operation for executing a particular activity is not selected until the activity is in fact executed. Thus, current execution parameters can be considered for service selection.

3. **Integration of event and exception handlers in process definition:**
   Several existing solutions for quality support of cooperative processes focus on detecting events such as timeouts by incorporating event and exception handlers in each cooperative process definition (for example, [THAT2003, CHAP2004]). The exception handling steps are executed during process execution if the specified exception occurs.

---

Fig. 1-2: Different categories for ensuring quality of cooperative process execution.
4. **Definition and monitoring of additional quality conditions for service execution and flexible event handling during process execution:**

This category provides a generic solution which allows for a homogeneous definition and monitoring of quality characteristics for cooperative processes defined and executed in different process engines. Furthermore, quality monitoring may be applied for different types of services, thus enhancing the execution quality of services that do not offer sufficient support themselves (for instance, legacy applications). Additionally, quality monitoring and exception handling may be widely automated to relieve administrators. However, still sufficient user interaction may be offered to provide control about possible changes in process instances.

In this thesis we propose an architecture that provides comprehensive support for quality-oriented execution of cooperative processes in the fourth category, accomplishing quality support offered in the first three categories. It offers an infrastructure to define and monitor quality characteristics and to handle events / exceptions dynamically and semi-automatically. The quality monitoring may be used for different types of services. Furthermore, recommendations for possible optimizations of processes are generated based on analysis of logged execution data. The architecture is independent of concrete process engines and thus is usable with different process engines.

In the following, we first present the technical concepts and standards for Web services and Web-service-based cooperative processes which provide the basis of the infrastructure described in this thesis (Section 1.1), before we formulate requirements for quality-oriented process execution and optimization in Section 1.2. Finally, we discuss the contributions of this thesis (Section 1.3) and give an outline (Section 1.4).

### 1.1 Web Services and Cooperative Processes

In this section we shortly sketch the main concepts and technologies for Web services, and Web-service-based cooperative business processes.

#### 1.1.1 Web Services

Web services enable organizations to expose business functions provided by their internal applications to service consumers which can access them using standard Web technology. According to the definition of Web services given by the World Wide Web consortium (W3C) a Web service is

> “a software application that is identified by an URI [Uniform Resource Identifier], whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.” [W3C2002]
Introduction

So, Web services should be available through the Internet using Internet protocols, such as HTTP (Hypertext Transfer Protocol) or SMTP (Simple Mail Transfer Protocol), and they should be advertised so that clients may be written that interact with the Web services. The interaction either may be a remote procedure call to an application hosted by the service provider or may be used to exchange complex business documents represented in XML [CHAP2002].

Further properties of Web services are:

- Web service execution can be stateless, i.e., it does not depend on the outcome of preceding service executions, as well as stateful, i.e., it considers preceding service results. Stateful Web service executions occur if Web services are used to encapsulate processes [LEYM2002].
- Web services may be executed synchronously or asynchronously. For the former the service consumer waits until the Web service returns a response, for the latter it just calls the service and then executes different functions. Results of an asynchronous service call may be retrieved at a later time.
- Service provider and service consumer are loosely coupled. That means that changes in the application implementing the Web service do not affect the service consumer as long as the Web service interface remains unchanged. If the interface of a Web service is changed, the service consumer does not have to change the business functions using the service but only the service calls.

A Web service typically comprises one or more operations that may be called by the service consumer sending an XML-based input message. Whether the Web service returns an output message depends on the type of the operation (synchronous or asynchronous). Additionally, in case of execution failures each service operation may also return fault messages which are defined in the service description.

Web services are published and used in a service-oriented architecture (SoA) [BURB2000]. Fig. 1-3 shows the main actors and parts of a SoA and their relationships. The service provider „publishes“ its service descriptions in a service directory in which a service consumer may „find“ an appropriate service for a task. Finally, the service consumer „invokes“ the service by sending XML messages to the service provider. Several Web service related standards have been defined to support the different tasks executed in a SoA. Currently, there are three main XML-based languages respectively protocols supporting the implementation and use of Web services: SOAP [SOAP2003], WSDL [WSDL], and UDDI [BELL2003]:

- SOAP [SOAP2003], provides a mechanism for the transport of XML documents over different standardized Internet protocols such as SMTP, HTTP, or FTP (File Transfer Protocol).
- WSDL, the Web Service Description Language [WSDL], allows for a standardized XML-based description of a Web service. A WSDL specification typically is divided in an abstract and a concrete part. The abstract part contains the definitions of the input, output and fault messages exchanged with the operations of a Web service. These definitions are usually
Web Services and Cooperative Processes

based on datatype definitions in an XML schema. The datatypes and messages may be used solely between two cooperation partners or may e.g. be industry-wide.

Furthermore, the abstract part of a WSDL service description contains a description of the operations offered by a Web service and defines the input, output and fault messages for each operation. The operations are grouped in porttypes which define a sequence of messages which a Web service sends and/or receives to execute a particular business function. Several porttypes may be specified to define different message sequences in dependence of the protocol used to call the operations.

The concrete part of a WSDL service description defines bindings for the protocols over which a service may be contacted, e.g. SOAP, CORBA Internet Inter-ORB Protocol (IIOP), .NET, Java Message Service (JMS), or WebSphere MQ. Furthermore, the concrete part contains the network addresses for the different porttypes, the so called ports or endpoints over which the operations of the Web service may be accessed.

- UDDI, the Universal Description, Discovery and Integration [BELL2003], provides a specification for a directory in which Web services may be published by the service provider and searched by the service consumer. It supports the description and discovery of businesses, organizations, and other Web service providers, the Web services they make available, and the technical interfaces which may be used to access those services.

1.1.2 Workflows and Web-Service-based Cooperative Processes

Workflows are used to model and execute clearly structured, frequently repeated processes whose definition is only seldom changed. They offer a strict separation of application program code from
Introduction

the overall process logic as well as the integration of automated and manual activities and of data from different sources (for example, [AALS2002A, GEOR1995, JABL1996, LAWR1997]). Workflows consist of a set of tasks that are executed in a particular order. The ordering of the tasks is explicitly specified in the workflow definition. For instance, in the process executed inside the PC manufacturer’s organization in Fig. 1-1 first a credit check is performed and then the order fulfillment is initiated. Workflow management systems offer tools to support the definition and execution of workflows as well as the integration of applications and users during workflow execution [WFMC].

Workflows typically focus on supporting the processes executed inside one company or organization. As cooperative processes have gained significant importance during the last years workflows have been extended to also support inter-organizational workflows which cross organizational boundaries. As Web services emerged several proposals for the composition of Web-service-based business processes appeared, for instance, BPML [ARKI2002], WSFL [LEYM2001] or XLANG [THAT2001]. However, none of these became really accepted. In 2002 the Business Process Execution Language for Web Services (BPEL for short) [THAT2003] was proposed as a standard for the definition of Web-service-based cooperative business processes by BEA, IBM, and Microsoft. In the following year many other vendors on the field of business process management and Web services such as Oracle, SAP, or Siebel joined the standardization committee for BPEL. Now BPEL is developed and standardized by a technical committee of OASIS (Organization for the Advancement of Structured Information Standards1).

BPEL is based on standard languages and protocols such as XML, WSDL, or SOAP. Each cooperative process defined with BPEL has a WSDL description and can thus be published as a complex Web service itself. In this thesis, we primarily focus on cooperative processes defined with BPEL to be close to the existing standards. Furthermore, it may then be expected that our proposed architecture is compatible with several process engines from different vendors supporting the development and implementation of BPEL.

1.1.3 Cooperation Models for Cooperative Processes

Two different cooperation models for Web-service-based cooperative processes are considered in this thesis. A simple cooperation model corresponds to a workflow whose activities refer to external Web services. Thus, the service providers of the integrated services do not know the overall process. This type of cooperation is also known as orchestration of a process. The cooperative process in Fig. 1-1 may be executed according to the simple cooperation model if only the PC manufacturer knows the overall process and the partners do not. In a complex cooperation model, the involved partners agree upon a cooperative process (including the services used and the order in which they are called) that is then executed symmetrically, i.e. with equal partners. The collabor-
The cooperative fulfillment process in Fig. 1-1 is executed in the complex cooperation model if all four partners (PC manufacturer, monitor and printer supplier, distribution center) are aware of the common process they execute. In the complex model, the control over a process instance may be shared between the partners, as, for example, one partner is only involved in the first half of the process and another partner in the second half. Especially in long running processes it may be a good solution to share the control over one process instance to relieve the resources of the partners as soon as possible. The sharing of control implies that changes on the process definition have at least to be propagated to or even agreed upon by all partners. This model represents one form of process choreography in which a central instance exists controlling the execution of the process. Another form of choreography that also allows virtual processes that are not executed and controlled by a central instance is not addressed in this thesis. In the following we compare the two cooperation models in more detail.

The different service operations may either be called synchronously, i.e., the process engine waits for a response from the operation before continuing the process, or asynchronously, i.e., the engine does not wait for a response before continuing the process execution. If response messages for asynchronous service calls may be received later in the process access points may be offered for the external services to which those may send the response. Furthermore, each response message has to be correlated to the request message using some identifier (for instance application data). This identifier has to be agreed upon between the partners which may be aligned with the complex cooperation model in which the cooperation partners know about the cooperative process. Thus, in the complex cooperation model synchronous and asynchronous service calls may be used. In the simple cooperation model only synchronous service calls and asynchronous service calls without response are possible, as the partners do not know that their service is used in a cooperative process. Thus, no recipient for the response message is known and no correlation mechanism is available.

| Knowledge about process at all partners | Simple cooperation model | Complex cooperation model |
| Equal of partners | no | yes |
| Sharing of control | no | yes |
| Service calls | synchronous (asynchronous without response) | synchronous and asynchronous |
| Coupling | loose | tight |
| Autonomy | maximal | restricted |
| Durability of cooperation | long-term or short-term | long-term |

Table 1-1: Comparison of the cooperation models.
Introduction

Regarding the coupling of the cooperation partners and their autonomy the simple cooperation model only needs a loose coupling of the partners. They just publish the minimal information that is needed to call their services and do not make any other agreements. Thus, the simple cooperation model offers the maximal autonomy for the service providers. In the complex cooperation model the partners have agreed on a common cooperative process and possibly also a business protocol, i.e., a set of messages and the order in which those are exchanged. Thus, the partners are coupled more tightly than in the simple cooperation model. Also, their autonomy is restricted as they cannot change the type or order of the messages sent during the process without threatening the successful execution of the cooperative process. If one partner wants to perform some changes, all partners have to agree on the changes and possibly have to change their process part as well.

Due to these dependencies between the partners cooperative processes using the complex cooperation model are mainly useful for long-term partnerships, as the additional effort to set up those processes is high. In contrast, the simple cooperation model can be used for both long-term and short-term cooperations. Only the partner controlling the process decides how long he wants to use a service of a particular service provider or a particular cooperative process definition.

Tab. 1-1 summarizes the features of the two cooperation models. The simple cooperation model supports maximal autonomy for service providers but implies that quality monitoring and event/exception handling have largely to be performed by the partner controlling the cooperative process. The complex cooperation model focusses more on cooperative scenarios with equal partners that may also share the control over a cooperative process, for instance, to optimize resource usage. This restricts the autonomy of the partners to a certain degree. Event handling becomes more sophisticated in this model as additionally implications for the services executed by partners have to be communicated to them during event handling.

1.2 Requirements for Quality-Oriented Process Execution

Cooperative business processes are long running and may include many different services and partners. To ensure that those processes reliably serve their purpose a comprehensive support for quality-oriented execution is required. In the following we formulate requirements for appropriate support of quality-oriented process execution to whose fulfillment this thesis may contribute.

A quality-oriented process execution should support dynamic handling of events and exceptions to support flexibility and robustness of cooperative processes. This means that actions are executed after an event or exception occurred to ensure that a process may be successfully continued despite this event. However, Web services used in cooperative processes are heterogeneous and differ in their complexity (for instance, legacy applications as well as complex interactive applications can be encapsulated in a Web service). Thus, dynamic event handling should provide actions that are applicable to all these different types of services. The actions should be executed automatically, without violating the autonomy of the organizations providing the services, and
Contributions of the Thesis

should cover a large number of events and exceptions. However, additionally sufficient control functions such as user confirmations should be provided before an action is in fact executed. To reduce the necessity of manual intervention to handle events a rule-based approach should be used to derive appropriate actions automatically. It is typically not possible to cover all possible events with rules. Furthermore, large rule sets become difficult to maintain. Thus, the rule-based approach should be accomplished by an automatic derivation of actions based on logged execution and event handling data.

Automatic event detection is another requirement that is closely related to dynamic event and exception handling. To be able to detect events that may violate the execution quality of a cooperative process a formal description of the desired execution quality is needed. Therefore quality characteristics should be defined that can be imposed on services executed in cooperative processes. These quality characteristics should be applicable to heterogeneous services and should be monitored efficiently and automatically without violating the autonomy of the cooperation partners. They should be observable without knowing details of the service implementation. Also, they must not conflict with any quality characteristics already offered by the application implementing a Web service. Thus, the additional characteristics should not replace but accomplish existing ones. The availability of such quality characteristics reduces again the necessity of manual intervention and helps to detect events in time. Furthermore, it ensures that no events are overlooked if at the same time many instances of different process definitions are executed in parallel in one process engine.

An architecture supporting a quality-oriented execution of cooperative processes should offer a generic solution that may be used with different cooperative process engines without major adaptations. Thus, the execution quality of several cooperative processes executed in different process engines may be defined and monitored in a uniform way throughout one organization. The mechanisms offered by the architecture should be compatible with Web services and be based on Web service standards such as XML, WSDL, SOAP, and UDDI as these help to abstract from the concrete execution platform of services.

As cooperative processes may use different cooperation models (cf. Section 1.1.3) the provided solution should support different models to make it flexibly usable with different engines. Process engines may also already provide different exception handling mechanisms. As a generic solution should work with different engines the offered quality-related mechanisms should not replace but accomplish the existing mechanisms.

1.3 Contributions of the Thesis

Based on the discussed requirements we now describe the contributions of this thesis. The main contribution is the design of the service-based Web-Flow architecture which provides the necessary infrastructure to support quality-oriented execution and optimization of processes in cooper-
Introduction

ative scenarios. The following list describes the contributions in detail.

1. **The Web-Flow architecture:** The Web-Flow architecture provides the necessary infrastructure for the support of quality-oriented execution and optimization of cooperative processes. Web-Flow offers automatic quality monitoring and event detection as well as event / exception handling which accomplishes mechanisms already available in cooperative scenarios. Web-Flow is Web-service-based and supports quality support for process orchestration as well as for process choreography (cf. Section 1.1.3). The infrastructure is implemented in a dedicated component called Web-Flow MaX (Monitoring and eXception handling) which may be used together with different process engines executing cooperative processes.

2. **A metamodel for the description of quality-related concepts:** A metamodel of concepts that are related to a quality-oriented process execution is provided. The concepts comprise amongst others:

   - a classification of quality constraints to describe the desired execution quality of heterogeneous services as well as of whole cooperative processes.
   - semantic information about a service such as service categories, operation functionality, and parameter roles,
   - events (such as the violation of quality constraints or failed service executions) that may occur during process execution and may lead to the violation of the desired execution quality of the cooperative process,
   - actions that can be executed automatically to handle events and that ensure that the process may be continued despite an event with the best execution quality possible,
   - rules that define appropriate actions to handle occurring events.

   The metamodel additionally offers concepts to describe cooperative processes, activities, services, and operations which are used for quality monitoring, event detection, and event / exception handling. The XML-based schema WXS (Web-Flow quality and eXception Schema) is defined as a machine- as well as human-readable representation of these concepts.

3. **Mechanisms for quality monitoring and automated event detection:** We provide mechanisms that allow for the automated monitoring of the quality constraints defined in the metamodel. The quality monitoring can be applied to different types of services even if they do not offer quality support by themselves (as it is the case, for instance, for legacy applications). Furthermore, we provide mechanisms to monitor other events that may occur during the execution of Web-service-based cooperative processes and may lead to exceptions. These events may be fault messages sent by Web services to the process engine, manual fault messages (i.e., fault messages received over phone or e-mail) or external events. Predefined message formats allow for an automated evaluation of manual faults and external events.
Outline

4. **Dynamic event / exception handling and semi-automatic action determination**: We provide a rule-based approach to handle events / exceptions and to ensure the execution quality of cooperative processes. This approach is accomplished by a log-based mechanism to derive appropriate actions semi-automatically from logged execution data. For the latter we provide mechanisms to determine similar cases for an event under the consideration of auxiliary conditions such as the success of the former event handling or the violation of further quality constraints. We provide a set of operators that implement the actions and can be executed semi-automatically, i.e., after a final confirmation by an administrator.

5. **Log-based optimization**: The execution quality of cooperative processes may also be improved by regular revision and optimization of the cooperative process definition based on recommendations derived from execution data logged during former process executions. Recommendations may, for instance, consider the response time of services for optimizing the overall execution time of a cooperative process or measure the correctness of the results delivered by a service [KÜNG2003]. We offer mechanisms that analyze the log data and generate recommendations for process optimizations. The recommendations may be used by process designers as a starting point for process revision.

A prototypical implementation of the developed concepts and mechanisms is also described in this thesis. This implementation is used to evaluate the usability of our approach of a generic architecture to support quality-oriented execution of cooperative processes.

1.4 **Outline**

We conclude this chapter with an outline of the thesis. In Chapter 2 we discuss related work to distinguish our contributions from other approaches. The Web-Flow architecture will be introduced in Chapter 3 and the main components will be described. Chapter 4 describes the Web-Flow metamodel and introduces the Web-Flow quality and eXception Schema WXS.

In Chapter 5 we describe the mechanisms used in Web-Flow to automatically monitor events and quality constraints. Chapter 6 describes the rule-based exception handling approach and the operators that implement the derived actions and realize the dynamic event and exception handling. The semi-automatic derivation of appropriate action from logged execution data that accomplishes the rule-based event / exception handling is subject of Chapter 7. The log-based optimization of cooperative processes is also described there. Chapter 8 provides an overview over a prototypical implementation of the Web-Flow architecture. Chapter 9 concludes the thesis with a summary and an outlook on future work.
Introduction
2 Related Work

As discussed in the introduction execution quality for cooperative processes may be supported in four different categories (see the front pane of the box in Fig. 2-1). For the evaluation of work related to the Web-Flow architecture developed in this thesis we consider an additional dimension: we separate the approaches whether they focus on the Web service provider side or on processes / the service consumer side. Thus, we get eight categories to classify related work.

To clearly characterize and separate our approach we discuss related work from the different categories and compare them to the following main requirements for quality-oriented execution of cooperative processes:

- a classification and metamodel of quality characteristics that can be applied to heterogeneous services and can be monitored without violating the autonomy of the service providers;
- automated monitoring of the execution quality and automated detection of events such as violation of quality conditions, fault messages, and external events which may occur during process execution;
- flexible, dynamic handling of events and exceptions based on a rule-based approach accomplished by a log-based, semi-automatic derivation of appropriate actions;
- a generic solution which can be used with different process engines and supports different cooperation models.

These requirements all relate to approaches in the category in the front top right corner of the category dice (denoted by thick borders in Fig. 2-1). They focus on support of event detection, flexible exception handling during process execution thus considering the current process execution context, and are applied on the service consumer side.

Related work covers the following aspects: quality of service for Web service and workflow execution, quality-oriented composition of complex Web services as well as cooperative processes and exception handling. For the last item we consider two types of approaches. Before the emergence of Web services, cooperative processes were mostly realized as distributed or inter-organizational workflows. As Web services and Web service implementations became available several researchers and standardization efforts focused on service composition to define cooperative pro-
cesses on the basis of Web services. Both fields have found considerable interest. But as for this thesis approaches supporting event and exception handling for cooperative processes are of particular interest, we emphasize approaches addressing this problem.

In the following, we first discuss related work on the field of quality of service for Web service and workflow execution and Web service composition (Section 2.1). Then we concentrate on cooperative processes and particularly consider approaches supporting exception handling (Section 2.2 and 2.3). Finally, we relate the discussed approaches to the categories shown in Fig. 2-1 and summarize the discussion of related work (Section 2.4).

2.1 Quality of Service for Web Services and Workflows

Supporting quality of service for Web services and workflows has found considerable interest recently. We discuss two main fields to support Web service quality: the definition of quality characteristics for Web service execution (also known as service level agreements) which are monitored at the provider side and the support of a quality-oriented composition of complex Web services by the service consumer. Quality of service for workflow execution has also been considered by several researchers. In the following, we first discuss several approaches on these fields and also focus on approaches addressing Web service search in the context of service composition.
2.1.1 Quality-oriented Web Service and Workflow Execution

The approaches for quality-oriented Web service execution cover dynamic resource management (for instance, load balancing) to support sufficiently fast Web service execution (for instance, [KEID2003]), the definition and monitoring of service level agreements [CASA2003, KELL2003, SAHA2002], response time guarantees [KRAI2002], and the description of policies that have to be fulfilled to use a Web service successfully [HOND2003]. All these approaches consider the current process execution context, focus on the service provider, and deal with exception prevention (i.e., they relate to the back lower right sector in Fig. 2-1).

Keidl et al. [KEID2003] describe a generic dispatcher service which augments Web services with load balancing and high availability features, without having to consider these features at service development time. The dispatcher service is also capable of automatic service replication which allows to install new instances of static services on demand. The dispatcher service has been implemented within the ServiceGlobe system [KEID2002], an open Web service platform on which Web services can be implemented, stored, published, discovered, deployed, and dynamically invoked at arbitrary Internet servers participating in the ServiceGlobe federation.

Keller and Ludwig [KELL2003] provide the “Web Service Level Agreement“ (WSLA) framework, an approach for the definition of service level agreements (SLAs) to describe quality of service parameters for Web service execution, such as response time, availability, reliability, cost, or throughput. The WSLA framework consists of a flexible and extensible language based on XML schema and a runtime architecture comprising several SLA monitoring services that enforce the SLAs. The monitoring services may be outsourced to third parties to ensure a maximum of objectivity. Sahai et al. [SAHA2002] propose an automated and distributed SLA monitoring engine that allows for precise and unambiguous specification of SLAs. It collects the right measurements, models the data, and evaluates the SLAs at certain times or when certain events happen. Furthermore, it obtains measurements at the server as well as the client sites and guarantees SLAs on them which is especially important in cross-enterprise scenarios such as Web service execution.

The Web Services Policy Framework (WS-Policy, [HOND2003]) provides a general purpose model and corresponding XML-based syntax to describe and communicate the policies of a Web service. Policies must be followed by service consumers in order to be able to interact with a Web service successfully. WS-Policy defines a base set of constructs that can be used to describe a broad range of service requirements, preferences, and capabilities, such as authentication or encryption algorithms supported by a Web service. Policy expressions allow for both simple declarative assertions and more sophisticated conditional assertions.

Casati et al. [CASA2003] have addressed monitoring and business-oriented management of Web service execution. The focus is on supporting service providers to monitor the execution of their Web services and the execution environment to ensure they operate with the desired quality levels. The logged service execution data may be mapped to appropriate, business-relevant metrics (e.g., response times or cost of service execution). Using these metrics reports may be generated which
Related Work

can be used by managers to check if their services fulfill all promised service level agreements or to detect weak points.

Other approaches addressed quality of service not on the Web service level but for whole workflows. In the context of the METEOR-S project\(^2\) Cardoso et al. [CARD2004] present an approach to support quality of service (QoS) for workflows and Web service processes used to manage services or to create products. The authors state that the underlying workflow engine must accept the QoS specifications and be able to estimate, monitor, and control the QoS promised to customers. They present a predictive QoS model that makes it possible to compute the quality of service for workflows automatically based on atomic task QoS attributes. The QoS model considers three dimensions namely time, cost, and reliability. The time dimension considers the task response time, i.e., the time a task needs for setup and execution. The cost dimension describes the costs associated with the task execution, e.g., direct labor cost, direct material cost, or machine cost. The reliability dimension describes whether a task has been executed successfully or has failed. For each workflow task estimates for the different QoS dimensions are set at design time and recomputed from time to time taking into account log data of former workflow executions. For the overall workflow, quality of service is calculated based on the QoS estimates for the single tasks using the graph reduction algorithm SWR(w), which reduces the workflow graph until only one activity is left. This activity contains the QoS metrics for the whole workflow process.

The implementation of the QoS model requires modifications or adding of several components of the workflow system. Thus, the QoS model is tightly coupled with a particular workflow engine. Furthermore, the reliability dimension only considers whether a task was successful or has failed and does not consider the quality of inputs or results of single workflow tasks. The authors also describe that potential alternatives for workflow completion may be evaluated if the QoS metrics for the workflow are going to be violated. In order to complete a workflow according to initial QoS requirements, it is necessary to adapt, replan, and reschedule a workflow in response to unexpected progress, delays, or technical conditions. However, the generation and evaluation of alternatives is not discussed in detail.

Other approaches addressing quality of service for workflows are [GILL2002] and [KLIN1999b]. Gillmann et al. [GILL2002] describe a tool that largely automates the task of configuring a distributed workflow system with several replicated workflow servers to meet guaranteed quality of service in terms of throughput, response time, and tolerable downtime. The tool is based on a suite of mathematical models such as continuous-time Markov chains and derives the necessary degrees of replication for the various server types.

Klingemann et al. [KLIN1999b] address the support of quality of service for cross-organizational workflows using a service-oriented workflow model. They present a technique how to derive a model of external services, based on continuous-time Markov chains, by analyzing the externally observable behavior of a service. The derived model describes the quality of a service, e.g. in

---

Quality of Service for Web Services and Workflows

terms of response time or cost, based on log data gathered during former service executions, and may be used to predict the quality of service for new service executions. The approach allows to assess the quality of external services without compromising the autonomy of the service providers. However, other dimensions of quality such as conditions on input or result parameters of services are not considered.

2.1.2 Quality-Oriented Web Service Composition

Several studies focus on a quality-oriented composition of complex Web services. Some address the dynamic selection of the best providers for a particular Web service (for example, [KEID2003, SCHU2003]) depending on the current execution state of a process. These approaches are applied by the service consumer during process execution and aim at exception prevention (i.e., they belong to the front lower right sector in Fig. 2-1). Others use ontology-based semantic annotations for a sophisticated description of services which may improve service selection and composition (for example, [ANKO2002, OWLS2003, TESC2002, MEDJ2003A]). A programming language for the implementation of complex Web services is described in [FLOR2002A]. These approaches focus on exception prevention and do not consider the process execution context as they are applied during process definition time by the service consumer (i.e., they belong to the front lower left sector in Fig. 2-1).

Dynamic service selection as described by Keidl et al. [KEID2003] provides a layer of abstraction for service invocation offering Web services the possibility of selecting and invoking other Web services at runtime based on a technical specification of the desired service (a so-called tmodel). The selection can be influenced by using different types of user-defined constraints. For instance, metadata and location constraints may be used as filters on all services returned for a particular tmodel from a UDDI registry. Mode constraints allow to define whether one, some or all services returned by a service search are to be invoked. Other conditions apply for service execution. Reply constraints are evaluated on replies received from a service and may either select some parts of the reply or check whether properties such as encryption, signature or age of data are fulfilled. Furthermore, maximal waiting times for service replies may be defined or it may be specified that after receiving replies from a particular number of services all pending service calls may be aborted. The dynamic service selection has also been implemented within the ServiceGlobe system [KEID2002] which allows for the definition of complex services based on several simple Web services.

Schuler et al. [SCHU2003] propose a distributed and decentralized process engine that routes process instances combining a number of services to applications directly from one node to the next ones that may execute the next service. Thus, it is able to dynamically balance the load of processes and services among all available service providers. Navigation costs only accumulate on nodes that are directly involved in the execution. Furthermore, the system OSIRIS (Open Service Infrastructure for Reliable and Integrated Process Support) described in [SCHU2003] uses a publish/
subscribe based replication scheme together with freshness predicates to significantly reduce replication costs.

[OWLS2003] describes an ontology of services, called OWL-S (formerly DAML-S, see [ANKO2002]) that will make it possible for users and software agents to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties. OWL-S is part of the DARPA (Defense Advanced Research Projects Agency) Agent Markup Language program and intended for the use in the Semantic Web which should enable greater access not only to content but also to services on the Web. The OWL-S ontology has three main parts: the service profile for advertising and discovering services, the process model for the detailed description of the operation of services, and the grounding which connects the OWL-S ontology with low-level XML-based descriptions of Web services (such as WSDL) and provides details on how to interoperate with a service via messages.

A more general approach for the semantic intermediation between business process models and software components (such as for instance Web services) is presented by Teschke [TESC2002]. An ontology consisting of a natural language ontology and a domain specific vocabulary allows for specifying the requirements formulated in business process models and the solutions characterized by software component (Web service) descriptions. To find suitable software components semantic matches between the process requirements and the component description are calculated. The final selection of a software component from a list of semantic matching proposals is performed manually by a domain expert.

A different approach is taken by Florescu et al. in [FLOR2002A]. The authors present an XML programming language designed for the implementation of complex Web services. The language called XL is portable and fully compliant with W3C standards such as XQuery, XML Protocol and XML Schema. It allows programmers to concentrate on the logic of their application and to develop Web services with good performance. Therefore, it provides high-level and declarative constructs for actions which are typically carried out in the implementation of a Web service; e.g. logging, error handling, retry of actions, workload management, events, etc. Furthermore, performance tuning tasks (for instance, caching, horizontal partitioning, etc.) should be provided by an implementation of the language as, for example, described in [FLOR2002B].

### 2.1.3 Web Service Search

Finding appropriate actions in logged execution data requires to find events that are similar to the current event in terms of the event itself and the Web service operation which has been executed when the event occurred. Recently, several approaches in the field of Web service composition also have addressed the problem of finding similar Web services. In the following, we discuss two approaches, namely the METEOR-S project and the Woogle search engine.

Besides quality-oriented execution of workflows (cf. Section 2.1.1) the METEOR-S project aims at extending standards for Web services and service-oriented architectures such as BPEL4WS.
Quality of Service for Web Services and Workflows

WSDL, and SOAP with Semantic Web technologies to improve Web service discovery and composition. Verma et al. [VERM2004, SIVA2003] focus on adding semantics to WSDL and UDDI by annotating Web services based on shared, domain specific ontologies. The annotations are added using extensibility in elements and attributes as supported by WSDL specification version 1.2. The annotations are stored in UDDI and are used for semantics-based discovery of relevant Web services. Patil et al. [PATI2004] describe a framework for semi-automatically marking up Web service descriptions with ontologies. They provide algorithms to match and annotate WSDL files with relevant ontologies and use domain ontologies to categorize Web services into domains.

Cardoso and Sheth [CARD2003] address the problem how to efficiently discover Web services based on functional and operational requirements in the context of the composition of workflow processes that model e-service applications. Therefore, tasks executed in a workflow and Web service interfaces are semantically described by concepts taken from ontologies. In particular, the input and output parameters of Web services are associated with ontological descriptions. For searching appropriate Web services for tasks service templates are used which indicate the characteristics of the Web service that is needed. A service template consists of the service name, a service description, quality of service (QoS) parameters (such as time, cost, and reliability as, for instance, discussed in [CARD2004]), and a set of output and input parameters. A service object holds the description of a real Web service consisting of the same part as the service template. To find a Web service for a given task a service template is matched against a set of service objects. The similarity between service templates and service object is calculated based on three kinds of similarity: syntactic similarity, operational similarity, and semantic similarity. Syntactic similarity is based on service names and service descriptions, it is calculated using „string-matching“ algorithms. Operational similarity is based on QoS metrics, it is calculated as geometric distance of the QoS dimensions specified in service template and service object.

Semantic similarity evaluates the similarity of concepts and properties used to describe the input and output parameters of service template and service object. The goal is to calculate the degree of integration between service template and service object, i.e., it is derived how many input properties of a service template have an associated output property in a service object respectively how many output properties of a service template have an associated input property in a service object. The association of input an output properties of two operations describes whether the operations have a similar functionality. As outputs and inputs are semantically annotated the semantic similarity is derived from a comparison of the ontological concepts assigned to input and output parameters. Therefore, sophisticated algorithms are provided which consider two cases: first that concepts for input and output are taken from the same ontology, second that they are taken from different ontologies.

Sivashanmugam et al. [SIVA2004] employ the approach described in [CARD2003] in a framework for semantic Web process composition. The search for services executing an activity in a Web process is guided by a semantic activity template specified by a user. The approach assumes that all services are semantically annotated describing the functionality, input, output, precondi-
Related Work

tions, and effects of a service by mapping it to ontological constructs. Services are searched in registries specified by the user using the semantic activity template; the search results are ranked using the semantic match value between the activity template and the service.

Dong et al. [DONG2004] describe the algorithms underlying the Woogle search engine for Web services. Woogle helps to locate desired Web services by supporting similarity search for Web services. Therefore, the underlying structure and semantics of the Web services are exploited and an accurate specification of users’ information needs is supported. A user can ask for Web service operations that are similar to a given one, for operations that have similar input or output parameters as a given operation, and for services that compose with a given one, i.e., one operation produces an output that may be used as input for the second one. Furthermore, also template search is supported, i.e., the user specifies the functionality, input, and output of the desired operation and the search engine returns a list of matching operations. The algorithms for similarity search combine multiple sources of evidence. They take into account the similarity between Web service descriptions, the similarity between operations, and the similarity between input and output parameters of operations. The authors consider operations to be similar if they take similar inputs, produce similar outputs and the relationships between input and output are similar.

To determine similar inputs and outputs a clustering algorithm is used which groups the names of parameters of Web service operations into semantically meaningful concepts. Therefore, association rules are used which formulate that parameters tend to express the same concept if they often occur together. The similarity between parameters is then derived by comparing the concepts produced by the clustering algorithm. Thus, similarity between parameters and service operations is only calculated based on their names, no additional semantic information is used. This is due to the fact, that the similarity search of Woogle is designed for the large number of services published in publicly available service registries for which annotations may not be assumed.

2.2 Dynamic Exception Handling for Cooperative Processes

Cooperative processes are subject to recent standardization efforts, commercial products, and research approaches [AALS2003]. The research approaches may be divided in those using inter-organizational workflows to support cooperative processes and those more recent approaches using Web services. As dynamic event and exception handling is one of the main features of Web-Flow we especially discuss related work which also provides dynamic exception handling facilities. The approaches are located in the top front sector of the classification dice in Fig. 2-1 as they focus on exception handling on the process level. We consider standardization efforts, commercial systems, inter-organizational workflows, Web-service-based systems, and active databases.
2.2.1 Standards and Commercial Systems

We describe the event and exception handling facilities of the BPEL standard proposal, some other standards supporting cooperative processes, and shortly discuss some commercial systems.

2.2.1.1 The BPEL4WS Standard Proposal

A recent standardization effort on the field of cooperative business processes is the Business Process Execution Language for Web Services (BPEL4WS, [THAT2003]). BPEL processes resemble workflows defined and executed in workflow management systems [WFMC2000]. They have a control flow consisting of basic and structured activities. The latter can group a set of other structured or basic activities using control flow constructs such as sequences, parallel and conditional branchings, or loops. Fig. 2-2 shows a procurement process as it may, for example, be executed at

---

**Fig. 2-2**: Printer supplier procurement process represented as structured activities in BPEL.
(adopted from [ALON2004])
the printer supplier involved in the collaborative fulfillment process described earlier (cf. Fig. 2-2). To process an order the supplier first receives the order, then checks its local stock and confirms the order if the goods are on stock. Otherwise, it is checked whether the goods may be shipped by an external partner. If a partner is found, the order is confirmed, otherwise canceled. Structured activities have thicker borders and denote the control flow construct used, for instance, „sequence“ or „switch“. Activities with thin borders are basic activities which either call Web service operations („invoke“) or wait for responses from Web service operations („receive“ or „reply“).

The data flow between the different Web services used in a cooperative process can be defined by mappings between the Web services’ output and input parameters and process internal variables which transport the data between the Web services. The control and data flow are defined by a process designer when a new cooperative process definition is generated. For more details on the control and data flow constructs of BPEL see [THAT2003].

The BPEL standard is implemented, at least partly, in different commercial as well as publicly available products, for example, in the Oracle BPEL Process Manager (formerly Collaxa BPEL server [ORAC2004]), IBM Websphere [IBM2004B], Microsoft’s BizTalk Server [CHAP2004], or in the IBM alphaworks Business Process Execution Language for Web Services Java Run Time (BPWS4J) [IBM2004A].

BPEL offers several event and exception handling mechanisms for faults and events occurring during business process execution. As business processes are of long duration and use asynchronous messages exception handling in BPEL relies on the concept of compensation, i.e., it tries to reverse the effects of failed activities in an application-specific manner. There is a long history of work in this area regarding, for example, the use of Sagas [GARC1987], ConTracts [WÄCH1992], open nested transactions [GRAY1993], or spheres of compensation [LEYM1995]. BPEL provides a variant of such compensation protocols, called „Long-Running (Business) Transactions“ (LRTs), by providing the ability to define fault handling and compensation (see [THAT2003]). We will shortly describe this approach in the sequel.

In BPEL each activity is executed in the context of a „scope“. For instance, Fig. 2-3 shows the procurement process from Fig. 2-2 with the scope of the structured activity „searchExternal“ which looks for an external partner who may supply the goods. Each scope may provide „fault“, „compensation“, and „event handlers“ for the activities contained in the scope. A fault handler defines a set of „catch“ activities which are to be executed if a particular fault occurs, related for instance to a fault message returned from a Web service or explicitly thrown by a „throw“-activity inside this scope. For instance, if the Web service executing the activity „confirmOrder“ sends a fault message instead of a regular response this fault message would be propagated up to the activity „searchExternal“ as the fault handlers for the scope containing the activity „confirmOrder“ are defined there.

A compensation handler contains activities that reverse the effects of the whole scope for which it is defined. It is called in a fault or compensation handler of the immediately enclosing scope to
undo the effects of the activities that have been executed inside this scope. A compensation handler can only be invoked once after the scope execution has completed normally. Each scope has a default fault handler as well as a default compensation handler assigned which both call the compensation handlers of all enclosed scopes in reverse execution order. The default fault handler additionally re-throws the fault to its enclosing scope. To express the fault and compensation handling relationship between scopes BPEL uses the protocol framework of WS-Transaction [CABR2002].

Event handlers can be defined for two types of events: „message events“ related to incoming messages, e.g. a cancellation or a status query, and „alarm events“ that are triggered after a user-set time, denoting for instance a timeout while waiting for a Web service response. An event handler is called as soon as its associated event occurs and it can be triggered several times.

However, the compensation-based approach has some disadvantages as they have been described, for instance, in [HÄRD1999]. The compensation handlers have to be defined manually for each
Related Work

scope, i.e., a user has to specify which basic or structured activities are to be executed for compensation. Furthermore, the execution of the compensation activities can also fail which may lead to re-throwing the fault up to the top activity representing the whole process. In this case, the whole process has failed. Another problem of the fault handler approach of BPEL is that event detection is limited to particular places in the cooperative process and events that are not covered by Web service messages or timeouts (for instance, violations of other quality characteristics) cannot be detected automatically.

2.2.1.2 Further Standards

In this section we focus on the exception handling capabilities of two other standards supporting cooperative processes, namely the Web Services Choreography Description Language (WS-CDL, [W3C2004]) and ebXML Business Process Specification Schema [OASI2001].

WS-CDL [W3C2004] is a standard proposal developed by the W3C and aims at being able to precisely describe collaborations between partners regardless of the supporting platform or programming model used by the implementation of the hosting environment. Using WS-CDL a contract containing a „global“ definition of the common ordering conditions and constraints under which messages are exchanged is produced. The contract describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. The global specification is in turn realized by combining the resulting local systems on the basis of appropriate infrastructure support. Exceptions occurring during choreography are handled in exception handling blocks that describe what additional interactions should be executed to handle the exceptions. Different types of exceptions may occur in a choreography and may be handled in the exception block, for instance, interaction failures, protocol based exchange failures, security failures, timeout errors, validation errors (related to the conformance of a message to its XML schema definition), or application failures. The exception block may also contain different exception handlers that are executed under different conditions. Thus, WS-CDL supports the use of predefined exception handlers.


Regarding exception handling control exceptions and business protocol exceptions may be handled. Control exceptions deal with the mechanisms of message exchange such as verification, validation, authentication and authorization and will occur up to message acceptance. These
Dynamic Exception Handling for Cooperative Processes

exceptions are signalled back to the organization that called the partner and lead to the termination of the business transaction. Business protocol exceptions deal with the mechanisms that process the business transaction and will occur after the request message has been verified and validated by the partner organization. Business protocol exceptions are for instance an invalid structure of a message, a violation of business rules, syntax failures in the business document, or authorization exceptions, i.e., a role is not authorized to participate in a business transaction. These exceptions are also signalled back to the organization that sent the request message and the business transaction is terminated. Thus, exception handling in ebXML Business Process Specification Schema only comprises sending notifications to the organization requesting the execution of a business transactions. Exceptions always lead to a termination of the business transaction and no exception handling is offered.

2.2.1.3 Commercial Systems

Commercial systems for the development, deployment, and management of cooperative business processes and XML-based Web services such as Microsoft’s BizTalk Server [CHAP2004, MEHT2001, ROXB2001], Intersystems Ensemble [INTE2004], or the AmberPoint product family [AMBE2004] mainly focus on monitoring processes and Web services in a business-oriented manner. That means they support monitoring and viewing real-time information about the status of active messages and process activities, historical data and real time performance metrics of business processes.

BizTalk Server uses processes defined in BPEL. So, BizTalk Server also uses a compensation-based exception handling to support long running business transactions as described in the last section.

The AmberPoint Exception Manager [AMBE2004] is able to detect both operational exceptions (such as system or implementation failures) and business exceptions such as data entry errors or orders lost in processing in Web services executed in an organization. For exception handling compensating applications or Web services may be called or a user may be notified to perform some manual exception handling.

These approaches are only able to automatically detect exceptions that are related to Web service messages or timeouts, other (for instance, external) events which may lead to exceptions cannot be detected. Furthermore, predefined exception handling steps are executed. Thus, additional conditions such as the success of the former exception handling or changes in laws and policies cannot be considered flexibly.

Other commercial systems offer support for measuring and monitoring process performance and quality. In the following, we shortly discuss ARIS Process Performance Manager [IDS2004], IBM Workbench Business Integration Monitor [IBM2004d], and ADONIS [BOC2004].

ARIS Process Performance Manager (ARIS PPM, [IDS2004]) is a browser-based solution imple-
2.2 Inter-organizational Workflows and Exception Handling

Many approaches supporting cooperative processes with inter-organizational or distributed workflows use common process definitions (for example, [AALS2001, BON2002, LENZ2001, MUTH1998, SCHÖ2001, VOSS1999, ZENG2001]). As the cooperation partners have to agree on definitions and changes of such common process definitions, their autonomy is reduced when participating in inter-organizational workflows. In most approaches only simple exception handling is provided, typically for the monitoring of deadlines. The Workflow Management Coalition has specified an interface for the cooperation of several workflow engines (Interface 4, [WFMC2000]). However, this interface only focuses on data exchange and does not address exception handling.

A more flexible approach for the definition of inter-organizational workflows is presented by Aalst and Khumar [AALS2002b]. The authors propose an eXchangeable Routing Language (XRL) using XML syntax. A routing schema defined in XRL can be used to support flexible routing of documents between cooperation partners in the Internet environment. Thereby, the control over the whole process instance may either be passed between the cooperation partners or one partner can keep the control (i.e., different cooperation models are supported). The formal seman-
tics of XRL is expressed in terms of Petri nets.

Approaches such as the Crossflow system [GREF2001, KLIN1999A], WISE [LAZC2000, LAZC2001] or [GEOR1999] all describe workflow-based approaches for business-to-business (B2B) interaction and virtual organizations based on the integration of services in cooperative processes. Thereby, also common workflow definitions are used and the interfaces and formats for data exchange have to be agreed upon separately with each partner. Thus, the autonomy of the service providers is reduced.

Several recent research approaches have addressed dynamic exception handling for centralized and distributed workflows (for instance, [CASA1999, CHIU2001, MÜLL2002, REIC1998, REIC2001, SADI2001, WESK2001, ZHOU1999]). For example, in CHIMERA-EXC (Casati et al. [CASA1999]) Datalog-based rules can be defined to monitor events and to derive appropriate actions. The ADEPTflex workflow system (Reichert et al. [REIC1998, REIC2001]) provides an operator set to adapt running workflow instances to a changed situation (for example, by dropping or inserting nodes). Thereby correctness and consistency of adapted workflows is preserved by the operators and temporal implications of workflow adaptations such as deadline violations for workflow activities are considered (Dadam et al. [DADA2000]). However, exception detection and handling are executed manually, i.e., a sophisticated user has to detect possible exceptions and decide which operators are to be applied. Recently, ADEPTflex has been extended to also support workflow schema changes [RIND2004A]. Also propagation of changes to running workflow instances is supported (Rinderle et al. [RIND2004B]). A similar approach for dynamic workflow adaptation is provided by the WASA system [WESK2001], which also allows for schema changes and does not impose any restrictions on the structure of the workflow definitions. However, workflow adaptation also has to be executed manually. Sadiq et al. [SADI2001] describe an approach in which partially defined workflows can be executed that contain so-called „pockets of flexibility” with a set of workflow fragments and rules stating how these fragments may be refined at runtime. Thus, a workflow can be adapted at predefined places.

The AGENTWORK system [MÜLL2002, MÜLL2004] presents a rule-based approach for automatic event detection and operators for automatic adaptation of workflow instances during runtime. The operator set comprises constructs for dropping, adding, replacing, or delaying workflow activities. Whenever possible, workflow adaptation follows a predictive strategy, i.e., the remaining part of affected workflow instances is analyzed and adapted in advance. Thus, users are informed timely about necessary changes (e.g., the preparation of new examinations in a medical context). Alternatively, reactive adaptation is possible, i.e., for each activity it is checked directly before execution whether it has to be adapted. Müller and Rahm [MÜLL2000] address the impacts of workflow adaptations on cooperating workflows. For instance, due to adaptations data needed in cooperating workflow instances may be sent too late or not at all if the producing activity has been deleted. The predictive adaptation strategy tries to foresee such delays and to inform the cooperating workflow in advance. However, this requires sophisticated workflow systems at the cooperation partner’s side which are able to handle such notifications.
Related Work

2.2.3 The AdaptFlow Prototype

Based on the concepts developed in [MÜLL2002] we have built the AdaptFlow prototype for event-oriented automatic workflow adaptation [GREI2005, GREI2004]. It has been built in a cooperation project with the Department of Medical Informatics at the University of Leipzig. It offers workflow-based support for the execution of clinical trials in medicine, for example in oncology (i.e., treatment of cancer). However, it may also be used in other application domains. In many medical domains clinical trials are used to introduce and compare new treatments. In clinical trials patients are treated according to trial protocols. Trial protocols contain detailed instructions for the execution of medical therapies and additionally specify reactions to exceptional situations (for instance, an infection or a toxicity). Such exceptional situations occur quite frequently during the application of a treatment, for example, as patients react differently to the same treatment.

