

Ph.D. Thesis

Model Driven Development of Inter-organizational Workflows

Conducted for the purpose of receiving the academic title
'Doktor der technischen Wissenschaften'

Supervisor

Gerti Kappel

Institute of Software Technology and Interactive Systems [E188]

Submitted at the Vienna University of Technology

Faculty of Informatics

by

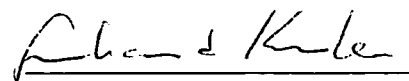
Gerhard Kramler

9255117

Feldsdorf 9

4201 Gramastetten

Vienna, May 18, 2004



Contents

Abstract	v
Zusammenfassung	vi
1 Introduction	1
1.1 Need for Model Driven Development	2
1.2 Contribution of This Thesis	3
I XML-based Technologies	5
2 Specification Languages	6
2.1 Introduction	6
2.2 Framework of Requirements	7
2.2.1 Functional Perspective	7
2.2.2 Operational Perspective	9
2.2.3 Behavioral Perspective	9
2.2.4 Informational Perspective	9
2.2.5 Interaction Perspective	10
2.2.6 Organizational Perspective	10
2.2.7 Transactional Perspective	11
2.3 Comparison of Specification Languages	11
2.3.1 WSDL - Web Service Description Language	13
2.3.2 WSFL - Web Service Flow Language	14
2.3.3 XLANG	14
2.3.4 BPML - Business Process Modeling Language	15
2.3.5 WSCL - Web Service Conversation Language	15
2.3.6 ebXML - Electronic Business XML	15
2.3.7 WPDL - Workflow Process Definition Language	16
2.4 Summary Of Results And Classification Of Languages	16
2.5 Conclusions	17

3	Comparing WSDL-based and ebXML-based Approaches	19
3.1	Introduction	19
3.2	Framework for Comparison	21
3.3	Overview of Approaches	22
3.4	Comparison	24
3.4.1	Information Items Layer	24
3.4.2	Documents Layer	25
3.4.3	Interactions Layer	25
3.4.4	Services Layer	30
3.5	Summary and Outlook	33
4	Example B2B Protocol Specification	36
4.1	Introduction	36
4.1.1	Overview of Approaches	38
4.2	Information Items Layer	39
4.2.1	WSDL-based Approach	39
4.2.2	ebXML-based Approach	39
4.3	Documents Layer	40
4.3.1	WSDL-based Approach	41
4.3.2	ebXML-based Approach	41
4.4	Interactions Layer	41
4.4.1	WSDL-based Approach	42
4.4.2	ebXML-based Approach	44
4.5	Services Layer	50
4.5.1	WSDL-based Approach	50
4.5.2	ebXML-based Approach	54
4.6	Discussion	55
II	Model Driven Development	57
5	Representing XML Schema with UML	58
5.1	Introduction	58
5.2	Transformation Rules and UML Profile	60
5.2.1	Schema Document	61
5.2.2	Complex Type Definition	61
5.2.3	Simple Type Definition	62
5.2.4	Element Declaration and Usage	64
5.2.5	Attribute Declaration and Usage	67
5.2.6	Model Group	67
5.2.7	Identity Constraint Definition	69
5.2.8	Group Definition and Reference	71
5.2.9	Annotations	71

5.2.10	Notation	72
5.3	UML Profile Implementation	72
5.3.1	Package Stereotypes	73
5.3.2	Class Stereotypes	74
5.3.3	Property Stereotypes	76
5.3.4	Generalization Stereotypes	77
5.3.5	Datatype Stereotypes	78
5.3.6	Comment Stereotypes	80
5.4	Comparison and Outlook	81
6	Approaches to Extending XML Schema	83
6.1	Introduction	83
6.2	An Overview of Active XML Schema	84
6.3	Evaluation Criteria	85
6.4	Approaches	86
6.4.1	Proprietary Schema Approach	86
6.4.2	Side by Side Approach	88
6.4.3	Framework Approach	89
6.4.4	Specialized XML Schema Approach	92
6.5	Comparison and Related Work	94
6.6	Conclusion	96
7	Outlook	97
8	Acknowledgements	99
	List of Figures	100
	List of Tables	101
	Bibliography	102
	Curriculum Vitae	110

Abstract

The rise of the web has spurred automation of cooperation among organizations. Inter-organizational workflows support such cooperations in a way similar to traditional intra-organizational workflows that support business processes within an organization. The distinct characteristics of inter-organizational workflows, such as heterogeneity and autonomy of the participating software systems, has lead to the development of several new XML-based technologies supporting inter-organizational cooperation. These technologies, however, introduce additional complexity into the development of inter-organizational workflows. Model driven development is an approach to master these complexities by using higher-level models as main development artifacts. In the model driven architecture (MDA), UML can be employed as common modelling language for models at various levels of abstraction and various technologies.