To adequately support the execution of trial protocols and to relieve the medical staff (even a specialized physician has to administer many different protocols to different patients at the same time) a system is needed that handles well-structured but flexible therapy processes efficiently. A convenient system should observe the status of the therapies currently being applied, offer automatic recognition of exceptional situations, and appropriate decision support for handling such situations. Furthermore, the system should be able to automatically adapt affected therapy processes to adequately handle necessary treatment modifications.

The AdaptFlow prototype aims at offering this support by combining a workflow management system with a rule base. The workflow system is used for the execution of therapy workflows and the integration of different users, data, and applications. Furthermore, it has to support dynamic adaptation of running workflows to handle the flexibility of therapy processes. The rule base handles the medical knowledge represented in the protocols and is used to detect exceptional situations. The AdaptFlow prototype offers the following advanced features to support adaptive treatment workflows. Besides rule based exception detection with extended Event-Condition-Action (ECA) rules (see [GREI2005] for more details) and dynamic workflow adaptation with obligate user confirmation (i.e., if the dynamic adaptation is not appropriate the user can perform an alternative exception handling manually) it supports two strategies for automatic workflow adaptation. Whenever possible AdaptFlow tries to predictively adapt the remaining part of a running therapy workflow as soon as an exception is detected (predictive adaptation). This informs users early about necessary changes and supports a timely and effective treatment of patients. This strategy is based on temporal estimates about the duration of future workflow parts, which are not always possible. In those cases the second strategy, reactive adaptation, is used which adapts affected activities directly before their execution.

AdaptFlow offers a set of so called control actions. A control action is a technical description of the workflow adaptations that are necessary to implement the therapy modifications which handle an exceptional situation. In particular, AdaptFlow supports the following control actions:
Dynamic Exception Handling for Cooperative Processes

- dropping an activity,
- adding a new activity,
- replacing an activity by another one,
- delaying an activity,
- changing input parameters of an activity,
- aborting a workflow instance,
- suspending a workflow instance.

The control actions are applied to all affected activities in the workflow. For this purpose the API functions of the underlying workflow system (ADEPTflex [REIC2000]) are used, for instance, to drop or add nodes. The adaptations are shown to an authorized medical user for confirmation before the modified workflow instance is resumed. The workflow fragment in Fig. 2-4 illustrates the

![Workflow Diagram](image)

**Fig. 2-4:** Screenshot from the AdaptFlow worklist client.

The screenshot shows a running workflow instance in which an activity has been dropped predictively (marked with the cross).
result of the application of a drop control action on a therapy workflow as it is presented to a medical user that works with the system.

2.2.4 Web-Service-based Cooperative Processes

For the realization of Web-service-based cooperative processes the cooperation model used is important as it influences the degree of autonomy of the cooperation partners and the possibilities for dynamic exception handling (cf. Section 1.1.3). In a simple, workflow-like cooperation model only one partner controls the execution of the cooperative process and calls the services of the partners. This model allows for maximal autonomy of the partners, i.e., no process details have to be published, only the interfaces are used. However, this implies that event detection and exception handling has to be performed totally on the service consumer side. With a complex cooperation model the partners providing the services agree upon the overall process, may act both as service provider as well as consumer and may share the control over the process. To reach sufficient autonomy of the service providers, the cooperative process definitions only contain the minimal information that is necessary for establishing the cooperation [BUSS2002]. A complex cooperation model allows for distributed exception handling at several cooperation partners.

Besides the standardization effort BPEL (discussed in Sections 1.1.2 and 2.2.1.1) several studies addressed the definition and execution of Web-service-based cooperative processes (e.g., [AMBR2003, BARE2003, BENA2002, CHIU2002, FAKA2003, JANG2003, ORRI2003, PERR2004]). To offer high execution quality and flexibility for cooperative processes most of them focus on the selection of the most appropriate service provider during runtime thus aiming at preventing exceptions similar to the approaches discussed in Section 2.1.2. Dynamic exception handling is supported by calling alternative service providers (if available), executing pre-defined alternative paths or compensation steps. Perrin and Godart [PERR2004] describe a model for a middleware that provides support for cooperative process management and coordination esp. in virtual enterprises (i.e., organizations that allow enterprises to create a partnership for a specific project). It offers pertinent information about work progress while maintaining adequate privacy of information, and supports both long-time transactions and dynamic process definition. Thus, the main focus is on management and coordination of complex and flexible cooperative processes, execution quality is not explicitly considered.

Casati and Shan [CASA2001] describe the eFlow system which offers a platform for the specification, enactment, and management of complex business processes using simple and complex services provided by cooperation partners. The desired services may be described using some quality characteristics and an appropriate provider may be selected dynamically during runtime. Events raised by the eFlow system itself or by external event managers may be detected with event nodes at predefined places of a process. The eFlow system provides exception rules that define for each service call how failures occurring for this service call are to be handled. Additionally, so-called transaction regions with assigned compensation activities can be defined. Furthermore, manual
adaptations of running process instances are possible, e.g. by selecting an alternative service provider for a particular activity or by migrating running process instances to new process definitions that have been adapted to the changed situation.

A different approach to service-based process execution is presented by Wetzel and Klischewski [WETZ2002] who describe Serviceflow Management which puts the service nature of inter-organizational processes into the center of modeling, design, and architectures. Serviceflow management distincts between the serviceflow, the portion of the process where the customer’s concern is evaluated and cared for, and background processes. Furthermore, it considers social and quality aspects in service delivery. The authors provide instance-based XML process representations and generic components and architectures for their exchange and for the provision of service tasks. However, dynamic exception handling to enhance flexibility and robustness of the serviceflows is not addressed in this work.

Hung and Chiu [HUNG2004] present an approach for workflow-based information integration with exception support in a Web service environment. Therefore, an implementation framework is proposed comprising four layers, namely, application layer, workflow layer, service layer, and message layer. The workflow layer supports control-flows, data-flows and exception-flows by using the Business Process Execution Language for Web Services (BPEL4WS) with extensions for data-integration and exception-handling assertions. The exception-handling assertions are based on a classification of expected exceptions that comprises control, data, temporal, and external exceptions. Three exception handling procedures are available, namely remedy, forward recovery, and backward recovery, which are implemented as exception-flows in the workflow layer. The exception handling assertions are linked to SOAP-fault implementations which trigger the execution of exception-flows. Thus, this approach focusses on an extension of the exception handling of BPEL4WS but does not consider the definition and monitoring of additional quality conditions. Furthermore, the exception-flows are not defined independently of the workflow but are contained in the standard workflow definition (for forward and backward recovery) or require manual interaction (for remedy).

### 2.2.5 Active Databases and Rule Engines

Rule-based mechanisms related to dynamic exception handling may be found in active databases [WIDO1996], transaction processing [DAYA1991], and rule engines [ILOG2004]. These approaches are typically based on Event Condition Action (ECA) rules. ECA rules specify which action is to be executed after a particular event has occurred. The condition part contains additional constraints that may influence the selection of an appropriate action [WIDO1996]. Active databases use triggers based on ECA rules to react automatically to events that are related to insert, update, or delete operations on database tuples. In the context of active databases actions typically are again data manipulation operations executed on the database but may as well be calls to applications outside the database.
Related Work

In these approaches actions can only be derived if an appropriate rule has been defined for an occurred event. Otherwise, the event cannot be handled by the rule-based approach. Furthermore, rules have to be specified and maintained manually which may become quite tedious for large rule sets.

Business rules engines such as ILOG [ILOG2004] support users in defining and maintaining business rules that have to be considered in applications and cooperative business processes. The ILOG rules engine allows business analysts to define, manage, and store business rules that should be followed by all organizations in an enterprise. Thus, the rules are not directly related to handling events in particular process instances. However, the central business rule repository may be considered during event handling to ensure that derived actions are in line with the business rules.

2.3 Deriving Actions for Exception Handling

Several researchers have addressed the possibility of using logged workflow execution data for exception detection and automatic derivation of appropriate exception handling steps in standard and inter-organizational workflows (e.g., [GRIG2001, HWAN1999, LEO2003, MOUR2003, SCHI2003]). These approaches are related to the log-based, semi-automatic derivation of actions.

Luo et al. [Luo2003] propose exception handling techniques that support conflict resolution in cross-organizational settings. Conflicts may occur because of failures of underlying applications, workflow system component failures, or insufficient user input. In particular, the authors propose a “bundled” exception handling approach consisting of exception knowledge sharing, coordinated exception handling, and intelligent problem solving. Exception knowledge sharing means that cooperation partners share exception specifications and exception handling experiences. Therefore, exception patterns, exception handler patterns, and a description of exception handling experiences are all stored in a case repository and shared by all cooperating business processes. Exception handling experience is stored in a so called case data structure that consists of an exception information block, a context information block with workflow application data, and an action information block. Coordinated exception handling is realized over a coordinator that is responsible for solving conflicts based on the information stored in the case repository. To derive similar cases for a new conflict intelligent problem solving using case based reasoning is utilized. Therefore, former exception handling experiences may be reused. However, the necessity of sharing knowledge about exceptions and exception handling and the use of a coordinator reduces the autonomy of the cooperation partners. Furthermore, no mechanisms for deriving recommendations for process optimizations from the exception handling data are provided.

Grigori et al. [GRIG2001] focus on analyzing, predicting, and preventing the occurrence of exceptions, i.e., deviations from the desired or acceptable behavior of business processes. The authors propose a solution which is based on applying data warehousing and data mining techniques on workflow execution logs. Workflow execution logs record all important events that occur dur-
ing workflow execution. These comprise the start and completion time of activities, the input and output data, the used resources, as well as any failure that occurred during activity execution. This data is imported in a data warehouse and analyzed using data mining techniques. Thus, knowledge about the circumstances of exceptions can be extracted and be used to predict future exceptions. The authors also describe an architecture and implementation of a tool suite that enables exception analysis, prediction, and prevention.

Hwang et al. [HWAN1999, HWAN2002] present an approach to facilitate manual exception handling for the users by proposing solutions for resolving a given exception. Specifically, several algorithms are defined which scan through the previous records about handling former exceptions, looking for those that are close to the current exception. In this context close means that the exceptions are similar e.g. in terms of the activity during whose execution the exception occurred or in terms of the role who has executed the activity. The ways in which those exceptions were handled serve as useful information in determining how to handle the current one. This is similar to the approach of Web-Flow, but does not take into account the special needs of Web-service-based cooperative processes (e.g., considering the autonomy of the service providers).

2.4 Summary

The Web-Flow architecture developed in this thesis is related to work on the fields of quality-oriented execution and composition of Web-services and workflows, on dynamic exception handling for cooperative processes as well as on log-based action derivation and Web service search. In this chapter we have discussed relevant work from the different areas. Fig. 2-5 summarizes the results by aligning related work to the eight categories for execution quality support we identified at the beginning of the chapter.

The approaches addressing quality-oriented execution of Web services mainly focus on exception prevention by defining quality characteristics for single Web services that are supervised by the service provider. Quality-oriented service composition also aims at preventing exceptions by selection of appropriate services by the service consumer. These approaches do not cover the requirements for quality-oriented process execution identified at the beginning of the chapter that characterize the approach of Web-Flow. Thus, the Web-Flow architecture may accomplish the quality support provided by these approaches.

With regard to automatic event detection and flexible dynamic exception handling for events occurring in a cooperative process several interesting approaches have been proposed for inter-organizational workflows. These approaches belong to the categories in the top front sector of the dice in Fig. 2-5. Most of them define exception handling steps during process definition and thus do not consider the process execution context. They belong to the category in the left top front part. The exception handling mechanisms provided by the different approaches support manual as well as automatic and log-based exception handling and some also allow for a rule-based, predic-
Related Work

Fig. 2-5: Alignment of related work with categories for execution quality support.
(The sector with thick borders denote the category fulfilling the postulated requirements for quality-oriented process execution.)

tive adaptation of workflow instances. However, these approaches require the use of particular process engines and thus are not generic. Most approaches addressing inter-organizational workflows also use common process definitions, i.e., they only support a complex cooperation model. In summary, none of the related approaches addresses all requirements we postulated for comprehensive support of a quality-oriented process execution. Thus, the Web-Flow architecture developed in this thesis extends and accomplishes these approaches. The classification of quality characteristics that can be monitored automatically at the service consumer side during process execution accomplish approaches that aim at exception prevention by selection of the best fitting service either during process definition or execution. Web-Flow also accomplishes approaches that provide exception handling with a classification of automatically observable quality characteristics and a rule-based exception handling approach. Furthermore, it offers a generic solution and supports quality monitoring for process orchestration as well as for process choreography.
3 The Web-Flow Architecture

This chapter introduces the main concepts of the Web-Flow architecture. The Web-Flow architecture aims at offering comprehensive quality support for cooperative business processes integrating heterogeneous and autonomous services available locally or from external partners. Therefore, it provides quality monitoring and advanced event and exception handling functionality implemented in a dedicated component, the Web-Flow MaX. The dynamic event and exception handling aims at enhancing the robustness and flexibility of the cooperative processes.

We first present the concepts underlying the Web-Flow architecture (Section 3.1) and sketch its main components (Section 3.2). Then we give an overview of the mechanisms for event detection, dynamic event handling, and log-based selection of actions used by the Web-Flow MaX component (Section 3.3).

3.1 Web-Flow Architecture: Concepts

As already discussed in the introduction the execution quality of cooperative processes can be supported in four categories. Each category has a particular focus (exception prevention versus exception handling, consideration of the process execution context versus concentration on process definition time) and thus addresses different aspects of process quality support. In the following we have a more detailed look at the different categories and identify shortcomings with respect to a comprehensive support of process execution quality.

1. Selection of the best fitting service operation during process definition:
   That means that the service operation is chosen to execute a particular activity in a cooperative process which promises to fulfill the task best. The selection is based on descriptions and promises given by the service provider and experiences from earlier service calls. Thus, during process execution it may turn out that the selected service operation may not achieve the

3. Monitoring and eXception handling
promised execution quality, for instance, due to lack of resources for operation execution or
due to unavailability of the service.

2. Selection of the best fitting service provider during process execution:
   In this category the service operation for executing a particular activity is not selected until
   the activity is in fact executed. Thus, in contrast to the first category it may be assumed that
   the selected service provider may really execute the operation with the promised execution
   quality as current execution parameters can be considered.
   However, the selection of a service operation during process execution may delay the execu-
   tion of the whole process instance as possibly different service directories have to be
   searched for an appropriate service. If no appropriate service is found even a manual selec-
   tion of a service provider may be necessary. This delay is tolerable in a large number of applica-
   tion domains as the processes are typically long running and thus may compensate the
delay. However, in application domains with strict time limits such as health care or produc-
tion the additional delay for searching an appropriate service may lead to the violation of
deadlines in the process. For instance, a service cannot be selected earlier than 10 minutes
before an activity is to start as only then the necessary runtime parameters are available. If
the service search takes more than these 10 minutes (e.g., due to unavailability of resources)
and the start time is a hard deadline as there has to be a specific timespan between two activ-
ities, the delay leads to a violation of the deadlines.

Despite the careful selection of the service still exceptions may occur during the execution of ser-
vices in a cooperative process that hinder the successful execution of the process. Such exceptions
can for instance be service faults, that may lead to a delayed delivery of service results or to insuf-
ficient service results. The service may also not respond at all. These exceptions lead to the violation
of quality characteristics which the service consumer expects in order to reach a particular
overall process execution quality. To ensure the process execution quality despite the exceptions
exception handlers can be integrated in the process definition as it is supported in the third cate-
gory:

3. Integration of event and exception handlers in process definition:

   Several existing solutions for quality support of cooperative processes focus on detecting and
handling events and exceptions such as timeouts and service faults by incorporating event
and exception handlers in each cooperative process definition (for instance, [THAT2003,
CHAP2004]). The exception handlers are activated if their associated exception occurs and
execute the steps that have been defined during process definition. They are to ensure that the
process can be continued despite the exception. As several different exceptions may occur
during process execution possibly a large number of event and exception handlers have to be
maintained for each process definition. They may not be reused in different process defini-
tions. Furthermore, the approaches support the quality monitoring only for processes that
have been defined and are executed with a particular process engine.
In cooperative scenarios it cannot be expected that the service consumer is always notified about service faults, as the partners are often only loosely linked. In particular, if a simple cooperation model is used (cf. Section 1.1.3) the service consumer does not know that the service execution is part of a cooperative process and that a particular execution quality is desired. Furthermore, services that are based on legacy applications may not be able to offer execution quality support at all. To allow for monitoring the execution quality for different types of services by the partner that controls the cooperative process the fourth category for quality support is introduced:

4. **Definition and monitoring of additional quality characteristics for service execution and flexible event handling during process execution:**

   This category provides a generic solution which allows for a homogeneous definition and monitoring of quality characteristics for services and cooperative processes defined and executed in different process engines. The quality characteristics can be applied to different types of services and thus allow a quality monitoring for services that do not offer this themselves (for example, legacy applications). Additionally, rule-based dynamic event and exception handling is supported to ensure the overall process execution quality if quality characteristics are violated. Quality monitoring and exception handling are widely automated to relieve process administrators. However, still sufficient user interaction is offered to provide control about possible changes in process instances.

   In this category, service operations are selected during process definition (as in the first category). Event handling actions may be reused as they are not integrated in the process definitions, thus facilitating the maintenance of cooperative process definitions and event handlers. Events that may be caused by unsatisfying service execution as discussed for the first category may be handled by the flexible event handling mechanisms. Thus, the process instance may be continued successfully despite the events.

The Web-Flow architecture developed in this thesis provides comprehensive support for quality-oriented execution of cooperative processes and belongs to the fourth category. It is based on a classification of quality characteristics that can be imposed on service executions and contribute to the overall execution quality of a cooperative process. A main property of the quality characteristics is that they do not make any assumptions about the applications realizing a service. Thus, they can be applied to heterogeneous services (for instance, legacy applications as well as highly interactive applications). Monitoring can be performed completely at the process side and does not violate the autonomy of the service provider. This also allows for monitoring the execution quality of services realized by legacy applications that do not offer any quality support. If a service operation already uses internal exception handling to ensure execution quality, the quality characteristics can be used to define and monitor accomplishing quality conditions that are not covered by the service operation. For instance, some quality characteristics may become relevant only in the context of a concrete cooperative scenario and may then be monitored on process level.

To handle violations of quality characteristics and to ensure the overall execution quality of cooperative processes the Web-Flow architecture provides dynamic exception handling. It is based on
3.2 Web-Flow Architecture: Components

The components of the Web-Flow architecture realizing the concepts described above are implemented in a dedicated component, Web-Flow MaX, which interacts with a process execution infrastructure. Fig. 3-1 illustrates the Web-Flow architecture and the process execution infrastructure. The infrastructure part comprises a process management system, a process database, a Web service framework and the Web services themselves. As the functionality of Web-Flow is independent of a concrete process management system Web-Flow MaX does not use any process engine specific interfaces, but observes the messages sent between the process engine and the application services.

The Web-Flow architecture is a service-oriented architecture. Services used in cooperative processes executed in the Web-Flow architecture are assumed to provide a defined and published service interface. If necessary this interface has to be provided through a wrapper, for example, for workflows encapsulated in Web services. This allows for a uniform access to the service descriptions for the process engine as well as for the Web-Flow MaX component. Furthermore, the use of a service-oriented architecture ensures that Web-Flow MaX may be used with several process engines as long as the activities used in the processes provide a Web service interface specified in WSDL (cf. Section 1.1.1).

The process management system typically consists of an editor for the definition of cooperative processes and a runtime component for the execution of the processes. The available cooperative
process definitions are kept in the process database. During process execution the engine calls local and external services using a Web service framework (Interface 1 in Fig. 3-1). The process engine also may cooperate directly with process engines of partners (Interface 2 in Fig. 3-1). The interface between the process management system and the Web-Flow MaX component is used to retrieve information about cooperative processes, available services, or for negotiations about cooperative process definitions in the complex cooperation scenario.

The Web-Flow MaX component provides the functionality to offer support for quality-oriented execution and optimization of cooperative processes. It has four core parts, namely the constraint and rule definition, the constraint and event monitoring, the dynamic event and exception handling, and the logging and analysis part. Furthermore, it comprises a Web service search component, the Web-Flow metadata and the Web-Flow log data repository. All components will be shortly described in the sequel.

**Constraint and Rule Definition**

The constraint definition part is used to specify the desired execution quality of services and co-
operative processes in terms of quality constraints which describe, for instance, the desired response time for a service execution. The constraint and rule definition component is also used to annotate services and their operations with semantic information (such as the service category) which is used for dynamic event and exception handling. The specification has to be done manually which is facilitated by an editor with a graphical user interface (also called Web-Flow MaX editor in the following). This manual effort is necessary to allow for an automated monitoring of the quality characteristics and semi-automatic event handling during process execution. Therefore, an XML-based description of the quality constraints is generated. Cooperative process definitions may either be imported from the process database or read directly from the Web-Flow metadata repository (for example, if the process designer wants to revise quality constraints or to define more quality constraints).

The constraint and rule definition component also supports the definition of rules to specify appropriate actions for exception handling. Rules are also specified manually and stored in the Web-Flow metadata repository for later use during dynamic event handling.

**Constraint and Event Monitoring**

The constraint and event monitoring part observes different sources for events that may lead to quality constraint violations or other exceptional situations occurring during process execution. Exceptional situations may, for instance, be a fault messages sent by the service provider to notify the service consumer about a failed service execution. The main event source are the service calls made during process execution and the corresponding responses received by the process engine. Other event sources are an interface for the notification about fault messages, received, for example, by e-mail or phone, and an interface for notifications about external events such as a change in general business policies of a company (for instance, for deadlines of invoices). This interface is denoted as Interface 4 in Fig. 3-1. After an event has been detected the constraint and event monitoring part of Web-Flow MaX evaluates the quality constraints defined for the activities of all possibly affected running process instances to check whether any of these have been violated. How many process instances are affected depends on the type of event.

**Dynamic Event and Exception Handling**

The main goal of Web-Flow’s event and exception handling is to ensure that the cooperative process successfully serves its purpose despite an occurred event or exception. Therefore possible event handling steps comprise performing additional actions, for instance, to provide necessary results or to compensate for the effects of the event. If appropriate, the process may also be adapted to prevent further events or exceptions when continuing the process execution. Further goals are that the event handling should not lead to violation of further quality constraints and that no or only few further events should occur.

The dynamic event and exception handling part of Web-Flow MaX supports two ways to derive appropriate actions to handle events and exceptions. For frequently occurring events rules may be
defined by the administrator which specify how to handle an event. The rule-based approach is accomplished by a log-based selection of appropriate actions if no appropriate rule is defined for a particular event. This approach allows to also successfully handle events for which no rules have been defined so far.

**Logging and Analysis**

The logging part of Web-Flow MaX records all events and exceptions together with the actions that have been executed to handle them. The log data is saved in the Web-Flow log data repository. This data is on the one hand used to derive appropriate actions during dynamic event and exception handling as described above. Therefore, the event descriptions are compared to derive whether an event similar to the current one has already occurred in the past. On the other hand, the logged execution data may also be analyzed offline (i.e., not in the context of dynamic event handling) to derive possible recommendations for process optimizations. The analysis may e.g. reveal that a particular service provider never answers in time if its services are called between 9 and 10 in the morning. Another result may be that a service operation often produces a violation of a response time constraint if it is called in a particular sequence of operations, for example, as a prior call to the same service provider blocks necessary resources.

The results of the analysis are used as a starting point for process optimizations which may improve the execution quality of the processes. Furthermore, frequently occurring events are identified which helps to define rules for appropriate handling the events. The analysis is performed semi-automatically, i.e., an interface is provided to the administrator in which he can specify the desired criteria for analysis (for instance, execution time, process definition type, service provider, etc.). The logging component then analyzes the log data according to these criteria and presents the result to the administrators. The generation of rules is also supported by a user interface.

**Web Service Search**

A further part of Web-Flow MaX is the Web service search that is used to search service directories for appropriate services executing a given task. This functionality is mainly needed during event handling, for example, if alternative services have to be invoked. Furthermore, it can also be used in combination with the editor of the process management system during the definition of new cooperative processes. The Web service search component searches globally available service directories such as UDDI (Universal Description, Discovery and Integration, [BELL2003]) directories, Web service search engines such as Woogle [DONG2004], and the local Web-Flow metadata repository. Additionally, Web-Flow metadata repositories of cooperation partners can be searched (Interface 3 in Fig. 3-1).

**Web-Flow Metadata Repository**

The Web-Flow metadata repository contains documents describing the cooperative processes, the
services used, and the quality constraints describing the desired execution quality of cooperative processes. The original service descriptions (specified in WSDL) and the original cooperative process definitions (specified for instance in BPEL) are also saved to be able to access them quickly if additional information is needed (for instance on data formats used for Web service calls).

3.3 Event Detection and Handling

Event detection and handling in Web-Flow is executed similarly as in active databases [WIDO1996]. After an event is detected, the rule set is searched for a matching rule. In contrast to these approaches, if no rule is found appropriate actions are derived automatically from logged event handling data. Finally, the derived actions are applied. In the following we sketch how quality monitoring, event detection, and rule-based event handling are performed in Web-Flow MaX by means of an example. The algorithms will be presented in detail in Chapter 5 (Event Detection), Chapter 6 (Dynamic Event and Exception Handling), and Chapter 7 (Log-based Selection of Actions and Recommendations).

3.3.1 Example scenario

As an example scenario we consider a fragment of the collaborative fulfillment process introduced in Chapter 1. To allow for a clearer presentation we only consider the process part executed at the PC manufacturer and at the monitor supplier as illustrated in Fig. 3-2. Execution quality is important for this sample cooperative scenario, to ensure that the PC is assembled and delivered in time to satisfy the customers so that they order their next PC from the same manufacturer. Also maximum execution durations for the different service calls are important, especially during the order entry part of the process, to ensure that the customer does not have to wait too long and perhaps leaves the site of the PC manufacturer without having placed an order.

Assume that the following quality constraints $q_c_1$ and $q_c_2$ have been defined for the collaborative fulfillment scenario:

$q_c_1$: The price for a monitor specified in the „ATPResponseMonitor“ response message from the „ATP check for monitor“ operation is to be less or equal the maximum price the customer has specified in the „OrderDetailsMonitor“ output message of the operation „Enter order details for monitor“. As $q_c_1$ checks the quality of a result returned by a service operation, we refer to $q_c_1$ also as result constraint.

$q_c_2$: The operation „Credit check“ has to answer in not more than 10 minutes, i.e., $q_c_2$ restricts the response time of an operation, so we refer to $q_c_2$ also as response time constraint.

Fig. 3-3 shows an XML representation of the two quality constraint in WXS (cf. Section 4.4). As the language is XML based and quite intuitive to understand we already present the example here. In Web-Flow several quality constraints may be combined in Boolean expressions using AND,
Event Detection and Handling

OR, or NOT operators. For instance, we can assign the combined quality constraint

$qc_1 \text{ AND } qc_2$  

(i)

to the operation „ATP check for monitor“.

3.3.2 Event Detection and Quality Constraint Monitoring

In this section we briefly sketch how the Web-Flow MaX component monitors events and evaluates quality constraints. The automated event detection for a particular process instance consists

---

45
The Web-Flow Architecture

of an initialization phase and of the actual monitoring of events and quality constraints. In Web-Flow the quality constraints are evaluated using a query processor of a relational database system to avoid a re-implementation of evaluation algorithms for Boolean expressions. Query processors are highly optimized so we also expect to reach a better performance by evaluating the constraint expressions on a relational database than evaluating the constraints on XML documents in an XML database. Furthermore, some constraints cannot directly be related to XML documents sent to or received from a Web service, for instance, response time constraints. Thus, it would be necessary to transform the constraints into XML documents. In the following, we sketch the initialization and monitoring phase of Web-Flow’s automated event detection.

3.3.2.1 Initialization Phase

In cooperative business scenarios the start of a new process instance is typically triggered by a message sent to the process engine from a partner or program (for instance, a website as interface to the customer) or from internal programs. When the Web-Flow MaX monitoring component receives a message indicating the start of a new process instance it starts the initialization phase. Therefore, it first retrieves the Web-Flow specific process and service definitions for the new pro-
Event Detection and Handling

cess instance from the metadata repository. They contain all quality constraints that have been assigned to service operations and activities used in that process. The quality constraint contains the desired values with which the constraint monitoring is initialized.

To evaluate the quality constraints automatically so called constraint evaluation queries $Q_{ce}$ are constructed. The constraint evaluation queries are standard SQL queries which return ‘true’ if the quality constraint is fulfilled and an empty result set if the quality constraint is violated. Therefore, they compare the actual values, for instance for the current response time measured with timers, to the desired values specified in the quality constraints (for example, 10 minutes as specified in $q_{c2}$). The queries are generated during the initialization phase and saved in the repository for the monitoring phase together with a reference to the process instance and the activity for which they have been defined.

3.3.2.2 Monitoring Phase

After the quality monitoring has been initialized the monitoring phase starts. The Web-Flow MaX monitoring component observes different event sources, for example, the messages sent between the process engine and the services. Another source for events are interfaces over which the Web-Flow MaX is notified about manual fault messages or external events. Such interfaces are, for instance, a graphical user interface for manual notifications or an application programing interface for automatic notifications. If a relevant event has been detected, for instance, a result message has been received from the activity „ATP check for monitor“, the necessary data to evaluate the quality constraints is extracted from the message.

For instance, for the result constraint $q_{c1}$ the XPath queries defined for the left and the right operand are executed on the XML documents. That means that the value of the $<$price$>$ element is retrieved from the result document „ATPResponseMonitor“ and the $<$maximumPrice$>$ element is retrieved from the result document of activity „Enter order details for monitor“. As this activity has been executed earlier the document is saved until activity „ATP check for monitor“ is actually executed. To determine which documents should be saved Web-Flow MaX analyzes dependencies between parameters of activities during the initialization phase. Comparisons of data retrieved from different messages can only be performed automatically if both data elements have the same data format. This is checked during constraint definition. If data elements have different formats a mapping should be specified to transform one data element in the format of the other element.

The constraint evaluation query $Q_{ce}$ for the actual activity is executed on the data extracted from the messages by the query engine. If the constraint expression is evaluated to false, at least one quality constraint has been violated. Then the Web-Flow MaX dynamic event and exception handling component is called with the information about the occurred event.
3.3.3 Web-Flow Event and Exception Handling

During the event handling the execution of the process instance is stopped by holding back the actual response message in the Web-Flow Max component. After event handling has been finished and an appropriate output message is available this message is sent to the process engine. In case of manual fault messages or external events the next message for an affected process instance is blocked by the Web-Flow Max component. This may either be a message sent from the process engine to a Web service or received from a Web service. In the following, we first sketch the Web-Flow event and exception handling model before we give an overview of the dynamic event and exception handling performed in Web-Flow.

3.3.3.1 Event Handling Model

The Web-Flow architecture aims at offering quality-oriented execution of Web-service-based cooperative processes. Thus, it supports the enforcement and control of quality constraints during the execution of a cooperative process. Furthermore, any event that may interfere with the successful execution of a cooperative process, for instance, manual fault messages or external events that may occur at any time during process execution, are also handled dynamically.

Events and faults that are already considered in the third category for quality support (cf. Section 3.1) and are intercepted in the cooperative process definition are handled directly by the process engine. For instance, in BPEL processes Web service fault messages may be handled using the „try-catch-throw“ constructs or timeouts may be handled with event handlers (cf. Section 2.2.1). Nevertheless, the Web-Flow Max event and exception handling component also observes these events before they are possibly handed over to the process engine. The exception handlers contained in a process definition are extracted during the initial parsing of the process definition in the Web-Flow Max constraint definition component. So, the process designer specifying quality constraints is informed about the event handling already provided when he specifies quality constraints for a process definition. Additionally, the exception handlers are saved as available actions in the metadata repository so that they may be selected for dynamic event and exception handling.

All events that are not covered in one of the first three categories for quality support (cf. Section 3.1) and all events that are related to the violation of quality constraints are handled by Web-Flow’s dynamic event and exception handling. To derive appropriate actions first the rules specifying actions for frequently occurring events are considered. If no matching rule was found appropriate actions are derived from logged execution and event handling data. This two-step approach has the advantage that it is not necessary to define and maintain a possibly large set of event handling rules. The process designer may concentrate on describing the desired execution quality.

Furthermore, rules may be derived semi-automatically based on the recommendations generated by the logging and analyses component. Furthermore, the readability and maintenance of the pro-
cess definition is facilitated as the standard execution and the event and exception handling logic are clearly separated. A further advantage is that actions are derived close in time to the actual occurrence of an event, so changes in business policies, laws, etc. can easily be taken into account without changes in the process definitions or in event rules. The use of rules for frequently occurring events improves the performance of the event handling as we may expect that it will take less time to search for a rule than to derive the action from a possibly large set of log entries.

3.3.3.2 Selection of Actions

For the selection of appropriate actions to handle an event the Web-Flow dynamic event and exception handling uses the data logged in the Web-Flow metadata repository. If an event occurs, for instance, the violation of the quality constraint $qc_f$ introduced in the example scenario in Section 3.3.1, and no appropriate rule may be found, the logging component is called to derive one or more appropriate actions from the log data.

The log data is searched for tuples which describe the handling of a similar event. Therefore, similarity values are calculated to describe the similarity between the current event and the event descriptions logged in the metadata repository. The similarity is calculated using the following attributes of a log entry:

- event type,
- service operation,
- semantic descriptions of a service operation such as service category, operation functionality, and parameter role,
- process definition in which the event occurred.

To improve performance of action selection the similarity values can be pre-calculated for instance during the night considering all log entries that are available up to this time. Based on the similarity values tuples are selected which may be used to handle an event. Therefore, thresholds are used that decide which tuples are similar enough to be considered. Depending on the number and kind of log entries the following three cases may occur:

- Exactly one action is found: The action is executed automatically after it has been confirmed by a responsible administrator in the organization currently controlling the cooperative process.
- No action is found: This case may occur especially during the introduction phase of Web-Flow when the log only contains few tuples or if completely new process definitions with new services and service providers are integrated in the Web-Flow system. In this case only a manual reaction is possible, i.e., a responsible administrator is notified about the event. He may either select one of the automatic actions which is then executed by Web-Flow or he performs some manual event handling. In either case the selected action is logged as an appropriate action for this type of event.
Several actions are found: In that case the priority assigned to the actions may be used to
derive which actions are most likely to be executed. Furthermore, impacts of an action on the
further execution of a cooperative process instance may be considered, e.g. whether the
action has been executed successfully or whether there have been more events when continu-
ing the process instance after applying the action. All actions are presented to a responsible
administrator who decides which of them are executed.

The priorities of actions are assigned manually when an administrator is notified about several
possible actions and orders them by defining their priorities. The tuples in the log database are al-
tered with the priority values. A detailed description of the algorithms for similarity calculation
and action selection used in Web-Flow is provided in Chapter 7 (Log-based Selection of Actions
and Recommendations).

3.3.3.3 Action Execution

To handle an event the Web-Flow MaX event and exception handling component executes the ac-
tions that have been derived from rules or by the logging and analysis component. The action types
supported by Web-Flow are introduced in Chapter 4 (Web-Flow Metamodel and Quality
Constraint Classification). The Web-Flow MaX event and exception handling component pro-
vides a set of operators that implement the different actions and allow for an automatic execution
of the actions after an obligatory user confirmation.

Besides the execution of the appropriate actions the dynamic event and exception handling of
Web-Flow also tries to consider impact of the event handling, for example, on the yet unexecuted
activities of the process instance and on cooperation partners. Additional service calls during
event handling may possibly delay the start time of future activities of the process instance. If a
quality constraint defines a fixed date as latest possible start time for one of the future activities,
it may already be derived that the delay may lead to a violation of the constraint although the ac-
tivity has not been executed yet.

As an illustration regard the following example. The response time constraint \(qc_3\) defining a max-
imal response time of 30 minutes for the operation „Credit check“ in the collaborative fulfillment
process is violated (see Fig. 3-4). The dynamic event and exception handling derives as appropri-
ate action to repeat the service call. The new service call is started at 11:10:00 a.m. and the exe-
cution of the following activities is delayed until the response message is finally received. Assume
further, that a quality constraint \(qc_4\) has been defined for the operation „Order monitor“ stating
that the operation is to be started not later than 11:10:00 a.m. (derived, for instance, from a dead-
line at which the price guaranteed in the ATP check response expires). It is clear that \(qc_4\) is vio-
lated when the execution of the process instance continues after event handling. So, the Web-Flow
MaX event and exception handling may check if all fixed response time or start time constraints
are met or if violations are already predictable and may for instance be prevented by relaxing some
deadlines. As time related constraints and time dependencies are quite common and important for
Event Detection and Handling

Example in medical cooperation scenarios or in supply chain scenarios, such impacts should be considered. Similar considerations are possible for the availability of output data. If an alternative service is executed that does not provide exactly the same output parameters some fields of the output document sent to the process engine may be left blank. If the Web-Flow MaX component may already derive during event handling that this currently unavailable data is needed by a following activity it may either notify an administrator or the service provider about the missing data in advance (note, that this is mainly useful for services which allow for manual interaction). Furthermore, such notifications are only useful if the complex cooperation model is used as only then a partner

Fig. 3-4: Temporal impacts of event handling for two sample quality constraints.
The Web-Flow Architecture

knows that its service is used in a particular process and is possibly able to derive the data from alternative sources.

3.4 Summary

In this chapter we first introduced the concepts and components of the Web-Flow architecture as well as the different parts of the Web-Flow MaX component comprising a constraint and rule editor, a quality constraint and event monitor, a dynamic event and exception handling, and a logging and analysis part. The Web-Flow architecture allows for the definition and monitoring of quality constraints for heterogeneous services that can be monitored completely at the service consumer side without violating the autonomy of the service provider. Furthermore, it offers automatic event detection and flexible exception handling. Exception handling is based on a rule-based approach which allows for deriving appropriate actions dynamically when an event really has occurred during process execution. To facilitate definition and maintenance of rules a log-based selection of appropriate actions is offered.

Finally, Web-Flow offers a generic solution that can be applied to cooperative processes executed in different process engines and in both cooperation models introduced in Section 1.1.3, namely process orchestration and process choreography. However, completely abstract processes are not supported, that means that we assume that there is always one particular organization controlling the execution of a process instance.

Compared to the different categories for execution quality support introduced in Chapter 2 Web-Flow belongs to the category in the front top right corner (cf. Fig. 3-5). Work in this category offers the comprehensive execution quality support that is required in cooperative scenarios. As discussed in Chapter 2 related approaches mainly focus on quality support in the other categories. Work on quality-oriented execution and composition of Web services covers the categories in the bottom half of the box (cf. Fig. 3-5). The categories in the upper half of the box are covered by work on exception handling for cooperative processes. Many approaches focus on the integration of pre-defined exception handlers in the process definition and are related to the top left category (cf. Fig. 3-5). Some approaches provide rule-based exception handling during runtime and belong to the front top right category. However, these approaches do not address all requirements that have to be addressed for a comprehensive execution quality support in cooperative scenarios (thus only one side of the according category dice is marked in Fig. 3-5). For instance, they do not offer a classification of quality characteristics for description and monitoring of process execution quality. Furthermore, they do not offer a combined exception handling approach consisting of a rule-based part and a log-based derivation of appropriate actions. Finally, they do not offer a generic solution applicable to different process engines and only support processes executed in a simple cooperation model.

Thus, the comprehensive support implemented in a generic solution as offered by Web-Flow ac-
complishes and extends existing approaches. Web-Flow also contributes to the realization of a multi-level execution quality support as it accomplishes any exception handling that is provided either in the applications realizing the service operations or in the cooperative process itself. Web-Flow allows to define and monitor additional quality characteristics even for services that do not offer any quality support themselves (for instance, as they are realized by legacy applications). Furthermore, Web-Flow handles violations of quality characteristics and other events that are not covered by exception handlers on service or process level.

Fig. 3-5: Comparison of Web-Flow to the categories for execution quality support.
The Web-Flow Architecture
4 Web-Flow Metamodel and Quality Constraint Classification

The algorithms of the Web-Flow architecture supporting a quality-oriented execution of cooperative processes depend on a formal description of the processes and services as well as of the desired quality characteristics specified by a process designer. To automate event and exception handling also an appropriate representation of events and actions is needed. This information is provided by a metamodel which supports the following aspects of quality-oriented process execution:

- structured description of the desired execution quality for heterogeneous and autonomous services and cooperative processes,
- widely automated quality monitoring and event detection,
- dynamic event / exception handling with reduced manual interaction,
- optimization of cooperative processes.

Web-Flow aims at offering a generic solution to support the execution quality of cooperative processes. Thus, the metamodel has to provide a representation of services and cooperative processes that abstracts from process engine or service specific models. The quality characteristics defined in Web-Flow should be applicable to heterogeneous services and should be monitored without violating the autonomy of the service provider. Furthermore, the constraints should allow for a comprehensive description of the execution quality. It has to be identified which quality characteristics satisfy these conditions and can also be monitored automatically.

To cover all these different aspects a comprehensive metamodel is defined for Web-Flow. Existing metamodels, for instance for quality characteristics, service descriptions, or exception handling, only cover some of the aspects and thus cannot provide the comprehensive and coherent representation needed in Web-Flow.

In this chapter we first introduce the Web-Flow metamodel for a quality-oriented description of cooperative processes (Section 4.1) and describe the classification of quality characteristics supported in Web-Flow (Section 4.2). We also discuss the exception handling concepts contained in the metamodel (Section 4.3). Finally, we present the XML-based language WXS (Web-Flow
4.1 Metamodel for Quality-Oriented Process Descriptions

To define the metamodel we investigated existing standards respectively standardization proposals for Web services and cooperative processes (for instance, [THAT2003, WSDL]) as well as metamodels for ontology-based service composition [OWLS2003, MEDJ2003A]. We derived basic properties of Web services and cooperative processes that are commonly used in these models. As the investigated metamodels provide a comprehensive view on services and cooperative processes, the derived concepts allow for a generic representation of services and cooperative processes. To identify appropriate quality characteristics we investigated different cooperative scenarios and related work already supporting quality characteristics mainly for service selection (for instance, [KEID2003]). We used these characteristics as a starting point for a definition of the Web-Flow quality characteristics. We extended these approaches to make them applicable for automated monitoring during runtime. The concepts for the support of rule-based exception handling have been derived from Event-Condition-Action (ECA) rules as, for instance, used in active databases [WIDO1996].

Fig. 4-1 provides an overview of the main classes of the metamodel. In the following we sketch the main classes and their relationships before providing detailed definitions of the quality-related concepts in the following sections. Appendix B (ODL Class Definitions for Web-Flow Metamodel) provides the ODL definitions for the main classes.

4.1.1 Overview

The Web-Flow architecture focusses on Cooperative Processes consisting of Activities which are executed by Service Operations. Each service operation belongs to a particular Service that is provided by an Organization (for an illustration of these concepts see Fig. 4-2). If the complex cooperation model is used (cf. Section 1.1.3) organizations are conscious of participating in a cooperative process which is expressed in the „participates in“ relationship. A service operation may have input, output, and fault parameters that are modeled in the „Parameter“ class. Semantic information about services, operations, and parameters such as service category, functionality of an operation, or role of an parameter is contained in attributes of the respective classes.

The remaining classes of the metamodel, namely Quality Constraint, Constraint Expression, Context, Event, Action, Condition, and Rule represent the concepts used for quality-oriented process description and dynamic event / exception handling. Quality Constraints describe quality charac-
Fig. 4-1: Main classes of the Web-Flow metamodel in UML. Some attributes have been omitted for clarity. The left number of a cardinality specifies the minimal number of associations an object may have to associated objects, the right number specifies the maximal number ("*" stands for "unlimited").
Web-Flow Metamodel and Quality Constraint Classification

Quality constraints may be assigned to services, service operations, activities, or an organization running a Web-Flow MaX component. Furthermore, quality constraints may be assigned to a particular context, i.e., the constraint only applies if a service operation is executed in a particular sequence of preceding and succeeding activities. The scope attribute of a quality constraint specifies the range of a quality constraint, e.g., one particular operation or all services used in an organization (more details are given in Section 4.2.1). Several single quality constraints holding for one service operation, activity, service, or organization may be combined into a constraint expression using the boolean operators AND, OR, or NOT. For a quality condition set by a constraint expression to be valid it is necessary that the whole constraint expression is valid. So, it is not sufficient that the single quality constraints mak-

Fig. 4-2: Illustration of metadata classes.
Exceptions occurring during the execution of cooperative process instances are related to Events. Events are, for instance, the violation of quality constraints, Web service fault messages, or manual fault messages received, for example, by phone or e-mail. Also external events may occur, such as notifications about changes in general business policies of a company or about the sudden unavailability of a preferred service provider. Web-Flow aims at handling those events dynamically by applying appropriate Actions to ensure that the cooperative processes may successfully be executed despite the exception. This helps to raise the flexibility and robustness of cooperative processes. Rules may be defined that describe which actions are to be executed to handle particular events. Rules may restrict the application of an action to particular operations during whose execution the event has occurred. Furthermore, a rule may have a Condition which specifies particular circumstances under which the action is applied, for instance, the number of repeated service calls. The use of rules also may accelerate handling of frequent events as appropriate actions may be found faster.

4.1.2 Semantic Description of Services, Operations, and Parameters

Semantic information about the category of a particular service, the functionality of its operations, and the roles of the operations’ parameters is used in Web-Flow in addition to the syntactical information about services such as its operations, input and output parameters. These attributes are used during the log-based derivation of appropriate actions for exception handling (cf. Section 3.3.3.2). The annotations help to identify log entries for events that have happened during the execution of service operations similar to that one for which the current exception occurred. Similarity is considered between service categories, operation functionalities, and parameter roles.

The semantic annotations in the Web-Flow metamodel are similar to some of the attributes used for instance in OWL-S [OWLS2003], in the VirginiaTech project [MEDJ2003A], or in METEOR-S [SIVA2003, VERM2004]. These approaches focus on providing comprehensive ontological information to support service discovery, composition, and also execution. For instance, a service profile in OWL-S describes the functionality of the service, the pre- and post-conditions as well as the effects of the service execution. [MEDJ2003A] provides an ontology-based framework for the automatic composition of Web services. Additionally, composability rules are provided that ensure a meaningful composition of services. METEOR-S [SIVA2003, VERM2004] annotates WSDL service descriptions with concepts from domain ontologies to capture the semantic information about the service to improve service discovery. As the focus of Web-Flow is on automated quality monitoring and exception handling only a subset of the information provided by these related approaches is used. Also no complete ontologies are needed to support the mechanisms offered by Web-Flow, but attributes deriving their values from taxonomies are sufficient. Thus, to reduce the overhead when using ontologies and to provide a generic solution that could also be applied to services without WSDL descriptions (in contrast to the approach of METEOR-S) we
did not use one of the existing approaches but added the necessary semantic attributes to the Web-Flow metamodel.

The category of a service describes to which domain a service belongs, for instance „insurance“, „health care“, or „manufacturing“. The values of the domain attribute are provided by pre-defined taxonomies for service categories which are referenced using a URI (uniform resource identifier [W3C2001B]). Possible taxonomies for service categories are, for example, NAICS (North American Industry Classification System, [NAIC2002]) or UNSPSC (Universal Standard Products and Services Classification, [UNSP2004]). Also synonymous category descriptions may be assigned to a service taken from the same or a different taxonomy. This ensures that during action derivation a similar service may be found although it has been annotated with a more general category or a value taken from a different taxonomy. In Web-Flow the service category is defined as follows:

Definition 4-1 - Category. The category of a service is defined by a tuple \((domain, taxonomy, synonyms)\) where

- \(domain\) is the domain to which a service belongs.
- \(taxonomy\) is the URI of the taxonomy used.
- \(synonyms\) is a set of tuples \((domain, taxonomy)\) defining alternative \(domain\) values taken from \(taxonomy\) where the synonymous taxonomy may be different from the top level taxonomy.

An example for a semantic description of the service category of a monitor seller service is the following:

\[(43211900, http://www.unspsc.org,\]
\[ (43211901, http://www.unspsc.org), \]
\[ (43211902, http://www.unspsc.org), \]
\[ (43211903, http://www.unspsc.org), \]
\[ (43211904, http://www.unspsc.org), \]
\[ (43211905, http://www.unspsc.org))\]

All category values have been taken from the UNSPSC taxonomy which is hierarchically organized. The first domain value, category number 43211900, denotes the top level category of computer displays indicating that the service belongs to this domain. The synonymous domain values 43211901 to 43211905, denote different types of displays (for instance, LCD displays). As a monitor seller service typically deals with different types of displays, it also belong to these more specialized domains. Assume that service A has been annotated with the values shown above and service B has only been annotated with the value 43211905 from the UNSPSC taxonomy:

\[(43211905, http://www.unspsc.org)\]
Nevertheless, it is possible to identify the two services as similar because service A provides the synonymous categories.

The *functionality* of a service operation describes which business function is provided by the operation, for instance, „request availability“, „purchase order“, or „notify of manufacturing work order“. The possible values are again provided by taxonomies that define a common vocabulary for the description of operation functionalities, for instance, RosettaNet [ROSE2004], cXML (commerce XML, [CXML2004]) or EDI (for a description see [MEDJ2003B]). The used taxonomy is denoted in the *taxonomy* attribute of the operation functionality using a URI. It is again possible to specify synonyms for an operation’s functionality such as „check availability“ as a synonym for „request availability“. In Web-Flow the operation functionality is defined as follows:

**Definition 4-2 - Functionality.** The functionality of a service operation is defined by a tuple *(function, taxonomy, synonyms)* where

- *function* is the business functionality of the operation defined in the taxonomy denoted by *taxonomy*.
- *taxonomy* is the URI of the taxonomy used.
- *synonyms* is a set of tuples *(function, taxonomy)* defining alternative *function* values taken from *taxonomy* where the synonymous taxonomy may be different from the top level taxonomy.

The *role* of an input, output, or fault parameter of a service operation denotes the semantics of a parameter, i.e., to which business object this parameter is related. Business objects are, for instance, the order status, an invoice, or an acknowledgement of acceptance. The possible values for the parameter’s role are taken from a taxonomy to give each parameter a well-defined semantics according to this taxonomy. A possible taxonomy for parameter roles is, for example, the RosettaNet Business Dictionary [MEDJ2003B] which provides a common vocabulary for the description of business entities. Again it is possible to specify synonyms, for instance taken from a company-specific taxonomy. In Web-Flow a parameter role is defined as follows:

**Definition 4-3 - Parameter role.** The role of an input, output, or fault parameter of a service operation is defined by a tuple *(role, taxonomy, synonyms)* where

- *role* is the business object assigned to the parameter defined in the taxonomy denoted by *taxonomy*.
- *taxonomy* is the URI of the taxonomy used.
- *synonyms* is a set of tuples *(role, taxonomy)* defining alternative *role* values taken from *taxonomy* where the synonymous taxonomy may be different from the top level taxonomy.

A message representing a service parameter may consist of several parts each of which represents a different business object. To capture this more detailed description of a parameter, it is also possible to annotate the different message parts with parameter roles in the Web-Flow metamodel.

The semantic information about categories, functionalities, and parameter roles is assigned man-
ually when a service is imported in Web-Flow for the first time. Besides the use during log-based action selection categories of services and functionality of operations may be considered during analysis of the logged execution data. Furthermore, semantic characteristics may be useful to restrict the search space for alternative services during event and exception handling. For instance, all services used by a company may be grouped according to their category. When the search query denotes a desired service category, then only those services with a matching category have to be considered.

4.2 Quality Constraints

Quality constraints are used in Web-Flow to specify quality characteristics for the execution of Web services in a cooperative process. Therefore, they allow for a description of the desired execution quality of a cooperative process. These characteristics complete the conditions a service may offer itself, for instance, by an advanced implementation that is able to check service output parameters before returning them to the service user. Additionally, a service provider may use policies to describe how a service is to be used (for instance, as provided by the WS-Policy framework [HOND2003]). In contrast to such policies quality constraints refer to execution characteristics such as start time, response time, iteration count, name or location of service provider, or constraints on service input and output parameters (for instance, price limits, product configurations, or delivery deadlines). We have derived different types of quality constraints from the base class shown in Fig. 4-1 which we introduce in detail in this section.

4.2.1 Types and Scope of Quality Constraints

We distinguish several types of quality constraints, namely metadata constraints, execution constraints and input respectively result constraints. The different constraint types are used for different purposes. Metadata constraints refer to rather static information, which is specified when a service is registered in a repository (for example, the service provider or its location). These constraints are primarily useful for service selection, i.e., to find a suitable service during process execution or to find an alternative service to handle an event or exception. The other types of constraints are more dynamic in the sense that they refer to information related to the actual execution of Web services, i.e., to a concrete Web service call in a cooperative process. Fig. 4-3 gives an overview over the different quality constraint types.

Quality characteristics can be imposed on complete services (i.e., on all operation a service offers) as well as on single service operations. The quality characteristics may also depend on the type of the process in which a service operation is used. For example, quality characteristics may only be valid in a collaborative fulfillment scenario but not in a medical scenario. The scope of a quality constraint describes for which objects a constraint is valid. It relates a constraint either to a service
Quality Constraints

(service specific), an operation (operation specific) or to the use of an operation in a particular process (process specific). Service specific constraints are valid for all operations of a service, e.g., to restrict the provider of a service in a metadata constraint. Operation specific constraints hold for each use of a service operation independent of a particular process, i.e., the constraint for a payment service applies regardless if the service is used in a travel booking process or in a collaborative fulfillment process. In contrast to that process specific constraints are only valid for the use of a service operation in a particular process. If company-wide regulations are to be enforced such as default values for maximal response times or a list of preferable service providers, quality constraints may have a global scope. That means that they apply to several or all services used in a company.

If quality constraints should only apply if a service operation is used in a particular context, i.e., in a certain sequence of service calls, the scope of the constraints is context dependent. To illustrate the usage of context dependent constraints consider the following scenario. In a medical context the allowed ranges in which a blood value is regarded to be critical may depend on the drugs applied beforehand. Thus, a context dependent quality constraint with reduced blood ranges may be assigned to a service operation examining the blood value after such a drug has been applied.
In all other use cases of this examination operation the standard blood ranges apply. If no context dependent constraints are available it has to be checked manually whether a particular context of a service operation applies in a particular process. Thus, context dependent constraints relieve the process designer as the constraint only has to be defined once independently of a concrete process. The checks whether the constraint applies in a particular process can be performed automatically in the Web-Flow MaX component. This also reduces the overall number of constraints that have to be maintained.

To specify different contexts Web-Flow uses the class `context`. In Web-Flow a context is defined by a set of service operations that are executed in a certain range before respectively after a particular operation. For instance, a preceding service operation describing a context may not necessarily be executed directly before the particular operation, but may, for instance, be the pre-preceding operation. If the context is determined by several preceding or succeeding operations the context description also specifies the order in which the activities have to be executed to contribute to the context. The context is typically checked on the process definition, i.e., on the planned process execution. A check on process instance level is only performed if the preceding or succeeding operations are part of conditional branchings. Then it may not be determined until runtime whether an operation is executed in a particular context. The context class also contains the quality constraints which apply for the operation that is executed in this particular context. The context class is defined as follows:

**Definition 4-4 - Context.** A context $c$ is a tuple $(forOperation, predecessors, successors, predecessorRange, successorRange, qualityConstraint)$ where

- $forOperation$ contains the particular service operation for which this context is defined.
- $predecessors$ is an ordered list of service operations which have to be executed before the operation for which the context is defined. The ordering in the list describes the order in which the operations have to occur in the process definition.
- $successors$ is an ordered list of service operations which have to be executed after the operation for which the context is defined. The ordering in the list again describes the order in which the operations have to occur in the process definition.
- To contribute to the context the first of the predecessor operations has to be executed at most $predecessorRange$ activities before $forOperation$ and the last of the successor operations has to be executed at most $successorRange$ activities after $forOperation$.
- $qualityConstraint$ denotes the constraint or constraint expression that should apply if the operation is executed in this context.

An example for a context specification is the following:

$$(op, (pred1, pred2), 3, (succ1, succ2), 2, qualityConstraint)$$

It specifies the context of the operation $op$ consisting of the operations $pred1$ and $pred2$ which have to be executed in the order „pred1, pred2“ within the last three operations before $op$. The
context is completed by the operations \textit{succ1} and \textit{succ2} which have to be executed in the order \textquotedblleft\textit{succ1, succ2}\textquotedblright within the next two operations after \textit{op}. This context is illustrated in Process 1 in the upper part of Fig. 4-4.

Context dependent constraints aim at defining quality constraints independently of concrete process definitions. Thus, we do not specify the particular control flow constructs in which the predecessor and successor operations occur. This would make the context definition less general as we would have to specify the control flow in detail. For instance, we would have to specify the exact number of parallel paths or the conditions for conditional branchings. To specify quality constraints that depend on the particular control flow of a process Web-Flow provides process specific quality constraints. Thus, in Web-Flow the activities of a context do not necessarily have to occur in a sequence. The context is also fulfilled if the predecessors are for instance the last activities of a parallel path while the operation itself and the successors are part of the sequence following the parallel flow construct. This case is illustrated in the process fragment \textit{Process 2} shown in the middle of Fig. 4-4.

\textbf{Fig. 4-4:} Illustration of the same context in different control flow constructs.
However, we have to consider the case that some of the context nodes are part of a conditional path (for instance, the first predecessor node in the Process 2 in Fig. 4-4). In this case it may not be determined until process execution time whether an activity is in fact executed in a particular context. Thus, before context dependent constraints may be evaluated it has to be checked during process execution whether the context is in fact fulfilled, i.e., whether the specified preceding operations have been executed.

However, when a context dependent quality constraint is evaluated for an operation $op$ it is not always possible to determine whether the succeeding operations specifying the context will really be executed. In particular, if the successor nodes are part of a conditional branching (as shown in Process 3 in Fig. 4-4) we can only try to predict which path will in fact be executed. For instance, if the data needed for evaluating the conditions $cond_1$ and $cond_2$ in Fig. 4-4 is already known when operation $op$ is executed the context can be evaluated.

As the prediction of conditional paths is out of scope of this thesis we refer to the literature (for instance, [MÜLL2002]). If no prediction is possible we still evaluate the context dependent quality constraint. If the constraint is violated we inform a responsible administrator that the complete context could not be determined automatically together with the request for confirming the exception handling actions. Thus, the administrator has the final decision in automatically undecidable cases.

### 4.2.2 Comparison Operators

Before we go into the details of the different quality constraint types, we take a look on the comparison operators used in the constraints. Each quality constraint is defined as a comparison either between two values specified in the constraint (for input, result, and metadata constraints) or between a value defined in the constraint and an execution characteristic that is derived dynamically during process instance execution by the Web-Flow MaX monitoring component (for execution constraints). Tab. 4-1 lists the comparison operators supported by Web-Flow.

These comparison operators cover the most common comparison types for the built-in data types of XML schema [W3C2001A] typically used in WSDL messages. Furthermore, the WXS language described later in this section is also built on XML schema. All comparisons may be executed automatically by the Web-Flow MaX monitoring component using the mechanisms of programming languages such as JAVA [JAVA2004]. The restrictions, for instance w.r.t. the use of the less operator with string operands, may be already checked in the Web-Flow editor component when the constraints are specified. So, we think most quality constraints may be defined with these operators. Of course, an extension of the operator set is possible in the future.
4.2.3 Definitions for Quality Constraints

Quality characteristics can be applied to rather static information that is relevant during service selection as well as to dynamic information that is relevant during service execution in concrete processes. Thus, Web-Flow supports different types of quality constraints. Metadata constraints apply to more static information whereas execution constraints and input/result constraints refer to quality characteristics relevant during execution. In this section we provide detailed definitions of the different quality constraint types.

**Metadata constraints** refer to conditions on the description of Web services, e.g. on the services’ UDDI metadata (Universal Description, Discovery and Integration, [BELL2003]) or WSDL metadata. They can restrict many different aspects, e.g. the provider of a service (specific companies, branch of business, specific geographic locations, etc.) or a fee that may have to be paid for using a service. Furthermore, they may restrict the service category, the functionality of service operations, or the roles of operation parameters as defined above in Section 4.1.2. Thus, metadata constraints restrict static characteristics of a service or its operations and are mainly used during service selection. They are not used for activities and therefore they do not have a process specific or context dependent scope. With the strictness attribute an administrator may specify that a constraint should be fulfilled but that its violation may also be tolerated if otherwise no appropriate service may be found. The allowed values for the strictness attribute are „high“ and „low“ (similar classifications are described in [KEID2003] for constraints for dynamic service selection and in [KIEß2002] for conditions in SQL statements). A metadata constraint is defined as follows:

**Definition 4-5 - Metadata constraint.** A metadata constraint $mc$ is a tuple $(\text{hasScope}, \text{strictness}, \text{refersTo}, \text{comparisonOperator}, \text{leftOperand}, \text{rightOperand})$ where

- $\text{hasScope}$ denotes the scope of $mc$. The possible values are global, service specific, operation specific.
Web-Flow Metamodel and Quality Constraint Classification

- **strictness** denotes whether the constraint is mandatory or whether it only should be fulfilled.
- **refersTo** denotes the object to which \( mc \) is assigned, i.e. a service or an operation. For global metadata constraints this attribute is left empty.
- **comparisonOperator** denotes the operator to be used to compare the left and the right operand of \( mc \).
- **leftOperand** denotes an XPath query and a parameter name that describe how the actual value of the left operand can be retrieved. The parameter name specifies an XML document containing metadata (e.g., a service description document retrieved from an UDDI directory or the Web-Flow metadata repository). The XPath query is used to extract the particular data element from this document.
- **rightOperand** either contains an XPath query and a parameter name as described for the leftOperand attribute or a fixed value to which the leftOperand is compared. The fixed value may also contain the path to a data source (e.g., a database system) and a query which retrieves the fixed value.

As an example consider the following metadata constraint that specifies a constraint for the selection of a distribution center that ships the PC to the customer in the collaborative fulfillment scenario (cf. Fig. 1-1). The scope and strictness attributes are omitted for better readability (also in all subsequent examples):

\[ mc_1: (...) \text{ distribution center service, equal,} \]
\[ \text{(serviceProviderDescription, /distributionCenter/location/state),} \]
\[ \text{(customerDetails, /customerData/address/state))} \]

The constraint specifies that the **distribution center service** should be located in the same state as the customer. The locations of the distribution center and the customer are specified in the parameters **serviceProviderDescription** respectively **customerDetails** and are accessed using the XPath expressions specified in the constraint. The background for this constraint may be that a long transport from the PC manufacturer to the distribution center is cheaper than a long transport from the distribution center to the customer as the PC manufacturer may bundle its deliveries for particular states.

**Response time constraints** restrict the time interval during which a response from a service operation is expected. They are defined on execution characteristics which are not accessible until a service operation is called to execute an activity. Response time constraints may also specify a fixed date (for instance, relative to the start time of a process instance) as the earliest or latest date at which a response is expected. Typically the type (earliest or latest) is specified by the comparison operator. An earliest response date may be useful if the service user knows that the response cannot be processed before a particular date, e.g., as the necessary machines are not working due
Quality Constraints

to maintenance operations. To prevent the response from being lost, an earliest response time may be specified. However, typically a response time constraint specifies a duration value specifying how long the user wants to wait for a response. The strictness attribute of a response time constraint is mainly used during event handling and specifies that a constraint should be fulfilled but that its violation may also be tolerated under certain conditions (for more details see Chapter 6). The allowed values for the strictness attribute are „high“ and „low“. A response time constraint is defined as follows:

**Definition 4-6 - Response time constraint.** A response time constraint $rtc$ is a tuple $(\text{hasScope}, \text{strictness}, \text{refersTo}, \text{comparisonOperator}, \text{durationOperand}, \text{dateOperand})$ where

- $\text{hasScope}$ denotes the scope of $rtc$. The possible values are global, operation specific, process specific, context dependent.
- $\text{strictness}$ denotes whether the constraint is mandatory or whether it only should be fulfilled. Per default, response time constraints are mandatory.
- $\text{refersTo}$ denotes the object to which $rtc$ is assigned, i.e., an operation or an activity of a process. This attribute is left empty for global response time constraints.
- $\text{comparisonOperator}$ denotes the operator to compare the actual response time to the expected response time specified by $rtc$.
- $\text{durationOperand}$ defines the expected response time interval.
- $\text{dateOperand}$ defines the expected response date.

Response time constraints as all other execution constraints refer to activities or service operations but not to whole services as they refer to quality characteristics that are not available until process execution. Therefore, the possible scope values exclude the type „service specific“.

The following response time constraint specifies that the operation $\text{CreditCheck}$ called by the PC manufacturer in the fragment of the collaborative fulfillment process in Fig. 4-2 should answer in at most 10 minutes:

$rtc_1: (..., \text{CreditCheck}, \text{lessEqual}, 10 \text{ minutes})$

As Web-Flow is based on a service-oriented architecture a service operation call in a process may represent a whole process executed at the service provider organization. Thus, response time constraints may also be used to specify temporal constraints on whole processes. Therefore, they are defined on the activity in the top-level process (e.g., as process-specific response time constraint) and are then forwarded to the Web-Flow MaX component of the partner executing the process together with the process call. The Web-Flow MaX component then may additionally consider the overall process execution time e.g. during event handling.

Start time constraints are similar to response time constraints. They define an earliest or latest start date for a service. Whether a particular constraint defines an earliest or latest startpoint depends
on the comparison operator used in the constraint. The actual values for start time constraints are often dynamically assigned not until process execution time. Therefore, the start time constraint may either contain the desired start time or an XPath expression specifying how the desired value may be retrieved, for example, from an output document. Start time constraints cannot have a global scope as they pose a condition on the execution of a particular activity. Start time constraints are monitored by the partner that controls the whole cooperative process as only he may decide when an activity is started. So, they are mainly applicable when one dedicated partner controls the whole process. A start time constraint is defined as follows:

**Definition 4-7 - Start time constraint.** A start time constraint \( stc \) is a tuple \((\text{hasScope}, \text{refersTo}, \text{comparisonOperator}, \text{dateOperand})\) where

- \( \text{hasScope} \) denotes the scope of \( stc \). The possible values are operation specific, process specific, and context dependent.
- \( \text{refersTo} \) denotes the object to which \( stc \) is assigned, i.e. an operation or an activity of a process.
- \( \text{comparisonOperator} \) denotes the operator to compare the actual start time to the desired start time specified by \( stc \).
- \( \text{dateOperand} \) defines the expected start date either using a real date value or an XPath expression describing how the value may be retrieved from an XML document.

**Iteration constraints** specify a maximal number of re-trials for calls to the same service operation during the execution of an activity in a process instance. They may, for instance, restrict the number of repeated operation calls after a failed service execution. Iteration constraints are defined as follows:

**Definition 4-8 - Iteration constraint.** An iteration constraint \( itc \) is a tuple \((\text{hasScope}, \text{refersTo}, \text{comparisonOperator}, \text{iterationOperand})\) where

- \( \text{hasScope} \) denotes the scope of \( itc \). The possible values are global, operation specific, process specific, and context dependent.
- \( \text{refersTo} \) denotes the object to which \( itc \) is assigned, i.e. an operation or an activity of a process. This attribute is left empty for global iteration constraints.
- \( \text{comparisonOperator} \) denotes the operator to compare the actual iteration number to the iteration number specified by \( itc \).
- \( \text{iterationOperand} \) defines the maximal number of re-trials for calls to the same service operation.

The following iteration constraint specifies that the operation *CreditCheck* in the collaborative fulfillment process in Fig. 4-2 is called at most three times. For example, that means that the operation call may only be repeated twice if the response time constraint \( rtc_1 \) specified above has been violated:

\( itc_1: (..., \text{CreditCheck}, \text{lessEqual}, 3) \)
Input and result constraints pose conditions on the input respectively the output parameters of a Web service call. Thus, they typically refer to XML documents sent to or returned by a service. An input or result constraint compares a data element of an input or output document either to a fixed value specified in the constraint or to another data element of the same or a different XML document. As the concrete messages are not available until a service operation is in fact executed, input and result constraints apply during process execution. For instance, with a result constraint it can be checked whether the price of a monitor returned in an ATP check answer is less or equal to the maximum price that has been specified by the customer in the monitor details earlier (cf. Fig. 4-2). Input and result constraints cannot have a global scope as they refer to input respectively output documents of specific service operations. Input and result constraints are defined as follows:

**Definition 4-9 - Input / Result constraint.** An input constraint \(ic\) / result constraint \(rc\) is a tuple \((\text{hasScope}, \text{refersTo}, \text{comparisonOperator}, \text{leftOperand}, \text{rightOperand})\) where

- \(\text{hasScope}\) denotes the scope of \(ic/rc\). The possible values are operation specific, process specific, and context dependent.
- \(\text{refersTo}\) denotes the object to which \(ic/rc\) is assigned, i.e. an operation or an activity.
- \(\text{comparisonOperator}\) denotes the operator to be used to compare the left and the right operand of \(ic/rc\).
- \(\text{leftOperand}\) denotes an XPath query and a parameter name that describe how the actual value of the left operand can be retrieved. The parameter name specifies an XML document containing input or output data. The XPath query is used to extract the particular data element from this document.
- \(\text{rightOperand}\) either contains an XPath query and a parameter name as described for the \(\text{leftOperand}\) attribute above or a fixed value to which the \(\text{leftOperand}\) is compared. The fixed value may also contain the path to a data source (for instance, a database system) and a query which retrieves the fixed value.

The following example specifies a result constraint checking the price of a monitor as described above:

\[rc_1: (\ldots, \text{ATP check for monitor}, \lesseqqref{\ldots}{\text{ATP check for monitor}}, \text{lessEqual}, \text{lessEqual}, \text{lessEqual}, (\text{ATPResponseMonitor, }/\text{ATPResponse/monitorDetails/price}), (\text{OrderDetailsMonitor, }/\text{OrderDetails/Monitor/maximumPrice}))\]

Similar to execution constraints input and result constraint are defined on items that are not available until a service operation is in fact called to execute an activity. Therefore, they may not have a service specific scope as they cannot apply to all operations of a particular service as the different operations have different parameters.
The quality constraint types presented in this section cover those quality characteristics which may be observed automatically if we only use the messages sent between the process engine and the different services as data source. For each message we can check if the message is sent or received in time respectively check its content. Additionally, we may use search results as data source for the monitoring of metadata constraint. As the Web-Flow architecture provides a generic infrastructure we cannot assume that other sources than the messages and the search results are available. So these quality constraint types represent all quality constraints which are automatically observable in a generic infrastructure for the quality-oriented execution of Web-service-based cooperative processes.

4.2.4 Constraint Expressions

As described in the last section quality constraints are specified as logical comparison predicates. Each quality constraint may be evaluated to either true or false depending on the outcome of the defined comparison. That means, a constraint is true, if the content of the XML document, the iteration number, or the response respectively start time fulfill the condition imposed by the constraint.

For one particular service, operation, or activity more than one quality constraint may be defined. These quality constraints can be concatenated to a more complex condition using boolean AND, OR, and NOT operators and parenthesis. We call this complex condition constraint expression as it is similar to a boolean expression. For instance, a response time constraint $rtc$ and a result constraint $rc$ can be combined to specify that the monitor ATP check should answer in less than 10 minutes ($rtc$) and that the price for a monitor should be less than 200 EUR ($rc$). The constraint expression formed of these two constraints is

$$(rtc \text{ AND } rc).$$

A constraint expression may consist of an arbitrary number of nested quality constraints of the same or different types. All constraints combined in one constraint expression must have the same scope, i.e., they refer to the same object. Thus, it is not allowed to combine an operation specific constraint with a process specific constraint in one constraint expression. This ensures that constraint expressions with different scopes are clearly separated and allows for using only constraints of one scope for a particular activity even if other constraints exist. For instance, if a process specific constraint expression (referring to activity $a_1$) and an operation specific constraint expression (referring to the service operation $o_1$ called to execute $a_1$) exist for a particular activity $a_1$ the process designer may decide that only the process specific constraint expression is to be used in this particular process definition (cf. Chapter 5). Formally a constraint expression is defined as follows:
**Definition 4-10 - Constraint Expression.** A constraint expression is defined as a logical expression consisting of a set of quality constraints which are combined using the Boolean operators AND, OR, and NOT and parenthesis. The quality constraint combined in one constraint expression must have the same scope, i.e., they must be defined for the same object namely service, operation, or activity.

Existing query languages such as SQL, the standard query language for relational databases, also provide advanced mechanisms for building conditional expressions. The constraint expressions provided by Web-Flow do not cover all functions offered, for instance, by SQL. However, they cover a great part of possible constraint expressions and thus are sufficient for providing comprehensive quality support in most applications. An extension of Web-Flow’s constraint expression to reach the expressional power of SQL is not in the scope of this thesis.

### 4.3 Exception Handling Concepts

Exception handling concepts are the third main part of the Web-Flow metamodel besides quality constraints and the concepts related to processes and services. Web-Flow uses a rule-based exception handling approach, thus the main concepts we introduce in this section are events, actions, and rules.

#### 4.3.1 Events

An event in the Web-Flow MaX metamodel represents things that may happen during the execution of a process instance and may possibly lead to an exception that has to be handled appropriately to ensure successful process execution. Possible events are, for instance, a failed service execution marked by a service fault message returned to the service consumer or the violation of a quality constraint detected by the Web-Flow MaX monitoring component. Events are related to data provided by different sources that are automatically observable such as fault messages, the evaluation of quality constraints, or external data sources (for instance, databases). Thus, four different classes of events are considered in Web-Flow as shown in Fig. 4-5: constraint violations, Web service fault messages, manual fault messages, and external events. In the following, we provide definitions for these event classes that may be automatically detected in the Web-Flow MaX component.

The first event class *constraint violation* is related to the quality constraints defined in the last section. An event of this type signals that a quality constraint respectively a constraint expression defined for a service, an operation, or an activity has been violated. For instance a response time deadline has exceeded or a result constraint has been evaluated to false. The violated quality constraint is contained in the event description as well as the object (i.e., service, operation, or activity) for which the event occurred. The violated constraint may also be a global constraint, as the
global constraints are assigned to services, operations or activities during the monitoring of a process instance. Additionally, the event contains the process instance in which the event occurred (if one could be assigned) and the point in time when it occurred. This information is needed to derive an appropriate action for handling the event from log data.

The event class Web service fault message describes events that are related to the reception of a Web service fault message. This type of event may either directly be handled by exception handlers contained in the process definition but can also be handled by Web-Flow specific event handling actions (the actions supported by Web-Flow will be described in Section 4.3.2). The fault messages that may be received from a Web service operation are described in its WSDL description.

The event class manual fault message is related to fault messages that are received from the service provider, for instance, per e-mail or phone and relate to the execution of a certain service in a particular process instance. To make these messages available to the Web-Flow MaX component an appropriate interface has to be defined over which an administrator can enter the fault message. To enable Web-Flow to evaluate the information automatically a number of pre-defined XML messages are available in Web-Flow that can be processed automatically by the Web-Flow MaX monitoring component. In particular the following message types are supported:

- a message that the service execution failed,
- a notification that the service execution will be delayed for a certain time span,
- a message that a particular service provider is out of business for some time,
- a message with free text that can be used for all events that do not match one of the types above. These events cannot be handled automatically but at least an administrator can be notified (for the appropriate action see Section 4.3.2).

If the interface should support further messages, these would have to be added to the Web-Flow MaX component and the interface has to be extended with the appropriate forms and the evaluation algorithms for the new messages. A manual fault message event is defined as follows:
The definitions of the first three event types have the same structure and differ only in one attribute. This attribute specifies the particular event type, either a violated quality constraint or a fault message. The definition is the following:

**Definition 4-11 - Constraint violation respectively fault message event.** A constraint violation respectively fault message event is a tuple \((\text{occursFor}, \text{occurredAt}, \text{occurredIn}, \text{constraint} | \text{faultMessage} | \text{manualFaultMessage})\) where

- \(\text{occursFor}\) denotes the object (service, operation, or activity) for which the event has occurred.
- \(\text{occurredAt}\) is the point in time at which the event has been detected by the monitoring component.
- \(\text{occurredIn}\) specifies the process instance in which the event has occurred.
- \(\text{constraint}\) specifies the particular quality constraint that has been violated by the event.
- \(\text{faultMessage}\) contains the XML document with the service fault message.
- \(\text{manualFaultMessage}\) contains the XML document with the manual fault message that has been entered in the interface.

The event class *external event* is related to notifications of the Web-Flow MaX component about events that have occurred independently of a particular process execution (i.e., are not related to an execution of a service operation in the context of a process instance) but may influence the successful execution of cooperative process instances. For instance, if the budget of a department has been reduced by 30%, result constraints assigned to activities in running process instances may have to be changed to use lower price limits. If a service provider is suddenly not able to execute its services anymore (e.g., as he has gone bankrupt or due to a nature catastrophe) all running process instances have to be checked if they use services of this partner and alternative service providers have to be searched to ensure that the process instances may be continued. External events also may notify the Web-Flow MaX about any changed data that is used in input or result constraints (as e.g. fixed values may be read from external data sources). The Web-Flow MaX component provides an interface over which external events, sent for instance by database triggers, can be received. As discussed for the manual fault message event class above Web-Flow provides a set of predefined XML messages that can be used for notifications about external events to allow for an automatic evaluation. External events are formally defined as follows:

**Definition 4-12 - External event.** An external event \(ee\) is a tuple \((\text{occurredAt}, \text{dataSource}, \text{notificationMessage})\) where

- \(\text{occurredAt}\) is the point in time at which \(ee\) has been detected by the monitoring component.
- \(\text{dataSource}\) specifies the database or any other data source in which the event occurred.
- \(\text{notificationMessage}\) contains the XML document with the notification about \(ee\) that has been sent over the interface to the Web-Flow MaX component.
Other events that may be observed by the Web-Flow MaX monitoring component are timeouts, sending an input, or receiving an output message. These events typically trigger the evaluation of quality constraints. Thus, they are only indirectly related to exceptions which may occur during process execution. However, as these events may also be used in rules (cf. Section 4.3.3) we shortly describe them here.

A timeout event describes that something has not happened in time, i.e., until a certain timespan has elapsed. In Web-Flow timeout events are particularly relevant in combination with response time or start time constraints. For instance, if a response time constraint has been specified for a service operation a timer is started with the interval specified in the constraint when the service operation is called. The timer generates a timeout if the operation has not sent a response in the interval specified by the response time constraint. A timeout event is a tuple \((\text{timeoutInterval}, \text{occurredAt}, \text{affectedObject})\). The attributes specify the time interval as it has been defined, for instance, in the response time constraint, the point in time when the timeout occurred and the object monitored by the timer (for example, a service call).

Events that describe the sending of an input message and the reception of an output message are mainly used to trigger the evaluation of quality constraints, for instance, of input or result constraints. The event definitions for sending an input or receiving an output message contain the parameter that has been sent and the Web service operation to which or from which the message has been sent. Furthermore, the process instance in which the operation is used is contained and the point in time when the message has been sent. Send input message events respectively receive output message events are specified as a tuple \((\text{inputMessage} | \text{outputMessage}, \text{operation}, \text{occurredIn}, \text{occurredAt})\).

### 4.3.2 Actions

In Web-Flow actions describe steps that are to be taken to handle exceptional events occurring during the execution of a cooperative process instance. The execution of an action is to ensure that the cooperative process instance reliably serves its purpose despite the exceptional event. Web-Flow aims at a semi-automatic execution of the actions to give the process administrator the greatest support possible with as little manual interaction as possible. Actions may affect the complete process in which an event occurred or only the single service call that caused the exception. Web-Flow offers different types of actions supporting these different scopes as illustrated in Fig. 4-6.

Process related actions either continue the process without executing any further actions (action *Continue process* in Fig. 4-6) or abort the whole process (action *Abort process* in Fig. 4-6). Continuing a process is useful if exception handlers have already been defined as part of the cooperative process (if the process definition language offers the necessary constructs). They may be used to handle Web service fault messages or timeouts if a service operation does not answer in time.
Aborting a process instance after an exception occurred may be a useful solution if a failed service call should provide some data without which the continuation of the process instance is not possible. For instance, an ordering process may not be continued without an offer and if all service calls to get offers failed the process instance may be aborted. Aborting a process instance may have some effects on other instances that wait for data produced by this process instance. These effects may be taken into account during event and exception handling which will be described in Chapter 6. Definitions of the process related action types can be found in Appendix B (*ODL Class Definitions for Web-Flow Metamodel*). Note that all action types defined in Web-Flow have a priority attribute. The action priority is used during action selection to determine an order if several possible actions have been found (cf. Chapter 7).

Service call related actions in Web-Flow either repeat the service call, call an alternative service, or suspend the service call for a particular time span. Further action types accomplishing the possible reactions in Web-Flow are *manual reaction* and *procedure call*. With a *procedure call* action advanced exception handling facilities of the process engine executing the cooperative process can be used. A *manual reaction* is used if none of the other actions could be derived. In the sequel we discuss these five action classes in more detail.

The action class *repeat service call* specifies that the same Web service operation is to be called again with the same input parameter values. This action may e.g. be used to handle violations of response time constraints and is defined as follows:
Definition 4-13 - Repeat service call. A repeat service call action \( rsc \) is a tuple \((operation, processInstance, hasPriority)\) where

- \( operation \) denotes the operation that is to be called again.
- \( processInstance \) is the identifier of the process instance for which \( rsc \) is to be executed.
- \( hasPriority \) denotes the priority of \( rsc \).

The definition of a repeat service call action contains the operation that is to be repeated and the process instance in which the original action has been executed, but it does not contain the actual parameter values. As the same parameter values as for the original operation call are used, these are derived from the execution data logged by the Web-Flow MaX component. The process instance ID is not assigned until the action has in fact been derived by the Web-Flow MaX event and exception handling component.

The action class alternative service call is similar to the action repeat service call. It has three different subtypes:

1. It may specify that the same operation is to be called again but with different parameter values that are specified in the action. This may be used to search again for cheaper offers from the same provider after a result constraint defining a maximum price has been violated.

2. It may explicitly specify an alternative Web service operation by naming the service and the operation to be called. So an alternative provider may be named from which an offer is requested if the first one is not satisfying. This action type may also be used to specify that a different access channel (described as a port type) is to be used, for instance, if the high speed access failed.

3. It may use metadata constraints to describe an alternative service that is to be called, e.g. by defining the service category and the desired functionality of the operation (as described above in Section 4.1.2). In this case less stringent restrictions may be set on the alternative service than by directly denoting the service. The Web-Flow MaX component then tries to find an appropriate service which corresponds to the given information.

The alternative service call action is formally defined as follows:

Definition 4-14 - Alternative service call. An alternative service call action \( asc \) is a tuple \((affectedOperation, parameters, operation, operationProperties, processInstance, hasPriority)\) where

- \( affectedOperation \) specifies the original operation which failed and caused the alternative service call.
- \( parameters \) specify the parameter values for the alternative service call.
- \( operation \) specifies the name of an operation that is to be called.
- \( operationProperties \) specify the metadata constraints that describe an alternative service operation.
Exception Handling Concepts

- processInstance is the identifier of the process instance for which asc is to be executed.
- hasPriority denotes the priority of asc.

The parameter values for the alternative service call action may either be specified explicitly in the action or may be derived from the parameter values of the failed service call. For the latter the input parameters of the failed service operation and the alternative operation have to be mapped as it may not be assumed that they are identical. More details on the mapping of the input parameters can be found in Chapter 6 about the implementation of the different actions. The particular subtype of an alternative service call action may be derived from the values that are actually provided.

Sometimes an appropriate solution may be not to repeat a service operation call immediately or to call an alternative service operation but to suspend a failed Web service operation call for a particular time (for example, if it may be assumed that the service call may fail again if it is called immediately). For example, a service call has failed due to a system crash at the service provider. As the provider needs some time to recover from the crash, the repetition of the service call should be delayed for a certain time. If the service operation is then called again it is much more likely that the service call will be executed successfully. To repeat the service call the same service operation is called again with the same input parameter values. The suspend service call action is defined as follows:

Definition 4-15 - Suspend service call. A suspend service call action ssc is a tuple (operation, suspensionInterval, processInstance, hasPriority) where
- operation denotes the operation that is to be called again.
- suspensionInterval denotes how long the service operation call is suspended.
- processInstance is the identifier of the process instance for which ssc is to be executed.
- hasPriority denotes the priority of ssc.

The suspension of a service call suspends the execution of the whole process instance until the suspension interval has elapsed. If an activity in one of several parallel paths is suspended the other paths may be continued until they reach the activity at which the parallel paths join again. Then the execution of the instance is suspended until the path with the suspended activity has also reached the join node.

If the process engine executing the cooperative process instances offers advanced exception handling capabilities such as process adaptation an action of the class procedure call may be defined. It specifies a procedure external to the Web-Flow MaX component that may be called to execute the exception handling. Such procedures may offer dynamic workflow adaptation as described in [GREI2005, MÜLL2004]. Furthermore, some parameters may be defined that are needed to call the external procedure. The action class procedure call is defined as follows:
Definition 4-16 - Procedure call. An procedure call action $pc$ is a tuple $(procedure, processInstance, parameters, hasPriority)$ where
- $procedure$ denotes the external function that is to be called.
- $processInstance$ is the identifier of the process instance that is e.g. to be adapted.
- $parameters$ specify some additional parameters for the procedure call.
- $hasPriority$ denotes the priority of $pc$.

The action class manual reaction is used if no automatic executable action may be derived for an event as the event has not yet occurred for a particular or a similar operation. Furthermore, the administrator may always decide to handle a particular event or exception manually. A manual reaction action is defined as follows:

Definition 4-17 - Manual reaction. A manual reaction action $mr$ is a tuple $(recipient, message, hasPriority)$ where
- $recipient$ contains the contact data of the message recipient.
- $message$ specifies the text of the message to be sent or displayed.
- $hasPriority$ denotes the priority of $mr$.

The manual reaction action class ensures that no event respectively exception is lost but that at least a user denoted by the recipient attribute is notified with a message specified in the action. The notification may, for instance, be an e-mail or an automatic alert sent to a user.

4.3.3 Rules

The rules represented in the Web-Flow metamodel provide information about which actions are to be used to handle particular events. The rules are similar to event condition action (ECA) rules as used, for instance, in active databases [PATO1999]. The information provided in the rules is exploited by the Web-Flow MaX dynamic event and exception handling component to allow an efficient handling of exceptions (cf. Chapter 6).

Rules may contain an operation attribute if the administrator wants to express that a particular action is only suitable if the event occurred during the execution of a particular Web service operation. Another optional part of a rule is a condition which may for instance be an iteration constraint specifying that the action should only be applied if the service operation has been repeated three times. Another possible condition is a context which specifies that the rule should only apply if an operation has been executed in a particular sequence of activities. A context condition may also be defined in addition to an operation condition. Formally a rule is defined as follows:

Definition 4-18 - Rule. A rule $r$ is a tuple $(event, action, condition, operation)$ where
- $event$ denotes an event object of one of the types defined in Section 4.3.1.
**WXS: The Web-Flow quality and eXception Schema**

- *action* denotes an action object of one of the types defined in Section 4.3.2.
- *condition* is an optional attribute which either represents a *quality constraint* as defined in Def. 4-5 to Def. 4-9 or a *context* as described in Def. 4-4. If a condition is specified a rule only applies if the condition is fulfilled.
- *operation* is an optional attribute representing a service operation. If an operation attribute is defined, the rule only applies if the event occurred while a service operation of type *operation* has been executed.

In Web-Flow rules may be generated semi-automatically with support from the Web-Flow MaX logging and analysis component. The semi-automatic generation of rules is described in Chapter 7. As an administrator may define arbitrary rules Web-Flow offers some automatic checks to ensure the consistency of the rule set. For instance, if the administrator specifies different actions for the same event occurring for the same operation he is notified about this possible contradiction. A possible solution to solve this contradiction may be that a condition is added to the rule to distinguish between the different actions.

As the use of rules during event handling may lead to cycles we have to use some mechanisms which ensure that event handling finishes after a finite number of iterations. For instance, if a service call is repeated and fails again the same rule that has triggered the first repetition may apply again. That means the service call is repeated again. If the rule does not contain a condition part restricting the maximum number of iterations this loop continues until either the service responds or a user detects the loop and stops the exception handling. As it is not in the scope of this thesis to develop algorithms for identifying rule dependencies that may cause cycles during exception handling, the solution adopted by Web-Flow is to specify a global maximum number of inter-dependent event handling steps, the so called *Global Iteration Number for event handling* $GIN_{event}$.

Inter-dependent event handling steps are steps that are taken to handle events that occurred while executing an action handling an earlier event. Thus, a failed service call may be repeated at most the number of times specified by $GIN_{event}$. However, it has to be considered that any iteration conditions specified in rules are not overwritten by $GIN_{event}$. That means, that $GIN_{event}$ has to be larger than the biggest iteration number specified in a condition of a rule contained in the Web-Flow MaX metadata repository. To ensure this $GIN_{event}$ has to be checked each time new rules are added to the metadata repository. As new rules are not added to frequently but, for instance, only once per day this additional effort is justifiable.

**4.4 WXS: The Web-Flow quality and eXception Schema**

The algorithms used in Web-Flow to support quality-oriented execution of cooperative processes need a structured, machine-processable representation of the Web-Flow metadata. The representation should also be human-readable to ease the presentation of metadata to the user, e.g. for con-
Web-Flow Metamodel and Quality Constraint Classification

constraint definition or rule generation. As the Web-Flow architecture focusses on Web-service-based cooperative processes XML is a good solution for such a representation of the metadata. Most Web service related standards and standard proposals are based on XML respectively XML schemas defining the types and constructs of the respective language. Furthermore, Web services use XML messages for communication. Thus, we define WXS, the Web-Flow quality and eXception Schema, which is based on an XML schema that contains types for the concepts of the Web-Flow metamodel. Concrete metadata class instances are saved in XML documents containing instances of the according XML types in the Web-Flow metadata repository.

The Web-Flow MaX metadata repository contains WXS documents as well as additional documents such as WSDL files of the used services and the original process definitions, for instance,

![Fig. 4-7: Organization of documents in Web-Flow MaX metadata repository. (The first document in the second row contains log entries.)](image-url)
defined in BPEL (shown in the lower part of Fig. 4-7). There are six types of WXS documents (see the upper part of Fig. 4-7):

- documents with a Web-Flow specific description of the services which also contain the service specific and the operation specific quality constraints,
- documents with a Web-Flow specific description of cooperative processes which also contain the process specific quality constraint,
- documents with global quality constraints which may not be assigned to a service, service operation, or activity,
- documents containing the log entries,
- documents containing event handling rules, and
- documents with context descriptions.

Log entries and event handling rules will be described in more detail in Chapter 7. The service and process documents replicate some information from the additional documents and enhance it, for instance, with semantic service descriptions or service, operation respectively process specific quality constraints.

The complete XML schema of WXS may be found in the appendix (cf. Appendix A (XML Schema for WXS)). As the definition of the WXS schema is straightforward, we only present one example here. Further examples will be given in the subsequent chapters. All WXS definitions are created semi-automatically with support of the Web-Flow MaX editor component respectively the logging and analysis component.

Recall the collaborative fulfillment scenario described in Chapter 1. If we want to provide semantic descriptions for the service of the monitor seller that provides two operations “ATPCheckMonitor” and “orderMonitor” we may use the WXS service description shown in Fig. 4-8. Each operation has an input and an output parameter (line 13 and 14 respectively 23 and 24), the first operation also has a fault parameter (line 15). The semantic information about the service category, the functionality of its operations or the parameter roles is kept in sub-elements of the related type. In Fig. 4-8 a service category has been defined taken from the UNSPSC taxonomy (see lines 4 to 6) and the functionality and role description have been derived from the RosettaNet taxonomy (see lines 9 to 11, line 19 to 21 and line 25 to 27).

4.5 Summary

In this chapter we have introduced the Web-Flow metamodel which contains concepts, classes and types that provide the information necessary for the quality-oriented execution of cooperative processes. The main contribution of the metamodel is the classification of quality constraints that describe quality characteristics that can be applied to heterogeneous services, service operations, or cooperative processes. Quality constraints can be defined for static information about a service (for instance, UDDI metadata) or for dynamic information that is related to service execution. Ex-
Web-Flow Metamodel and Quality Constraint Classification

```xml
<service name="MonitorSellerService" namespace="...">
  <provider organizationName="monitorseller" .../>
  <serviceCategory>
    <category value="43211900"
      taxonomyURI="http://www.unspsc.org"/>
  </serviceCategory>
  <operation opID="1" name="ATPCheckMonitor"
    porttype="MonitorProcurementPT" ...>
    <opFunctionality>
      <functionality value="PIP 3A2"
        taxonomyURI="http://www.rosettanet.org"/>
    </opFunctionality>
    <parameters paramType="InputParameter" message="monitorDetails"/>
    <parameters paramType="OutputParameter" message="availability"/>
    <parameters paramType="FaultParameter" message="ATPFault"/>
  </operation>
  <operation opID="2" operationName="orderMonitor"
    porttype="MonitorProcurementPT" ...>
    <opFunctionality>
      <functionality value="PIP 3A4"
        taxonomyURI="http://www.rosettanet.org"/>
    </opFunctionality>
    <parameters paramType="InputParameter" message="monitorOrder"/>
    <parameters paramType="OutputParameter" message="acknowledgement"/>
    <paramRole>
      <role value="OrderConfirmation"
        taxonomyURI="http://www.rosettanet.org"/>
    </paramRole>
  </operation>
</service>
```

Fig. 4-8: Monitor purchasing service with semantic attributes described in WXS.
Summary

Examples for dynamic quality constraints are response time constraints or input respectively result constraints imposed on input or output documents of services. The metamodel also allows to define different scopes for quality constraints, which describe whether a quality constraint applies to a service, to particular service operations, or only if an operation is called in a particular process. Furthermore, also a context-dependent scope is supported. It allows to specify that a quality constraint only applies if the operation is executed in a particular sequence of operations.