The goal of this thesis is to exploit the application of MDA for model driven development of inter-organizational workflows. In this respect, several contributions are made. First, a *survey of current XML-based technologies* is given, discussing the commonalities and differences of the various languages and identifying requirements on any modelling language supporting them as target technologies. Second, an extension of UML for *platform-specific modelling of XML documents* is defined, specifically addressing the problem of round-trip engineering. Third, different ways of *extending schema specifications for XML documents* are investigated, addressing the lack of expressiveness of XML schemas as compared to UML models.

Zusammenfassung

Das Web wird vermehrt zur automatisierten Abwicklung organisationsübergreifender Kooperationen genutzt. Inter-organisationale Workflows unterstützen solche Kooperationen in einer Weise, vergleichbar mit klassischen Workflows, welche Geschäftsprozesse innerhalb einer Organisation unterstützen. Die spezifischen Merkmale inter-organisationaler Workflows, wie Heterogenität und Autonomie der beteiligten Softwaresysteme, führten zur Entwicklung verschiedener neuer, XML-basierter Technologien zur Implementierung inter-organisationaler Workflows. Die Verwendung dieser neuen Technologien bedingt jedoch eine erhöhte Komplexität der Softwareentwicklung. Modellbasierte Softwareentwicklung ist ein Ansatz zum einfachen Umgang mit der zunehmenden Technologiekomplexität durch Verwendung von technologieunabhängigen, abstrakten Modellen als Basis für die Softwareentwicklung. Im Rahmen der Model Driven Architecture (MDA) wird UML als einheitliche Modellierungssprache für Modelle auf verschiedenen Abstraktionsebenen und für verschiedene Technologien eingesetzt.

Ziel der Dissertation ist es, die MDA für die modellbasierte Entwicklung inter-organisationaler Workflows zu adaptieren. Dazu erfolgt zunächst ein Überblick und Vergleich aktueller XML-basierter Technologien, speziell im Hinblick auf deren unterschiedliche Ausdrucksstärke und der damit verbundenen Anforderungen an eine unterstützende Modellierungssprache. Auf Basis dieser Anforderungen wird eine Erweiterung der UML zur plattformspezifischen Modellierung von XML-Dokumenten vorgenommen, welche insbesondere eine Grundlage für das Round-Trip Engineering bildet. Schliesslich werden verschiedene Möglichkeiten zur Erweiterung von XML Schema-Spezifikationen aufgezeigt, die es ermöglichen, die grössere Ausdrucksstärke von UML auch in XML abzubilden.

Chapter 1

Introduction

The idea of capturing and controlling business processes by means of computer technology is relatively old, the first systems dating back to the 70ies [100]. Mainly due to immature technology, however, it took more than 15 years, until business process automation spread beyond research communities and conquered the market as workflow management systems (WFMS) [49]. Nowadays, not least due to recent developments in object-oriented technology, WFMS are able to keep their promise of raising the productivity and competitive edge of an organization. Traditional WFMS support the design, execution, and monitoring of long lasting business processes that typically involve multiple activities and multiple collaborating resources within an organization [42, 45].

Practice has shown, however, that it is more and more required to relax the assumption that a process is executed by a single WFMS within the limits of a single organization [86, 2, 58]. A certain process may span multiple, geographically distributed WFMS employed by, e.g., multiple cooperating organizations in a value chain. Thanks to the rise of the Internet and the Web in particular, along with basic building block technologies such as XML and Web Services, the participation of several organizations in shared business processes has been made possible. Business processes are able to cross organizational boundaries to an extent never experienced before. The Web and its accompanying technologies are used for conducting global business transactions and coordinating the business processes. Workflow technology appears as a possible key to integrate Web applications in a process-centered manner. For example, collaborative telecommunication services led to the coupling of business processes of telecom partners resulting in shared workflow processes, virtual enterprises combine services from different companies leading to virtual business processes that go beyond a single enterprise boundary.