The class definitions for services and cooperative processes are based on existing standards (for instance, [THAT2003, WSDL]) and allow for a generic description of services and cooperative processes. The rule-based exception handling in Web-Flow is accomplished by an approach for log-based derivation of appropriate actions. These algorithms are supported by semantic descriptions of services, operations, and parameters which are also contained in the metamodel. The semantic annotations are similar to those provided by existing semantic metamodels (for instance, [OWLS2003, VERM2004, MEDJ2003A]). These metamodels provide comprehensive ontological frameworks mainly for service composition or execution. However, the Web-Flow mechanisms for quality monitoring and exception handling only use a subset of the information provided by these approaches. Thus, to reduce the overhead of using ontologies semantic attributes referencing taxonomies have been added to the Web-Flow metamodel.

WXS, the Web-Flow quality and eXception Schema, provides an XML schema of the concepts defined in the Web-Flow metamodel. WXS allows for an XML based representation of the classes and provides datatypes that have been defined based on the class definitions of the metamodel.

Some data contained in the metamodel classes is imported from existing cooperative process definitions and service descriptions and is therefore replicated in the Web-Flow specific classes. We accept this replication as we want homogeneous data for the algorithms of the Web-Flow MaX component to work on. As it may not be assumed that service descriptions from different service providers contain homogeneous information it is more suitable to replicate the data locally in proprietary data structures.
5 Event Detection

The first step to quality-oriented process execution is the detection of events that may hinder a successful process execution. As process execution should be influenced as little as possible by event detection efficient mechanisms are needed. Event sources should be monitored and detected without delaying the process execution. Furthermore, mechanisms for an efficient evaluation are necessary to detect possible exceptions before the next task in the process is started. This ensures that exception can be handled in time.

An automation of event detection relieves the user from manually observing the different events and ensures that no event is overlooked. This is especially necessary if many process instances of different process definitions are executed in parallel. Appropriate event sources should be identified that allow for detecting events without user interaction.

As the event sources provide events for several process instances executed in parallel events have to be correlated with the process instances in which they have actually occurred. This ensures that events are evaluated properly as the same event may not lead to exceptions in all process instances. If in fact an exception has occurred, the correlation of the event with the process instance ensures that exception handling is performed for the appropriate process instance.

In Web-Flow the Web-Flow MaX monitoring component is responsible for the automated event detection. In this chapter we identify specific challenges for event detection in Web-service-based cooperative processes and present how the monitoring component addresses these (Section 5.1). We describe how events are detected and correlated with the appropriate process instance (Section 5.2) and discuss how events are evaluated efficiently (Section 5.3).

5.1 Event Detection Overview

In Web-Flow the event types that are described in the Web-Flow metamodel may be detected automatically, in particular the violation of quality constraints, fault messages sent by Web services or by users, and notifications about external events. The Web-Flow MaX monitoring component
is responsible for the automated event detection in Web-service-based cooperative processes monitored by Web-Flow. A particular difficulty that has to be addressed for Web-service-based processes is that the Web services called in cooperative processes may be heterogeneous and the autonomy of the service provider has to be preserved. Thus, event monitoring has to be based on event sources that are available for different types of services and that may be observed without violating the autonomy of the provider. Furthermore, the event sources should be independent of concrete process engines as Web-Flow aims at providing a generic solution for quality support.

In the context of Web-Flow the messages sent between the partner controlling the cooperative process and the Web services are an appropriate event source. Further event sources are interfaces incorporated in the Web-Flow MaX component about which manual notifications and external events can be observed and timers that are used internally to monitor response time or start time constraints. Efficient mechanisms are needed to monitor these even sources.

The violation of quality characteristics imposed on Web service executions in cooperative processes is an important event type monitored in Web-Flow. As efficient event evaluation is an important requirement for event detection appropriate algorithms are needed to quickly detect constraint violations based on the monitored event sources. Additionally, the other event types also should be evaluated efficiently to detect possible exceptions.

Fig. 5-1 gives an overview of the event detection as it is executed in the Web-Flow MaX monitoring component. The algorithms for event detection and evaluation of quality constraints are presented in the following sections.

After an event has been detected in one of the observed event sources its type is determined. If the event is a message requesting the start of a process instance from the process engine the Web-Flow MaX monitoring component initializes the monitoring of the quality constraints for this particular instance (left path of decision node 1 in Fig. 5-1). For this step we assume that process instances are started by sending an observable message to the process engine as it is the case in BPEL. For process instances that are started without an initial message no monitoring of quality constraints is possible. After initializing the quality constraint monitoring the process instance start message is passed to the process engine and the algorithm stops. The initialization step is also performed if the message notifies the monitoring component about a Web service search that is carried out during event and exception handling. For the evaluation of the metadata constraints defined for the Web service search an initialization of the monitoring algorithm is necessary. However, in this special case the start message is not passed on to any other recipient.

In the right path first the event type is checked. If the event is a timeout denoting the violation of a response or start time constraint the next step is the evaluation of the quality constraints to determine which constraint has actually been violated (left path of decision node 2 in Fig. 5-1). Then an WXS event description is created containing the violated quality constraint and sent to the Web-Flow MaX event and exception handling component before the algorithm stops.

For all other events, i.e., sending of an input message, reception of an output or fault message, or
Event Detection Overview

Fig. 5-1: Overview of event detection in Web-Flow. Bold activities mark sub-routine calls.
Event Detection

notification about an external event, first the data necessary for further event processing is extracted (right path of decision node 2 in Fig. 5-1). That means that transport protocol related information is removed. If the event is an external event (left path of decision node 3 in Fig. 5-1) an WXS event description is generated containing the extracted data and sent to the Web-Flow MaX dynamic event and exception handling.

For all other event types first the quality constraints assigned to the respective activity or service operation are evaluated (right path of decision node 3 in Fig. 5-1). If any of these constraints is violated (left path of decision node 4 in Fig. 5-1) a WXS event description is generated containing the violated constraint. Then the event is reported to the dynamic event and exception handling. If no constraint is violated (right path of decision node 4 in Fig. 5-1) the message is passed on to the process engine (for output messages) respectively to the Web service (for input messages).

5.2 Detection and Evaluation of Events

The first step of the automated event detection is the detection and evaluation of so-called basic events. Basic events are observable by the Web-Flow MaX monitoring component and do not directly correspond to the event types described in the Web-Flow metamodel. The automatically observable events comprise the following types:

(i) sending of an input message to a Web service,
(ii) reception of an output message from a Web service,
(iii) reception of a fault message from a Web service,
(iv) reception of a manual fault message,
(v) reception of a search result from the Web-Flow MaX Web service search component,
(vi) reception of a notification about an external event,
(vii) timeout.

To detect the basic events the Web-Flow MaX monitoring component observes different event sources that provide the necessary information. In particular, these are the following sources:

- the messages sent between the process engine and the Web services. This source provides the basic events of type (i), (ii), and (iii).
- a user interface over which manual fault messages are forwarded to the monitoring component. This source provides basic events of type (iv).
- an interface to the Web service search component. This source provides basic events of type (v).
- an interface over which the Web-Flow MaX monitoring component is notified about external events providing basic events of type (vi).
- Timers which are used to detect violations of response time or start time constraints providing basic events of type (vii).
In the following, we discuss in more detail how the Web-Flow MaX component monitors the different event sources.

### 5.2.1 Web Service Messages

As described in Chapter 1 Web services typically are called with SOAP messages [SOAP2003] that are transported using a network protocol such as HTTP (Hyper Text Transfer Protocol [W3C2000]). Thus, the messages sent between the process engine and the different Web services may be intercepted by an appropriate tool, for instance a network monitor, which watches the outgoing and incoming messages of the process engine. This additional component is needed as the Web-Flow MaX component is only loosely linked with the process engine executing the cooperative processes. Thus, the Web-Flow MaX has no direct access to the internal message processing of the engine.

The intercepted input, output, and fault messages are forwarded to the Web-Flow MaX compo-

---

**Fig. 5-2:** Interception of messages.
In this figure the messages are sent between the process engine and the Web services. The other direction is similar with redirected arcs.
Event Detection

... which processes each message. Then either the original message or a newly constructed message is forwarded to the original recipient (the process engine respectively a Web service). Fig. 5-2 shows an example of a network monitor catching a message \( m \) sent from the process engine to a Web service. After processing the message, the original message \( m \) or a new message \( m' \) are forwarded to the Web service, depending on the outcome of the message processing respectively the event handling.

5.2.1.1 Identification of Process Instances

To ensure that the Web-Flow MaX component evaluates the right quality constraints for a particular message the intercepted messages have to be correlated with the process instances currently executed. Therefore, a unique correlation ID may be used which should be contained in every SOAP message sent during the execution of the respective process instance.

Fig. 5-3 illustrates the correlation problem with an example. The Web-Flow MaX component observes a message sent from the process engine to the Web service „Service1“. As the process engine currently executes two process instances it is not clear which instance actually has called the service operation, especially as no engine internal information may be used. Therefore, the message contains a correlation ID that specifies the name of the process instance („Process1“). Using the correlation ID the Web-Flow MaX monitoring component is able to determine which quality

![Diagram](image_url)

Fig. 5-3: Correlation of messages with process instances.
constraints have to be evaluated based on this particular message.

We cannot assume that such a correlation ID is automatically generated and published by the process engine. BPEL for instance uses correlation IDs for asynchronous service calls whereas the messages sent for synchronous service calls are correlated internally.

The WS-Coordination specification [CABR2003] also contains a „coordination context“ which provides a data structure that uniquely marks messages belonging to the same conversation, i.e., the same process. The data structure is included in the header of all SOAP messages sent during the conversation. Generally, we cannot assume that the process engine and the used Web services support WS-Coordination.

As Web-Flow is to provide a generic infrastructure for quality monitoring working with different process engines, we need a mechanism that ensures that a correlation ID is contained in every Web service message. The correlation ID may be a process instance ID or application data that is unique within all process instances executed in a particular process engine. A process instance ID is generated by the Web service that starts a process instance and sent in the process start message. The ID is stored by the Web-Flow MaX component to identify further messages sent for this process instance. The process instance ID is contained in all further input, output, or fault messages sent during the execution of this process instance. The ID must not be manipulated by the Web services, i.e., a Web service operation simply copies the process instance ID from the input to the output or fault message.

We cannot assume that all Web services support the use of process instance IDs in the way described above. Especially if the simple cooperation model is used (cf. Section 1.1.3) Web service providers typically do not know that their services are used in cooperative processes. In this case the correlation of messages can be performed using application data. The data must uniquely identify a process instance within all process instances executed in the process engine monitored by the Web-Flow MaX component. Furthermore, it must be contained in all messages sent during the execution of the cooperative process. Examples for application data that fulfills these requirements are an order ID, an invoice ID, or a patient ID in medical cooperative scenarios.

5.2.1.2 Identification of Activities

Besides the use of process instance IDs to identify the right process instance for a particular message each message also has to be correlated to a particular activity of the process instance to derive which quality constraints have to be evaluated. Therefore, execution data logged in the Web-Flow MaX repository is used. The data specifies which activities are currently executed. With this information the Web-Flow MaX monitoring component may derive the activity to which a particular message belongs.

In this context we have to consider one critical situation in which the Web service output messages may not be related uniquely to one particular activity. This may occur if the same Web service
Event Detection

operation is called in two (or more) parallel paths of a process instance at approximately the same
time so that their execution intervals overlap. In the example scenario in Fig. 5-4 operation 1 of
service 1 is executed as first activity in both parallel paths (activity „s1.op1“) at time point t and
t’ which only differ slightly. For example, the activity „s1.op1“ may get offers for shipments with
different carriers defined in the input parameter. As the Web service output messages only contain
the Web service operation name and the process instance ID it is not clear to which particular ac-
tivity a response message from operation 1 of service 1 belongs. We also cannot assume that the
response for the first call will be returned first (if there is a short distance between the first and the
second operation call), as the messages may take different paths in the network. This may lead to
a shift in their order.

To solve this problem the Web-Flow MaX monitoring component simply evaluates the quality
constraints for all activities that may be the recipient of the message, i.e. for both activities in Fig.
5-4. For global or operation specific quality constraints this is always possible as they are valid
for all activities that call the same Web service operation. If process specific constraints exist for
one of the activities an evaluation of the wrong output message may lead to actions that have not
been intended by the administrator. However as the Web-Flow MaX component always demands
a user confirmation before executing event handling actions the execution of undesired actions
may be prohibited by the administrator.

5.2.2 Interfaces for Manual Fault Messages, External Events, and Web Service Search Results

Another important event source for the Web-Flow MaX component are the interfaces over which
this component is notified about manual fault messages or external events that are not directly re-
lated to Web service messages but may also lead to exceptional situations. Furthermore, results from the Web service search component are also an event source for the Web-Flow MaX constraint monitoring component as metadata constraints may be defined on the result messages. In the following, we present these interfaces in more detail.

5.2.2.1 Interface for Manual Fault Messages and External Events

Manual fault messages and external events may be handled similarly by the Web-Flow MaX component, thus we discuss them together in this section. However, the relation of these events to a particular Web service call in a process instance is different. While manual fault messages are related to a particular activity external events notify the Web-Flow MaX component about events that are not directly related to an activity.

Manual fault messages may be received by phone or e-mail, and are forwarded to the Web-Flow MaX component in an appropriate format that allows for automatic processing. Manual fault messages are related to the execution of a particular service operation in a particular process instance. However, they could not be sent by the Web service itself, as for instance the whole network of the service provider broke down. In particular, manual fault messages may occur during the execution of complex services comprising user interaction. As such services usually run for a longer time feedback given by the service provider during the service execution may contain valuable information for the partner. For example, it may inform about delays due to service execution failures.

The Web-Flow MaX component provides a graphical user interface over which the monitoring component is notified about the occurrence of fault messages. Automatic processing of messages is only possible if the structure and semantics of the notification messages are known to Web-Flow MaX. Thus, the interface offers forms providing input fields for entering fault details (for instance, the process instance ID). The interface translates the user input into a WXS representation of the manual fault message which is then further processed. Quality constraints are evaluated based on the information provided by the fault message. Alternatively, the message may be considered as event itself and be handled by the dynamic event and exception handling component of Web-Flow MaX.

In particular, for manual fault messages the following WXS types are supported by the respective user interface:

- a message indicating failure of service execution,
- a notification that the service execution will be delayed for a certain time span,
- a message that a particular service provider is out of business for some time,
- a message with free text that can be used for all events that do not match one of the predefined message types above. These events cannot be handled automatically, but at least a user can be notified.
Event Detection

If the interface shall support further messages the appropriate type definitions have to be added to the WXS schema and the interface is extended with the appropriate forms and the evaluation algorithms for the new messages.

An example of a WXS manual fault message is shown in Fig. 5-5. The message notifies the Web-Flow MaX component that a service provider is out of business until a particular date and cannot execute any services. If there are pending service calls for this provider with corresponding response time constraints these constraints may be violated. The resulting exceptions then have to be handled appropriately.

Examples for external events are the following notifications:
- The budget of a department has been reduced by 30%. Consequently, quality constraints assigned to activities in running process instances may have to be changed to use lower price limits.
- If a service provider is suddenly not able to execute its services anymore (for instance, as he has gone bankrupt or a natural disaster has occurred) all running process instances have to be checked if they use services of this partner and alternative service providers have to be searched to ensure that the process instances may be continued.
- Data that is used in input or result constraints has changed (for instance, company wide regulations for maximum prices) and thus the currently monitored quality constraints have to be updated.

External events may be received by the Web-Flow MaX component over several communication channels, for instance, manually. Therefore, the Web-Flow MaX component also uses the graphical interface and appropriate WXS message definitions to be notified about external events. These manually reported external events are directly sent to the Web-Flow MaX dynamic event and exception handling component for processing.

Furthermore, the Web-Flow MaX component offers an interface over which external messages
Monitoring of Quality Constraints

may be received from other programs. For instance, databases containing data used in quality constraints may provide triggers which inform the Web-Flow MaX component if this data is updated. The interface for the automatic notification about external events expects messages that are already in the WXS format. These external events are also sent to the Web-Flow MaX dynamic event and exception handling component for further processing.

5.2.2.2 Web Service Search

The messages received from the Web service search component of the Web-Flow MaX component contain the results of searching for a service to execute a particular task. As the desired service may be described using metadata constraints the search results have to be evaluated by the constraint monitoring component to check if all constraints are met or if some of them have been violated and if perhaps a new search has to be performed. Therefore, the Web-Flow MaX monitoring component offers an internal interface to which the Web service search component sends its results for evaluation.

5.2.3 Timers

A further event source for the Web-Flow MaX monitoring components are timers. These are used to observe response time or start time constraints for Web service calls. If a Web service does not answer or is not started in time a timeout occurs leading to the evaluation of the quality constraints assigned to the affected activity.

A timer for the monitoring of a response time constraint for a particular activity is started when the process engines invokes the service operation by sending the related message. A timer is stopped when an output or fault message is received from the Web service operation in time. Manual fault messages received over the user interface may as well lead to the stop of a timer, for instance, if the Web-Flow MaX component is notified manually about a failed service execution.

Timers for the monitoring of start time constraints are a bit different, as they are started as soon as the actual start time is assigned to the activity. This is done either at process instance start time or when the XPath query denoting the start date is evaluated. They are stopped when the process engine calls the related activity by sending an input message.

5.3 Monitoring of Quality Constraints

As described in Chapter 4 quality constraints are defined as logical comparisons and constraint expressions as Boolean expressions. To evaluate the constraints the logical comparisons and the Boolean expressions have to be evaluated automatically. Boolean expressions are, for instance, also part of SQL queries. Thus, query processors used in relational database systems already pro-
vide the necessary evaluation functionality. As query processors are highly optimized they allow for an efficient evaluation of Boolean expressions.

As efficient automated evaluation of quality constraints is an important requirement for event detection in Web-Flow the Web-Flow MaX monitoring component uses a query processor for the evaluation of constraint expressions. The translation of the XML-based descriptions of the quality characteristics into tuples of relational database tables has only to be done once when the process instance is started. Thus, it does not influence the overall performance of constraint violation. The extraction of data from the XML documents representing the events (for instance, messages sent between the process engine and the Web services) has to be performed anyway. Furthermore, if a query processor is used this extraction is not necessary for constraints which cannot directly be related to XML messages, for instance, response time or iteration constraints.

The monitoring of quality constraints for a particular process instance comprises two phases. When a process instance is started the constraint monitoring is initialized. During process execution the quality constraints are then evaluated based on observed events. Any detected violations are reported to the Web-Flow MaX dynamic event and exception handling. In the following, we describe the different phases in more detail.

5.3.1 Initialization

The relational database for the constraint monitoring contains one table for each constraint type which stores the desired and the actual values of a particular quality constraint. Furthermore, the database contains a table with the so called constraint evaluation queries which are sent to the query processor to evaluate the quality constraints respectively the constraint expressions.

The initialization phase has to ensure that all quality constraints that are valid for the service calls in the new process instance are really monitored during process execution. In particular, it is necessary to determine which quality constraints actually apply for the new process instance. As quality constraints with different scopes may have been defined, we have to select the constraints to be used automatically. For example, if context dependent constraints exist for a particular service operation, it has to be checked whether the context is fulfilled in this process instance. Furthermore, the constraint evaluation queries have to be constructed.

The main algorithm for the initialization of the constraint monitoring for a new process instance is shown in Fig. 5-6. In the sequel we describe this algorithm and then discuss its more complex sub-routines in more detail. After a process instance start message was received by the Web-Flow MaX monitoring component, the WXS cooperative process definition is retrieved from the metadata repository. It contains the activities of the process and the process specific quality constraints and constraint expressions as described in Section 4.4. Furthermore, all global quality constraints are retrieved as those are kept in separate WXS files in the metadata repository.
Then the process definition is traversed and for each activity the Web service operation executing the activity is determined and the WXS service definition is retrieved from the metadata repository. This description contains the operation specific quality constraints for a particular operation execution (cf. Section 4.4) which are retrieved in the next step together with the context dependent constraints. It is checked which context dependent constraints apply for an operation. Therefore, those context descriptions are retrieved from the Web-Flow MaX metadata repository that have been defined for the service operation called by the currently considered activity. Thus, the context description document shown in Fig. 5-7 would be retrieved if the current activity calls operation 2 of service 2 (denoted in the \texttt{<forOperation>} element in line 2). Then it is checked

- whether the directly preceding activity in the cooperative process is executed by operation 1 of service 1 (denoted in the \texttt{<predecessors>} element in line 3) and
- whether the two succeeding activities are executed by operation 1 of service 3 and operation 2 of service 4 (denoted in the \texttt{<successors>} element in line 7).

If all these checks are positive (i.e., the currently considered activity is executed in the context shown in Fig. 5-8), the quality constraint specified in the context document (line 12 of Fig. 5-7) is added to the set of quality constraints applying for this operation.

As it is possible that for a certain activity only constraints of particular scopes are to be used, in the next step the Web-Flow MaX monitoring component checks which of the quality constraints have to be used and therefore are to be monitored. After the constraints to be monitored have been determined the tables of the relational database (also called monitoring tables in the following) are initialized. In the last step the constraint evaluation queries $Q_{ce}$ are generated and saved in the relational database.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{algorithm.png}
\caption{Algorithm for initialization of constraint monitoring.}
\end{figure}
Event Detection

```
1 <context contextID="1" contextName="sampleContext">
2  <forOperation operationName="op2"
    porttype="opPT2"
    operationTargetNamespace="opTNS2"
    serviceName="s2"/>
3  <predecessors>
4   <precOperation operationName="op1"
      porttype="opPT1"
      operationTargetNamespace="opTNS1"
      serviceName="s1"/>
5  </predecessors>
6  <predecessorRange>1</predecessorRange>
7  <successors>
8   <succOperation operationName="op1"
      porttype="opPT1"
      operationTargetNamespace="opTNS1"
      serviceName="s3"/>
9   <succOperation operationName="op2"
      porttype="opPT2"
      operationTargetNamespace="opTNS2"
      serviceName="s4"/>
10 </successors>
11 <successorRange>2</successorRange>
12 <qualityConstraint>
13 ...  
14 </qualityConstraint>
15 </context>
```

**Fig. 5-7:** Example for a context description in WXS.
(for context dependent quality constraints)

![Diagram](#)

**Fig. 5-8:** Example for the context described in Fig. 5-7.
The second task is the currently considered task, the light grey one is the predecessor and the dark grey ones are the successors.
During the initialization phase the Web-Flow MaX monitoring component creates a full copy of the quality constraints valid for a particular process instance and its services at that point in time the process instance is started. Thus, we avoid the problem to propagate modified quality constraints to the monitoring component during process instance execution. That means the administrator may change quality constraints for process definitions or services while instances of these processes or services are currently executed. These changes will apply to the next process instance started after the changes have been saved. To avoid conflicts during the initialization phase no changes can be saved to the Web-Flow MaX metadata repository as long as the Web-Flow MaX monitoring component is reading data for the initialization from the metadata repository.

As cooperative processes are often long running this approach may be too restrictive. It would be desirable to propagate changes of quality constraints to currently executed process instances. This would allow to reflect, for example, changed business policies in all running process instances as soon as possible. However, the solution of this problem is out of scope of this thesis.

5.3.1.1 Selection of Constraints

As Web-Flow provides different scopes for quality constraints there may be quality constraints with different scopes for one activity, for instance, an operation specific one and a process specific one. However, this may lead to ambiguous situations as two constraints of the same type but with different scopes may be defined. In particular, two types of ambiguity can be identified:

- **Contradiction**: The first one are contradictory comparisons for one parameter value (for instance, \(price > x\) and \(price < x\)) which never could be fulfilled at the same time.
- **Subsumption**: The second type especially occurs for execution constraints (i.e., response time, start time, and iteration constraints). Two constraints with the same scope, the same comparison operator, and different comparison values may not be contradictory but subsume each other. That means that one constraint will never apply as it is always covered by the second constraint.

To avoid such contradictions the Web-Flow constraint editor checks automatically during constraint definition if several quality constraints may possibly lead to contradictions. However, it is often not possible to check the semantics of the quality constraint automatically (especially for metadata, input, or result constraints). It may also be difficult to determine automatically whether two comparisons are contradictory if arbitrary complex constraint expressions may be used. Thus, the Web-Flow editor only performs basic checks using a set of heuristic rules. For example the following rules may be considered:

- If two execution constraints of the same type but with different scopes are defined for one activity respectively the service operation called by the activity they may possibly be contradictory.
- If two execution constraints of the same type and scope but with conflicting comparison operators are defined for one activity they may also be contradictory. Conflicting comparison
operators are for instance „>“ and „<“. As Web-Flow only allows a restricted number of comparison operators (cf. Section 4.2.2) this reduces the number of rules that have to be defined to detect contradictions.

- If two result (input/metadata) constraints with different scopes are defined for one activity that reference the same part of an XML document they may possibly be contradictory.

When the Web-Flow editor detects possible contradictions it notifies the administrator and requests to resolve the contradiction by specifying which quality constraints of which scopes are to be used for a particular activity. Therefore, the WXS type definition for activities provides three dedicated attributes as shown in lines 4, 5, and 6 of Fig. 5-9. The attributes contain Boolean values stating whether quality constraints of a particular type are to be used or not. The Web-Flow editor also checks whether subsuming constraints are defined and notifies the administrator to delete the subsumed constraint. This ensures that no constraints are saved that will never apply.

During the initialization phase of the constraint monitoring the information in the activity definition is used if quality constraints of more than one scope have been retrieved for a particular activity. The Web-Flow MaX monitoring component uses only those constraints and constraint expressions for which the attribute is set to „true“ and ignores the other constraints. This is possible as constraint expressions may only contain quality constraints of the same scope (as defined in Section 4.2.4). Thus, the ignorance of some constraints may not lead to ambiguous constraint expressions.

---

Fig. 5-9: Attributes used in WXS to determine valid quality constraints.
Monitoring of Quality Constraints

5.3.1.2 Initialization of Monitoring Tables

The next step of the initialization algorithm initializes the monitoring tables. Therefore, the desired values specified in the quality constraint are extracted and saved in the monitoring table for this constraint type. The attributes for the actual values are initialized with null as these values will not be available until process execution. To ensure that the right constraints are evaluated for a particular activity in a particular process instance dedicated attributes are used for every tuple. In particular, these are the process instance ID, the activity ID, and the number of the constraint in a constraint expression. A further special attribute relates each tuple to the constraint evaluation query that is used to automatically evaluate the quality constraint respectively constraint expression.

Execution constraints directly specify the desired values in the constraint. Thus, the values can be copied directly to the monitoring table. For instance, for a response time constraint a tuple with the following attributes is inserted in the monitoring table:

\[(exprKey, processInstanceID, activityID, queryID, responseDuration, responseDate, actualExecutedDate)\]

For the response time constraint shown in Fig. 5-10 the following values are added (the first four attributes relating the constraint to a particular activity have been omitted):

\[(..., 00:05:00, null, null)\]

The monitoring tables for start time and iteration constraints have similar structures and thus will not be discussed here (for details see [BERR2004]).

As metadata, input, and result constraints evaluate constraints on documents not available until process execution, the constraints specify an XPath expression how the data element on which the constraint is defined may be retrieved from the documents during process execution. Furthermore, the constraints either specify a fixed comparison value or again an expression how to retrieve the values from the document. To make this information available for automated constraint evaluation
Event Detection

The XPath queries also have to be stored in the monitoring tables. For example, for a result constraint a tuple with the following attributes is inserted in the monitoring table:

$(exprKey, processInstanceID, activityID, queryID, xpathQuery_leftOp, parameter_leftOp, value_rightOp, xpathQuery_rightOp, parameter_rightOp, actualValue_leftOp, actualValue_rightOp)$

For the sample result constraint in Fig. 5-11 the following attribute values are added (again the first four attributes are omitted):

$(..., /ATPResponse/monitorDetails/price, ATPResponseMonitor, null, /OrderDetails/Monitor/maximumPrice, OrderDetailsMonitor, null, null)$

The tables used for monitoring of input and metadata constraints have the same attributes and will not be discussed here.

5.3.1.3 Construction of Constraint Evaluation Queries

In the last step of the initialization algorithm for each activity the constraint evaluation queries $Q_{ce}$ are constructed which allow for the automated evaluation of the quality constraints and the constraint expressions using the query engine of the relational database. If more than one constraint
expression is valid for an activity, for instance, as the administrator has specified that the global as well as the process specific constraints are valid, separate queries are used for each scope. A constraint evaluation query is a standard SQL query whose WHERE part contains the constraint expression valid for a particular activity and a particular scope. The basic format of constraint evaluation queries is the following:

\[
SELECT \ 'true' \\
FROM \ monitoring-table \ t_1, \ monitoring \ table \ t_2, ...
WHERE \ constraint-expression
\]

The SELECT part of the constraint evaluation query is equal for each query. It defines that the query returns ‘true’ if the constraint expression is fulfilled, i.e., tuples matching the condition exist, and an empty result set if the constraint expression is violated, i.e., no matching tuples exist. The FROM clause contains all tables that are referenced in the constraint expression and assigns correlation names that are used in the WHERE clause (for instance, \(t_1, t_2\) in query (i) above). In the following, we will explain the construction of constraint evaluation queries by means of examples.

Let us first assume that we have the following constraint expression consisting of only one quality constraint, namely the response time constraint \(rtc\) shown in Fig. 5-10:

\[
\text{constraint-expression: rtc}
\]

The FROM clause of the evaluation query \(Q_{ce1}\) references only the monitoring table for response time constraints (named RESPONSE_TIME_CONSTRAINT). The WHERE clause contains the constraint expression shown above. Additionally it uses the attributes exprKey, processInstanceID, activityID, and queryID to reference the right tuple. The complete query \(Q_{ce1}\) is the following:

\[
SELECT \ 'true' \\
FROM \ \text{RESPONSE\_TIME\_CONSTRAINT} \ R \\
WHERE \ (R.exprKey='...' \ \text{AND} \\
R.processInstanceID='...' \\
R.activityID='...' \\
\text{AND} \ R.queryID='...' \\
\text{AND} \ R.actualExecutedDate} \ \leq \ R.responseDate)
\]

The operator to compare the desired with the actual values is derived from the WXS representation of the single quality constraints (cf. Fig. 5-10). To uniquely identify a query within all valid queries for a particular activity in a process instance, the value for the queryID attribute is determined from the number of valid constraint expressions for this activity.
As a second example consider the following constraint expression consisting of a response time constraint \( rtc \) and a result constraint \( rc \), each denoting the constraints shown in Fig. 5-10 and Fig. 5-11:

\[
\text{constraint-expression: } rtc \text{ AND } rc
\]

For this constraint expression the FROM clause of the query \( Q_{ce2} \) has to reference two monitoring tables, that for response time constraints and that for result constraints (named RESULT_CONSTRAINT). To avoid that a possibly large cartesian product is calculated between the two tables we join the tables in the WHERE clause using the attributes \( \text{processInstanceID}, \text{activityID}, \) and \( \text{queryID} \). Inside one constraint evaluation query \( Q_{ce} \), these attributes have the same values, as each query \( Q_{ce} \) evaluates the quality constraints for one particular activity in one particular process instance. The complete query \( Q_{ce2} \) is the following:

\[
\begin{align*}
\text{SELECT 'true' (iii)} \\
\text{FROM RESPONSE\_TIME\_CONSTRAINT R, RESULT\_CONSTRAINT P} \\
\text{WHERE (R.processInstanceID = P.processInstanceID} \\
\text{AND R.activityID = P.activityID} \\
\text{AND R.queryID = P.queryID) AND} \\
\text{((R.exprKey='1' AND} \\
\text{R.processInstanceID='...'} \\
\text{AND R.activityID='...'} \\
\text{AND R.queryID='...'} \\
\text{AND R.actualExecutedDate \leq R.responseDate) \\
\text{AND (P.exprKey='2' AND} \\
\text{P.actualValue\_leftOp \leq P.actualValue\_rightOp))}
\end{align*}
\]

The first three lines of the WHERE clause contain the join condition, the further lines contain the constraint expression. The values of \( \text{processInstanceID}, \text{activityID}, \) and \( \text{queryID} \) only have to be checked for one of the tuples, for the other ones they are automatically the same because of the join condition. The use of the \( \text{queryID} \) attribute in the join condition ensures that the tuples for one particular constraint expression for one scope are evaluated together.

For the evaluation of metadata constraints on results received from the Web-Flow MaX Web service search (cf. Section 5.2.2.2) also the strictness of a particular constraint may be considered. As described in Section 4.2.3 the strictness can either be \textit{high} or \textit{low}. It indicates whether the violation of a particular constraint makes the whole service useless for the task to be executed or whether it may still be used. If the strictness of a metadata constraint \( mc \) is high the WHERE clause of the constraint evaluation query \( Q_{ce} \) may be constructed as described above. However, if the strictness
of mc is low the WHERE clause becomes more complex as the constraint expression may be true if mc is fulfilled as well as if mc is violated. Consider the following constraint expression:

\[ \text{constraint-expression} = mc_1 \text{ AND } mc_2 \]  

(iv)

with \(mc_1\) being a metadata constraint with high strictness and \(mc_2\) being a metadata constraint with low strictness each using the comparison operator ‘\(\leq\)’. The constraint evaluation query for this constraint expression is the following:

\[
\begin{align*}
\text{SELECT} & \quad \text{’true’} \\
\text{FROM} & \quad \text{METADATA\_CONSTRAINT } M_1, \text{ METADATA\_CONSTRAINT } M_2 \\
\text{WHERE} & \quad (M_1.\text{processInstanceID} = M_2.\text{processInstanceID} \\
& \quad \text{AND } M_1.\text{activityID} = M_2.\text{activityID} \\
& \quad \text{AND } M_1.\text{queryID} = M_2.\text{queryID}) \text{ AND} \\
& \quad ((M_1.\text{exprKey} = ‘...’ \text{ AND} \\
& \quad \text{M_1.\text{processInstanceID} = ‘...’} \text{ AND} \\
& \quad \text{M_1.\text{activityID} = ‘...’}) \text{ AND} \\
& \quad \text{M_1.\text{queryID} = ‘...’}) \\
& \quad \text{AND } M_1.\text{actualValue\_leftOp} \leq M_1.\text{actualValue\_rightOp} \text{ AND} \\
& \quad ((M_2.\text{actualValue\_leftOp} \leq M_2.\text{actualValue\_rightOp}) \text{ OR} \\
& \quad \text{NOT (M_2.\text{actualValue\_leftOp} \leq M_2.\text{actualValue\_rightOp}))})
\end{align*}
\]

As the last two lines will always evaluate to ‘true’ we may just as well skip them. However, as we want that the constraint evaluation queries reflect the whole constraint expression we save the whole query \(Q_{ce}\) and leave the optimization to the query optimizer of the relational database system. As the metadata constraint expression (iv) contains two quality constraints of the same type, the FROM clause of the Query (v) specifies two different correlation names for the same table. This allows for distinguishing the tuples that are part of the constraint expression.

For response time constraints the strictness attribute may be considered the same way. However, if it is set to „high“ it may be omitted and the standard constraint evaluation query as for example shown in Query (iii) may be used.

5.3.2 Constraint Evaluation

Constraint evaluation is performed after an event has been detected and the relevant data has been extracted from the message. Regarding the different constraint types only some kinds of events actually lead to an evaluation of a particular quality constraint for an activity. The following dependencies exist:
Event Detection

<table>
<thead>
<tr>
<th>Quality Constraint</th>
<th>Associated Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata constraint</td>
<td>reception of the output message from the Web service search</td>
</tr>
<tr>
<td>Response time constraint</td>
<td>timeout or reception of output message or reception of fault message (manual as well as Web service faults)</td>
</tr>
<tr>
<td>Start time constraint</td>
<td>timeout or sending input message</td>
</tr>
<tr>
<td>Iteration constraint</td>
<td>sending input message</td>
</tr>
<tr>
<td>Input constraint</td>
<td>sending input message</td>
</tr>
<tr>
<td>Result constraint</td>
<td>reception of output message</td>
</tr>
</tbody>
</table>

Table 5-1: Dependencies between evaluation of quality constraints and events.

- **Metadata constraints** are evaluated when a result message from the Web service search component is received. As these messages are forwarded directly to the Web-Flow MaX they may be clearly identified.

- **Response time constraints** are evaluated either when a timeout occurs or when an output message or a fault message (manual or Web service faults) are received from a Web service operation for which a timer is running. The latter events denote that the operation execution has finished and typically it may be assumed that the response has arrived in time. However, as in a constraint expression the response time may be negated it is evaluated for both cases.

- **Start time constraints** are evaluated either when a timeout occurs or when the activity is started. Again, the evaluation is performed in both cases as the constraint may be negated in a constraint expression. The start of an activity is signalled by a message sent by the process engine calling a Web service operation.

- **Iteration constraints** are mainly used for event handling to determine, for instance, whether an activity may be repeated. Thus, they are typically evaluated when an input message is sent to a Web service operation.

- **Input constraints** are evaluated when the process engine sends a message calling a Web service operation to execute a particular activity.

- **Result constraints** are evaluated when the output message is received from an activity.

Tab. 5-1 summarizes the correlations of events and quality constraints. As we can see in the table one event, for instance, the reception of the output message of a Web service operation, may lead to the evaluation of several quality constraints. This is reasonable as a constraint expression may contain different quality constraints.

Two further events which do not lead to the evaluation of quality constraints but to an update of the monitoring tables are not listed in Tab. 5-1. First, if a response time constraint specifies a du-
ration value, the expected response date has to be added in the tuple as soon as the activity is started. The value is calculated by adding the response duration to the start date. Second, start time constraints often may not be assigned until the process instance is running. So, the monitoring table has to be updated when the output message specifying the desired start date is received. The update of the monitoring tables is realized using standard SQL update statements.

In the following we describe in more detail how the different quality constraints are evaluated after an event was detected.

5.3.2.1 Execution Constraints

For the evaluation of a response time constraint the related tuple in the monitoring table possibly has to be updated twice. If the response time constraint does not specify a response date, the first update sets the response date when the activity is actually started to allow for the comparison as specified in the constraint evaluation query, for example Query (ii). The expected response date is calculated by adding the response duration interval to the start time of the activity. The second update is executed either when the output or fault message has been received or when a timeout occurs. In the first case the point in time when the message was received is saved in the tuple. In the second case a timestamp denoting a timeout is saved. If a timer is running and an output or fault message is received, the timer is stopped to avoid unwanted timeouts.

For instance, for the response time constraint in Fig. 5-10 the following tuple may result after both updates have been performed:

\[(..., 00:05:00, 2005-01-23T17:06:00, 2005-01-23T17:04:00)\]

The last attribute specifies the point in time when the response actually has been received, the last but one attribute specifies the latest point in time at which the response should be received.

For the evaluation of a start time constraint the monitoring tuple is updated either with the actual timestamp when the activity is started or a timestamp denoting the timeout of the related timer.

To evaluate iteration constraints the current iteration number for a particular activity is retrieved from the log data written by the Web-Flow MaX when an output or fault message is received or a timeout occurs. Then the related tuple of the monitoring table is updated with the current iteration number. If there is more than one iteration constraint for an activity, each tuple is updated with the same iteration count.

The tuples to be updated may be determined by using the processInstanceId and the activityId. Technically, there may be several response time or start time constraints which are valid for one activity, even though this does not make much sense from a semantic point of view as one constraint typically subsumes the others. Furthermore, contradictory and subsuming constraints should be avoided by the preprocessing steps described in Section 5.3.1.1. If there are several response time or start time constraints the same timestamp is entered for every tuple as there may only be one actual response respectively start date for one activity.
Event Detection

The queryID attribute of the updated tuples specifies which constraint evaluation queries are assigned to the quality constraint. However, the queries are not executed until all update operations for all tuples in the different monitoring tables have been executed as for instance, an output message may lead to the evaluation of a response time constraint as well as of a result constraint.

5.3.2.2 Input, Result, and Metadata Constraints

Input, result, and metadata constraints compare an element of an XML document either to a fixed value or to an element of another XML document. The documents may be Web service input or output messages or search results from the Web-Flow MaX Web service search. For the update of the affected tuples in the monitoring tables first the actual values of the left and possible right operand are extracted from the XML documents. Therefore, the XPath query is evaluated on the related message document and the tuple is updated with the returned value. For instance, for the result constraint in Fig. 5-11 the following tuple may result from updating the attribute values (the last line indicates the values retrieved from the documents):

\[(..., /ATPResponse/MonitorDetails/price, ATPResponseMonitor, null, /OrderDetails/Monitor/maximumPrice, OrderDetailsMonitor, 120.00 \text{ EUR}, 150.00 \text{ EUR})\]

Several input, result, or metadata constraints may be assigned to the same activity. For each constraint one tuple has been inserted in the monitoring table during the initialization phase, containing the document name and the XPath query. This ensures that the correct actual value is inserted in each tuple as the XPath query specifies how to retrieve the value. If the XPath query is executed on the wrong document no result would be returned and the actual value is not updated. The left and the right operand may be defined on different messages, which may be received by the Web-Flow MaX monitoring component at different point in times. Thus, messages possibly have to be saved until the second message necessary to evaluate a particular constraint is also available. For instance, the result constraint in Fig. 5-11 compares the price contained in an output document to a maximum price contained in an output message of a preceding activity of the process (see Fig. 5-12 for an illustration). Thus, the output document has to be saved until the output document of the activity to which the result constraint is assigned will be available. The Web-Flow MaX monitoring component analyzes such dependencies between messages and quality constraints during the initialization of the constraint monitoring and stores the necessary messages.

The tuples which have to be updated and the constraint evaluation queries are determined in the same way as for execution constraints. Again, evaluation queries are not executed as long as all update operations (for instance, triggered by an input or output message) have been completed.

As a final remark to the evaluation of quality constraints we state that an automatic evaluation of
input, result, and metadata constraints is only possible if we have sufficiently structured XML documents as input respectively output. This is the prerequisite to successfully retrieve fine-grained element contents using the XPath queries specified in the constraints. If a message only contains a single text field the Web-Flow MaX monitoring component may only check whether a particular string is contained in the text.

Furthermore, if an element of a message that should provide a value is left blank, the XPath query will result in an empty document, the monitoring table will be filled with the \texttt{null} value, and the quality constraint will be evaluated to "violated". This is correct if the missing value is required, but has to be seen as an undesired outcome if the required value is optional. To prevent this, during constraint definition the Web-Flow MaX constraint editor checks whether the XPath query of an input, result, or metadata constraint references a required or optional document element. Therefore, it may use the type definitions for the input and output messages contained in the WSDL description of the respective service. If an optional element is referenced the administrator is asked to confirm the application of the constraint though it may lead to an unwanted constraint violation.

5.3.2.3 Updates of Monitoring Tables

We also have to consider the case that the tuples in the monitoring tables containing the desired values of a quality constraint may be updated independently of the evaluation of the quality constraint. That means that the monitoring tables are updated after the initialization phase but before the activity is in fact executed and an event has occurred. Possible reasons for such updates may for instance be the reception of an external event with a notification that a global result constraint

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{dependencies.png}
\caption{Dependencies between input and output messages of different activities.}
\end{figure}
Event Detection

defining the maximal price for monitors has been reduced to 100,00 EUR. Thus, all tuples that compare an ATP response message to this maximal price are identified and updated (for instance, also the constraint shown above).

The update mechanism is also necessary to assign the desired values for start time constraints. As start time constraints typically are not known until the process is actually executed, the desired values are initialized with *null*. The according tuple in the monitoring table is updated as soon as a message is received that specifies the desired start time. Therefore, the start time constraint may specify a parameter and an XPath expression describing how the desired values may be derived from the parameter. Alternatively, some calculations may be defined specifying how the desired value may be calculated considering for instance the reception time of a particular message plus a temporal distance. As the realization of the update mechanism and the calculation of desired values for start time constraint are more technical problems we do not discuss them further in this thesis.

5.4 Summary

In this chapter we have described the mechanisms used in the Web-Flow MaX monitoring component to observe quality constraints and events. Main requirements for event detection are to monitor events without influencing the process execution and to evaluate events efficiently. Furthermore, correlation of events and process instances is necessary. Additionally, event detection should be executed automatically to ensure that no events and exceptions are overlooked. Special challenges in the context of Web-Flow are that the Web services called in cooperative processes are heterogeneous and the service providers are autonomous.

Web-Flow fulfills these requirements as it monitors the messages sent between the process engine and the service provider. These messages can be monitored independently of the process engine. Web-Flow also provides mechanisms to correlate the observed messages with the currently executed process instances. Furthermore, these messages are available for different types of services and can be monitored without violating the autonomy of the service providers. Further event sources are interfaces for manual or automatic notifications which are part of the Web-Flow MaX component or timers monitoring temporal constraints. The use of these process engine independent event sources also contributes to Web-Flows goal of providing a generic solution for monitoring the process execution quality.

Efficient evaluation of events and in particular of quality constraints is provided by making use of the optimized evaluation features for boolean expression offered by query processors of relational databases. The constraint evaluation as well as the event monitoring are performed automatically. This ensures that no event is overlooked and that possible constraint violations and exception are detected in time.
6 Dynamic Event and Exception Handling

The main goal of dynamic event and exception handling in business processes is to ensure that a process instance may be continued with the best execution quality possible despite the exception. For instance, if an activity failed that should provide data for subsequent activities, alternative services may be used to generate the necessary data. The overall execution quality of the cooperative process instance may still be decreased (for instance, as overall time frames may be violated), but the process instance may be continued successfully and does not have to be cancelled.

Therefore appropriate exception handling actions have to be provided that can be executed efficiently without delaying the overall process execution too much. To relieve the users from handling exceptions manually the actions should be determined and applied semi-automatically. Depending on the current execution context the most appropriate actions should be applied. Thus, actions should not be derived until an exception has really occurred. It has to be ensured that no undesired actions are applied. Therefore, mechanisms are needed that allow administrators to control which actions are in fact executed. Furthermore, side effects of the action application on other process instances (for example, a delayed delivery of required data) should be considered.

To address these requirements the Web-Flow MaX dynamic event and exception handling component uses a rule-based exception handling approach. With this approach actions are not derived until an exception has really occurred. The approach is accomplished by a log-based derivation of appropriate actions if no matching rule could be found. The actions provided by Web-Flow are implemented in automatically executable operators. These operators aim at generating a state of the cooperative process instance in which the execution may be continued.

Furthermore, the Web-Flow MaX dynamic event handling offers mechanism to detect and avoid possible negative impacts of actions on quality constraints defined for future service calls in the same cooperative process instance. For instance, it is checked whether the execution of the action may lead to the violation of any temporal constraint defined on this or future activities of the process instance. Furthermore, if the complex cooperation model is used we may inform cooperation partners about possible delays or missing data in advance.

In this chapter we give an overview over Web-Flow’s dynamic event and exception handling...
Dynamic Event and Exception Handling

(Section 6.1) and discuss the selection of appropriate actions (Section 6.2). After shortly sketching the handling of pending service calls (Section 6.3) the operators implementing the different actions are described in detail (Sections 6.4, 6.5, 6.6). This also includes the detection and handling of effects on future activities in the remaining process instance or on cooperation partners.

6.1 Dynamic Exception Handling: Overview

This section introduces the basic algorithm for dynamic event and exception handling used by the Web-Flow MaX component and discusses the handling of events occurring during event handling.

6.1.1 Basic Algorithm

Besides the requirements formulated above dynamic event and exception handling in Web-Flow has to address some specific challenges. These are related to the focus of Web-Flow on providing a generic solution for quality monitoring and exception handling in Web-service-based cooperative processes. Exception handling in the Web-Flow MaX exception handling component has to be applied independently of the process engine. After applying the appropriate action the engine must be able to continue the process instance. That means that appropriate messages have to be generated and sent to the original receiver (for instance, the process engine). Furthermore, the Web services called in cooperative process instances are heterogeneous and the service providers are autonomous. Thus, exception handling operators have to be applicable without violating the autonomy of the service provider. Furthermore, the actions should be applicable in the same way to different types of services (for instance, legacy applications as well as services with manual interactions).

Fig. 6-1 shows the basic algorithm for exception handling. The bold activities in Fig. 6-1 mark subroutines which are presented in detail in the subsequent sections of this chapter. The Web-Flow MaX dynamic event handling is called by the monitoring component when an exceptional event has been detected. Then first an appropriate action is selected to handle the event. Therefore, either rules are used (cf. Section 6.2) or if no appropriate rule is found the action is derived dynamically from the log entries gathered by the Web-Flow MaX logging and analysis component (cf. Chapter 7). The action selection step results in one or more actions which are to be executed to handle the event.

As Web-Flow aims at both relieving the process administrators as much as possible and giving them sufficient control over event handling the selected actions have to be confirmed before they are executed. Thus, the administrator has the possibility to specify another action or to perform some manual event handling if the action suggested by Web-Flow is not appropriate. This confirmation step makes the Web-Flow architecture also usable in critical application areas such as production or health care in which wrong actions may lead to loss of money or threaten the recovery
Fig. 6-1: Overview of event handling in Web-Flow. Bold activities mark sub-routine calls.
Dynamic Event and Exception Handling

of patients. However, as this step is implicitly contained for instance in a manual reaction action, the confirmation is part of the operators implementing the different actions and is not shown in the overall algorithm.

The further processing of selected actions will be illustrated by means of examples. Assume that a repeat service call action has been selected. Thus, the algorithm enters the left arm of decision node 2 in Fig. 6-1. First, it is checked whether there is a pending service call (for instance, after the violation of a response time constraint). This service call is aborted to ensure that only one service call is valid at one point in time (cf. Section 6.3). Then the respective operator is called to execute the action (cf. Section 6.4). This includes an analysis and handling of effects of the action execution on future parts of the cooperative process instance and the cooperation partners. Finally, it is checked whether the result message produced by the action execution is in the format expected by the original recipient (typically the process engine). If the format is not the expected one, the message has to be mapped to make it processable for the recipient (cf. Section 6.4.3.4). Then the result is passed to the recipient.

As no operator is defined for the actions continue process and manual reaction the handling of these action types is also covered here. If a continue process action is selected (left arm of decision node 1 in Fig. 6-1) no event handling is performed by the Web-Flow MaX component as an appropriate event respectively exception handler has already been defined in the cooperative process. It is only checked, whether an event handler really exists in the cooperative process. Then, the original message is forwarded to the original recipient (typically the process engine) and the process execution continues. If no event handler could be found in the metadata repository (for instance, as the action selection has returned a wrong action) the algorithm switches to a manual reaction action. Thus, it is ensured that the process may successfully be continued even if the wrong action has been selected.

The action manual reaction applies if none of the other actions could be automatically executed. To realize this action a graphical user interface is provided in which the administrator is notified about the event and some additional information (such as the process instance, the affected activity, and a list of event handlers defined in the process instance). The administrator may then either select one of the automatically executable actions or he may decide to perform real manual event handling. Depending on the user input the Web-Flow MaX component either calls the operator for the selected automatic action or finishes this path without further steps.

The algorithm finishes after the log entry containing the event has been completed with the action that has been executed and some information whether event handling has been successful. Fig. 6-1 omits the case that several actions are to be executed to handle a particular event. Therefore, the whole algorithm except the selection step is repeated until all selected actions have been executed. Then the algorithm finishes.
6.1.2 Handling of Inter-Dependent Events

We cannot assume that no further events will occur during event handling. For instance, if a repeat service call action is executed the service call may again fail as the service is still unavailable. We call these events inter-dependent events as they occur while an action is executed in the Web-Flow MaX dynamic event handling component and therefore depend on an preceding event. These events may be handled with the same dynamic event handling algorithm as regular events. However, we have to consider that a service call may have to be repeated very often until it finally succeeds. This leads to a long delay of the whole process instance execution. In the worst case the service provider has shut down its service forever, so we will never get an answer and the process execution is suspended forever.

Related work addressing exception handling for long running transactions (for instance, [DAYA1991, THAT2003]) face similar problems. The proposed solution is to execute each exception handler only once. If it fails, the transaction is aborted and the failure is re-thrown to the enclosing exception handler (if available).

Web-Flow relaxes this approach by defining a global maximum number of inter-dependent event handling steps. This number specifies the maximum number of iterations for event handling which applies if event handling is not terminated earlier by a successful action execution. To calculate the total number of iterations for event handling all action executions are considered (i.e., the handling of the first „regular“ event as well as of all subsequent events) independently of the action type.

The decision whether an inter-dependent event is handled with one of the actions provided by the Web-Flow MaX component does not only depend on the total iteration number. It also considers whether a new iteration may violate any response time constraint defined for this activity or for the whole process. However, as this check is performed for every operator which may violate time constraints we do not discuss this here but refer to the description of the different operators in Section 6.4, 6.5, and 6.6.

6.2 Selection of Actions

After we have given an overview of the basic event handling algorithm we now discuss different sub-routines in more detail and start with the selection of actions. This step is performed to derive automatically which action is to be executed to handle an event. Note that although we often speak of one action which is selected in the following there may also be several actions that are to be executed for an event. However, we concentrate on the case that one action is executed as several events simply are executed sequentially in several iterations of the basic algorithm.

The Web-Flow MaX dynamic event and exception handling supports two ways for selecting actions. First, rules may be defined which relate events and actions, especially in case of frequent
Dynamic Event and Exception Handling

events. Second, Web-Flow may use the log entries gathered by the Web-Flow MaX logging and analysis component to derive an appropriate action dynamically. The algorithms used for this dynamic selection are presented in Chapter 7. The second step is more time consuming as there are typically more log entries than rules. Therefore, the basic strategy of Web-Flow MaX is as follows:

1. Search the existing rules for an appropriate action.
2. If no action has been found, use the dynamic action selection and call the Web-Flow MaX logging and analysis component.

Selection of Rules

The rules are stored as WXS documents in the Web-Flow MaX metadata repository. They have the following basic structure (as defined in Section 4.3.3):

```xml
<rule>
  <event>...</event>
  <action>...</action>
  <condition>...</condition>
  <operation>...</operation>
</rule>
```

Each rule has a mandatory event and action part, the condition and operation parts are optional. The rules used in Web-Flow are similar to event condition action (ECA) rules used, for instance, in active database systems [PATO1999]. ECA rules are used to model and implement active behavior in database systems. For example, they can trigger the execution of particular functions after new data has been entered in the database. Web-Flow rules also trigger an automatic handling of exceptions that have occurred during process execution.

The optional parts of the Web-Flow rules may be used to further restrict the application of the rule (again similar as in ECA rules), for instance, to a particular operation or only to cases where the total number of service call iterations has reached a given threshold. Rules which are only applicable after a specified number of service call iterations are necessary to ensure that event handling terminates after a finite period of time (cf. Section 6.1.2). The condition part may either contain a quality constraint (for instance, an iteration constraint) or a context that describes a sequence of operations in which the current operation is executed.

A rule applies to a particular event if one of the following cases is fulfilled:

1. The rule only consists of an `<event>` and an `<action>` part and
   - the occurred event matches the `<event>` part of the rule.

2. The rule consists of an `<event>`, an `<action>`, and an `<operation>` part and
   - the occurred event matches the `<event>` part of the rule and
   - the event has occurred while an operation of the type specified in the `<operation>` part has been executed.
3. The rule consists of an \(<event>\), an \(<action>\), and a \(<condition>\) part and
   - the occurred event matches the \(<event>\) part of the rule and
   - the condition is fulfilled. That means that either the quality constraint is fulfilled or the
     operation during whose execution the event occurred is embedded in the appropriate con-
     text.

4. The rule consists of an \(<event>\), an \(<action>\), an \(<operation>\), and a \(<condi-
   tion>\) part and
   - the occurred event matches the \(<event>\) part of the rule and
   - the event has occurred while an operation of the type specified in the \(<operation>\) part
     has been executed and
   - the condition is fulfilled.

Handling of Several Applying Rules

If exactly one rule matching the current event is found in the Web-Flow MaX metadata repository,
the action specified in the \(<action>\) part is executed to handle the event. If several rules are
found it is additionally checked whether one rule is a specialization of one of the others. Checking
for rule specializations can be regarded as a special kind of rule analysis as addressed on the fields
of active databases and artificial intelligence [BARA1999, ROAN2000].

A rule is a specialization of another rule if a stricter condition has to be fulfilled. In that case only
the action of the more special rule is applied (since it subsumes the action of the more general
rule). As rules are defined semi-automatically, i.e., with manual interaction, we can make this as-
sumption as users typically define stricter rules to cover special cases which are not covered by
the more general rule. The Web-Flow MaX component uses the following rule to decide whether
a rule \(r_1\) is a specialization of a rule \(r_2\):

\[
\text{\(r_1\) is more special than \(r_2\) if}
\begin{align*}
& \text{the event parts of \(r_1\) and \(r_2\) are the same and} \\
& \text{\(r_1\) has at least one more optional part (i.e., condition or operation) than \(r_2\) and} \\
& \text{if \(r_1\) and \(r_2\) have a common optional part this is the same in both rules.}
\end{align*}
\]

If \(r_1\) and \(r_2\) have different optional parts, i.e., \(r_1\) has a \(<condition>\) part and \(r_2\) has an \(<oper-
ation>\) part, we cannot decide automatically which rule is more special. In this case both actions
are applied. As this includes a user confirmation for the automatic execution of the actions the ad-
ministrator still has the possibility to cancel the application of one of these actions.

To illustrate the handling of rules let us consider an example. Assume that we have the following
two rules (the rule parts are only described informally, details are omitted for better readability):

\[
\begin{align*}
\text{<rule ruleID="1">} \\
\text{<event>event1</event>} \\
\text{<action>repeat service call</action>} \\
\text{</rule>}
\end{align*}
\]
Dynamic Event and Exception Handling

and

<rule ruleID="2">
  <event>event1</event>
  <action>alternative service call</action>
  <condition>already two iterations</condition>
</rule>

When the original event, which occurred at process execution time, and the first inter-dependent event are handled only the first rule applies. This leads to a repetition of the service call. But if the second service call also fails both rules will apply as then the service has already been called two times and the condition of the second rule is fulfilled. The second rule is stricter than the first one as it additionally specifies a condition for the same event. So only the action of the second rule will be executed.

6.3 Handling of Pending Service Calls

When an event is reported to Web-Flow the service call during which the event occurred may still be pending. That means, the service has not sent a response yet. This situation may occur due to the violation of response time constraints or if a manual fault message or an external event notification is forwarded to Web-Flow MaX. In both cases the response from the Web service has not been received when the event occurs and is handled.

Fig. 6-2 illustrates this scenario. At point in time \( t_1 \) the process engine calls a Web service operation. At \( t_2 > t_1 \) an event is detected (for instance, the violation of a response time constraint defined for this service operation). Web-Flow MaX handles the event by repeating the service call at \( t_3 \). The response message \( m_2 \) for this repeated service call is received by Web-Flow MaX at \( t_4 \) and forwarded to the process engine. Consequently, the execution of the process instance may be continued. However, as the original service call has not been aborted before the repetition of the service call, at \( t_5 \) the response message \( m_1 \) for the original service call is received by Web-Flow MaX. As message \( m_2 \) has already been sent to the process engine and the process instance has been continued, message \( m_1 \) is withdrawn by Web-Flow MaX.

To avoid such ambiguous service response messages, pending service calls are aborted before the selected action is executed. Therefore, three possible cases are considered:

- In the first case the process engine and the services execute the cooperative processes using a coordination protocol with transaction support such as WS-Coordination [CABR2003] and WS-Transaction [CABR2002]. Then the pending service calls are aborted by sending an abort message according to the transaction protocol. After this the new service call may be started and it may be assumed that the next received response message is correlated to this call.
- In the second case, no transaction protocol is available and Web-Flow MaX has no possibil-
Handling of Pending Service Calls

Web service

Web-Flow MaX

Process Engine

Web service

Web-Flow MaX

Process Engine

Fig. 6-2: Illustration of ambiguous Web service response messages. Message $m_2$ is the response from the repeated service call, message $m_1$ from the original service call.

ity to send an abort message to the Web service. So, it may send a notification to a responsible user. However, this solution is only applicable for long running services as for short running services the operations may already have been finished when the notification is received. Furthermore, whether an abort notification message is useful also depends on the type of occurring events. For instance, if the event is related to the violation of a response time constraint the service provider may not be reachable at all and thus cannot receive the notification.

To deal with these shortcomings, Web-Flow MaX utilizes message correlation information provided by the network monitor which intercepts the messages sent between the process engine and the services (cf. Section 5.2.1). Thus, it may be derived to which request a response message belongs and Web-Flow MaX only forwards the response for the repeated service call and withdraws the other message (as discussed for the example above). This strategy also ensures the correct handling of events occurring because of a network failure and not because of a failed operation execution. That means, the service operation has been executed correctly and the response was delayed during its way back to the process engine thus, leading to violation of a response time constraints.
Dynamic Event and Exception Handling

- Third, if no transaction protocol and no message correlation information is available the Web-Flow default strategy is used:

  Forward the first response message received after the new service call and destroy all response messages received later.

We are aware that messages may be permuted by this strategy, i.e., the response for the original service call is forwarded instead of the response from the repeated call. However, indistinguishable response messages only occur if the same service operation is called again possibly with different parameter values (one subtype of the alternative service call action). As in this case the event handling is completely performed by Web-Flow MaX the process instance still waits for the message of the original service call. Thus, it should not lead to failures if the response to the original message is delivered instead of that of the repeated operation call. So, the process execution may be continued after the response message has been received by the process engine.

6.4 Service Call Related Actions

In the following sections we introduce the different operators that are used in the Web-Flow MaX dynamic event and exception handling component to execute actions automatically. There are operators for the actions repeat service call, suspend service call, alternative service call, procedure call, and abort process. Fig. 6-3 gives an overview of all action provided by Web-Flow and also of the different subtypes. For the actions manual reaction and continue process no operators are provided (indicated by the grey boxes in Fig. 6-3). The action manual reaction just notifies a user and then possibly executes one of the other automatically executable actions. If the action continue process is triggered the exception handlers defined in the cooperative process are used. So, the Web-Flow MaX dynamic event handling simply forwards the message to the process engine.

Each operator is introduced with an WXS (Web-Flow quality and eXception Schema) example for the corresponding actions to show which information is available during event handling. We describe the main functionality of each operator and discuss how effects of the action execution on future parts of a process instance respectively on the cooperation partners may be considered.

In this section we start with the operators repeat-service-call, suspend-service-call, and alternative-service-call used to implement the service call related actions in the middle part of Fig. 6-3. We describe them together as their execution and effects are very similar. All three operators are executed by calling a service operation and forwarding the result (possibly after some additional steps) to the recipient of the original service call. The original service call is that one whose failure led to the action execution.

A suspension of a service call in Web-Flow means that the same service call is repeated but not
Fig. 6-3: Overview over Web-Flow exception handling actions.
Dynamic Event and Exception Handling

until some time has elapsed. Thus, the operator suspend-service-call is an extension of the operator repeat-service-call. The same holds for the operator alternative-service-call as it may either call the same service operation again with different parameters or it may call a different service operation. Thus, it demands the same basic operations as the operator repeat-service-call.

In the following, we introduce the three different operators starting with the most simple one, repeat-service-call, and ending with the most complex one, alternative-service-call. Then we describe additional tasks that are related to the operator alternative-service-call before we discuss the effects of the three operators on the further execution of the cooperative process instance respectively on the cooperation partners.

6.4.1 Operator repeat-service-call

A repeat service call action tells the Web-Flow MaX dynamic event handling component to repeat a service call that has produced a failure event immediately after the action has been derived. Possible failure events that may be handled with this action are the following:

- The service has not answered in time and a response time constraint has been violated.
- The service has sent a failure message that it was not able to process the request (without any further details).
- A manual fault message has been received that notifies the Web-Flow MaX that the service has been out of business but is working again now. Such messages may be sent for complex, long running services with manual interaction.

An example for a WXS specification of a repeat service call action is given in Fig. 6-4. It specifies that the operation „op2“ of service „s2“ (line 3) executed in the process instance with ID „xx1652“ (line 4) is to be repeated. The operator repeat-service-call processes this action in several steps:

Step 1. First it is checked whether there are iteration constraints which may prohibit the repeated execution.

Step 2. The operator checks whether any response time constraint is defined for this activity that might be violated when the activity is repeated. This step is only executed if the repeat

```
1  <action>
2   <repeatServiceCall>
3     <serviceOperation operationName="op2" ... serviceName="s2" />
4   </processInstanceID>xx1652</processInstanceID>
5  </repeatServiceCall>
6  </action>
```

Fig. 6-4: WXS description of a repeat service call action.

124
**Service Call Related Actions**

A *service call* action is the reaction to another event than the violation of a response time constraint.

Step 3. If no violations occur, the operator will request the administrator’s confirmation and repeat the service call.

Step 4. If any iteration constraint is violated, the operator will stop and switch to manual reaction.

Step 5. If any response time constraint is violated the operator will use additional criteria such as the strictness of a constraint to decide whether to repeat the service call or not.

In the following we describe the steps in more detail.

### 6.4.1.1 Steps 1 & 2: Consideration of Quality Constraints

For the execution of Step 1 it is checked whether there is any valid iteration constraint for this activity that specifies that the service operation may only be called once. Typically those constraints are already considered during action selection. In order to prevent that an iteration constraint has not been considered this additional check is included in the operator *repeat-service-call*. If the operation may be repeated, the operator continues with Step 2, otherwise it stops and switches to manual reaction as described in Section 6.1.

For the consideration of response time constraints (Step 2) first any response time constraint that is valid for the actual activity is derived from the Web-Flow MaX monitoring component (cf. Chapter 5). If no response time constraint exists, the operator may directly continue with Step 3. If such constraints exist an estimated execution duration $dur_{exec}$ of the service call that is to be repeated is needed. This value may be retrieved from the logged execution data gathered by the Web-Flow MaX logging and analysis component. For the exception handling an average execution duration is calculated as we assume that it represents the actual execution duration best. Details on the derivation of execution duration estimates are provided in Section 7.1.2.

If this value is available then the expected end time $end_{repeat}$ of the repeated service call is calculated as

$$end_{repeat} = now + dur_{exec}$$

where $now$ is a representation of the current point in time.

The point in time when a response time constraint is violated, $t_{violation}$, may be retrieved from the Web-Flow MaX monitoring component. Then for each response time constraint the following comparisons are calculated:

- If $t_{violation} > end_{repeat}$ then probably no response time constraint will be violated and the operator may continue with Step 3 and repeat the service call.
Dynamic Event and Exception Handling

If \( t_{\text{violation}} \leq end\_repeat \) then probably the response time constraint will be violated and the operator continues with Step 5.

The expected end time \( end\_repeat \) is only an estimated value as the execution duration is based on the values measured during past executions of a service operation. Thus, the actual execution may still take longer or last shorter, i.e., the calculated end time may turn out to be wrong. However, if a sufficient number of measured duration values exist (for instance, after six months of using Web-Flow MaX), the estimated duration value may be sufficiently precise and the comparisons will be fulfilled in a large number of cases. Furthermore, we consider it the better the solution to work with estimated values than not to consider possible response time constraint violations at all. The calculation of the expected end time is considered as a hint whether response time constraint violations may occur if the service call is repeated.

6.4.1.2 Step 3: Repetition of Service Call

During Step 3 the operator repeats the service call. That means, it first retrieves the following information that is necessary to call the service:
- the service address,
- the operation parameters,
- the parameter values of the original service call.

Service address and parameters may be retrieved from the documents in the Web-Flow MaX.
metadata repository (Step 1 in Fig. 6-5). They are contained either in the Web-Flow specific service description documents or in the original WSDL description of the service. The concrete parameter values for the repeated service call may be derived from the input message \( i \) that has been sent from the process engine to the Web service and that has been intercepted and stored by Web-Flow MaX (Step 2).

With this information a new service call with a new input message \( i' \) is executed (Step 3). The monitoring of quality constraints and events for this new service call is discussed below. If the repeated service call is successful, the result message \( o \) sent to Web-Flow MaX is forwarded to the process engine (Step 4) and the process instance execution continues.

As the Web-Flow MaX monitoring component also monitors the repeated service calls events may be detected with the same mechanisms as used for regular service calls. Thus, events occurring during exception handling are also handled with the basic algorithm introduced in Section 6.1. No further steps need to be taken.

6.4.1.3 Monitoring of Quality Constraints During Action Execution

Each quality constraint that is valid for the original activity is also valid for the repeated service call. Thus, the constraints are also monitored during the execution of the \textit{repeat-service-call} operator. Therefore, some additional preparation steps may be necessary. For instance, the monitoring of response time constraints has to be initialized again as the violation of a response time constraint may have caused the execution of the \textit{repeat service call} action.

The necessary preparation steps differ depending on which event caused the execution of the \textit{repeat service call} action. We distinguish the following cases\(^5\):

1. \textit{Violation of response time constraints}: In this case the Web-Flow MaX component has not received a response message from the service operation yet. Thus, no result constraints may have been evaluated. However, input or iteration constraints may have been evaluated when the service call was started. Thus, the monitoring of the following constraint types is initialized again: response time, input, and iteration constraints.

2. \textit{Violation of result constraints}: In this case Web-Flow MaX has already received a regular response message whose content does not meet the user’s expectations expressed in the result constraint. Input, result, response time, and iteration constraints may already have been evaluated. Thus, the monitoring of these constraints is initialized again before the service call is repeated. Note, that a \textit{repeat service call} action is indeed a rather unlikely reaction to a violation of a result constraint (an \textit{alternative service call} action would be more likely). But principally it may occur.

5. Metadata, input, and iteration constraints are not considered as their violation may not lead to a \textit{repeat service call} action as they are typically checked before a service call is executed.
3. **Web service fault and manual fault messages**: If the action was triggered by a Web service fault message or a manual fault message, input and iteration constraints might have been evaluated when the original service call was started. If a Web service fault message was received, also response time constraints may have been evaluated. Result constraints are not affected as they are related to regular output messages. Thus, the monitoring of input, iteration, and response time constraints is initialized again before the repeated service call is started.

For monitoring and evaluation of Web service fault messages and manual fault messages no initialization steps are necessary as they are not related to quality constraints. Therefore, they just have to be related to the new service call sent by the Web-Flow MaX dynamic event handling component. This may be realized with the standard mechanisms offered by the Web-Flow MaX monitoring component described in Sections 5.2.1 and 5.2.2.1.

### 6.4.1.4 Step 5: Decision about Repetition

The *repeat-service-call* operator may detect in Step 2 that possibly response time constraints which have not been violated yet may be violated when the service call is repeated. In this case it is difficult to decide automatically whether the service call may be repeated or not. For example, the violation of the response time constraint may be tolerable if then a result may be achieved from the Web service.

Web-Flow MaX uses the strictness attribute of a response time constraint to decide this question. As defined in Section 4.2.3 the strictness of a response time constraint may be specified as „high“ or „low“ with the following meaning:

- „low“ means that the constraint may be violated if then another constraint may be fulfilled or an event handling action may be executed.
- „high“ means that the constraint has to be fulfilled under all circumstances.

Thus, the decision whether a service call is repeated or not is calculated as follows:

- If strictness of $rtc$ is low, the operator repeats the service call. The possible violation of the response time constraint is tolerated.
- If strictness of $rtc$ is high, the operator does not repeat the service call. Instead it switches to a *manual reaction* action as the question of the right action is not decidable automatically.

In the first case, the response time constraint $rtc$ is not valid any more for this activity. Thus, the Web-Flow MaX monitoring component deletes $rtc$ from the set of quality constraints that are monitored for the currently executed activity. This prevents that an event is raised erroneously by the violation of $rtc$. 
6.4.2 Operator suspend-service-call

The operator *suspend-service-call* is similar to the operator *repeat-service-call*. It also repeats a service call after an event has occurred. However, the repeated service call is not started until a certain interval has elapsed. An example for a WXS specification of a *suspend service call* action is depicted in Fig. 6-6. It specifies that operation „op3“ of service „s2“ (line 3) executed in the process instance „xx1287“ (line 4) is to be suspended for one hour and 20 minutes (line 5).

Possible failure events that may be handled by means of this action are the following:

- The Web-Flow MaX component has received a manual fault message that notifies the service user that the service is not available for a certain period of time. Then the service call may be suspended until the time interval will have elapsed.
- A service call has been repeated for a certain number of times without getting a result. Then it may be a good solution to wait some time before calling the service again.
- An input constraint has been violated and may be handled by suspending the service call for a particular time (for instance, as one of the parameters contains an implicit start date for the operation that may be evaluated by an input constraint).
- A start time constraint has been violated as the desired start time has not yet been reached. Thus, the service call is suspended until the start time is reached.

The examples for failure events show that we have to consider two possible situations when the operator *suspend-service-call* may be called:

- before the original service call message is sent to the Web service (the last two examples) or
- after the original service call has been executed (the first two examples).

Thus, we get two variants of the operator *suspend-service-call* which we describe in the following.

6.4.2.1 Suspend-Service-Call Before Operation Execution

This variant of the operator is used when the action is executed after an input constraint or a start time constraint defined for the affected activity has been violated. In this case, simply the operator

```
1  <action>
2   <suspendServiceCall>
3    <serviceOperation operationName="op3" ... serviceName="s2"/>
4   </processInstanceID>xx1287</processInstanceID>
5   <suspensionInterval>PT1H20M</suspensionInterval>
6  </suspendServiceCall>
7  </action>
```

Fig. 6-6: WXS description of a *suspend service call* action.
Dynamic Event and Exception Handling

has to delay the delivery of the intercepted input message to the Web service until the suspension interval has elapsed. Fig. 6-7 illustrates the execution of this operator variant.

After the input message \( i \) has been intercepted by the Web-Flow MaX component, event \( e \) is detected and the action suspend service call is selected to handle the event. Then Web-Flow MaX starts a timer for the duration of the suspension interval (Step 1 in Fig. 6-7). After the timer has elapsed, the original input message \( i \) is forwarded to the Web service (Step 2). After successful service execution the output message is sent back to the process engine (Step 3).

As no original service call precedes the execution of the operator no re-initialization of the quality constraint monitoring is necessary. The handling of events which occur during the execution of the service operation is also handled with the standard mechanisms provided by the Web-Flow MaX component.

6.4.2.2 Suspend-Service-Call After Operation Execution

The second variant of the operator suspend-service-call is used when the event has occurred during the execution of the service operation. Thus, it is executed in similar steps as the operator repeat-service-call presented in Section 6.4.1:

Step 1. First the operator checks whether there are iteration constraints which may prohibit the repeated execution of the service call.

Step 2. The operator checks whether any response time constraint is defined for this activity that might be violated when the activity is repeated after the suspension interval has elapsed. This step is only executed if the suspend service call action is the reaction to another event than the violation of a response time constraint.
Step 3. If no violation occurs, the operator will request the user’s confirmation and suspend the service call for the specified time.

Step 4. If any iteration constraint is violated, the operator will stop and switch to manual reaction.

Step 5. If any response time constraint is probably violated the operator will use additional criteria (such as the strictness of a response time constraint) to decide whether to suspend the service call or not.

Steps 1, 4, and 5 are executed in the same way as described for the operator repeat-service-call in Section 6.4.1. For Step 2 and 3 some additional remarks become necessary.

For the consideration of response time constraints (Step 2), again the expected end time $end_{suspend}$ of the suspended service call is calculated, if any valid response time constraint exists for the actual activity. Therefore, the estimated execution duration $dur_{exec}$ of the service call that is to be suspended is retrieved from the logged execution data gathered by the Web-Flow MaX logging and analysis component (cf. Chapter 7). Then $end_{suspend}$ is calculated as

$$end_{suspend} = now + t_{suspend} + dur_{exec}$$

where $now$ is a representation of the current point in time and $t_{suspend}$ is the suspension interval. The expected end time $end_{repeat}$ is then compared to $t_{violation}$, the point in time when a response time constraint is violated:
- If \( t_{\text{violation}} > \text{end}_{\text{suspend}} \) then no response time constraint will be violated and the operator may continue with Step 3 and suspend the service call.
- If \( t_{\text{violation}} < \text{end}_{\text{suspend}} \) then the response time constraint will be violated and the operator continues with Step 5.

The suspension of a service call, after the original service call has been executed (Step 3), is illustrated in Fig. 6-8. Similar to the first variant of the operator first a timer set to the suspension interval is started (Step 1 in Fig. 6-8). While this timer runs the operator creates the new input message \( i' \) with information derived from the Web-Flow metadata repository and the original service input message \( i \) (Steps 2 and 3). After the timer has elapsed the service is called again (Step 4) and the output message \( o \) is forwarded to the process engine if no further event has occurred (Step 5).

The mechanisms for the monitoring of quality constraints during the repeated execution of a service operation and the handling of events occurring during the repeated execution are the same as described above for the operator \textit{repeat-service-call}. So, we do not further discuss these issues here.

### 6.4.3 Operator alternative-service-call

The action \textit{alternative service call} is used to call an alternative service operation if the execution of the original service has failed. The alternative service is called directly after the action has been selected. Failure events that may be handled with this action are for instance the following:

- The original service call has returned a response message that violates one or more result constraints, for instance, as the customer wishes are not considered or the price in an offer is too high. Thus, an alternative service may be called or the input parameters may be adjusted to get a satisfying result.
- A service call has been repeated for a number of times without finishing successfully (for instance, as the network of the service provider has broken down). Then it may be conceivable to try another service operation that provides the same functionality.
- Web-Flow MaX has received a Web service fault or a manual fault messages which notifies this component that the service will not be available for a quite long time (for instance, as a big system crash has occurred at the service provider). Thus, an alternative service may be used to get the desired results and to continue with the execution of the process instance.

As the examples show there are different types of actions for \textit{alternative service call} depending on the type of occurring event. In particular, we distinguish three different kinds of actions for \textit{alternative service call} which result in three variants of the \textit{alternative-service-call} operator (illustrated in Fig. 6-3):

1. The operator calls the same service operation again but with different parameter values.
2. The operator calls a different service operation with the same parameter values as the original service call. The service operation is explicitly specified in the action description.

3. The operator calls a different service operation with the same parameter values as the original service call. The service operation is specified by metadata constraints in the action description.

The fourth possible variant, the call of a different service operation with different parameter values, is not covered as the goal of the alternative service call is to provide the results expected from the original service operation by using an alternative one. If we call a totally different service operation with new parameter values this is not related to the original service call any more as then no values (neither for the operation nor for the parameters) may be derived from the original service call. Furthermore, it would be difficult to execute the new service call automatically.

The different variants of the operator have some common parts that may be re-used in a concrete implementation of the operator. However, for a conceptual description of the operators we discuss each variant of the operator completely separately in the following.

6.4.3.1 Variant 1: Same Operation - Different Parameters

This variant of the operator is similar to the repeat-service-call operator. The only difference is the generation of the input message for the new service call. Thus, the operator is executed in the following steps:

Step 1. First it is checked whether there are iteration constraints which may prohibit the repeated execution of the service call.

Step 2. The operator checks whether any response time constraint is defined for this activity that might be violated when the activity is reinvoked. This step is only executed if the alternative service call action is the reaction to another event than the violation of a response time constraint.

Step 3. If no violation occurs, the operator will request the administrator's confirmation, construct the new input message with the different parameter values and reinvoke the service operation.

Step 4. If any iteration constraint is violated, the operator will stop and switch to manual reaction.

Step 5. If any response time constraint is violated the operator will use additional criteria (e.g., the strictness of a constraint) to decide whether to execute the action or not.

Steps 1, 2, 4, and 5 are executed in the way described in Section 6.4.1 for the operator repeat-service-call. Thus, we do not describe them further. The execution of Step 3 is illustrated in Fig. 6-9 and is described in the remainder of this section.
Dynamic Event and Exception Handling

Fig. 6-9: Illustration of first variant of alternative-service-call operator. The bold numbers denote the steps referenced in the text.

```xml
1 <action>
2  <alternativeServiceCall>
3   <affectedOperation operationName="op2" ... serviceName="s1" />
4   <processInstanceID>x1453</processInstanceID>
5   <alternativeParamValue>
6     <parameterPart dataType="float">
7       <xPathQuery>/monitorDetails/maximumPrice</xPathQuery>
8       <parameter name="monitorOfferRequest" type="Input"/>
9     </parameterPart>
10    <assignValue>
11      <floatValue>150.00</floatValue>
12    </assignValue>
13   </alternativeParamValue>
14 </alternativeServiceCall>
15 </action>
```

Fig. 6-10: WXS description of an alternative service call action of the first type.
The operator first retrieves the information necessary to call the service from the Web-Flow metadata repository (Step 1 in Fig. 6-9). Then it constructs the new input message $i'$ for the service call with the parameter values specified in the WXS description of the alternative service call action (Step 2). If some parameters are not specified in the action description then the parameter values from the original input message $i$ will be used.

An example of an WXS action description is shown in Fig. 6-10. It specifies that Operation 2 of Service 1 executed in the process instance „xx1453“ (lines 3 to 4) is to be repeated but with a different price value. For instance, this operation is an operation searching for offers for monitors and has returned no results. Then we may call it again with a higher maximum price. The new price value is specified in line 11. The surrounding element `<assignValue>` specifies that the new value is assigned to the parameter specified in lines 6 to 9. Alternatively, the new parameter value may be copied from another input or output message available to Web-Flow MaX.

With the new input message $i'$, the service is called again (Step 3) and the output message $o$ from a successful operation execution is forwarded to the process engine (Step 4). For the monitoring of quality constraints and events during the new service call and for the handling of events occurring during action execution, the same mechanisms apply as they have been discussed for the operator `repeat-service-call` in Section 6.4.1. Thus, we refer to this section for further details on these topics.

6.4.3.2 Variants 2 & 3: Different Service Operation

In the second and third variant of the operator alternative-service-call an alternative service operation is specified in the WXS description of the action; either explicitly as shown in the following example or with a metadata constraint as it has been introduced in Chapter 4. As these two variants only differ in the way the new service operation is retrieved we discuss them together in this section.

The example depicted in Fig. 6-11 specifies that for an operation the alternative operation „op1“

```
1 <action>
2   <alternativeServiceCall>
3     ...
4   <alternativeService>
5     <serviceOperation operationName="op1" ... serviceName="s5" />
6   </alternativeService>
7 </alternativeServiceCall>
8 </action>
```

**Fig. 6-11:** WXS description of an alternative service call action of the second type.
Dynamic Event and Exception Handling

of service „s5“ is to be called. The parameter values for the new input message are not specified explicitly but are derived from the input message of the original service call (see below). The operator executes these action types in the following steps:

Step 1. The operator retrieves the information to call the alternative service

   a) either from the Web-Flow metadata repository based on the specification provided in the action description
   
   b) or from service directories based on the metadata constraints specified in the action description. Therefore, the Web service search component of Web-Flow MaX is used.

Step 2. The operator checks whether there is any response time constraint defined for this activity that might be violated when the new service operation is executed. This step is only executed if the alternative service call action is the reaction to another event than the violation of a response time constraint and an estimated execution duration is available for the alternative service operation.

Step 3. The new input message is constructed and the alternative service operation is called.

Step 4. If any response time constraint is violated the operator will use additional criteria (for instance, the strictness of the response time constraint) to decide whether to execute the action or not.

As the operator calls a new service operation no iteration constraints are considered here. Response time constraints may be only considered if an estimated execution duration is available for the new service operation. In that case Steps 2 and 4 are executed in the same way as described above for the operator repeat-service-call (cf. Section 6.4.1.1).

For the execution of Step 1, the operator first retrieves the necessary information to call the new service. This information involves the

- service address,
- operation porttype and name,
- operation parameters.

In case a) the information is retrieved from the Web-Flow metadata repository using the operation specification provided in the action description. The information is either contained in Web-Flow specific service descriptions or in the original WSDL descriptions of the service. This step is illustrated with Step 1a) in Fig. 6-12. Step 1b) illustrates the execution of case b) above. Therefore, the Web-Flow MaX Web service search component is used to find an appropriate alternative service (cf. Section 6.4.3.3).

In Step 3 (Step 2 in Fig. 6-12) the new input message \( i' \) is generated. However, as the original operation and the alternative operation may use different input parameters the values of the original input message may not simply be copied. The original input message \( i \) is mapped to the input message \( i' \) of the alternative service operation (cf. Section 6.4.3.4).
After the new input message $i'$ has been generated, the operator calls the new service (Step 3). Before the execution is started the monitoring of any quality constraint valid for this service operation is initialized. Therefore, the standard mechanisms of the Web-Flow MaX monitoring component are used considering all possible constraints except those with type process specific (cf. Section 5.3). We may exclude these constraints as they refer to the standard execution of an operation in a particular process and not to an exceptional one. Events occurring during the execution of the alternative service operation may again be handled with the standard mechanisms provided by Web-Flow MaX.

Similar to the input message, the output message $o$ of the alternative operation is mapped to the output message format expected by the process engine after the operation has finished successfully (Step 4, cf. Section 6.4.3.4). The generated output message $o'$ is then forwarded to the process engine (Step 5).

In the following two sections we describe the use of the Web-Flow MaX Web service search component and the mapping of input and output parameters to complete the discussion of the operator `alternative-service-call`.

**Fig. 6-12:** Illustration of second and third variant of `alternative-service-call` operator.

The bold numbers denote the steps referenced in the text.
6.4.3.3 Web Service Search in Service Directories

The Web service search component is used to automatically search for appropriate services for an alternative service call action. The desired service operation is described using a constraint expression consisting of metadata constraints that are used as input for the Web service search. Three possible sources may be used for the search:

- the Web-Flow metadata repositories of the local Web-Flow MaX component,
- UDDI registries,
- Web service search engines that allow for searching similar service operations such as Woogle [DONG2004] and provide an interface for automatic search.

For each of these sources, the information provided in the metadata constraints is translated in a format that may be sent to the interface of the respective source. As metadata constraints only specify parameters and parameter parts which contain a desired value, they may be translated to different interface formats. For instance, UDDI registries provide an API (Application Programming Interface) over which the registry may be queried automatically. For the Web-Flow metadata repositories XQuery or XPath queries may be used. As the translation process depends on the concrete interface a service directory offers, we omit details about the translation into the respective interface format here.

The Web-Flow MaX Web service search component processes a request in the following steps:

Step 1. In the first step the metadata constraints are sent to the Web-Flow MaX monitoring component to initialize the monitoring of these constraints. For the initialization and for the evaluation of the constraints later on the standard mechanisms of the monitoring component are used (cf. Chapter 5).

Step 2. Then the service description in the metadata constraints is translated into the interface formats of the sources to be used. Thereby, we also consider the strictness of the different metadata constraints:

a) If all metadata constraints have a high strictness only one search query is created as all constraints are to be fulfilled to make a service useful.

b) If at least one metadata constraint has a low strictness several search queries are generated: one that contains all constraints, one that contains only the constraints with high strictness and several queries each of which contains only a subset of the constraints with low strictness.

Step 3. In the next step all queries generated in Step 2 are sent to the respective sources. That service is selected from the search results that fulfills all constraints with high strictness and most constraints with low strictness. Therefore, the results from all sources are considered together. If no service is found the search continues with Step 5.

Step 4. To ensure that all metadata constraints are fulfilled (at least those with high strictness) the
selected service is sent to the Web-Flow MaX monitoring component for evaluation. If no metadata constraint is violated, the WSDL description of this service is returned to the alternative-service-call operator. If the service is found in a Web-Flow MaX metadata repository also the Web-Flow specific service description may be returned if it is available.

Step 5. If no appropriate service could be found in any of the available sources an administrator is notified and asked to specify a service that may be used for the alternative-service-call operator.

As an illustration for Step 2b), the generation of the queries, consider the following example. The constraint expression contains three metadata constraints $mc_1$, $mc_2$, and $mc_3$ which are concatenated with AND:

$$mc_1 \text{ AND } mc_2 \text{ AND } mc_3$$

Assume that $mc_1$ has a high strictness, whereas $mc_2$ and $mc_3$ have a low strictness. Thus, according to Step 2b) the following queries are generated (the queries are presented in a high level notation):

$$q_1: mc_1 \text{ AND } mc_2 \text{ AND } mc_3$$
$$q_2: mc_1$$
$$q_3: mc_1 \text{ AND } mc_2$$
$$q_4: mc_1 \text{ AND } mc_3$$

With these queries it is ensured that any service is found that matches the constraint expression. $q_1$ finds the services that match all three constraints, $q_2$ finds the services only matching the constraint with high strictness $mc_1$. $q_3$ and $q_4$ find those services that match the strict constraint $mc_1$ and one of the constraints with low strictness. Thus, the results for $q_1$ are the best fitting candidates. The results of $q_3$ and $q_4$ are the next appropriate candidates. Only if these three queries do not provide a result a service found by query $q_2$ is used as alternative service. If several services have been found (for instance, with $q_1$), simply the first one in the result set is used as all are considered equally appropriate.

A possible extension of the Web service search is to consider the execution duration of services and find a service that executes the given task with the least possible delay of the remaining process part. As not all service directories provide information about execution durations such optimizations may only be applied for selected service directories (for instance, the Web-Flow metadata repository).

6.4.3.4 Mapping of Input and Output Parameters

Principally, we may not assume that an alternative service retrieved from the Web-Flow MaX metadata repository or found by the Web service search component has exactly the same input and
output parameters as the original service. Thus, it is necessary to map the original message to the desired message format to make the message processable for the recipient. In this section, we discuss how the Web-Flow MaX component addresses this problem.

Consider the following example. A failed monitor search operation expects an input message of the following format (we use an XML notation as Web service messages typically are based on XML):

```xml
<monitorDetails>
    <brand>...</brand>
    <deliveryDate>...</deliveryDate>
    <driverVersion>...</driverVersion>
    <maximumPrice>...</maximumPrice>
</monitorDetails>
```

The alternative service operation found by the Web service search component expects an input message of the following format:

```xml
<monitorRequest>
    <brand>...</brand>
    <deliveryDate>...</deliveryDate>
    <price>...</price>
</monitorRequest>
```

Thus, the new service does not consider the driver version of the monitor and the element specifying the maximum price is only named `<price>`. To generate the input message for the new service automatically these differences between the two message formats have to be detected and handled appropriately.

Automatic schema matching is a complex problem that has been addressed by several researchers in the last years, for instance, [DOAN2001, RAHM2001, DO2002, MELN2004]. This problem is orthogonal to the problems addressed by Web-Flow. Thus, a solution to this problem is out of scope of this thesis. However, to provide a complete solution Web-Flow uses a manual approach for mapping messages. The accomplishment of the approach with solutions offered by existing systems (such as COMA [DO2002]) is subject of future work.

### 6.4.4 Side Effects of Operator Execution

The execution of the operators `repeat-service-call`, `suspend-service-call`, and `alternative-service-call` may have effects on the yet unexecuted parts of the cooperative process instance as well as on the cooperation partners. In particular the following effects may occur:

- As the operators execute additional service calls, the execution of the future activities of the process instance will be delayed. This will be not critical if there are no response time or start time constraints defined on the future activities. However, if such constraints are defined they
may be violated by the delay implied by the additional service call.
- There may be response time constraints which are valid for a whole process if this process is used as an activity in a superior process. The operator execution may violate such process-wide temporal constraints.
- The execution of an alternative service may lead to a loss of output parameters needed for future activities as the alternative service does not provide this data.
- The delay of the future activities may also influence the cooperation partners as they may have already scheduled their resources for the execution of future service calls which are also part of the cooperative process instance. This case especially applies if the complex cooperation model is used as only then the partners know in which processes their services are used and may prepare for future operation calls.

As such effects may further decrease the execution quality of a cooperative process which is to be ensured by Web-Flow MaX, the Web-Flow MaX dynamic event handling tries to anticipate these effects (if possible automatically). Therefore, the Web-Flow MaX dynamic event handling first checks which type of effects may occur after an operator has been executed. Then it takes appropriate steps to deal with these effects. In this section we describe how the different types of effects are detected and handled.

### 6.4.4.1 Handling of Temporal Effects

To consider possible violations of any temporal constraint execution durations for the activities in the unexecuted process part are needed. As already discussed for considering response time constraints during operator execution (cf. Section 6.4.1.1) these values may be derived from the log data gathered by the Web-Flow MaX logging and analysis component. However, this component may only provide values if an activity has been used earlier in at least one cooperative process instance monitored by Web-Flow MaX.

In order to decide whether a response time constraint \( \text{rtc}_{\text{process}} \) which is valid for the whole process instance is violated after the execution of an operator the duration of the whole remaining process instance \( \text{dur}_{\text{process}} \) has to be estimated. For this purpose, estimation algorithms as described in [MÜLL2004] may be used. As the estimation of future process parts is a quite complex problem and not in the main focus of this thesis we do not discuss it here but refer to the literature instead. An implementation of estimation algorithms is provided in the AdaptFlow prototype [GREI2005].

After \( \text{dur}_{\text{process}} \) has been estimated Web-Flow MaX may derive whether process-wide response time constraints are violated by executing the following comparisons:

- If \( \text{dur}_{\text{process}} < \text{rtc}_{\text{process}} \) the overall process execution time will not violate the response time constraint. Thus, no further steps are necessary.
- If \( \text{dur}_{\text{process}} > \text{rtc}_{\text{process}} \) the overall process execution time will probably violate the response time constraint \( \text{rtc}_{\text{process}} \).
Dynamic Event and Exception Handling

In the second case Web-Flow MaX may notify the cooperation partner about the possible violation of the response time constraint. Then the partner may consider the delay during the monitoring of the superior process instance. Note that the consideration of process-wide temporal constraints will only be possible if both partners use Web-Flow MaX or if the superior and the current process instance are both monitored by the same Web-Flow MaX.

The delay of the remaining process instance may lead to a violation of start time or response time constraints which specify a fixed date for future activities. That means that the date is reached before the activity is started at all (this case has been illustrated in Section 3.3.3.3). Another effect may be that the allowed execution time of an activity is reduced significantly thus leading to an inevitable violation of the response time constraint.

To handle these effects the Web-Flow MaX event handling component checks if the delay $d_{op}$ imposed by the operator execution influences response time or start time constraints in the remaining process part. Therefore, it first retrieves from the monitoring component all start time and response time constraints, $stc_i$ and $rtc_i$, which are valid for future activities in the process instance and specify a fixed date. Then each of these constraints is updated by adding the delay $d_{op}$ to the date value of the constraint:

\[
stc_i.date = stc_i.date + d_{op} \\
rtc_i.date = rtc_i.date + d_{op}
\]

The same mechanism may be used to update a response time constraint after a notification about the delay of a sub-process has been received. These updates may prevent the unwanted violation of temporal constraints caused by the execution of a repeat service call, suspend service call, or alternative service call action.

Another temporal effect that may be considered automatically is that the delay $d_{op}$ imposed by the execution of one of these operators leads to a delayed call of service operations in the remaining part of the cooperative process instance. With the complex cooperation model service providers know in which cooperative processes their services are used. Thus, they have the possibility to already schedule resources for subsequent service operations after the first of their service operations in a particular process instance has been called. Especially, this holds for long-running, complex services. If the delay $d_{op}$ is significant with respect to the execution time of the single service operations the Web-Flow MaX dynamic event handling component may notify the service providers in advance about the delay. Thus, they have the possibility to re-schedule their resources according to the new expected starting times of their service operations.
6.4.4.2 Missing Output Data

The new service operation that is executed by the alternative-service-call operator may not necessarily provide exactly the same output parameters as the original operation. Thus, some fields of the output message \(o'\) that is generated by mapping the new output message \(o\) to the format expected by the process engine may be left blank. This may still be the case even with manual mapping as the new service operation just may not provide some results.

If expected data may not be provided by the alternative operation Web-Flow MaX may check whether this data is needed as input of subsequent activities. Therefore, it may analyze the data flow of the cooperative process instance. If any dependency is detected two solutions are possible:

- Web-Flow MaX tries to derive the missing data from any other data source automatically or with user support.
- If no appropriate data source is found the service provider of the service operation expecting the data may be informed in advance. Then the service provider may either derive the data from any other data source (for instance, with database queries or other service calls) or decide that the operation may be executed without this data. Note, that this procedure is only useful if the complex cooperation model is used and if the affected service allows for manual interaction to provide the data.

An activity that would consume the missing output data may be part of a conditional branching in the unexecuted part of the process. It is not clear during exception handling whether or not the activity will really need the data. Thus, in this case the steps described above to retrieve the data are delayed until this question can be decided.

With these steps it is possible to consider missing data in advance and reduce the number of cases in which missing data may lead to further exceptions. The technical details about analyzing the data flow are omitted here as they strongly depend on the language used for the original cooperative process definitions. The Web-Flow specific process definitions may not be used for this purpose as they do not contain data flow information (cf. Chapter 4).

6.5 Operator procedure-call

In this section we describe the operator procedure-call and discuss which side effects its execution may have on the process instance and the cooperation partners.

6.5.1 Operator Description

The action procedure call tells the Web-Flow MaX dynamic event handling that the event handling is performed outside Web-Flow MaX using existing procedures. For example, these procedures may provide the functionalities for dynamic process adaptation as described in
Dynamic Event and Exception Handling

[MÜLL2002, REIC2000]. Dynamic process adaptation focusses on adapting unexecuted workflow parts to a changed logical situation (for instance, after an exception has occurred) to ensure that the process instance may be successfully continued in the changed situation. As this requires sophisticated algorithms and is only supported by specialized process engines we do not integrate them in Web-Flow MaX but use existing implementations (as described in [GREI2004]). As we assume that the external procedures request user confirmation itself before applying changes to a process instance, the step is omitted for this operator.

Fig. 6-13 shows an example for a WXS description of a procedure call action. It states that for event handling the procedure „ReplaceActivity“ (line 3) should be called for the process instance „xx1289“ (line 6). Furthermore, two input parameters are specified that are required by the external procedure (lines 4 and 5). We assume, that the parameters are specified in the right order in the WXS action description, i.e., in the order that is expected by the procedure, as we do not consider the semantics of the parameters here.

The intention of this procedure call is that all activities of type „activity A“ (for instance, those which call a particular service operation) in the future part of the process instance are replaced by activities of type „activityB“ as it is shown in Fig. 6-14. A possible scenario for this action may be that a manual fault message has been sent to Web-Flow MaX with a notification that a particular service is not available for a certain time. Thus, all activities calling this service are replaced by activities calling an alternative service to prevent violations of response time constraints when the process instance is continued.

The application of the procedure-call operator for this type of procedure is only possible if the process engine supports dynamic process adaptations. Nevertheless, a procedure call action may specify any other external procedure which may be used for dynamic event handling. Thus, it is important that procedure call actions only specify external procedures which match the capabilities of the process engine whose processes are monitored by Web-Flow MaX. However, this may not be checked automatically but is in the responsibility of the administrator specifying the procedures.

1 <action>
2  <procedureCall>
3   <procedure>ReplaceActivity</procedure>
4   <parameter>activityA</parameter>
5   <parameter>activityB</parameter>
6   <processInstanceID>xx1289</processInstanceID>
7  </procedureCall>
8 </action>

Fig. 6-13: WXS description of a procedure call action.
6.5.2 Side Effects of Operator Execution

The execution of the operator procedure-call may change the service operations called in the yet unexecuted part of the cooperative process instance. Initially planned service operations may not be called at all (for instance, Operation 2 of Service 4 in Fig. 6-14) and others may be called additionally (for instance, Operation 3 of Service 5 in Fig. 6-14). The changes depend on the type of the external procedure. If the complex cooperation model is used service providers know in which cooperative processes they participate and may schedule their resources accordingly. This holds in particular for complex, long-running services with manual interaction. To help the cooperation partners to optimize their resource schedules, Web-Flow MaX may send notifications about changes in the process instances to the service providers.

Therefore, the Web-Flow MaX dynamic event handling analyzes the future part of the cooperative process instance after an external procedure has finished execution. Then the necessary notifications are sent. However, if the simple cooperation model is used for a process instance no notifications are necessary, as the service providers do not know that their service is used in this particular process instance.

As the execution of the procedure-call operator may also cause a delay of the activities in the remaining part of the process instance violations of response time or start time constraints in the remaining process are also possible. These effects may be considered in the same way as described in Section 6.4.4.1 above for the operators repeat-service-call, suspend-service-call, and alternative-service-call.
6.6 Operator abort-process

In this section we describe the implementation of the operator *abort-process* and how side effects of the operator execution may be considered.

6.6.1 Operator Description

The operator *abort-process* aborts a whole process instance if it is not reasonable to continue the execution after an exception has occurred. This may be the case if a failed operation is to provide some results that are necessary for the successful execution of the cooperative process instance. For instance, if no suitable supplier could be found in a purchasing scenario even after calling alternative services, the whole purchasing process may be cancelled as it is unlikely that it will finish successfully.

An example for an *abort process* action is shown in Fig. 6-15 (described in WXS). It specifies the name (line 3) and the ID (line 4) of the process instance that should be aborted. If more than one process instance is to be aborted multiple process instances may be specified within one *abort process* action. For instance, if further process instances depend on the successful execution of the first instance, these should also be aborted.

The operator *abort-process* has two possibilities to abort a process instance as illustrated in Fig. 6-16:

a) If the process engine provides an appropriate interface to automatically abort process instances (denoted by the API box in Fig. 6-16), the Web-Flow MaX dynamic event handling component uses this interface and sends an „abort instance“ message. Before the process instance is really aborted, this operator also requests the administrator to confirm the abortion.

b) If no such interface is provided the Web-Flow MaX component blocks all further messages sent to or from a process instance to be aborted (denoted by the crossed out messages and the dotted lines in Fig. 6-16). The messages may be identified with the help of the correlation ID which is assumed to be part of every message sent during process execution (cf. Section

```
1 <action>
2   <abortProcess>
3     <processName>PurchaseProcess</processName>
4     <processInstanceID>xx1782</processInstanceID>
5   </abortProcess>
6 </action>
```

Fig. 6-15: WXS description of an *abort process* action.
5.2.1). Additionally, Web-Flow MaX notifies an administrator about the abortion of the process instance, so that the instance may be aborted manually in the process engine. This prevents pending process instances which arise when no response message is received for a service call made during process execution.

6.6.2 Side Effects of Operator Execution

After the abortion of a process instance service calls to partners contained in the yet unexecuted part of the process are not executed any more. This is no problem if the simple cooperation model is used, as the cooperation partners do not know that their services are used in the cooperative process. Thus, it is unlikely that they have prepared resources for the execution of future service calls and no further actions have to be taken by Web-Flow MaX.

However, in the complex cooperation model the partners have agreed on the cooperative process instance. Thus, they know that their services are involved in process instance execution and may prepare resources for future service operations after their first operation involved in a particular process instance has been called. To be able to optimize resource management and release the resources it would be helpful for the cooperation partners to be informed in time about aborted process instances. Then they may reschedule their resources according to the actual tasks.

This approach is especially useful for long running processes and complex services with manual interaction respectively use of machines that have to be scheduled. To realize this early information, Web-Flow MaX checks if the complex cooperation model has been used for a particular aborted cooperative process instances. Then it retrieves the service providers involved in the unexecuted part and notifies them about the abortion.
6.7 Processing of External Events

External events typically may not be handled directly with the operators described in the last sections. Thus, we have omitted their handling in the basic algorithm in Section 6.1 and discuss it separately. Different kinds of external events can be handled by the Web-Flow MaX dynamic event handling component:

- Events not related to process execution but with effects on event handling, for instance:
  - Web-Flow MaX is notified about changes in company-wide policies such as the list of preferred cooperation partners which is relevant for searching alternative services.

- Events not related to process execution but with effects on constraint monitoring, for example:
  - Web-Flow MaX is notified about data changes relevant for the monitoring of quality constraints, for instance, that the budget of a department has been reduced by 30%. This may have implications for price limits that are monitored with result constraints.

- Events that are not related to a particular process instance but may lead to exceptions in other process instances, for example:
  - Web-Flow MaX is notified about events that may influence the execution of several process instances, such as the sudden unavailability of a service provider.
  - Web-Flow MaX is notified about the delay of a process that is executed as sub-process of a cooperative process instance monitored by Web-Flow MaX. This may have effects on response time and start time constraints defined for the superior process instance.
  - Web-Flow MaX is notified that the execution of some service operations will be delayed due to events having occurred in the cooperative process instance.

To handle external events several mechanisms are used by the Web-Flow MaX dynamic event handling component:

1. If the external events affect quality constraints of currently executed cooperative process instances (as in the first two examples above) the Web-Flow MaX monitoring component is notified about the changes. Then it may update the desired values for the affected quality constraints accordingly. For instance, desired values for price limits may be reduced or a list of service providers may be changed. The update mechanism has been shortly described in Section 5.3.2.3.

2. If the external event notifies Web-Flow MaX about changes that may affect several process instances (third example above), all process instances are checked whether they are affected. For instance, if a service provider is suddenly not able to execute its services anymore (for instance, as he has gone bankrupt or due to a nature catastrophe) all running process instances are checked whether they use services of this partner. Then the operator alternative-service-call may be used to provide alternative service providers for the affected activities. To be able to perform these changes in advance, i.e., before the affected activities are in
fact executed, the operator may be adapted slightly. For instance, no concrete input message for the original service operation is available when the operator is applied in advance.

3. If the external event notifies Web-Flow MaX about the delay of a sub-process this may influence response time and start time constraints for activities in the remaining process part. To check for predictable violations and to enhance the temporal constraints if necessary (i.e., if date values are specified) the same mechanisms as described in Section 6.4.4.1 for the handling of temporal side effects of some operators may be used.

4. If the external event notifies Web-Flow MaX about delays of future service calls (for instance, as described in Section 6.4.4.1) a re-scheduling of resources for the activity executions may be necessary. Therefore external procedures may be used, that may be called automatically either using the operator procedure-call or directly. For direct calls the available procedures may be registered with Web-Flow MaX, for instance, in the metadata repository. As the realization of such re-scheduling algorithms is not in the focus of this thesis we do not discuss this topic further.

With these mechanisms different types of external events may be handled widely automatically. However, the notifications are expected to have a format that is understandable and automatically processable for the Web-Flow MaX dynamic event handling (as discussed for manual fault messages earlier). Therefore, a set of predefined formats for external events are used in Web-Flow MaX which may be enhanced if further event types are to be considered.

6.8 Summary

In this chapter we have described the dynamic event and exception handling component of Web-Flow MaX. This component is used to handle events that are detected by the Web-Flow MaX monitoring component. The component uses a rule-based approach that selects appropriate actions not until an event actually has occurred. In contrast to other approaches using predefined exception handlers contained in the process definition, this allows for considering the current execution context during action selection. The execution of actions ensures that the process instance may be continued despite the exceptional event with the best execution quality possible. Thus, a forward-oriented exception handling is provided that also considers process execution quality.

Web-Flow offers operators that implement the different actions and allow for an efficient execution. The operators are executed automatically after a final user confirmation which ensures that no undesired actions are executed. Administrators are relieved from manual exception handling by the semi-automatic application. The operators also consider side effects of the action execution, for instance, delayed data delivery. Thus, cooperation partners can be informed in time and may actively ensure the execution quality of their processes.

Furthermore, the operators can be applied independently of a concrete process engine and are ap-
Dynamic Event and Exception Handling

Applicable to different types of services. They mainly use the messages sent between process engine and service provider to derive necessary information for exception handling. Their application does not violate the autonomy of the service provider. Thus, they address the special requirements of Web-service-based processes where only limited interaction with service providers is possible and the communication is solely based on messages.
7 Log-based Selection of Actions and Recommendations

To relieve process administrators as much as possible during exception handling appropriate actions should be selected automatically. If only a rule-based approach is used rules have to be defined for all possible events. As rules have to be defined and maintained manually this is quite tedious for the administrator. Furthermore, it is nearly impossible to define rules for all possible events. A promising approach to reduce the manual effort for rule definition is to consider exception handling steps that have been used for former events. Therefore, it is necessary to gather appropriate log data during process execution and exception handling. To derive appropriate actions for a particular event algorithms are needed that determine the similarity between the new event and logged events. Based on these similarity values appropriate actions may be selected. Again these algorithms should work automatically to relieve administrators.

A further possibility to improve process execution quality are regular revisions and optimizations of the process definitions. Optimizations should be based on recommendations that point out weak points of the existing process definitions (for example, service calls that fail often). Such weak points may also be derived from logged exception handling data. Appropriate mechanisms are needed to analyze the log data accordingly. Based on the analysis results also exception handling rules can be derived that then help to improve the rule-based action selection.

The Web-Flow MaX logging and analysis component addresses these requirements. It gathers log data about process instance execution and events which is used to derive appropriate actions for dynamic event handling. The logging component offers mechanisms to calculate the similarity between events. Furthermore, it offers mechanisms for user-driven analysis of log data and generation of recommendations for process optimizations. The analysis results may also be used to derive rules for event handling.

In this chapter we describe the functionality of the Web-Flow MaX logging and analysis component starting with a description of the log data gathered by this component (Section 7.1). Then we focus on the automatic selection of appropriate actions to handle a particular event (Section 7.2). Finally, we discuss the analysis mechanisms and the generation of rules and recommendations (Section 7.3).
7.1 Gathering Log Data

Before appropriate actions may be selected log data is gathered during process execution. As Web-Flow aims at providing a generic solution that is independent of concrete process engines, no engine log data can be used. As the service providers are autonomous and the services are heterogeneous also no log data gathered at the service provider side can be accessed. The appropriate log data has to be derived solely from the event sources that can be observed by Web-Flow MaX, mainly the messages sent between process engine and Web services (cf. Chapter 5).

The log data has to contain information about process instance execution (for instance, which service operations have been called when) as well as about event handling. The data is gathered during process execution by the Web-Flow MaX logging component and is saved in the Web-Flow MaX metadata repository. In this section we describe the structure of log entries, i.e., which parts characterize a log set, and when and how the information for the log entry may be gathered. Furthermore, we describe how activity execution durations which may be used for dynamic event handling may be derived from the log entries.

7.1.1 Structure of Web-Flow Log Entries

The Web-Flow MaX logging component generates a log entry for each service operation that is called during the execution of a cooperative process instance. Log entries for regular executions of service operations (i.e., those for which no event occurs) consist of two parts: operation and process definition, shown in the first two columns of Fig. 7-1. If an event occurs during the execution of the service operation a third part event is added to the log entry (third column in Fig. 7-1). After the dynamic event and exception handling component has handled the event the fourth part is filled with the executed action.

Each of these parts is further structured. A service operation is described by the following values:

- operation name: the operation name as contained in the Web-Flow specific service description.
- porttype: the porttype distinguishes between two service operations of the same service with the same name that may be called over different communication channels (for instance, one operation is accessed using a form on a Web page, the other is accessed by phone).
- service: the name of the service to which the operation belongs.
- provider: the service provider offering and executing the operation.
- activity ID: the ID of the activity calling the service operation as specified in the Web-Flow specific process description.
- start and end time: the points in time when the operation has been called by the process engine and when the response has been returned to the process engine.
- regular: this value denotes whether the action has been called regularly, i.e. during standard process execution, or during event handling. Thus, we log each operation execution during
Gathering Log Data

dynamic event handling in a separate log entry.

The semantic service descriptions which characterize an executed operation may be derived from the Web-Flow specific service descriptions.

The process definition part contains the ID and the name of the process definition as they have been defined in the Web-Flow specific process description. Furthermore, the process instance ID is logged to be able to relate all operations that have been executed in one process instance. The information about the first two parts is sent to the Web-Flow MaX logging component by the monitoring component which intercepts the messages sent between the process engine and the service operations. Thus, it may derive the necessary information for the log entries.

If the Web-Flow MaX monitoring component detects an event it informs the logging component about the event type and the time when the event has occurred. Possible event types are:

- violation of a quality constraint,
- Web service fault message,

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>PROCESS DEFINITION</th>
<th>EVENT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation name</td>
<td>process name</td>
<td>violation of quality constraint</td>
<td>continue process</td>
</tr>
<tr>
<td>porttype</td>
<td>process ID</td>
<td>web service fault message</td>
<td>repeat service call</td>
</tr>
<tr>
<td>service</td>
<td>process instance ID</td>
<td>manual fault message</td>
<td>suspend service call</td>
</tr>
<tr>
<td>provider</td>
<td>operation name</td>
<td>external event</td>
<td>alternative service call</td>
</tr>
<tr>
<td>activity ID</td>
<td>operation functionality</td>
<td>occurred at</td>
<td>abort process</td>
</tr>
<tr>
<td>start time</td>
<td>service category</td>
<td></td>
<td>procedure call</td>
</tr>
<tr>
<td>end time</td>
<td>parameter role</td>
<td></td>
<td>manual reaction</td>
</tr>
<tr>
<td>regular</td>
<td></td>
<td></td>
<td>success</td>
</tr>
</tbody>
</table>

Fig. 7-1: Structure of a Web-Flow log entry.
Log-based Selection of Actions and Recommendations

- manual fault message, or
- external event.

The first three event types are related to the execution of a service operation which has already been logged. Thus, the Web-Flow MaX logging component searches for the according operation log entry with information provided by the monitoring component and adds the event description. However, no operation execution may be assigned to the last event type (external event). Thus, a new log entry is created with the first two parts (operation and process definition) left empty.

The action part is not filled until an appropriate action has been selected and executed by the dynamic event and exception handling component. Then, the type of the selected action is logged (for instance, repeat service call or procedure call) together with an attribute success that provides information whether the action execution was successful or not. Two notions of success are possible:

- success\textsubscript{action}: This is the simplest type of success. It denotes that the action has been applied without any further events occurring during its execution (for instance, a repeat service call action leads directly to a result of the service operation).
- success\textsubscript{process}: This type of success denotes whether any further events have occurred when the process instance was continued after action execution.

The success attribute consists of two boolean values that specify which of the two success parts is fulfilled. The value for success\textsubscript{action} may be set directly after the action has been executed whereas the value for success\textsubscript{process} may not be derived until the process execution has finished. The information about the success of a particular action may, for instance, be used for the derivation of rules as for instance actions that lead to further events during process execution are worse candidates for event handling rules than successful actions.

The information about the executed actions and the success\textsubscript{action} part of the success attribute are provided by the Web-Flow MaX dynamic event handling component. The value of the success\textsubscript{process} attribute is derived by the logging component when it logs the end time of the last operation of a process instance. Therefore, the following steps are executed:

1. All log entries related to operations executed in this process instance are selected.
2. Iterating over the operations of the process instance it is checked for each operation whether an event has occurred for this operation and for any subsequent operations (i.e., operations which are called later in the process):
   a) If the success\textsubscript{action} attribute of the executed action is 'true' and no event has occurred for subsequent operations (i.e., operations which are called later in the process) then success\textsubscript{process} is set to 'true'.
b) If the \(\textit{success}_\text{action}\) attribute of the executed action is 'true' and at least one event has occurred for subsequent operations then \(\textit{success}_\text{process}\) is set to 'false'.

c) If the \(\textit{success}_\text{action}\) attribute of the executed action is 'false' then the log entries for the operation execution during action processing are retrieved and the same checks are performed on these entries.

Note that in step 2b) it may not be determined whether the execution of this particular action caused the subsequent events. However, if a new event has occurred in an instance of the same process definition this may still be useful information to find appropriate actions.

As Web-Flows dynamic event handling ensures that event handling stops after a finite number of iterations (cf. Section 6.1.2) only a finite number of log entries have to be searched in step 2c). Information about the order of the operations in the cooperative process instance may be derived from the Web-Flow specific cooperative process definition.

\section*{7.1.2 Derivation of Execution Durations}

Besides appropriate actions, rules, or recommendations also execution durations of single service operations may be derived from the log data gathered by the Web-Flow MaX logging component. This information may for instance be used by Web-Flows dynamic event handling to consider temporal side effects of action execution on remaining parts of a cooperative process instance (see the operator descriptions in Sections 6.4, 6.5, and 6.6 for more details).

To derive an estimated execution duration \(\textit{dur}_{\text{exec}}\) for a service operation \(o\) from the log data the following steps are executed:

1. First all log entries \(l_{o,i}\) for already completed executions of the service operation \(o\) are selected.

2. Then the estimated execution duration \(\textit{dur}_{\text{exec}}\) is calculated as the average value of all \(n\) execution durations measured for \(o\) using the following formula:

\[
\textit{dur}_{\text{exec}} = \frac{\sum_{i=1}^{n} (l_{o,i,\text{endtime}} - l_{o,i,\text{starttime}})}{n}
\]

where \(n = \#(l_{o,i})\)

We use the average value for the calculation of the estimated execution duration as we expect to get values that meet the real values in most cases with the least deviations. The use of a maximum or a minimum function would stress extreme values too much which have possibly only occurred one or two times. As the estimated execution durations are used to determine side effects on a process instance which imply a reaction by the Web-Flow MaX component (for instance, a user notification) the use of maximal or minimal values may lead to too many...
or too little reactions. A similar discussion for the use of average execution durations in the context of workflow estimation may be found in [MÜLL2004].

To allow for an efficient execution of the exception handling operators, the execution durations are pre-calculated periodically, for example, every night. The calculated values are stored in the Web-Flow MaX metadata repository and retrieved when the dynamic event handling component asks for the duration value of a particular operation. When the Web-Flow MaX component is starting to work, the execution durations may be re-calculated in shorter time periods to get good estimated execution durations as fast as possible.

7.2 Selection of Actions for Event Handling

Based on the log data gathered during process instance execution and event handling appropriate actions for the handling of new events may be derived automatically. Therefore, the similarity between the log entry describing the new event and all other log entries is calculated. As the log entry describing the new event does not have an action part, only the operation part, the event part, and the process definition part are considered. An appropriate action is then derived from the log entries with the greatest similarity to the new log entry. In the following we describe how the similarity between two log entries is calculated and how an appropriate action is selected based on these similarities. At the end of this section we discuss some optimizations of the similarity calculation to provide a better performance of action selection.

7.2.1 Calculation of Similarity

To calculate the similarity between a new log entry and an existing log entry first the similarities between the different parts, event, operation, and process definition, are calculated using the functions

\[ ds_{\text{event}}(e_1, e_2), ds_{\text{op}}(op_1, op_2), \text{ and } ds_{\text{process}}(p_1, p_2). \]

Then these values are aggregated to the value \( ds \) representing the overall similarity between the two log entries. Fig. 7-2 illustrates this process.

All functions used during the calculation of \( ds \) return normalized values for the description of the similarity between two log entry parts respectively two log entries. That means, the values are taken from the interval \([0,1]\) with the following meaning:

\[ ds_i = 0 \] means that the attributes respectively entries are not similar at all.

\[ ds_i = 1 \] means that the attributes respectively entries are identical.
Selection of Actions for Event Handling

In the subsequent sections we define the different functions \( ds_{\text{event}}(e_1, e_2) \), \( ds_{\text{op}}(op_1, op_2) \), and \( ds_{\text{process}}(p_1, p_2) \), and \( ds \). We start with \( ds_{\text{process}}(p_1, p_2) \) as this is the most simple one and continue with \( ds_{\text{event}}(e_1, e_2) \) and \( ds_{\text{op}}(op_1, op_2) \) which is the most complex one.

**7.2.1.1 Similarity of Process Definition Parts**

To derive the similarity of two process definition parts, we compare the three attributes process name, process ID, and process instance ID. As process identifiers respectively process instance identifiers are unique these values can only be the same or differ from each other. Thus, we may omit the comparison of the process name attribute and calculate the similarity between two process definition attributes \( p_1 \) and \( p_2 \) using the function \( ds_{\text{process}}(p_1, p_2) \) which is defined as follows:

\[
\begin{align*}
    ds_{\text{process}}(p_1, p_2) &= \begin{cases} 
        1 & \text{if } p_1\.processID = p_2\.processID \text{ AND } p_1\.processInstanceID = p_2\.processInstanceID \\
        0.9 & \text{if } p_1\.processID = p_2\.processID \text{ AND } p_1\.processInstanceID \neq p_2\.processInstanceID \\
        0 & \text{if } p_1\.processID \neq p_2\.processID \text{ AND } p_1\.processInstanceID \neq p_2\.processInstanceID 
    \end{cases}
\end{align*}
\]

Fig. 7-2: Calculation of similarity between a new and an existing log entry.
If the process ID and the process instance ID are the same, the two process definition parts are identical and their similarity is set to 1. If the process ID is the same and the process instance ID differs then the log entries belong to two process instances of the same process definition. Thus, the two parts are considered quite similar expressed in a similarity value of 90%. (Note that this is an initial value which may be adapted after some evaluations of the selection algorithm.)

If neither process ID nor process instance ID are the same or if for one or both of the two log entries no process definition part is provided (as the log entry describes the occurrence of an external event), then the process definition parts are not regarded similar and their similarity is set to 0.

If the process ID is different and the process instance ID is the same one of the two log entries is erroneous as this would either mean that one process instance belongs to two different process definitions or that two process instances have the same ID. Both cases should be anticipated by the Web-Flow MaX component. If this case occurs yet the existing log entry is skipped and a user notification is generated to get a resolution for this conflict.

7.2.1.2 Similarity of Event Parts

To derive whether the event parts of two log entries are similar we consider the detailed type and structure of the event. In the Web-Flow metamodel four different event types are defined:

- violation of quality constraints,
- Web service fault messages,
- manual fault messages, and
- external events.

The events are further structured and contain for instance the quality constraint that has been violated or the fault message (cf. Section 4.3). Therefore, we consider the detailed structure of the events for the calculation of the similarity between the event parts of two log entries. To derive the similarity between two event parts $e_1$ and $e_2$ we use the matrix shown in Tab. 7-1. The time of occurrence („occurred at“ attribute in Fig. 7-1) is not relevant for the similarity of two events. Thus, the similarity is calculated using the function $ds_{\text{event}}(e_1, e_2)$ which is defined as follows:

$$ds_{\text{event}}(e_1, e_2) = \text{value-at}(e_1.\text{type}, e_2.\text{type})$$

The matrix contains all possible combinations of event types. Violations of quality constraints and manual fault messages have been split up in their sub-types depending on the violated constraint or the particular message format used. We have not further divided Web service fault messages as every service provider may use its own Web service fault message types. Thus, it is impossible to consider all of them in a matrix as the one used here. External events are dissimilar to other events anyway as they are not related to the execution of a service operation. Thus, we also do not further consider their subtypes in the overall matrix.

However, as Web-Flow uses predefined message formats for external events they may be com-
pared to each other similar as for manual fault messages. Therefore, a similar matrix may be used as the one shown in Tab. 7-1 only considering the sub-types of external events. If both log entries have an event part of type external event the second matrix is used to derive their similarity. As this may be done analogously to the other event types, we do not discuss this in more detail here.

The following list gives some explanations for the similarity values we have chosen for the matrix for the calculation of the function $d_{\text{even}}(e_1, e_2)$:

- **Response time and start time constraints as well as input and result constraints are quite similar to each other.** Response time and start time constraints both relate to temporal execution characteristics. Input and result constraints both check conditions on service messages. Thus their similarity is set to 0.9.

- **An external event may not be considered similar to any other event type as it is not related to the execution of a service operation in a cooperative process instance, so the similarity is set to 0.1.** An exception is the similarity to a manual fault message notifying the Web-Flow MaX component about an arbitrary other event which may also be an external event (0.2).

- **Row 1:** Start time constraints are more similar to input constraints than to result constraints as the first two are both checked when the service operation is called. As iteration constraints and start time constraint are both constraints on execution characteristics they are more similar to each other (similarity 0.7) than to input (0.6) respectively result constraints (similarity 0.5). As Web service fault and manual fault messages typically notify the service user after a service has been called they are considered quite dissimilar to start time constraints.

- **Row 2:** Accordingly response time constraints are more similar to result constraints (0.7) than to input (0.5) or iteration (0.6) constraints. The similarity to iteration constraints is also determined by the fact that response time and iteration constraints are both execution constraints. As violations of response time constraints typically may be related to a failed service call or a problem at the service provider they are considered quite similar to Web service fault messages and most types of manual fault messages.

- **Row 3:** As discussed for start time constraints iteration constraints are more similar to input than to result constraints. They are regarded quite dissimilar to Web service fault messages and manual fault messages.

- **Row 4:** An input constraint is considered more similar to a Web service fault message and a manual fault message notifying about a failed service execution than to other manual fault messages as the service execution may fail due to erroneous input data.

- **Row 5:** The same holds for result constraints. However, as they are checked after a service operation has finished they are more similar to Web service fault messages than input constraints (0.5 instead of 0.4 in row 4).

- **Row 6:** Metadata constraints are mainly used for service selection and not during process execution, so they are considered dissimilar to all other event types (similarity 0.1).

- **Row 7+8:** Web service fault messages are quite similar to manual fault messages notifying about a service failure (0.8) but less similar to other manual fault messages as these typically
are sent before the service execution finishes. Web service fault messages are sent after a service has finished execution. The same holds for the similarity of a manual notification about a failed service execution and other manual fault messages.

- **Row 9+10:** A manual notification about a service delay is considered quite similar to a manual notification that a provider is out of Business as both notifications will result in a service delay.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Violation of QC startTimeConstraint</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Violation of QC responseTimeConstraint</td>
<td>1.0</td>
<td>0.6</td>
<td>0.5</td>
<td>0.7</td>
<td>0.1</td>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td>0.8</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Violation of QC iterationConstraint</td>
<td>1.0</td>
<td>0.6</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Violation of QC resultConstraint</td>
<td>1.0</td>
<td>0.9</td>
<td>0.1</td>
<td>0.4</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Violation of QC metaDataConstraint</td>
<td>1.0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>WebServiceFault</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>ManualFault serviceFailed</td>
<td>1.0</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>ManualFault serviceDelay</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>ManualFault providerOutOfBusiness</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>ManualFault otherEvent</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>ExternalEvent</td>
<td>1.0</td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Table 7-1:** Similarity matrix for comparison of event types.
(QC stands for quality constraint. The fields below the diagonal are grey as the two parts of the matrix are symmetric. Thus the values have been omitted for clarity of the presentation.)
Selection of Actions for Event Handling

delay. The similarity to any other manual notification is only 0.4 just as for a manual notification that a provider is out of business.

Note, that the values shown in Tab. 7-1 are only intended to be used as start values for the Web-Flow MaX component. After a significant amount of log data is gathered the values may be refined using analysis results provided by the Web-Flow MaX logging component. For instance, the success of automatically selected actions may be an indicator how good some of these values are. If an action is selected quite often due to a high event similarity but its application results in subsequent events in the same process instance the similarity value may be reduced to enforce that different actions are selected. Another approach to improve the similarity values could be the applications of machine learning algorithms. However, this is out of scope of this thesis.

7.2.1.3 Similarity of Operation Parts

For the calculation of the similarity between two operation attributes $op_1$ and $op_2$ we support two strategies: syntactic comparison vs. semantic comparison. Syntactic comparison means that we compare the operation name, the porttype, the service name, and the provider of $op_1$ and $op_2$ and derive whether they correspond. The starttime, endtime, and activity ID of an operation execution are not considered here, also the attribute denoting whether the service operation has been called regularly or during event handling is omitted.

The second comparison strategy, semantic comparison, considers the functionality of the operation, the category of the service, and the roles of the parameters to derive the similarity between $op_1$ and $op_2$. Thus, we may still find some similar log entries for an operation attribute although no log entry with a directly matching operation exists, for instance, as the operation names are different. However, semantic comparison is only possible if semantic service descriptions exist for a particular operation.

To derive as much information as possible from the log entries we always calculate both similarity values for two operations $op_1$ and $op_2$, the syntactic similarity $ds_{op,syn}$ and the semantic similarity $ds_{op,sem}$. The functions to calculate these values are defined in Sections 7.2.1.4 and 7.2.1.5. Based on these two results the operation similarity is then derived using the function $ds_{op}(op_1, op_2)$ which is defined as follows:

$$ds_{op}(op_1, op_2) = \begin{cases} \frac{w_{op,1} \times ds_{op,syn} + w_{op,2} \times ds_{op,sem}}{\sum_{i=1}^{2} w_{op,i}} \text{ with } w_{op,i} \in [0,1] \\ 0 \quad \text{if } op_1 = \text{null OR } op_2 = \text{null} \end{cases}$$
Log-based Selection of Actions and Recommendations

The weights $w_{op,i}$ (i=1,2) control the impact of the syntactic and the semantic similarity on the overall operation similarity. The default values of the weights are 1, however they are automatically or manually configurable. For instance, if the semantic similarity cannot be calculated as the semantic service description is not available $w_{op,1}$ is set to 1 and $w_{op,2}$ is set to zero automatically. Thus, the operation similarity is equal to the syntactic similarity.

If the semantic similarity is to have more impact, the weight $w_{op,2}$ may be adjusted manually (for instance, using a configuration parameter of the Web-Flow MaX logging and analysis component) with a higher value. This step may be useful if often new services and process definitions are integrated in the Web-Flow MaX metadata repository and sufficient semantic service descriptions are available. In contrast to that the syntactic similarity may get more impact if only few semantic service descriptions are saved in the metadata repository.

If one or both of the two log entries has an empty operation attribute as it logs an external event the similarity value for the operation attributes is set to zero.

7.2.1.4 Syntactic Comparison of Operation Parts

For the syntactic comparison of two operation parts $op_1$ and $op_2$ the operation name, the porttype, and the service name (including the target namespace), are successively compared using appropriate functions and then the derived similarity values $ds_{op-name}$, $ds_{porttype}$, and $ds_{service-name}$ are aggregated to the similarity value $ds_{op,syn}$. As the service name includes the target namespace of the service, which implicitly contains the provider (for instance, a company’s URL), we do not compare the provider name separately.

As the names for the considered operation characteristics may be arbitrarily chosen by a service provider, in this step we may only determine whether two values correspond to each other or are different. More sophisticated functions to derive the similarity between two names as, for instance, described in [DONG2004] are not utilized here but may be subject of future enhancements of the Web-Flow MaX component.

All operation characteristics relevant for the calculation of $ds_{op,syn}$ are represented as strings. So, we use the comparison operators $=$ and $\neq$ denoting string comparisons which check whether the two strings are equal or differ also considering upper and lower case. In particular, the syntactic similarity is calculated as follows:
Selection of Actions for Event Handling

The second case in the function definition ensures that two operations whose names do not correspond are not considered as syntactically similar. As the same operation name is considered the main indicator for the syntactic similarity of two operations, this step is reasonable.

With the weights $w_{op,d,i} (i=1,2,3)$ the administrator may control the impact of the single operation characteristics on the overall syntactic operation similarity. For instance, if the porttype is considered less important than the operation and the service name $w_{op,d,2}$ may be reduced. Therefore, the default value 1 for the weights may be adjusted.

The single similarity values for the syntactic comparison of operation name, porttype and service name, are calculated using the following functions:

$$ds_{op,syn} = \begin{cases} \frac{w_{op,d,1} \times ds_{op-name} + w_{op,d,2} \times ds_{porttype} + w_{op,d,3} \times ds_{service-name}}{\sum_{i=1}^{3} w_{op,d,i}} & \text{with } w_{op,d,i} \in [0,1] \\ 0 & \text{if } ds_{op-name} = 0 \end{cases}$$

The second case in the function definition ensures that two operations whose names do not correspond are not considered as syntactically similar. As the same operation name is considered the main indicator for the syntactic similarity of two operations, this step is reasonable.

With the weights $w_{op,d,i} (i=1,2,3)$ the administrator may control the impact of the single operation characteristics on the overall syntactic operation similarity. For instance, if the porttype is considered less important than the operation and the service name $w_{op,d,2}$ may be reduced. Therefore, the default value 1 for the weights may be adjusted.

The single similarity values for the syntactic comparison of operation name, porttype and service name, are calculated using the following functions:

$$ds_{op-name} = \begin{cases} 1 & \text{if } op_1.operation-name = op_2.operation-name \\ 0 & \text{if } op_1.operation-name \neq op_2.operation-name \end{cases}$$

$$ds_{porttype} = \begin{cases} 1 & \text{if } op_1.porttype = op_2.porttype \\ 0 & \text{if } op_1.porttype \neq op_2.porttype \end{cases}$$

$$ds_{service-name} = \begin{cases} 1 & \text{if } op_1.service-name = op_2.service-name \\ 0 & \text{if } op_1.service-name \neq op_2.service-name \end{cases}$$
Log-based Selection of Actions and Recommendations

Tab. 7-2 shows the possible combinations of similarity values for the operation characteristics and the values of $d_{op, syn}$ if the weights $w_{op,d,i}$ ($i=1,2,3$) are set to 1. Line 3 represents the case that a Web service may have several porttypes grouping operations for the access over different channels or protocols (see [WSDL]). Thus, two operations $op_1$ and $op_2$ are considered quite similar if they belong to different porttypes of the same service (similarity of 2/3). In line 4 we assume that service providers choose similar names for similar operations and porttypes in different service. So, if operation name and porttype correspond for $op_1$ and $op_2$ we also consider $op_1$ and $op_2$ quite similar (similarity of 2/3). However, if only the operation name corresponds and neither the porttype nor the service name then the similarity is only 1/3.

7.2.1.5 Semantic Comparison of Operation Parts

With the semantic comparison strategy the semantic similarity $d_{op,sem}$ between two operations $op_1$ and $op_2$ is calculated. Semantic comparison considers the operation functionality, the service category, and the parameter roles as defined in Section 4.1.2. Analogously to syntactic comparison also first the similarity values between two semantic descriptions of

<table>
<thead>
<tr>
<th></th>
<th>$d_{op-name}$</th>
<th>$d_{porttype}$</th>
<th>$d_{service-name}$</th>
<th>$d_{op,syn}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2/3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7-2: Values of $d_{op,syn}$

The values have been calculated with syntactic comparison for all possible single similarities of operation characteristics. The weights $w_{op,d,i}$ are set to 1.
Selection of Actions for Event Handling

a service, namely \( ds_{\text{func}} \), \( ds_{\text{cat}} \), \( ds_{\text{param-role}} \), are calculated and then aggregated to \( ds_{\text{op,sem}} \). As semantic descriptions are not only strings but have a more complex structure we start with a common description how the similarity between two semantic description values may be derived before we define the functions for the calculation of \( ds_{\text{op,sem}} \).

**Similarity Between Semantic Descriptions**

The semantic descriptions of an operation are represented as tuples of the form:

\[(\text{value}, \text{taxonomy}, \text{synonyms})\]

where \( \text{value} \) is a string representing the actual value of a characteristic, \( \text{taxonomy} \) is a string with the URI (Uniform Resource Identifier) of the taxonomy from which \( \text{value} \) has been taken and \( \text{synonyms} \) defines a set of synonymous values each of which is a tuple \((\text{value}, \text{taxonomy})\). The first two tuple elements together are also referred to as the main-value of the semantic description in the following.

An example for a semantic description of the service category of a monitor seller service is given in example (ii). Thereby, 43211900 denotes the top level category of computer displays whereas the categories 43211901 to 43211905 denotes different types of displays (for instance, LCD displays). The category 43211600 stands for computer accessories which may also comprise computer displays, 43211500 denotes the category computer which may be seen as a superior term of computer displays. All category values are taken from the UNSPSC taxonomy.

\[(43211900, \text{http://www.unspsc.org}),\]

\[\((43211901, \text{http://www.unspsc.org}),\]

\[\(43211902, \text{http://www.unspsc.org}\),\]

\[\(43211903, \text{http://www.unspsc.org}\),\]

\[\(43211904, \text{http://www.unspsc.org}\),\]

\[\(43211905, \text{http://www.unspsc.org}\),\]

\[\(43211600, \text{http://www.unspsc.org}\),\]

\[\(43211500, \text{http://www.unspsc.org}\))\]

For the calculation of the similarity between two semantic descriptions it is first checked whether the main-values, i.e., the tuples \((\text{value}, \text{taxonomy})\), correspond. If this is not the case, further comparisons are executed considering the synonyms specified for each semantic value (details are given below). As the similarity between a main-value and its synonyms is not always 100% an additional table is kept in the Web-Flow MaX metadata repository which contains similarities between the main semantic description and its synonyms. An extract of this table is given in Tab. 7-3 containing the similarity values for the example (ii).
In the first version of the Web-Flow MaX component the similarities between main-values and synonyms are specified manually. That means if an administrator assigns a semantic description with synonyms it is checked for each pair \((\text{main-value}, \text{synonym}_i)\) whether a similarity value already exists. If no value is found for a particular pair, the administrator is asked to specify a similarity value for this pair. The specified values are bi-directional, i.e., one value applies independently whether \(\text{value}_1\) is the main-value and \(\text{value}_2\) the synonym or vice versa. As a future enhancement the synonym similarities may be derive automatically, for instance, by evaluating the hierarchy of the values in the taxonomy. That means that, for instance, two values which are children of the same parent are considered more similar than two values with a direct parent-child relationship. Algorithms for such calculations may be found in the literature (for instance, [BERG1999, CARD2003, DO2002, PALO2003]) and are not further discussed here.

If the main-values of two semantic description values of \(op_1\) and \(op_2\) differ the lists of synonyms are used to calculate the similarity for these semantic characteristic. Therefore, we provide the function

\[ ds_{\text{synon}}(\text{sem}_1, \text{sem}_2) \]

which takes two semantic description values \(\text{sem}_1\) and \(\text{sem}_2\) of the form \((i)\) for a particular semantic characteristic as input and calculates the similarity between these values using the synonyms of each. Therefore, we first compare the main-values of each semantic description against the synonyms of the other one:

<table>
<thead>
<tr>
<th>RowID</th>
<th>value(_1)</th>
<th>value(_2)</th>
<th>similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211901, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211902, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211903, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211904, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.8</td>
</tr>
<tr>
<td>5</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211905, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.8</td>
</tr>
<tr>
<td>6</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211600, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>(43211900, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>(43211500, <a href="http://www.unspsc.org">http://www.unspsc.org</a>)</td>
<td>0.3</td>
</tr>
</tbody>
</table>

*Table 7-3:* Extract of similarity table for comparison of semantic description values and their synonyms.
Selection of Actions for Event Handling

1. First, the main-value of $sem_1$ is checked against each synonym of $sem_2$. If a matching value is found, i.e. the main-value of $sem_1$ corresponds to synonym $i$ of $sem_2$, the search stops and the function is assigned the similarity value of the pair $(sem_2\text{-main-value}, sem_2\text{-synonym}_i)$. The value is derived from the similarity table:

$$ds_{synon}(sem_1, sem_2) = \text{similarity-value-of}(value_1 = sem_2\text{-main-value},$$

$$value_2 = sem_2\text{-synonym}_i)$$

To ensure that the value is found a second query may be used with inverted values for $value_1$ and $value_2$.

2. If no result has been found in step 1 the main-value of $sem_2$ is checked against each synonym of $sem_1$. If a matching value is found, i.e., the main-value of $sem_2$ corresponds to synonym $i$ of $sem_1$, the search stops and the function is assigned the similarity value of the pair $(sem_1\text{-main-value}, sem_1\text{-synonym}_i)$. The value is retrieved from the similarity table.

If still no result has been found after step 2 in a third step the synonyms of both operations are compared to each other. Therefore, each synonym $s_i$ of $sem_1$ is compared to each synonym $s_j$ of $sem_2$. If a matching pair of synonyms $(s_i, s_j)$ is found the similarity value is determined as the minimum of the similarity values between each synonym value and its main-value:

$$ds_{synon}(sem_1, sem_2) = \min\{\text{similarity-value-of}(value_1 = sem_1\text{-main-value}, value_2 = sem_1\text{-s}_i),$$

$$\text{similarity-value-of}(value_1 = sem_2\text{-main-value}, value_2 = sem_2\text{-s}_j)\}$$

Fig. 7-3 illustrates these steps. We use the minimum function as it matches best with the semantics that the synonym is only similar to the main-value to a certain degree. Another possibility would be the use of the average or the maximum function but in both cases the smaller similarity would be raised which may falsify the similarities intended by the administrators. However, the selection of the function is configurable so an administrator may decide to use average or maximum instead of the minimum.

If no result has been found after the last step the similarity between the two semantic descriptions is set to zero, i.e.

$$ds_{synon}(sem_1, sem_2) = 0,$$

as then no semantic similarity exists.

Calculation of Semantic Similarity

In the following, we describe how the semantic similarity of two operations is calculated using the semantic descriptions of the form as shown in the examples (i) and (ii). Thereby, the comparison
operators $\text{=} \text{ and } \neq$ again denote string comparisons which check whether two strings are equal or differ also considering upper and lower case. The semantic similarity between two operations is calculated using a weighted average function as follows:

$$d_{\text{op,sem}} = \frac{w_{\text{op,s,1}} \times d_{\text{funct}} + w_{\text{op,s,2}} \times d_{\text{cat}} + w_{\text{op,s,3}} \times d_{\text{param-role}}}{\sum_{i=1}^{3} w_{\text{op,s,i}}}$$

with $w_{\text{op,s,i}} \in [0, 1]$

The default values for the weights $w_{\text{op,s,i}}$ (i=1,2,3) are 1. However, again the weights may be adjusted if for instance, the similarity of the operation functionality and the service category are to have more impact on the semantic similarity of two log entries than the similarity of the parameter roles.

The similarity value $d_{\text{funct}}$ is calculated using the following function:
Selection of Actions for Event Handling

If both operations provide a value for the operation functionality and these values correspond in their main-values the similarity is set to 1. If at least one operation provides no value for the functionality the similarity is set to 0 as no comparison is possible. If both operations provide a semantic value but the main-values differ the synonyms are used to determine the similarity between the operations using the function \( ds_{\text{synon}}(\text{sem}_1, \text{sem}_2) \) introduced above.

For the service category a similar function is used:

\[
\begin{align*}
ds_{\text{cat}} & = \begin{cases} 
1 & \text{if } \text{op}_1.\text{functionality.value} = \text{op}_2.\text{functionality.value} \text{ AND } \text{op}_1.\text{functionality.taxonomy} = \text{op}_2.\text{functionality.taxonomy} \\
0 & \text{if } \text{op}_1.\text{functionality} = \text{null OR op}_2.\text{functionality} = \text{null} \\
\text{ds}_{\text{synon}}(\text{op}_1.\text{functionality}, \text{op}_2.\text{functionality}) & \text{otherwise}
\end{cases}
\end{align*}
\]

For the calculation of the semantic similarity between the parameters of \( \text{op}_1 \) and \( \text{op}_2 \) we have to consider that operations may have three different parameter types, input, output, and fault parameters, each of which may have its own semantic description. Thus, the similarity values for each type are calculated separately if semantic descriptions are provided for their roles. If one of the operations does not provide any semantic descriptions parameter roles, the overall similarity is set to 0. Otherwise it is calculated as the weighted average of the similarity values for the different parameter types:
Log-based Selection of Actions and Recommendations

The default values for the weights \( w_{p,i} \) (i=1,2,3) are 1. However, the weights may be adjusted if for instance, the similarity of fault parameters is to have less impact on the similarity of operation parameters than input or output parameters.

The semantic similarity values for the input and output parameters, \( ds_{input-param} \) and \( ds_{output-param} \), are calculated in the same way as described for functionality and category. If a semantic description is missing in at least one parameter (for instance, as an operation provides no output) then the corresponding similarity value for this parameter type is set to 0 as a comparison is not possible. We only present the function for the input parameters here, the function for output parameters is analogous:

\[
ds_{param-role} = \begin{cases} 
\frac{w_{p,1} \times ds_{input-param} + w_{p,2} \times ds_{output-param} + w_{p,3} \times ds_{fault-param}}{\sum_{i=1}^{3} w_{p,i}} & \text{with } w_{p,i} \in [0,1] \\
0 & \text{if } op_1.param-role = \text{null for all parameters of } op_1 \text{ OR } op_2.param-role = \text{null for all parameters of } op_2
\end{cases}
\]

As service operations may have several fault parameters the calculation of \( ds_{fault-param} \) becomes a bit more complex. We apply an each-to-each comparison of all fault parameters of both operations and combine the values to one value using the average. That means, that we calculate the similarity between all pairs of fault parameters using the same function as defined for input parameters. The overall similarity value of the fault parameters is calculated with the following function:

\[
ds_{input-param} = \begin{cases} 
1 & \text{if } op_1.input.role.value = op_2.input.role.value \text{ AND } op_1.input.role.taxonomy = op_2.input.role.taxonomy \\
0 & \text{if } op_1.input.role = \text{null OR } op_2.input.role = \text{null} \\
ds_{synon}(op_1.input.role, op_2.input.role) & \text{otherwise}
\end{cases}
\]
Selection of Actions for Event Handling

\[ ds_{\text{fault-param}} = \frac{\sum_{i=1}^{n} ds_{\text{fault-param},i}}{n} \]

with \( ds_{\text{fault-param},i} \) represents the similarity of one pair of fault parameters and \( n = \#(\text{fault-params of op}_1) \times \#(\text{fault-params of op}_2) \) denotes the number of all pairs which could be made up from the fault params with semantic description of \( \text{op}_1 \) and \( \text{op}_2 \).

7.2.1.6 Aggregation of Similarity Values

After the similarity values for the single parts of two log entries, event, process and operation have been calculated the overall similarity \( ds \) between the two log entries is derived. The functions presented in the preceding Sections 7.2.1.1 to 7.2.1.5 ensure that the single similarity values \( ds_{\text{event}} \), \( ds_{\text{op}} \), and \( ds_{\text{process}} \) are normalized, i.e., in the interval \([0,1]\). Thus, we aggregate the single similarity values to one value \( ds \) using the weighted average:

\[ ds = \frac{w_1 \times ds_{\text{event}} + w_2 \times ds_{\text{op}} + w_3 \times ds_{\text{process}}}{\sum_{i=1}^{3} w_i} \quad \text{with } w_i \in [0,1] \]  

(iii)

The default values for the weights \( w_i \) (i=1,2,3) are 1, thus giving every part the same impact on the overall similarity. However, the weights may also be adjusted. Thus, the event and the operation part may get more impact on the overall similarity than the process part.

Furthermore, if one or both log entries represent an event of type \text{external event} the formula (iii) is adapted automatically by the Web-Flow MaX logging component to prevent that the similarity value \( ds \) for the two log entries is reduced unnecessarily. As described earlier in this chapter the similarities for the operation part and the process definition part are set to zero for this event type. If we would use the formula (iii) with the default values for the weights, we would get the following result:

\[ ds = \frac{1 \times ds_{\text{event}} + 1 \times 0 + 1 \times 0}{3} = \frac{1}{3} \times ds_{\text{event}} \]

That means that the event similarity which is the only one different from zero would be reduced to one third. This leads to a significant reduction of the overall similarity which may lead to a bad ranking of this particular log entry (cf. Section 7.2.2). Thus, we adapt formula (iii) as follows if at least one of the two log entries logs an event of type \text{external event}:
Log-based Selection of Actions and Recommendations

\[
d_s = \frac{w_1 \times d_{\text{event}} + 0 \times d_{\text{op}} + 0 \times d_{\text{process}}}{w_1} = d_{\text{event}}
\]  

(iv)

We have performed some evaluations to determine good values for the different weights
- \(w_i\) (for the overall similarity),
- \(w_{op,i}\) (for the similarity of operations),
- \(w_{op,d,i}\) (for the syntactic comparison of operations),
- \(w_{p,i}\) (for the similarity of parameters), and
- \(w_{op,s,i}\) (for the semantic comparison of operations)

used during the calculation of the similarity between two log entries. The results are presented in Chapter 8.

7.2.2 Ranking of Actions

The similarity between a new log entry and the existing log entries whose calculation has been described in the last section gives a first hint which actions may be more suitable to handle the new event than others. Therefore, the set of all log entries may be ordered according to the similarity values starting with the entry with the highest similarity to the new log entry:

<table>
<thead>
<tr>
<th>log-entry</th>
<th>similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.93</td>
</tr>
<tr>
<td>2</td>
<td>0.93</td>
</tr>
<tr>
<td>3</td>
<td>0.87</td>
</tr>
<tr>
<td>4</td>
<td>0.7</td>
</tr>
<tr>
<td>5</td>
<td>0.65</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

As the Web-Flow MaX component aims at relieving the administrator as much as possible from event handling, the goal of the automatic action selection is to find exactly one action that is appropriate for handling the current event. Therefore, different steps are used to select those actions which are most appropriate for handling the new event. The steps are described in the following.

**Similarity Threshold**

A first step to reduce the set of log entries is to consider only those that are most similar to the new log entry. Therefore a similarity threshold, \(sim_{\text{threshold}}\), is used which is again manually configurable. For instance, if only few log entries are available \(sim_{\text{threshold}}\) may be assigned a lower val-
ue to get a significant number of similar log entries. If many log entries are available $sim_{threshold}$ may be increased to reduce the number of appropriate log entries. In the remainder of this chapter we assume a similarity threshold of 85\%, i.e.

$$sim_{threshold} = 0.85.$$  

This value may also be seen as a default value for the Web-Flow MaX logging and analysis component.

If we apply the similarity threshold to the list (v) we get a list which only consists of three entries:

- log-entry$\,_{1}$ | similarity = 0.93
- log-entry$\,_{2}$ | similarity = 0.93
- log-entry$\,_{3}$ | similarity = 0.87

This list illustrates one of three cases which may occur after the similarity threshold has been applied. In particular, we distinguish the following cases:

- exactly one action has been found,
- no action has been found,
- several actions have been found.

In the first case the goal of the action selection to find exactly one appropriate action is reached. Thus, this action is returned to the Web-Flow MaX dynamic event handling component which executes the action (cf. Chapter 6).

The second case that no action has been found may occur especially at the beginning of the use of the Web-Flow MaX component when only few log entries are available. This case may also occur if completely new process definitions with new services and service providers are monitored by the Web-Flow MaX component. Then, no comparable log entries are available. In this case a manual reaction action is derived as appropriate action and is returned to the Web-Flow MaX dynamic event handling component. During the execution of the manual reaction action the administrator may choose another automatically executable action which is then logged as an appropriate action for this type of event. Thus, next time a similar event occurs an appropriate action may be found automatically.

If several log entries have a sufficiently high similarity to the new event (the third case) Web-Flow tries to rank these actions further considering different criteria. This helps the administrator to decide which action is most appropriate. The ranking criteria is described in the following.

### Ranking Criteria

The Web-Flow MaX logging component uses different criteria to rank the log entries in the reduced set (vii) which remained after the application of $sim_{threshold}$. In the following we describe these criteria before we discuss their relevance for the possible ranking strategies:
1. **Similarity**: The first criterion is the similarity between the new event and the existing log entries.

2. **Success**: The second criterion checks whether the action has been executed successfully or whether further events have occurred. Therefore, the $success_{\text{action}}$ part of the success attribute of the action part of a log entry is checked. If available also the $success_{\text{process}}$ part of the success attribute may be considered. The consideration behind this criterion is that it is better to select an action that has been executed successfully than one during whose execution a new event has occurred. As possibly a log entry may not only contain one action but a list of actions that have been executed to handle an event, the success attributes of the different actions are aggregated. The aggregated values for $success_{\text{action}}$ and $success_{\text{process}}$ represent the success values for the whole log entry. The aggregation is calculated by concatenating the single values using a boolean AND. Thus, the aggregated value is only set to „true“ if every single value is true. In the following we do not distinguish between aggregated success values and single success values as it is irrelevant for the algorithms.

3. **Time of occurrence**: This criterion assumes that typically more recent actions are more appropriate to handle an event than older ones as then the execution conditions may have been different. The time of occurrence may be derived from the „occurred at“ attribute of the event part of a log entry which is present in every log entry.

4. **Priority**: Each action may have a priority which is assigned manually, for instance, when an administrator is notified about several possible actions or when an action is confirmed before execution (cf. Chapter 6). The log entries are updated with the priority value which ranges from 1 (highest priority) to 10 (lowest priority). However, in contrast to the „success“ attribute and the „occurred at“ attributes we may not assume that an action part of a log entry always has a priority attribute assigned. For the priority attribute a similar problem as for the success attribute arises. If a log entry contains a list of attributes each of this may have a priority. For the ranking we need one value representing the overall priority of the whole log entry. Thus, we use the maximum priority of all actions contained in the action list as the priority of the log entry.

5. **Selection rules**: The fourth criterion for the ranking of actions are selection rules. Such rules may define threshold values for the first three criteria which automatically lead to a high ranking of log entries that match a selection rule. An example for a selection rule is the following:

   \[
   \text{select log entry} \leftarrow (success_{\text{action}} = \text{true}) \text{ AND } (priority = 2) \text{ AND } (time-of-occurrence} \geq (\text{now - 3 days})
   \]
Selection of Actions for Event Handling

The rule states that a log entry whose action has been executed successfully in the last three days and has the second highest priority is to be selected from the list (vii).

Selection rules are generated manually and stored in the Web-Flow MaX metadata repository. They may help to reduce the number of actions which are suitable to handle a new event and thus may contribute to the goal of the action selection to find exactly one action.

The different criteria have different relevance for the ranking of a set of relevant log entries. Selection rules (criteria 5) have the highest relevance as they explicitly specify thresholds for the other criteria which decide about the selection of a log entry. Furthermore, as selection rules are specified by an administrator we assume that the administrator has chosen threshold values which may lead to good selection results. Contradictions between selection rules may occur but are no problem in this scenario, as the rules just group some criteria which have to be fulfilled to select an action. A log entry is simply compared with all selection rules and if it fulfills the criteria specified in one rule it is added to the final set of appropriate actions and the search stops. Thus, also rule termination is no problem for selection rules as they may not trigger further rules.

The similarity between a new log entry and the existing ones is of high relevance as it determines similar cases for the current event.

The success of an action execution (criterion 2) is also of high relevance for the ranking of actions as we consider it important whether an action has been executed successfully or not. As the success\textsuperscript{process} part is not assigned until the process instance finishes the value may not be available when a log entry is ranked. Thus, we have to consider different value combinations for the parts of the success attribute.

The relevance of the remaining two criteria, time of occurrence and priority, may be considered equal. However, for the ranking strategies we have to consider that the priority may only be available for subset of the log entries to rank.

Ranking Strategy

Based on the criteria above the Web-Flow MaX logging component ranks the entries of the list \(L_{le}\) of log entries with a sufficiently high similarity. In the following we present the applied strategy which leads to a new list \(L_{le,final}\) with the same log entries but in a ranked order:

1. In the first step the selection rules are applied. Therefore, each entry \(le\) of \(L_{le}\) is compared to the selection rules. If a matching rule is found for an entry \(le\) the search stops and \(le\) is added to \(L_{le,final}\) and deleted from \(L_{le}\). After this step the log entries which matched a selection rule are placed at the top of \(L_{le,final}\).

2. In the next step the similarity of the log entries to the new log entry is evaluated to get a ranking for all remaining entries \(le\) in \(L_{le}\). The log entries are added to \(L_{le,final}\) with decreasing similarity.
Log-based Selection of Actions and Recommendations

3. If several log entries in $L_{le,final}$ have the same similarity the entries in each group with equal similarity are ranked further evaluating the success attribute. The result are up to six groups $G_{le,i}$ $(i=1,..,6)$ of log entries:

- $G_{le,1}$ contains the log entries with $success_{action}=true$ and $success_{process}=true$.
- $G_{le,2}$ contains the log entries with $success_{action}=false$ and $success_{process}=true$.
- $G_{le,3}$ contains the log entries with $success_{action}=true$ and $success_{process}=false$.
- $G_{le,4}$ contains the log entries with $success_{action}=false$ and $success_{process}=false$.
- $G_{le,5}$ contains the log entries with $success_{action}=true$ and $success_{process}=nil$.
- $G_{le,6}$ contains the log entries with $success_{action}=false$ and $success_{process}=nil$.

The last two groups represent those log entries for which the value of $success_{process}$ is still unknown, for instance, as the process execution has not yet finished. The resulting groups are ranked in the following order which also determines the position of their members in the final list $L_{le,final}$ as the groups are added to $L_{le,final}$ in this order:

$G_{le,1}, G_{le,5}, G_{le,3}, G_{le,2}, G_{le,6}, G_{le,4}$

As we consider the $success_{action}$ attribute more important and it is available for every log entry we add first those log entries for which both success attributes are true, then those where $success_{action}$ is true and $success_{process}$ is either unknown (first) or false (second). Then the log entries for which $success_{action}$ is false are added in the same order determined by the value of the $success_{process}$ attribute.

The order „true - unknown - false“ for the $success_{process}$ attribute is determined by the consideration that an „unknown“ attribute value still may be updated to a „true“ attribute value whereas a „false“ attribute value remains false. Thus, we rank an attribute value „unknown“ higher than „false“.

4. In the third step for each group of log entries resulting from the second step the priority of the actions and the time of occurrence are evaluated. This determines the order of the log entries within one group and thus their order in the final list $L_{le,final}$. For the evaluation of these two attributes two different strategies are possible:

a) priority → time (priority before time): The log entries le in a group $G_{le,i}$ are first ordered according to the priority assigned to their actions starting with the entries with highest priority. Those log entries whose actions do not have a priority attribute are placed at the end of the list.
Selection of Actions for Event Handling

Then the log entries with the same priority are ordered according to their time of occurrence starting with the most recent log entries.

b) time → priority (time before priority): The log entries le in a group G_le,i are first ordered according to their time of occurrence starting with the most recent log entries. If then two or more log entries with the same time of occurrence exist, those may be ordered according to the priority assigned to their actions starting with the entries with highest priority. If a log entry does not have a priority attribute it is placed at the end of the list.

If the final ranked list L_le,final still contains more than ten log entries only the first 10 entries are presented to the administrator. Note, that the value of 10 log entries is only the default value and may be adjusted. Furthermore, the administrator may decide that only those log entries are relevant which match selection rules. Thus, the list L_le,final is cut off after those entries if such exist. All settings may be made using configuration parameters which are accessible over a user interface.

As it is difficult to decide which of the two possible strategies in step 4 is better, the strategy is also manually configurable. That means a user may decide which strategy is used and also may change this decision if he is not satisfied with the log entries presented in the final list L_le,final. For instance, it is possible that in the beginning high priorities have been assigned to the actions whereas after a while much lower or no priorities are assigned. Thus, if the first strategy is used more recent log entries will always be at the end of the list as they have a lower priority and will possibly be cut off.

To conclude the description of the ranking of several log entries which all have a sufficiently high similarity to the current log entry we present an example of a ranked list. We assume that the ranking strategy with sub-strategy a) (priority before time) has been applied on the list (vii) and has produced the final list (ix):

log-entry_3 | similarity = 0.87 [selection rule]  \[(ix)\]

log-entry_1 | similarity = 0.93 [success action=true, success process=Nil, priority=2]

log-entry_2 | similarity = 0.93 [success action=true, success process=Nil, priority=6]

The third log entry is on top of the list as it matches a selection rule (for instance, the one shown in (viii)). For the remaining two entries first the similarity is considered. As they have the same similarity the success attributes are evaluated which groups the two entries together in one group. Thus, in the third step the priority is considered which leads to the final ranking.
7.2.3 Performance Improvements

It may be quite time consuming to calculate the similarity between a new log entry and all existing log entries in the metadata repository especially when a large number of log entries is available. As the selection of appropriate actions may delay the event handling and thus the continuation of the affected process instance, performance improvements should be considered.

The main problem is the size of the search space for the similarity calculation, i.e., the number of log entries to which the new entry is compared. To reduce the search space a separation of the log entries according to particular log entry parts may be useful. A good candidate for a separation is the event attribute as it is present in all log entries whereas, for instance, the operation attribute may be omitted for particular events.

The log entries may be separated into different classes according to their event type using the types defined in the similarity matrix in Tab. 7-1. Using these separate groups a new log entry is first compared to log entries of the class with the same event type, i.e., those entries for which holds:

\[ ds_{\text{event}}(e_1, e_2) \geq 0.95 \text{ (using the values of the matrix in Tab. 7-1)} \]

If no entry is contained in this class or the similarity between the new log entry and all contained log entries is lower than the similarity threshold \( \text{sim}\_\text{threshold} \), the next classes of log entries may be searched.

That means the demanded value for \( ds_{\text{event}}(e_1, e_2) \) is decreased by 0.05 and the similarity between the new log entry and the entries in the respective classes are searched. These steps are repeated until either one or more log entries which are sufficiently similar are found (i.e., \( ds \geq \text{sim}\_\text{threshold} \)) or until all log entries have been searched without a satisfying result. The result of the search serves again as input for the ranking algorithm described in Section 7.2.2.

Another possible performance improvement would be the periodic pre-calculation of similarity values, for instance, for particular classes of log entries. However, the number of possible operations and process definitions is infinite, thus we would have to consider a large number of different classes. If a new entry occurs that does not match one of the pre-computed classes (for example, as the operation part does not match), we would have to calculate the similarity between the new log entry and the classes to decide which class of similarity values applies. Thus, it has to be investigated carefully, whether a pre-calculation really helps to improve performance.

7.3 Generation of Recommendations and Rules

Besides the selection of appropriate actions to handle particular events the log entries gathered by the Web-Flow MaX logging and analysis component may also be used to generate rules for event handling and recommendations for process optimizations. Event handling rules may accelerate the whole event handling process as described in Chapter 6 as appropriate actions may be found fast-
The recommendations may be used by a process designer to optimize process definitions or quality constraints to prevent future events. Some examples for recommendations are the following:

- A Web service fault message of a particular service operation is handled by a process internal event handler in one process definition but not in another process definition. An analysis of the log entries may reveal that the same event occurring for the same operation in different process instances (i.e., belonging to different process definitions) is handled differently. If a process internal event handler is available a *continue process* action was executed otherwise an *alternative service call* action was used.
  
  If the process internal event handler provides a good solution the administrator may adapt the second process definition and also add the event handler (if the process definition language supports event handlers).

- A service operation always violates its response time constraint if it is executed in a particular context, i.e., a particular sequence of activities. This may, for instance, be caused by the fact that a preceding operation of the same service provider blocks some resources and so the operation execution is delayed. To solve this problem the process designer may either relax the quality constraint (for instance, with a context dependent constraint) or replace one of the activities by one of another service provider.

The Web-Flow MaX component does not derive recommendations automatically rather it offers a tool for analyzing the log data after particular criteria. The results of the analysis may then be interpreted by the administrator to derive possible optimizations. The analysis tool also offers the possibility to derive event handling rules from the log entries. These functionalities are described in detail in the remainder of this section. However, the analysis tool does not aim at re-implementing a data warehouse or complex analysis tools but is tailored to the log data and the tasks occurring in the Web-Flow MaX component.

### 7.3.1 Analysis of Log Data

For the analysis of log entries the parts (namely operation, process definition, event, and action) and attributes (for instance, the event type) of a log entry may be used. In the following we discuss which parts are available and which analysis may be performed on them. We discuss each main part separately. The possible analysis queries mentioned below are only examples and do not cover all possible queries. This set is infinite and depends on the goals of the different users.

**Operation Part**

The *operation* part has two kinds of attributes which either provide a syntactic description of the related operation or a semantic description of the operation using, for instance, the operation functionality or the service category. In detail the following attributes are available (cf. Fig. 7-1):
- **operation name**: The operation name may be used in combination with the porttype and the service to find all log entries for the same operation, for instance, to find out which events have occurred for a particular operation and how they have been handled in different processes.
- **porttype**: The porttype is used in combination with the operation name to identify a particular operation.
- **service**: The service name may be used in combination with the operation name and the porttype to clearly identify a particular operation. It may also be used to derive all events that have occurred for operations of a particular service.
- **provider**: All events that happened for service operations of a particular service provider may be identified with this attribute.
- **start time / end time**: Start time and end time may be used to restrict the number of interesting log entries to operation executions that started / ended / have been performed at a certain time point or during a certain time interval.
- **regular**: This attribute separates log entries for regular service executions and log entries of events that occurred during event handling. For instance, it may be used to derive how many inter-dependent events occurred for a particular operation in a particular process instance (inter-dependent events are events that occur when an automatic action is executed to handle an event or exception).

**Process Definition Part**

The process definition part has three attributes:
- process ID (the unique ID used in the Web-Flow specific process description),
- process name (not unique, used in the Web-Flow specific process description), and
- process instance ID (the unique ID used in Web-Flow to identify the monitored process instances, for details see Section 5.2.1).

As the process name is not unique most queries will use the process ID respectively the process instance ID for instance to find events that occurred in particular processes or process instances. If the process ID is used all events that occurred in any process instance of this process definition are considered. To restrict the search on process instances (for instance, as many events have occurred during the execution of a particular process instance) the user may specify a process instance ID.

**Event Part and Action Part**

The event part of a log entry contains
- a detailed description of the event that has occurred and
- the point in time when the event has occurred.
The event attribute may be used to filter the log entries after particular event types, for instance, the violation of response time constraints or Web service fault messages. Furthermore, events that occurred at particular dates or during particular day times (for instance, between 10 and 12 in the morning) may be filtered.

The action part contains the following attributes:
- a description of the action that has been executed to handle the event of this log entry,
- the success attribute that describes whether the action has been executed successfully and whether further events have occurred when the process instance has been continued.

With these attributes we may filter whether particular actions have been executed successfully or which actions often lead to subsequent events in the affected process instance.

We close this section with two sample queries filtering a set of log entries. With the first query we retrieve all inter-dependent events that have occurred while an event has been handled for a particular operation in a particular process definition (i.e., we consider all process instances that have been derived from this process definition):

\[
\text{operation name} = \text{"op1" AND porttype} = \text{"pt1" AND service} = \text{"s1" AND} \quad (x)
\]
\[
\text{processID} = \text{"p15" AND}
\]
\[
\text{regular} = \text{"false" AND}
\]
\[
\text{event is not null AND}
\]
\[
\text{success action} = \text{"false"}
\]

We do not specify the event itself as the subsequent events may have different types than the original event. The third line (regular = \text{"false"}) selects all log entries that have been written for events that occurred during event handling. The fourth line (success action = \text{"false"}) selects only those log entries whose action failed again. If we omit the last line also the last successful action will be selected.

This query may, for instance, be used to check whether a particular event handling rule that has been defined for a particular event and a particular operation is well suited as it does not lead to inter-dependent events. For example, if always a subsequent event occurs after the application of the rule the rule may be changed.

Another example query selects events that occur during particular daytimes, for instance, between 8:00 and 10:00 in the morning, in process instances derived from a particular process definition:

\[
\text{occurred at} \geq 8:00 \text{ a.m. AND occurred at} \leq 10:00 \text{ a.m. AND} \quad (xi)
\]
\[
\text{processID} = \text{"p27"}
\]
Log-based Selection of Actions and Recommendations

The first line checks whether the sub-attribute "occurred at" has the right value. The second line restricts the events to those that occurred while an instance of process definition with ID p27 has been executed. With this query we may check which events occur quite frequently during particular day times. Based on the result a administrator may, for instance, relax response time constraints or change the process definition by using different service operation to prevent the events.

7.3.2 Generation of Rules

The analysis functionality described in the last section may also be used to derive event handling rules (described in Section 4.3.3) semi-automatically. These rules are used to find appropriate actions for handling events as it has been described in Section 6.2.

Event handling rules consist of four parts:
- the event,
- the action,
- an optional condition such as an iteration constraint or a context,
- an optional operation if a rule should only apply for particular operations.

Events, actions, and operations are also part of the log entries (cf. Section 7.1.1) and thus may be retrieved directly from the log data. Optional conditions are specified by an administrator who, for instance, may also use the Web-Flow specific process descriptions or the original process definitions to derive context descriptions.

To derive rules analysis queries are used which, for instance, select all events that have occurred for a particular operation:

\[
\text{operation name} = \text{"op2" AND porttype} = \text{"pt2" AND service} = \text{"s2" AND event is not null} \tag{xii}
\]

This query selects all log entries that contain the specified operation and whose event part is not empty. In contrast to Query (x) also the log entries describing regular service calls and those with a successful action execution are selected. The administrator may additionally sort the analysis results to find those events that occurred most frequently for this operation. Based on this information the administrator may define a rule which specifies a particular action for the events which occur most often for the operation specified in Query (xii).

If the administrator additionally wants to consider a particular operation in a particular process definition (for instance, as this operation is executed in a particular context) a query similar to the following one may be used:

\[
\text{operation name} = \text{"op2" AND porttype} = \text{"pt2" AND service} = \text{"s2" AND activity ID} = \text{"a64" AND process ID} = \text{"p24" AND} \tag{xiii}
\]
event is not null

The two additional lines restrict the log entries to those which describe the execution of the service operation for a particular activity in a particular process definition. The activity ID and the process ID are derived manually from the Web-Flow specific process description saved in the Web-Flow MaX metadata repository. If the context is to be contained in the rule it has to be added manually respectively with the help of the rule generation tool as it is not possible to derive the context intended by the user automatically. For instance, the Web-Flow MaX analysis component cannot know how many preceding and succeeding activities are part of the context.

To prevent rule contradictions the semi-automatic rule generation of the Web-Flow MaX component compares new rules to the rules already contained in the metadata repository before storing them permanently. Rule contradictions may occur if the user specifies

- different actions for the same event without any further conditions.
- different actions for the same event and the same operation.
- different actions for the same event and the same additional quality constraints.
- different actions for the same event, the same operation, and the same additional quality constraint.
- different actions for the same event, the same operation, and the same context.
- different actions for the same event, the same operation, the same additional quality constraint, and the same context.

The list contains all possible combinations of the event and optional condition parts of Web-Flow rules as these parts determine which rule applies if a particular event occurs for a particular operation. The notion „same event“ means that the events are not only of the same type but really describe the same event, for instance, the violation of an operation specific response time constraint that specifies a maximal response time of five minutes. The same holds for actions, i.e. actions of the same type may still be different, for instance, when they apply to different operations.

To resolve such contradictions additional conditions may be specified or existing conditions may be adapted. For instance, if an iteration constraint of the existing rule specifies a maximum number of three iterations (as shown in lines 16 to 23 of the rule in Fig. 7-4) the new rule may specify a minimum number of four iterations using the following condition part:

```xml
<condition>
  <qualityConstraint>
    <iterationConstraint name="itConstraint2">
      <comparisonOp>greaterEqual</comparisonOp>
      <iterationNumber>4</iterationNumber>
    </iterationConstraint>
  </qualityConstraint>
</condition>
```
Log-based Selection of Actions and Recommendations

That means that the existing rule applies if the activity has not been repeated more than three times and the new rule applies if the activity has been repeated more than three times.

If the administrator does not solve a contradiction detected by the Web-Flow MaX analysis component the contradictory rule is not saved in the metadata repository as this may lead to ambiguous states later when the rules are used to derive appropriate actions.

Furthermore, if a new rule is saved that contains an iteration constraint it is checked automatically, whether the number of iterations specified in the constraint is less or bigger than the global itera-

Fig. 7-4: WXS description of a rule.
The rule describes that a violation of a response time constraint is to be handled with the repetition of the service call if the operation has not been repeated more than three times.

That means that the existing rule applies if the activity has not been repeated more than three times and the new rule applies if the activity has been repeated more than three times. 
tion number $GIN_{event}$. $GIN_{event}$ is used to restrict the maximum number of inter-dependent events to prevent an infinite repetition of event handling steps for one operation (cf. Section 6.1.2). Thus, the biggest iteration number specified in any rule in the Web-Flow MaX metadata repository has to be less then $GIN_{event}$ to prevent that some rules may not be triggered as the maximum number for inter-dependent event handling steps has already been reached.

As the number of possible analysis queries is infinite and depends on the intention of the users the analysis tools also offers the possibility to save often used queries as templates in the Web-Flow MaX metadata repository. As this is a more technical feature that facilitates the use of the tool for the administrator we do not discuss it further in this thesis.

7.4 Summary

In this chapter we described the functionality provided by the Web-Flow MaX logging and analysis component, the last major component of the Web-Flow MaX component. To reduce the manual effort for defining and maintaining exception handling rules the rule-based action selection of Web-Flow is accomplished with a log-based selection of appropriate actions.

Therefore, appropriate log data has to be gathered during process instance execution. As Web-Flow aims at providing a generic approach, log data of the process engine as well as from the service provider may not be used. The Web-Flow MaX logging component achieves this by gathering log data from the messages sent between the process engine and the services. Thus, the autonomy of the service providers is not violated. Furthermore, the messages are available for different types of services.

To select appropriate actions from the log data semi-automatically, the Web-Flow MaX logging component offers algorithms to calculate the similarity between log entries. Thus, appropriate actions may also be found, if no log entry describes exactly the same event. The log entries are then ranked according to their similarities to the new event and the administrator selects the action that actually should be applied. This ensures, that no undesired actions are selected.

Quality-oriented execution of cooperative processes may also be supported by regular revision of processes based on recommendations identifying weak points. The Web-Flow MaX logging and analysis component also supports this with a user-guided analysis of the log data to generate recommendations for process optimizations. Also mechanisms are provided to semi-automatically generate exception handling rules based on the analysis results. These rules then again improve the dynamic, rule-based exception handling offered by Web-Flow.
Log-based Selection of Actions and Recommendations
8 Implementation and Evaluation

The concepts and algorithms of the Web-Flow architecture presented in the preceding chapters have been implemented in a prototypical realization of the Web-Flow MaX component. The first version of this prototype is runnable and demonstrates the integrated monitoring of the execution quality of a cooperative process instance. This starts with the definition of quality constraints, continues with the monitoring of quality constraints and events, the selection of appropriate actions from log entries, and finishes with event handling with a restricted number of the actions introduced in this thesis. We also have built a prototypical implementation of the collaborative fulfillment scenario described in Chapter 1 to show the applicability of our approach in a real-world scenario. In this chapter we present the main parts of the Web-Flow MaX prototype and some evaluations we have performed.

8.1 Web-Flow Prototype

The complete Web-Flow prototype consists of the prototypical implementation of the Web-Flow MaX component, a process engine, and a sample application scenario to demonstrate the applicability of our approach. The whole prototype has been built using freely available or open source software and technologies. Fig. 8-1 gives an overview over the technologies used for the different parts of the Web-Flow prototype.

For the infrastructure necessary to execute Web services and cooperative processes we used the Apache AXIS toolkit [AXIS2004], an implementation of the Simple Object Access Protocol (SOAP, [SOAP2003]), together with the Jakarta Tomcat application server [TOMC2004]. The cooperative processes are defined and executed with the BPWS4J process engine which is an implementation of BPEL4WS provided by IBM alphaworks [IBM2004A].

For the implementation of the Web-Flow MaX component we used JAVA [JAVA2004] together with some additional packages such as JAXB [JAXB2004] and WSDL4J [IBM2004C]. JAXB is the Java Architecture for XML Binding and provides a convenient way to bind an XML schema
Implementation and Evaluation

to a representation in Java code. Thus, the classes and types of the WXS schema (cf. appendix A) may easily be integrated in the implementation of the Web-Flow MaX component. The Web Services Description Language for Java Toolkit (WSDL4J) allows the creation, representation, and manipulation of WSDL documents describing services. It is used, for instance, in the Web-Flow MaX constraint definition component to parse the WSDL files of the services used in a cooperative process.

The Web-Flow Max metadata repository comprises an XML database (eXist, [EXIS2004]) and a relational database for the monitoring component (MySQL, [MYSQ2004]). We use an XML database for the Web-Flow specific XML documents as a database system offers support for structured storing of XML documents, for instance, by creating collections of related documents. For instance, we can group documents related to process and service descriptions in one collection and documents with logged execution data in another. Furthermore, an XML database offers a more efficient management and retrieval of XML files than it is possible in the file system. A database system also offers query support to extract data from XML files (for instance, using XPath or

Fig. 8-1: Technologies and software used in the Web-Flow prototype.
XQuery).
The Web services used in the sample scenario have also been implemented using JAVA and are accessible over the Apache AXIS toolkit in combination with the Jakarta Tomcat application server. As the implementation of the sample scenario is not the main focus of this thesis we concentrate in the following on the Web-Flow MaX prototype. Additionally we present some evaluation results that were performed to evaluate the applicability of the concepts and algorithms developed in this thesis.

8.1.1 Web-Flow MaX Prototype

The integrated support of the execution quality of cooperative processes provided by the Web-Flow MaX component starts with the definition of quality constraints for existing process definition. Therefore, the Web-Flow MaX constraint definition component (also called Web-Flow constraint editor) is used. The constraint editor has the following main tasks:

1. Import of cooperative process definitions:
   The cooperative process definitions may, for instance, be defined in BPEL4WS (Business Process Execution Language for Web Services [THAT2003]) or another process-engine proprietary process definition format. Thus, the Web-Flow editor needs import filters for the different process definition formats of the cooperative processes which are to be monitored by the Web-Flow MaX component.
   During the import the event and exception handlers contained in a process definition are extracted and saved in the Web-Flow MaX metadata repository to be available for dynamic event handling (cf. Chapter 6). Additionally, the WSDL files of the services used in a cooperative process definition are loaded. If a service has also been used in another process instance which has already been imported in Web-Flow then the Web-Flow specific service description is retrieved from the metadata repository. Furthermore, global and context dependent quality constraints are also retrieved, the latter only if their context is contained in the current process instance. That means the activity for which the context is specified is contained in the process instance and its preceding and succeeding activities match those described in the context.

2. Definition and storing of quality constraints:
   The quality constraints are defined over a graphical user interface which contains on the left side a tree structure of the activities of a cooperative process (see Fig. 8-2). For each of these activities quality constraints may be defined. The top right pane in Fig. 8-2 shows a sample result constraint that checks whether the delivery date returned by the „ATPCheck“ operation of the monitor seller is less than the desired delivery date specified by the user. To facilitate the definition of input and result constraints the Web-Flow editor also provides some information about the input and output parameters of a particular operation and the XML schema definitions of the complex data types that are used in the parameters. This information may,
Implementation and Evaluation

for instance, be derived from the WSDL service descriptions of a particular service. Context dependent and global constraints are specified using a similar pane. However, the constraints are not linked to particular activities or operations but are saved separately in the Web-Flow MaX metadata repository. If several quality constraints with the same scope have been defined for one activity or operation they may be linked together in a constraint expression. Therefore, also an appropriate user dialogue is provided.

When the constraints are stored in the Web-Flow MaX metadata repository together with the Web-Flow specific process and service descriptions some automatic checks are performed. For instance, it is checked whether both parts of input and result constraints use consistent datatypes. Additionally, contradictions between quality constraints may be checked as described in Section 5.3.1.1.

3. Revision of quality constraints for particular process definition:

As quality constraints may change over time the Web-Flow editor also supports a revision of quality constraints. Therefore, Web-Flow specific process or service descriptions may be

Fig. 8-2: Screenshot of Web-Flow MaX constraint definition component.
Web-Flow Prototype

loaded from the Web-Flow MaX metadata repository. After the changes have been performed the updated descriptions and quality constraints are saved in the metadata repository. Additional tasks of the Web-Flow editor are, for instance, user management to ensure that only administrators can define quality constraints and edit process and service descriptions. Therefore, a login mechanism is provided. The information about the currently logged in user may also be used as default contact information, for instance, in manual reaction actions, as it may be assumed that process administrators may also be responsible for event handling.

The next part of the Web-Flow MaX component which is responsible for the quality support of cooperative process instances is the Web-Flow MaX monitoring component. It is responsible for monitoring the quality constraints while a process instance is executed. Furthermore, it detects events such as Web service or manual fault messages or external events which may occur during process execution and may have to be handled to achieve the desired execution quality. All events (for instance, also violations of quality constraints) are forwarded to the Web-Flow MaX dynamic event handling component. The monitoring component of the Web-Flow MaX prototype implements the algorithms described in Chapter 5. Thus, no further discussion is provided here. More details especially on the technical realization of the component may be found in [BERR2004].

The dynamic event and exception handling component of the Web-Flow MaX component implements the selection of appropriate actions using the Web-Flow MaX logging component and two of the operators introduced in Chapter 6. We have chosen the operators for the repeat service call and the manual reaction action to demonstrate the integrated monitoring of process execution quality. Fig. 8-3 shows a screenshot of the user dialogue for the manual reaction action in which the user is informed about an exception that has occurred and the possible actions he may choose for handling. Further enhancements of the dynamic event handling component are a subject to future work.

The logging and analysis component of the Web-Flow MaX prototype offers an interface over which the dynamic event handling component may ask for appropriate actions to handle a particular event. The logging component selects the actions from the log entries in the Web-Flow MaX metadata repository using the algorithms described in Chapter 7. Furthermore, the logging component provides a graphical user interface over which analysis may be performed on the log data and recommendations for process optimizations may be derived. Based on the analysis results also rules may be generated which are stored in the metadata repository and may be used by the dynamic event handling component to accelerate the selection of appropriate actions for event handling as it has been described in Chapter 6. More details on the realization of the logging and analysis component may be found in [FRIT2004].
8.1.2 Evaluation

We have performed some evaluations of the concepts and algorithms of the Web-Flow architecture. Thereby, we considered two aspects:

- The overall applicability and usability of the Web-Flow MaX component to describe, monitor, and ensure the execution quality of cooperative process instances.
- The similarity functions used to derive appropriate actions for event handling automatically.

This is due to the fact that there are several weights which may be adjusted to influence the results of these functions and thus the overall performance of the dynamic event handling.

Regarding the first aspect we have not yet performed a complete evaluation with a large number of cooperative process definitions and process instances. However, the completely runnable Web-Flow prototype shows first that WXS, the Web-Flow quality and eXception Schema, may be used...
to describe the desired execution quality in real-world scenarios. Second, it shows that the algorithms for event monitoring and dynamic event handling may help to support the execution quality of cooperative processes.

Furthermore, we already have some detailed results for the performance of the Web-Flow MaX monitoring component in the context of our sample scenario. For instance, we have analyzed the delay that is imposed on the process instance execution by the initialization of the constraint monitoring. Therefore we have considered two scenarios:

- In the first scenario four process specific quality constraints have been defined on the collaborative fulfillment process, namely three input respectively result constraints and one response time constraint.
- In the second scenario also four process specific input respectively result constraints have been defined and additionally one global response time constraint has been defined.

With the second scenario we wanted to test whether global constraints extend the initialization significantly as they are valid for each activity of the cooperative process. For the evaluation we executed 19 instances of each scenario and measured the initialization time. Each process instance consists of 20 activities. The results are listed in Tab. 8-1 below.

As a first result we can derive that the initialization takes only some seconds. In comparison to overall process instance execution durations of hours, days, or even weeks this additional delay is acceptable. Furthermore, we can see that global constraints which are valid for each activity in a cooperative process instance in fact increase the initialization time. In our scenarios, the duration has been increased about a factor of 1.6 (for minimal and average) and a factor of 1.97 (for maximal). However, compared to the overall process execution durations this delay is not significant.

<table>
<thead>
<tr>
<th></th>
<th>scenario1</th>
<th>scenario2</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimal initialization duration</td>
<td>962 ms</td>
<td>1573</td>
</tr>
<tr>
<td>maximal initialization duration</td>
<td>2083 ms</td>
<td>4096</td>
</tr>
<tr>
<td>average initialization duration</td>
<td>1351 ms</td>
<td>2126</td>
</tr>
</tbody>
</table>

*Table 8-1: Initialization durations measured in milliseconds for 19 process instances.*

We also measured the message evaluation time for these two scenarios, i.e., the time the Web-Flow MaX monitoring component needs for evaluating each intercepted message and the quality
Implementation and Evaluation

With these measurements we may estimate how long the execution of the particular activities will be delayed by the additional quality monitoring. The results are summarized in Tab. 8-2.

As we can see the message evaluation does not delay the execution of the single activities significantly, the maximum value is less than 7 seconds. Projected on the whole process instance with 20 activities we get an average delay (imposed by the message evaluation) of approximately 23 seconds for the first scenario and 12 seconds for the second scenario. Compared to the overall process execution duration of hours, days, or weeks this does not reduce the process execution quality. Furthermore, we can see that global constraints do not increase the message evaluation time.

We plan to perform more evaluations with several real-world scenarios and real-world application data in the future to get more significant results. For instance, we also want to discover if the robustness of cooperative processes may be significantly raised by the use of the Web-Flow MaX component. Therefore, the number of events detected and handled by Web-Flow may be compared to the number of all events that occurred for a process instance.

We also have performed an evaluation of the functions used for action selection (cf. Chapter 7). The main goal of the automatic action selection is to get a high selectivity of the similarity measures, i.e. in the best case only one log entry should have a sufficiently high similarity to the new log entry. Then the action of this log entry may be chosen for execution without any further user interaction. The number of results of the automated action selection may be influenced by adjusting the weights used to aggregate the similarity measures (cf. Section 7.2.1). Furthermore, also the event similarities may affect the results (cf. 7.2.1.2) but we have not evaluated this aspect further as this aspect may be better evaluated with real world than with generated test data. To derive possible dependencies between the weights and the number of appropriate actions we have performed several evaluations which we will describe in the following.

For the evaluations we used four test data sets representing log data that may have been gathered during real process executions. Therefore, we used the services of our sample scenario and gen-

<table>
<thead>
<tr>
<th></th>
<th>scenario1</th>
<th>scenario2</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimal message duration</td>
<td>140 ms</td>
<td>100</td>
</tr>
<tr>
<td>maximal message duration</td>
<td>6549 ms</td>
<td>6860</td>
</tr>
<tr>
<td>average message duration</td>
<td>1145 ms</td>
<td>607</td>
</tr>
</tbody>
</table>

Table 8-2: Message evaluation durations measured in milliseconds for 19 process instances.
erated log entries representing occurred events. Thereby, we assumed that events of the four main types, namely violation of quality constraint, manual fault message, Web service fault message, and external event, occurred with different probabilities. For the distribution of the sub-types of quality constraint violations and manual fault messages we assumed a uniform distribution. We generated test data for the following cases:

1. All four event types are equally distributed.
2. Events of type violation of quality constraint, manual fault message, and Web service fault message occur with equal probability and events of type external event occur less often than other events.
3. Events of type manual fault message occur more often than events of type Web service fault message and with the same probability as events of type violation of quality constraint. Again, events of type external event occur less often than events of all other types. This case represents the case that a high percentage of Web service operations used in cooperative processes have manual interaction.
4. The fourth case is similar to the third case with the only difference that events of type violation of quality constraint occur less often than events of type manual fault message but more often than events of type Web service fault message. This gives still more relevance to Web service operations with manual interaction.

Each of the test data sets comprises 100 log entries. Tab. 8-3 summarizes the probability distributions in the different test data sets.

The action selection algorithm uses different types of weights which may, for instance, stress the direct comparison values or the semantic comparison value (cf. Section 7.2.1). The values of the different weights may, for instance, influence the number of similar log entries that are found for a new event. The default values for all weights are 1, i.e. all similarity values are weighted equal.

<table>
<thead>
<tr>
<th></th>
<th>violation of quality constraint</th>
<th>manual fault message</th>
<th>Web service fault message</th>
<th>external event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.35</td>
<td>0.35</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>0.3</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 8-3: Probability distributions of test data sets for evaluation of action selection.
However, this may lead to a great number of appropriate actions. To evaluate which of the single similarity values influences the number of results to which degree several evaluations may be performed by adjusting the weights. In the following we present the evaluation results for four simple weight sets which represent the basic possibility for adjusting the similarity calculation:

1. In the first weight set we use the default values for all weights.
2. In the second weight set we set the weight for the porttype to zero to evaluate whether this technical parameter has much influence on the number of similar entries. For all other weights the default values are used.
3. In the third weight set we set the weight for the direct similarity value of operations to zero, thus we only consider the result of the semantic similarity. For all other weights the default values are used.
4. In the fourth weight set we set the weight for the semantic similarity value of operations to zero, thus we only consider the direct similarity. All other weights are set to the default values.

We calculate the similarity between one randomly selected log entry and all other log entries of each data set using the different weight sets. As the similarity threshold introduced in Section 7.2.2 also may influence the number of selected actions we also have considered four different thresholds in the evaluation, namely 70 %, 80 %, 90 %, and 95 %. Tab. 8-4 shows the results for two runs, i.e. the similarity calculation for one log entry of each data set using each weight set. The values already confirm most of our assumptions which guided the design of the action selection algorithm.

For instance, we can see that a similarity threshold of 90 % or higher is a good choice as it leads to a sufficiently high selectivity, i.e., it reduces the number of appropriate log entries. The consideration of technical parameters such as the porttype may also reduce the number of similar log entries esp. if a low similarity threshold is used (the number of appropriate log entries is higher for weight set 2 than for weight set 1).

If we only use semantic comparison of operations (weight set 3) we typically get more possible operations however with a reduced overall similarity of 70 % to 80 %. In contrast to that we may find completely matching log entries (similarity of 100 %) if we only consider the direct operation similarity (weight set 4). That means that the use of the semantic operation similarity together with the direct operation similarity may reduce the overall similarity for a log set. Thus, it may possibly be a good solution to reduce the impact of the semantic operation similarity in comparison to the impact of the direct operation similarity. However, such more sophisticated evaluations are a subject of future work. Thereby, we also want to consider more runs to get more expressive results about the selectivity of the action selection algorithm and the different weights.
<table>
<thead>
<tr>
<th>similarity threshold</th>
<th>data set 1</th>
<th>data set 2</th>
<th>data set 3</th>
<th>data set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>run 1</td>
<td>run 2</td>
<td>run 1</td>
<td>run 2</td>
</tr>
<tr>
<td>weight set 1 (default)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>0.8</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>0.9</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0.95</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max. sim.</td>
<td>0.948</td>
<td>0.851</td>
<td>0.981</td>
<td>0.962</td>
</tr>
<tr>
<td>weight set 2 (without porttype)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>6</td>
<td>26</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>0.8</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>0.9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>0.95</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max. sim.</td>
<td>0.948</td>
<td>0.907</td>
<td>0.981</td>
<td>0.962</td>
</tr>
<tr>
<td>weight set 3 (only semantic)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>6</td>
<td>15</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>0.8</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>0.9</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0.95</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>max. sim.</td>
<td>0.929</td>
<td>0.814</td>
<td>0.962</td>
<td>0.925</td>
</tr>
<tr>
<td>weight set 4 (only direct)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>18</td>
<td>10</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>0.8</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>0.9</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0.95</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max. sim.</td>
<td>0.966</td>
<td>0.888</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 8-4: Number of found log entries for two runs of evaluation of action selection.
8.2 Summary

In this chapter we have presented the runnable Web-Flow prototype which shows the integrated realization of the Web-Flow architecture presented in this thesis. It consists of the Web-Flow MaX prototype, a process engine, and a sample real-world scenario. The Web-Flow MaX prototype implements the algorithms of the different parts developed in the preceding chapter of this thesis. The Web-Flow prototype shows the applicability of process execution quality support based on quality monitoring and event handling.

Some evaluations have been performed on the performance of the Web-Flow MaX constraint and event monitoring component. Furthermore, the automatic action selection provided by the Web-Flow MaX logging and analysis component has been evaluated to find out which weights are appropriate to achieve a high selectivity of the selection algorithms. A comprehensive evaluation of the usability of the Web-Flow approach, for instance, regarding the additional effort for process designers to define quality constraints and exception handling rules, has not been performed. Therefore a significant number of cooperative processes has to be monitored by Web-Flow. Thus, a more comprehensive evaluation of the Web-Flow prototype is not possible until Web-Flow is not used to continuously monitor execution quality in several cooperative scenarios.
9 Summary and Future Work

This chapter closes the thesis with a summary of the achieved results and an outlook on future work that will continue and complete the results of this thesis.

9.1 Summary

In this thesis we developed the Web-Flow architecture which offers a new approach to support the quality-oriented execution and optimization of cooperative processes. Cooperative processes use services provided by different organizations to execute the tasks in the process. The applications encapsulated in the services are heterogeneous and of different complexity. They may reach from legacy applications over complex applications to manual activities. Furthermore, the service providers use Web services to preserve their autonomy with respect to the internal realization of their services and applications. Cooperative processes can be executed with different cooperation models, a simple one in which only one partner knows the overall process and a complex one in which the partners agree on the process (cf. Section 1.1.3). Especially if the complex cooperation model is used a cooperative process may be executed in different process engines.

The first main contribution of the thesis is the Web-Flow architecture which provides a generic infrastructure to support quality-oriented execution and optimization of cooperative processes. Web-Flow is Web-service-based and supports quality support for both cooperation models. The infrastructure offers quality support for cooperative processes independently of the capabilities of the process engine as well as of the services. The infrastructure is implemented in a dedicated component, called Web-Flow MaX\(^6\), which may be used together with different process engines. Ensuring that cooperative processes reliably serve their purpose and meet the users’ expectations, is based on achieving a high process execution quality. As the process tasks are executed by Web services, this implies achieving a high execution quality of Web services. Approaches supporting

6. Monitoring and eXception handling
Summary and Future Work

The execution quality of cooperative processes may be classified in four categories:

1. selection of the best fitting service operation during process definition,
2. selection of the best fitting service provider during process execution,
3. integration of event and exception handlers in process definition,
4. definition and monitoring of additional quality conditions for service execution and flexible event handling during process execution.

The process engine executing the cooperative processes may already offer exception handling belonging to one of the first three categories. This is accomplished by the Web-Flow architecture providing comprehensive support for quality-oriented process execution in the fourth category. Therefore, Web-Flow provides as the second main contribution a classification of various quality characteristics that may be imposed on services. As Web services are heterogeneous and the service providers are autonomous the quality characteristics are applicable to different types of services and can be monitored without violating the autonomy of the service provider. Besides the quality characteristics Web-Flow also provides a metamodel of concepts related to a quality-oriented execution of cooperative processes. The metamodel comprises the following main concepts:

- a classification of quality constraints to describe the desired execution quality of heterogeneous services as well as of whole cooperative processes (for instance, response time, metadata, input, or result constraints),
- semantic information about a service such as service categories, operation functionality, and parameter roles,
- events (such as the violation of quality constraints or failed service executions) that may occur during process execution and may lead to the violation of the desired execution quality of the cooperative process,
- actions that are executed to handle events and that ensure that the process may be continued despite an event with the best execution quality possible,
- rules that define appropriate actions to handle occurring events.

A machine- as well as human-readable representation of these concepts is provided with the Web-Flow quality and eXception Schema (WXS). As the main focus of the Web-Flow architecture is on Web-service-based cooperative processes and XML is widely used in the context of Web services, WXS is based on an XML schema.

To achieve a high flexibility and robustness of cooperative processes violation of any quality characteristics should be handled dynamically. This also applies for other events that may hinder a successful and reliable process execution such as fault messages indicating a failed service execution. Therefore, the Web-Flow architecture provides as a third main contribution rule-based dynamic event and exception handling that accomplishes exception handling already provided by the process engine. In particular, Web-Flow’s dynamic event and exception handling offers the following features:
Future Work

- **Automated quality monitoring and event detection:** Web-Flow allows for the automated monitoring of quality constraints imposed on service executions in cooperative processes. The quality monitoring can be applied to different types of services even if they do not offer quality support by themselves (as, for instance, in legacy applications). The automatic monitoring also covers other exceptional events that may occur during the execution of Web services and may hinder the successful execution of cooperative processes. These events may be fault messages sent by Web services to the process engine, manual fault messages (i.e., fault messages received by phone or e-mail) or external events. Predefined message formats for manual fault messages and external events described in WXS allow for an automated evaluation. With the automated monitoring it may be ensured that no exceptional events are overlooked and the administrators are relieved.

- **Rule-based dynamic event / exception handling and semi-automatic action determination:** Web-Flow provides a rule-based approach to handle events / exceptions and helps to ensure the execution quality of cooperative processes. To relieve administrators from having to define and maintain a large number of exception handling rules, this approach is accomplished by a log-based mechanism to derive appropriate actions semi-automatically from logged execution data. Therefore, Web-Flow searches in the log data for similar events that have occurred earlier. Thereby, auxiliary conditions such as the success of the former event handling or the violation of further quality constraints are considered. The actions for handling the exceptions are implemented with a set of operators. These allow for a semi-automatic execution of the actions. Semi-automatic means, that an administrator responsible for the cooperative process has to confirm the action before execution.

- **Log-based optimization:** Web-Flow also offers mechanisms to analyze the execution data logged during former process executions and to derive recommendations for process revisions. For instance, recommendations may reveal service calls that frequently lead to exceptions. Regular process revisions taking into account recommendation also help to improve the process execution quality.

A prototypical implementation of the Web-Flow architecture realizes the developed concepts and mechanisms. Several evaluations have been performed on the Web-Flow prototype that show the usability of our approach of a generic infrastructure for execution quality support for cooperative processes.

### 9.2 Future Work

The work presented in this thesis comprises all necessary parts of the Web-Flow architecture including a prototypical realization. However, some open issues remain which are to be investigated in the future to continue and complete the work of this thesis. In the following, we shortly discuss these points and sketch how they may contribute to a further improvement of the Web-Flow architecture.
Summary and Future Work

With respect to the Web-Flow metamodel further types of context should be investigated in the future to allow for the description of more complex context dependent quality constraint. For instance, time or user dependencies may be considered. To allow for the presentation of time dependencies the Web-Flow metamodel has to be extended with temporal information representing for instance the temporal distance between two service calls. This information cannot be captured in the current version of the metamodel.

Regarding the mechanisms for quality constraint monitoring and event detection described in Chapter 5 we want to investigate mechanisms to propagate updates of quality constraints to running process instances. Currently, changed quality constraints do only apply to the process instances started in the future. However, especially in long running processes an update during process execution would be desirable.

Considering the mechanisms for dynamic event and exception handling described in Chapter 6 the mapping of messages may be supported with existing schema matching systems (for instance, [DO2002]). Furthermore, the Web service search component of the Web-Flow MaX may be enhanced to find alternative services that not only match the given constraints but also minimize the delay of the remaining part of a process instance. This will help to further improve the execution quality of a cooperative process instance.

Regarding the algorithms for automatic action selection described in Chapter 7 we also plan to evaluate some possible enhancements of the calculation of the similarity between two log entries. For instance, strings may not only be compared for equality, but also the distance between two strings, i.e., the number of characters in which they differ, may be taken into account. Additionally, if two concepts taken from the same taxonomy differ, it may be derived how far their distance is in the taxonomy. For instance, two concepts with a parent-child-relationship may be considered less similar than two concepts with a brotherhood relationship. This similarity may be derived automatically using algorithms which analyze the hierarchy of taxonomy entries.

The last issue for future work are further evaluations of the Web-Flow prototype described in Chapter 8. Therefore, Web-Flow should be installed and used for continuous process monitoring to get a significant amount of data. We want to derive some results describing the quality and usefulness of the recommendations and rules generated from analysis results. Therefore, user interaction is needed, first to derive rules and then to rate the quality.

With a significant number of process definitions for which quality constraints have been defined by administrators we may also derive whether the scopes of quality constraints supported by Web-Flow, namely global, operation specific, process specific, service specific, and context specific, are useful. Therefore, we may for instance analyze how often each type of scope is used. Furthermore, it may be possible to derive which scope is applicable for which type of constraint and to formulate some recommendations for the administrator defining quality constraints. Additionally, users may rate the effort to define quality constraints, perform log data analysis, and generate rules.
References

[A]


References

[B]

[C]
<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Journal/Publication Details</th>
</tr>
</thead>
</table>

[D]-[E]

<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Journal/Publication Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIS2004</td>
<td></td>
<td>eXist: Open Source native XML database.</td>
<td>Available at <a href="http://www.exist-db.org">http://www.exist-db.org</a></td>
</tr>
</tbody>
</table>

205
References

[F]


[G]


References


[H]


[I]-[J]


References


JAVA2004 Java Technology. Available at: http://java.sun.com


[K]


References


[L]


[M]-[N]


References


MYSQ2004 MySQL. Available at http://www.mysql.com


[O]-[P]


References


[R]
ROSE2004 RosettaNet. Available at http://www.rosettanet.org

[S]
References


[T]-[V]


UNSP2004 Universal Standard Products and Services Classification. Available at: http://www.unspsc.org


W3C2000 W3C: Hyper Text Transfer Protocol. Available at http://www.w3.org/Protocols/
W3C2001b W3C: URLs, URLs, and URNs: Clarification and Recommendations 1.0. Available at http://www.w3.org/TR/uri-clarification/

[X]-[Z]
Figures

Fig. 1-1: Collaborative fulfillment (swimlane presentation). ............................. 2
Fig. 1-2: Different categories for ensuring quality of cooperative process execution. .... 4
Fig. 1-3: Web-service-oriented architecture. 7 ..............................................

Fig. 2-1: Classification of approaches for ensuring execution quality. ................. 16
Fig. 2-2: Printer supplier procurement process represented as structured activities in BPEL. 23
Fig. 2-3: Scope of activities in BPEL. ......................................................... 25
Fig. 2-4: Screenshot from the AdaptFlow worklist client. .................................. 31
Fig. 2-5: Alignment of related work with categories for execution quality support. .... 36

Fig. 3-1: Web-Flow architecture. ............................................................... 41
Fig. 3-2: Fragment of collaborative fulfillment process to illustrate Web-Flow event detection and handling. ................................................................. 45
Fig. 3-3: Two sample quality constraints for collaborative fulfillment process. .......... 46
Fig. 3-4: Temporal impacts of event handling for two sample quality constraints. .... 51
Fig. 3-5: Comparison of Web-Flow to the categories for execution quality support. .... 53

Fig. 4-1: Main classes of the Web-Flow metamodel in UML. ............................ 57
Fig. 4-2: Illustration of metadata classes. ...................................................... 58
Fig. 4-3: Quality constraint types. ............................................................... 63
Fig. 4-4: Illustration of the same context in different control flow constructs. .......... 65
Fig. 4-5: Event classes supported in Web-Flow metamodel. ............................... 74
Fig. 4-6: Action types supported by Web-Flow. ............................................ 77
Fig. 4-7: Organization of documents in Web-Flow MaX metadata repository. ......... 82
Fig. 4-8: Monitor purchasing service with semantic attributes described in WXS. .... 84

Fig. 5-1: Overview of event detection in Web-Flow. ...................................... 89
Fig. 5-2: Interception of messages. ............................................................. 91
Fig. 5-3: Correlation of messages with process instances. .................................. 92
Figures

Fig. 5-4: Scenario for an ambiguous identification of activities.
Fig. 5-5: Example for a WXS message representing a manual fault message.
Fig. 5-6: Algorithm for initialization of constraint monitoring.
Fig. 5-7: Example for a context description in WXS.
Fig. 5-8: Example for the context described in Fig. 5-7.
Fig. 5-9: Attributes used in WXS to determine valid quality constraints.
Fig. 5-10: Sample response time constraint.
Fig. 5-11: Sample result constraint.
Fig. 5-12: Dependencies between input and output messages of different activities.

Fig. 6-1: Overview of event handling in Web-Flow.
Fig. 6-2: Illustration of ambiguous Web service response messages.
Fig. 6-3: Overview over Web-Flow exception handling actions.
Fig. 6-4: WXS description of a repeat service call action.
Fig. 6-5: Illustration of repeat-service-call operator.
Fig. 6-6: WXS description of a suspend service call action.
Fig. 6-7: Illustration of first variant of suspend-service-call operator.
Fig. 6-8: Illustration of second variant of suspend-service-call operator.
Fig. 6-9: Illustration of first variant of alternative-service-call operator.
Fig. 6-10: WXS description of an alternative service call action of the first type.
Fig. 6-11: WXS description of an alternative service call action of the second type.
Fig. 6-12: Illustration of second and third variant of alternative-service-call operator.
Fig. 6-13: WXS description of a procedure call action.
Fig. 6-14: Illustration of replacement of activities executed by operator procedure-call.
Fig. 6-15: WXS description of an abort process action.
Fig. 6-16: Illustration of operator abort-process.

Fig. 7-1: Structure of a Web-Flow log entry.
Fig. 7-2: Calculation of similarity between a new and an existing log entry.
Fig. 7-3: Comparison of two semantic service descriptions.
Fig. 7-4: WXS description of a rule.

Fig. 8-1: Technologies and software used in the Web-Flow prototype.
Fig. 8-2: Screenshot of Web-Flow MaX constraint definition component.
Fig. 8-3: Screenshot of manual-reaction dialogue of Web-Flow MaX event handling comp.
Tables

Tab. 1-1: Comparison of the cooperation models. ........................................ 9
Tab. 4-1: Comparison operators supported in Web-Flow. .................................. 67
Tab. 5-1: Dependencies between evaluation of quality constraints and events. ............. 108
Tab. 7-1: Similarity matrix for comparison of event types. ................................ 160
Tab. 7-2: Values of dsop,syn. ................................................................. 164
Tab. 7-3: Extract of similarity table for comparison of semantic description values and their synonyms. ......................................................... 166
Tab. 8-1: Initialization durations measured in milliseconds for 19 process instances. ....... 193
Tab. 8-2: Message evaluation durations measured in milliseconds for 19 process instances. . 194
Tab. 8-3: Probability distributions of test data sets for evaluation of action selection. .... 195
Tab. 8-4: Number of found log entries for two runs of evaluation of action selection. ...... 197
A XML Schema for WXS

<?xml version="1.0" encoding="UTF-8"?>
<!-- defines a cooperative process template not an instance -->
<element name="cooperativeProcess" type="gwxl:tCooperativeProcess"/>
<complexType name="tCooperativeProcess">
  <sequence>
    <element name="responsible" type="gwxl:tOrgUnit"/>
    <element name="activity" type="gwxl:tActivity" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="processTemplateID" type="long"/>
  <attribute name="processTargetNamespace" type="string"/>
  <attribute name="processServiceName" type="string"/>
</complexType>

<complexType name="tOrgUnit">
  <sequence>
    <element name="user" type="gwxl:tParameterPart" minOccurs="0" maxOccurs="unbounded"/>
    <element name="email" type="gwxl:tParameterPart" minOccurs="0" maxOccurs="unbounded"/>
    <element name="phone" type="gwxl:tParameterPart" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="organizationName" type="string"/>
  <attribute name="responsibleUser" type="string"/>
  <attribute name="phoneContact" type="string"/>
</complexType>

```xml
<!-- definition of a cooperative process and of contact -->
<!-- information of a partner company -->
```

```xml
</schema>
```
<attribute name="emailContact" type="string"/>
</complexType>

<!--
# definition of activities that represent the service calls
# in a cooperative process
-->

<element name="activity" type="gwxl:tActivity"/>
<complexType name="tActivity">
  <sequence>
    <element name="referenceOnOperation" type="gwxl:tReferenceOnOperation"/>
    <element name="qualityConstraint" type="gwxl:tBoolean-expr" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name="ID" type="long" use="required"/>
  <attribute name="controlFlowType" type="gwxl:tControlFlowType"/>  
  <attribute name="useGlobalConstraints" type="boolean" use="required"/>
  <attribute name="useOperationSpecificConstraints" type="boolean" use="required"/>
  <attribute name="useProcessSpecificConstraints" type="boolean" use="required"/>
</complexType>

<complexType name="tControlFlowType">
  <restriction base="string">
    <enumeration value="sequence"/>
    <enumeration value="and-split"/>
    <enumeration value="or-split"/>
    <enumeration value="join"/>
    <enumeration value="loop-start"/>
    <enumeration value="loop-end"/>
  </restriction>
</complexType>

<complexType name="tReferenceOnOperation">
  <attribute name="operationName" type="string"/>
  <attribute name="porttype" type="string"/>
  <attribute name="operationTargetNamespace" type="string"/>
  <attribute name="serviceName" type="string"/>
</complexType>
<!-- definition of Web services and operations  -->
<!-- information is extracted from the WSDL description of a -->
<!-- service respectively defined manually -->

<complexType name="tService">
  <sequence>
    <element name="provider" type="gwxl:tOrgUnit"/>
    <element name="operation" type="gwxl:tOperation" maxOccurs="unbounded"/>
    <element name="serviceCategory" type="gwxl:tSemanticsType" minOccurs="0" maxOccurs="1"/>
    <element name="qualityConstraint" type="gwxl:tBoolean-expr" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name="name" type="string"/>
  <attribute name="namespace" type="string"/>
</complexType>

<complexType name="tOperation">
  <sequence>
    <element name="parameters" type="gwxl:tParameter" minOccurs="0" maxOccurs="unbounded"/>
    <element name="qualityConstraint" type="gwxl:tBoolean-expr" minOccurs="0" maxOccurs="1"/>
    <element name="opFunctionality" type="gwxl:tSemanticsType" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name="name" type="string"/>
  <attribute name="opID" type="long"/>
  <attribute name="porttype" type="string"/>
</complexType>

<complexType name="tParameter">
  <sequence>
    <element name="paramRole" type="gwxl:tSemanticsType" minOccurs="0" maxOccurs="1"/>
    <element name="messageParts" type="gwxl:tMessagePart" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="message" type="string"/>
  <attribute name="paramType" type="gwxl:tParameterType"/>
</complexType>
<complexType name="tMessagePart">
  <sequence>
    <element name="messagePart" type="string"/>
    <element name="partRole" type="gwxl:tSemanticsType" minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>

<complexType name="tParameterType">
  <restriction base="string">
    <enumeration value="OutputParameter"/>
    <enumeration value="InputParameter"/>
    <enumeration value="FaultParameter"/>
  </restriction>
</complexType>

<complexType name="tPrice">
  <attribute name="amount" type="float"/>
  <attribute name="currency" type="string"/>
</complexType>

<complexType name="tSemanticsType">
  <sequence>
    <element name="description" type="gwxl:tSemanticsDescription" minOccurs="1" maxOccurs="1"/>
    <element name="synonyms" type="gwxl:tSemanticsDescription" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="tSemanticsDescription">
  <attribute name="value" type="string" use="required"/>
  <attribute name="taxonomyURI" type="string" use="required"/>
</complexType>

<complexType name="tBoolean-expr">
  <sequence>
    <choice>
      <element name="term" type="gwxl:tTermType"/>
      <element name="andTerm" type="gwxl:tBinaryLogicOperator"/>
      <element name="orTerm" type="gwxl:tBinaryLogicOperator"/>
      <element name="notTerm" type="gwxl:tUnaryLogicOperator"/>
    </choice>
  </sequence>
</complexType>
<complexType name="tTermType">
    <choice>
        <element name="metadataConstraint" type="gwxl:tMetadataConstraint"/>
        <element name="inputConstraint" type="gwxl:tInputConstraint"/>
        <element name="resultConstraint" type="gwxl:tResultConstraint"/>
        <element name="responseTimeConstraint" type="gwxl:tResponseTimeConstraint"/>
        <element name="startTimeConstraint" type="gwxl:tStartTimeConstraint"/>
        <element name="iterationConstraint" type="gwxl:tIterationConstraint"/>
    </choice>
</complexType>

<complexType name="tBinaryLogicOperator">
    <sequence>
        <element name="expression" type="gwxl:tBoolean-expr" minOccurs="2" maxOccurs="unbounded"/>
    </sequence>
</complexType>

<complexType name="tUnaryLogicOperator">
    <sequence>
        <element name="expression" type="gwxl:tBoolean-expr"/>
    </sequence>
</complexType>

<!--    ############ ########### ############ ############ ########### -->
<!--    # definition of element qualityConstraint     # -->
<!--    ############ ########### ############ ############ ########### -->
<complexType name="tQualityConstraint">
    <choice>
        <element name="startTimeConstraint" type="gwxl:tStartTimeConstraint"/>
        <element name="responseTimeConstraint" type="gwxl:tResponseTimeConstraint"/>
        <element name="iterationConstraint" type="gwxl:tIterationConstraint"/>
        <element name="metadataConstraint" type="gwxl:tMetadataConstraint"/>
        <element name="inputConstraint" type="gwxl:tInputConstraint"/>
        <element name="resultConstraint" type="gwxl:tResultConstraint"/>
    </choice>
</complexType>
<complexType name="tContext">
    <sequence>
        <element name="forOperation" type="gwxl:tReferenceOnOperation"/>
        <element name="predecessors" type="gwxl:tPredecessors" minOccurs="0" maxOccurs="1"/>
        <element name="predecessorRange" type="integer" minOccurs="0" maxOccurs="1"/>
        <element name="successors" type="gwxl:tSuccessors" minOccurs="0" maxOccurs="1"/>
        <element name="successorRange" type="integer" minOccurs="0" maxOccurs="1"/>
        <element name="qualityConstraint" type="gwxl:tBoolean-expr" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="contextID" type="long"/>
    <attribute name="contextName" type="string"/>
</complexType>

<complexType name="tPredecessors">
    <sequence>
        <element name="precOperation" type="gwxl:tReferenceOnOperation" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="number" type="int"/>
</complexType>

<complexType name="tSuccessors">
    <sequence>
        <element name="succOperation" type="gwxl:tReferenceOnOperation" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="number" type="int"/>
</complexType>

<complexType name="tResponseTimeConstraint">
    <sequence>
        <element name="comparisonOp" type="gwxl:tCompareOp"/>
        <choice>
        <!-- type definitions for response time and start time constraints -->
    </sequence>
</complexType>
<complexType name="tIterationConstraint">
  <sequence>
    <element name="comparisonOp" type="gwxl:tCompareOp"/>
    <element name="iterationNumber" type="int"/>
  </sequence>
</complexType>

<!--   ############ ########### ############ ############ ########### -->
<!--   # definition of type iterationConstraint                      -->
<!--   ############ ########### ############ ############ ########### -->
<complexType name="tIterationConstraint">
  <sequence>
    <element name="comparisonOp" type="gwxl:tCompareOp"/>
    <element name="iterationNumber" type="int"/>
  </sequence>
</complexType>

<!--   ############ ########### ############ ############ ########### -->
<!--   # definition of type iterationConstraint                      -->
<!--   ############ ########### ############ ############ ########### -->
<complexType name="tIterationConstraint">
  <sequence>
    <element name="comparisonOp" type="gwxl:tCompareOp"/>
    <element name="iterationNumber" type="int"/>
  </sequence>
</complexType>
<complexType name="tMetadataConstraint">
  <sequence>
    <element name="metadataCondition" type="gwxl:tComparison"/>
  </sequence>
  <attribute name="name" type="string" use="required"/>
  <attribute name="scope" type="string"/>
  <attribute name="strictness" type="gwxl:tStrictness" default="high"/>
</complexType>

<complexType name="tStrictness">
  <restriction base="string">
    <enumeration value="high"/>
    <enumeration value="low"/>
  </restriction>
</complexType>

<complexType name="tInputConstraint">
  <sequence>
    <element name="parameterCondition" type="gwxl:tComparison"/>
  </sequence>
  <attribute name="name" type="string" use="required"/>
  <attribute name="scope" type="string"/>
</complexType>

<complexType name="tResultConstraint">
  <sequence>
    <element name="parameterCondition" type="gwxl:tComparison"/>
  </sequence>
  <attribute name="name" type="string" use="required"/>
  <attribute name="scope" type="string"/>
</complexType>
<!-- type definitions for comparisons of XML document parts  -->
<!-- with values or other document parts  -->
<!--  
<complexType name="tComparison">
    <choice>
        <element name="greater" type="gwxl:tBinaryRelation"/>
        <element name="less" type="gwxl:tBinaryRelation"/>
        <element name="equal" type="gwxl:tBinaryRelation"/>
        <element name="lessEqual" type="gwxl:tBinaryRelation"/>
        <element name="greaterEqual" type="gwxl:tBinaryRelation"/>
        <element name="in" type="gwxl:tBinaryRelation"/>
        <element name="contains" type="gwxl:tBinaryRelation"/>
    </choice>
</complexType>

<!-- basic definition of a comparison operation  -->
<complexType name="tBinaryRelation">
    <sequence>
        <element name="leftOperand" type="gwxl:tParameterPart"/>
        <choice>
            <element name="fixedValue" type="gwxl:tValueType"/>
            <element name="rightOperand" type="gwxl:tParameterPart"/>
        </choice>
    </sequence>
</complexType>

<!-- references a part of an variable XML document  -->
<complexType name="tParameterPart">
    <sequence>
        <element name="xPathQuery" type="string"/>
        <element name="parameter" type="gwxl:tParameter"/>
    </sequence>
    <attribute name="dataType" type="gwxl:tDataType" use="required"/>
</complexType>

<simpleType name="tDataType">
    <restriction base="string">
        <enumeration value="string"/>
        <enumeration value="complex"/>
        <enumeration value="int"/>
        <enumeration value="float"/>
        <enumeration value="date"/>  
        <enumeration value="duration"/>
        <enumeration value="stringlist"/>
        <enumeration value="intlist"/>
        <enumeration value="floatlist"/>
        <enumeration value="datelist"/>
    </restriction>
</simpleType>
<enumeration value="durationlist"/>
</restriction>
</simpleType>

<complexType name="tValueType">
  <choice>
    <element name="stringValue" type="string"/>
    <element name="floatValue" type="float"/>
    <element name="integerValue" type="integer"/>
    <element name="dateValue" type="dateTime"/>
    <element name="durationValue" type="duration"/>
    <element name="stringList">
      <simpleType>
        <list itemType="string"/>
      </simpleType>
    </element>
    <element name="floatList">
      <simpleType>
        <list itemType="float"/>
      </simpleType>
    </element>
    <element name="integerList">
      <simpleType>
        <list itemType="integer"/>
      </simpleType>
    </element>
    <element name="dateList">
      <simpleType>
        <list itemType="dateTime"/>
      </simpleType>
    </element>
    <element name="durationList">
      <simpleType>
        <list itemType="duration"/>
      </simpleType>
    </element>
    <element name="dataSource" type="gwxl:tDataSource"/>
  </choice>
</complexType>

<complexType name="tDataSource">
  <sequence>
    <element name="path" type="string"/>
    <element name="query" type="string"/>
  </sequence>
</complexType>
<element name="event" type="gwxl:tEvent"/>
<complexType name="tEvent">
  <sequence>
    <choice>
      <element name="violationOfQualityConstraint">
        <complexType>
          <sequence>
            <element name="constraint" type="gwxl:tQualityConstraint"/>
          </sequence>
        </complexType>
      </element>
      <element name="webserviceFaultMessage">
        <complexType>
          <sequence>
            <element name="faultMessage" type="gwxl:tParameter"/>
            <element name="affectedActivityTemplateID" type="long"/>
            <element name="affectedOperation" type="gwxl:tReferenceOnOperation"/>
          </sequence>
        </complexType>
      </element>
      <element name="manualFaultMessage" type="gwxl:tManualEventNotification"/>
      <element name="externalEvent" type="gwxl:tExternalEvent"/>
    </choice>
    <element name="reaction" type="gwxl:tAction" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="occurredAt" type="dateTime"/>
</complexType>
<element name="tAction">
  <sequence>
    <choice>
      <element name="continueProcess" type="tContinueProcess"/>
      <element name="manualReaction" type="tManualReaction"/>
      <element name="repeatServiceCall" type="tRepeatServiceCall"/>
      <element name="alternativeServiceCall" type="tAlternativeServiceCall"/>
      <element name="abortProcess" type="tAbortProcess"/>
      <element name="procedureCall" type="tProcedureCall"/>
    </choice>
    <element name="priority" default="5" minOccurs="0" maxOccurs="1">  
      <simpleType>
        <restriction base="positiveInteger">  
          <minInclusive value="1"/>
          <maxInclusive value="10"/>
        </restriction>
      </simpleType>
    </element>
    <element name="successAction" type="boolean" minOccurs="0" maxOccurs="1"/>
    <element name="successProcess" type="boolean" minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>

<complexType name="tContinueProcess">
  <sequence>
    <element name="processName" type="string"/>
    <element name="processTemplateID" type="long"/>
  </sequence>
</complexType>

<complexType name="tManualReaction">
  <sequence>
    <element name="recipient" type="tOrgUnit"/>
    <element name="notificationMessage" type="string"/>
  </sequence>
</complexType>
<complexType name="tAlternativeServiceCall">
  <sequence>
    <element name="affectedOperation" type="gwxl:tReferenceOnOperation"
      minOccurs="1" maxOccurs="1"/>
    <element name="processInstanceID" type="string" minOccurs="1"
      maxOccurs="1"/>
    <choice>
      <element name="alternativeParamValue" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <sequence>
            <element name="parameterPart" type="gwxl:tParameterPart"/>
            <choice>
              <element name="assignValue" type="gwxl:tValueType"/>
              <element name="copyValue" type="gwxl:tParameterPart"/>
            </choice>
          </sequence>
        </complexType>
      </element>
      <element name="alternativeService" minOccurs="0">
        <complexType>
          <sequence>
            <element name="serviceOperation" type="gwxl:tReferenceOnOperation"/>
          </sequence>
        </complexType>
      </element>
      <element name="serviceProperties" minOccurs="0">
        <complexType>
          <sequence>
            <element name="metadataExpression" type="gwxl:tBoolean-expr"/>
          </sequence>
        </complexType>
      </element>
    </choice>
  </sequence>
</complexType>

<!--  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # -->
<!--  definition of action AlternativeServiceCall  # # # -->
<!--  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # -->
<complexType name="tRepeatServiceCall">
  <sequence>
  </sequence>
</complexType>
<element name="serviceOperation" type="gwxl:tReferenceOnOperation"/>
<element name="processInstanceID" type="string" minOccurs="1"
maxOccurs="1"/>
</sequence>
</complexType>

<!--  ############ ########### ############ ############ ########### -->
<!--  ############ definition of action SuspendServiceCall  ########### -->
<!--  ############ ########### ############ ############ ########### -->
<complexType name="tSuspendServiceCall">
<sequence>
<element name="serviceOperation" type="gwxl:tReferenceOnOperation"/>
<element name="processInstanceID" type="string" minOccurs="1"
maxOccurs="1"/>
<element name="suspensionInterval" type="duration"/>
</sequence>
</complexType>

<!--  ############ ########### ############ ############ ########### -->
<!--  ############ definition of action abortProcess      ########### -->
<!--  ############ ########### ############ ############ ########### -->
<complexType name="tAbortProcess">
<sequence>
<element name="processName" type="string" minOccurs="1"
maxOccurs="unbounded"/>
<element name="processInstanceID" type="string" minOccurs="1"
maxOccurs="unbounded"/>
</sequence>
</complexType>

<!--  ############ ########### ############ ############ ########### -->
<!--  ############  definition of action procedureCall    ########### -->
<!--  ############ ########### ############ ############ ########### -->
<complexType name="tProcedureCall">
<sequence>
<element name="procedure" type="string"/>
<element name="parameter" type="string" minOccurs="0"
maxOccurs="unbounded"/>
<element name="processInstanceID" type="string" minOccurs="1"
maxOccurs="unbounded"/>
</sequence>
</complexType>
<complexType name="tRule">
  <sequence>
    <element name="event" type="gwxl:tEvent" minOccurs="1" maxOccurs="1"/>
    <element name="action" type="gwxl:tAction" minOccurs="1" maxOccurs="1"/>
    <element name="condition" type="gwxl:tCondition" minOccurs="0" maxOccurs="1"/>
    <element name="operation" type="gwxl:tReferenceOnOperation" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name="ruleID" type="long"/>
</complexType>

<complexType name="tCondition">
  <choice>
    <element name="qualityConstraint" type="gwxl:tQualityConstraint" minOccurs="1" maxOccurs="1"/>
    <element name="context" type="gwxl:tContext" minOccurs="1" maxOccurs="1"/>
  </choice>
</complexType>

<element name="manualEventNotification" type="gwxl:tManualEventNotification"/>
<complexType name="tManualEventNotification">
  <choice>
    <element name="serviceFailed" type="gwxl:tServiceFailedMessage"/>
    <element name="serviceDelay" type="gwxl:tServiceDelayMessage"/>
    <element name="providerOutOfBusiness" type="gwxl:tOutOfBusinessMessage"/>
    <element name="otherEvent" type="gwxl:tOtherEvent"/>
  </choice>
</complexType>
<complexType name="tServiceFailedMessage">
  <sequence>
    <element name="operation" type="gwxl:tReferenceOnOperation"/>
    <element name="affectedActivityTemplateID" type="long"/>
    <element name="processID" type="long"/>
  </sequence>
</complexType>

<complexType name="tServiceDelayMessage">
  <sequence>
    <element name="operation" type="gwxl:tReferenceOnOperation"/>
    <element name="affectedActivityTemplateID" type="long"/>
    <element name="processID" type="long"/>
    <element name="delay">
      <complexType>
        <choice>
          <element name="for" type="duration"/>
          <element name="until" type="dateTime"/>
        </choice>
      </complexType>
    </element>
  </sequence>
</complexType>

<complexType name="tOutOfBusinessMessage">
  <sequence>
    <element name="operation" type="gwxl:tReferenceOnOperation"/>
    <element name="affectedActivityTemplateID" type="long"/>
    <element name="processID" type="long"/>
    <element name="provider" type="gwxl:tOrgUnit"/>
    <choice>
      <element name="for" type="duration"/>
      <element name="until" type="dateTime"/>
    </choice>
  </sequence>
</complexType>
<complexType name="tOtherEvent">
    <sequence>
        <element name="operation" type="gwxl:tReferenceOnOperation"/>
        <element name="affectedActivityTemplateID" type="long"/>
        <element name="message" type="string"/>
    </sequence>
</complexType>

<!-- definition of element externalEvent -->
<complexType name="tExternalEvent">
    <sequence>
        <element name="dataSource" type="string"/>
        <element name="notificationMessage" type="string"/>
    </sequence>
</complexType>

<!-- definition of Web service calls -->
<complexType name="tServiceCall">
    <sequence>
        <element name="processID" type="long" minOccurs="1" maxOccurs="1"/>
        <element name="processServiceName" type="string" minOccurs="1" maxOccurs="1"/>
        <element name="providerName" type="string" minOccurs="1" maxOccurs="1"/>
        <element name="serviceName" type="string" minOccurs="1" maxOccurs="1"/>
        <element name="serviceNamespace" type="string" minOccurs="1" maxOccurs="1"/>
        <element name="serviceCategory" type="gwxl:tSemanticsType" minOccurs="1" maxOccurs="1"/>
        <element name="activityID" type="long" minOccurs="1" maxOccurs="1"/>
        <element name="referencedOperation" type="gwxl:tReferenceOnOperation" minOccurs="1" maxOccurs="1"/>
        <element name="opFunctionality" type="gwxl:tSemanticsType" minOccurs="0" maxOccurs="0"/>
    </sequence>
</complexType>
maxOccurs="1"/>

<element name="opParameters" type="gwxl:tParameter" minOccurs="0"
  maxOccurs="unbounded"/>

<element name="startTime" type="dateTime" minOccurs="1" maxOccurs="1"/>
<element name="endTime" type="dateTime" minOccurs="1" maxOccurs="1"/>

<!-- regular service call or part of an exception handling ? -->
<element name="regularOrExceptionCall" type="gwxl:tRegularOrExceptionCall"
  minOccurs="1" maxOccurs="1"/>

<element name="occurredEvent" type="gwxl:tEvent" minOccurs="0"
  maxOccurrs="unbounded"/>
</sequence>

<attribute name="serviceCallID" type="long"/>
</complexType>

<simpleType name="tRegularOrExceptionCall">
  <restriction base="string">
    <enumeration value="regular"/>
    <enumeration value="exception"/>
  </restriction>
</simpleType>
</schema>
B ODL Class Definitions for Web-Flow Metamodel

class Organization
{
attribute string organizationName;
attribute string responsibleUser;
attribute string phoneContact;
attribute string emailContact;
relationship set<CooperativeProcess> participatesIn
    inverse CooperativeProcess:isControledBy;
relationship set<Service> provides inverse Service:isProvidedBy;
relationship set<QualityConstraint> hasConstraint
    inverse QualityConstraint:isAssignedTo;
}

class CooperativeProcess
{
attribute long processTemplateID;
attribute string processTargetNamespace;
attribute string name;
attribute Organization responsibleOrganization;
relationship set<Organization> isControledBy
    inverse Organization:participatesIn;
relationship set<Activity> consistsOf inverse Activity:isPartOf;
}

class Activity
{
attribute long activityID;
attribute string name;
relationship CooperativeProcess isPartOf
    inverse CooperativeProcess:consistsOf;
relationship set<QualityConstraint> hasConstraint
    inverse QualityConstraint:isAssignedTo;
relationship ServiceOperation isExecutedBy
    inverse ServiceOperation:executes;
}
class Service
{
    attribute string name;
    attribute string serviceTargetNamespace;
    attribute string address;
    attribute enumeration category;
    relationship Organization isProvidedBy inverse Organization:provides;
    relationship set<ServiceOperation> hasOperation
        inverse ServiceOperation:isPartOf;
    relationship set<QualityConstraint> hasConstraint
        inverse QualityConstraint:isAssignedTo;
}

class ServiceOperation
{
    attribute string name;
    attribute string porttype;
    attribute string port;
    attribute enumeration functionality;
    relationship Service isPartOf inverse Service:hasOperation;
    relationship set<Activity> executes inverse Activity:isExecutedBy;
    relationship set<Rule> isPartOf inverse Rule:appliesTo;
    relationship set<Context> isSuccessor inverse Context:successor;
    relationship set<Context> hasContext inverse Context:for;
    relationship set<Context> isPredecessor inverse Context:predecessor;
    relationship set<Parameter> hasParameter inverse Parameter:of;
    relationship set<QualityConstraint> hasConstraint
        inverse QualityConstraint:isAssignedTo;
}

class Parameter
{
    attribute string name;
    attribute string datatype;
    attribute enumeration role;
    attribute enumeration type;
    attribute set<MessagePart> messageParts;
    relationship ServiceOperation of inverse ServiceOperation:hasParameter;
}

class MessagePart
{
    attribute string messagePart;
    attribute enumeration role;
}

class Context
{
    attribute long ID;
    attribute string name;
    relationship set<ServiceOperation> successor
        inverse ServiceOperation:isSuccessor;
    relationship ServiceOperation for
inverse ServiceOperation:hasContext;
relationship set<ServiceOperation> predecessor
  inverse ServiceOperation:isPredecessor;
relationship set<QualityConstraint> hasConstraint
  inverse QualityConstraint:isAssignedTo;
relationship set<Condition> isPartOf inverse Condition:isContext;
}
class QualityConstraint
{
  attribute string name;
  attribute string scope;
  relationship Activity isAssignedTo inverse Activity:hasConstraint;
  relationship Organization isAssignedTo inverse Organization:hasConstraint;
  relationship Service isAssignedTo inverse Service:hasConstraint;
  relationship ServiceOperation isAssignedTo
    inverse ServiceOperation:hasConstraint;
  relationship Context isAssignedTo inverse Context:hasConstraint;
  relationship set<Condition> isPartOf inverse Condition:isConstraint;
  relationship set<Event> isViolatedBy inverse Event:violates;
  relationship set<ConstraintExpression> belongsTo
    inverse ConstraintExpression:consistsOf;
}
class ConstraintExpression
{
  attribute string expression;
  relationship set<QualityConstraint> consistsOf
    inverse QualityConstraint:belongsTo;
}
class Rule
{
  attribute long ID;
  relationship Condition hasCondition inverse Condition:isPartOf;
  relationship Event hasEvent inverse Event:isPartOf;
  relationship Action hasAction inverse Action:isPartOf;
  relationship ServiceOperation appliesTo ServiceOperation:isPartOf;
}
class Event
{
  attribute long ID;
  attribute date occurredAt;
  relationship set<Rule> isPartOf inverse Rule:hasEvent;
  relationship set<Action> isHandledBy inverse Action:handles;
  relationship QualityConstraint violates
    inverse QualityConstraint:isViolatedBy;
}
class ExternalEvent extends Event
    { attribute string dataSource;
      attribute string notificationMessage;
    }

class ConstraintViolation extends Event
    { attribute Service|ServiceOperation|ActivityoccursFor;
      attribute string processInstanceID;
      attribute QualityConstraint constraint;
    }

class WebServiceFaultMessage extends Event
    { attribute Service|ServiceOperation|ActivityoccursFor;
      attribute string processInstanceID;
      attribute Parameter faultMessage;
    }

class ManualFaultMessage extends Event
    { attribute Service|ServiceOperation|ActivityoccursFor;
      attribute string processInstanceID;
      attribute string manualFaultMessage;
    }

class TimeoutEvent extends Event
    { attribute duration timeoutInterval;
      attribute ServiceOperation affectedOperation;
    }

class SendInputMessage extends Event
    { attribute Parameter inputMessage;
      attribute ServiceOperation operation;
      attribute string processInstanceID;
    }

class ReceiveOutputMessage extends Event
    { attribute Parameter outputMessage;
      attribute ServiceOperation operation;
      attribute string processInstanceID;
    }

class Condition
    { attribute long ID;
      relationship set<Rule> isPartOf inverse Rule:hasCondition;
      relationship QualityConstraint isConstraint
        inverse QualityConstraint:isPartOf;
      relationship Context isContext inverse Context:isPartOf;
    }
class Action
{
    attribute long ID;
    attribute integer priority;
    relationship set<Rule> isPartOf inverse Rule:hasAction;
    relationship set<Event> handles inverse Event:isHandledBy;
}

class ContinueProcess extends Action
{
    attribute string processDefinition;
    attribute string processInstance;
}

class AbortProcess extends Action
{
    attribute string processInstance;
}

class RepeatServiceCall extends Action
{
    attribute ServiceOperation operation;
    attribute string processInstanceID;
}

class AlternativeServiceCall extends Action
{
    attribute ServiceOperation affectedOperation;
    attribute set<Parameter> parameters;
    attribute ServiceOperation alternativeOperation;
    attribute set<QualityConstraint> operationProperties;
    attribute string processInstanceID;
}

class SuspendServiceCall extends Action
{
    attribute ServiceOperation operation;
    attribute duration suspensionInterval;
    attribute string processInstanceID;
}

class ProcedureCall extends Action
{
    attribute string procedure;
    attribute string processInstanceID;
    attribute set<Parameters> parameters;
}

class ManualReaction extends Action
{
    attribute Organization recipient;
    attribute string message;
}
Wissenschaftlicher Werdegang

Persönliche Angaben: Ulrike Greiner

Geburtsort: Straubing
Geburtstag: 18.01.1976

Schulbildung:
Sept. 1982 - Juli 1986 Grundschule Hirschau

Studium:
Okt. 1995 - Sept. 2000 Informatikstudium an der Universität Leipzig, Diplom
Schwerpunkt: Praktische Informatik

Diplomarbeit: „Konzeption und Implementierung eines Agenten zur ereignisorientierten Adaptation von Workflows“

Berufstätigkeit:
Nov. 2000 - Sept. 2004 Wissenschaftliche Mitarbeiterin an der Abteilung Datenbanken (Leiter: Prof. Dr. E. Rahm), Institut für Informatik, Universität Leipzig

Mitarbeit im DFG-Projekt „Wissensbasiertes dynamisches Workflow-Management in klinischen Studien“

Arbeit an Dissertation „Quality-Oriented Execution and Optimization of Cooperative Processes: Model and Algorithms“

seit Okt. 2004 Researcher Business Process Modelling and Management
SAP Research, SAP AG, Karlsruhe
Selbständigkeiterklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichen Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Eggenstein-Leopoldshafen, den 25.01.2005