This evolution more and more turns *intra*-organisational workflows into *inter*-organizational workflows, thus focusing on workflows across organizational boundaries based on Web technology. The uniqueness of an inter-organizational workflow is characterized by several issues, including among others *heterogeneity* with respect to hardware, software, automation levels and workflow control policies and *autonomy* of the local systems leading to a lack of cross-company access to workflow resources (such as agents, tools, and information) and the missing of a complete view of the whole workflow [99].

These peculiarities of inter-organizational workflows make it necessary to explicitly specify

the interfaces of the cooperating software systems in order to achieve interoperability and loose coupling, representing the most crucial prerequisites for realizing inter-organizational workflows. So-called B2B protocols provide for the formal specification of relevant aspects of an interface, ranging from document types to transactions. Several languages for the specification of B2B protocols have been already proposed (e.g., BPEL [4] or ebXML [82]) each of them having different origins and pursuing different goals for dealing with the unique characteristics of inter-organizational workflows.

1.1 Need for Model Driven Development

To provide the basis for a sound engineering approach when developing inter-organizational workflows on basis of these B2B protocol specification languages, the employment of appropriate modelling formalisms is essential [86, 23]. Carefully designed models not only facilitate development, integration and maintenance but provide also the basis for automating at least some of the construction process itself. This idea is followed by OMG's Model Driven Architecture (MDA) [31], which could be therefore employed as the key component for facilitating the model driven development of inter-organizational workflows.

The MDA is a software development approach in which models are the key part of the definition of a software system. Abstract, technology independent models (so-called PIMs - platform independent models) are refined to more concrete models, eventually resulting in platform specific models (PSMs). Instead of handing the PSMs over to programmers for implementation, the structure and behavior of a system which is captured within a PSM is automatically transformed into executable artifacts (such as code or configuration files) [29]. With this approach, the specification of system functionality is separated from the specification of the implementation of that functionality on a specific platform. Although the standard does not prescribe a certain modelling formalism, it is recommended to apply (plain) UML for the PIM, whereas for the PSM, UML tailored to the target technology could be employed. The knowledge of the platform is encoded into transformations which are reused for many systems rather than redesigned for each new system¹, according to the motto "design once, build it on any platform".

The MDA approach promises a number of benefits including improved *portability* and *interoperability* due to separating the application knowledge from the mapping to a specific implementation technology, increased *productivity* due to automating the mapping, improved *quality* due to reuse of well proven patterns and best practices in the mapping and the possibility of validating the correctness of the models, and improved *maintainability* and *resilience to changes* caused by emerging technologies due to better separation of concerns [19, 57].

Despite of these various benefits, the application of the MDA to inter-organizational workflow has not yet been fully exploited. The current situation concerning model driven development of inter-organizational workflows can be characterized as follows. First, there is a proliferation of languages for describing inter-organizational workflows at a very low level of abstraction, most

¹Since in the MDA, automated transformations play a key role, it is important that transformations can be developed as efficiently as possible. Therefore, in April 2002, the OMG issued a Request for Proposals (RFP) for a standard syntax and execution semantics for transformations [29].

of them being XML-based. Second, some of these languages provide also some more abstract modeling formalisms, using, however, proprietary modeling languages as basis. Third, there are extensions of UML which support modeling for a particular target technology, only. The main problems resulting from this situation are the following:

- specification concepts of low-level languages are not sufficient for being employed in an abstract PIM
- proprietary specification and modelling languages do not support integration into an overall engineering process based on a standard modelling language such as UML
- the lack of an integrated modelling formalism for all languages prevents the support of business partners using different target technologies and languages
- round-trip engineering is not possible, thus hampering the integration with pre-existing B2B protocol specification languages

1.2 Contribution of This Thesis

In regard of these crucial problems, the contribution of this thesis is threefold. First, a *survey of current XML-based target technologies* is given, discussing the commonalities and differences of the various languages and identifying requirements for any modelling language supporting them as target technologies. Second, an *extension of UML for platform-specific modelling of XML documents* is defined, specifically addressing the problem of round-trip engineering. Third, different ways of *extending schema specifications for XML documents* are investigated, addressing the lack of expressiveness of XML schemas as compared to UML models.

According to this contribution, this thesis is structured into two parts, considering XML-based technologies and model driven development, respectively. Fig. 1.1 provides an overview of the considered applications of MDA to inter-organizational workflows and the relationship to the structure of this thesis.

Within Part I, Chapter 2 provides a survey of seven XML-based specification languages for inter-organizational workflows, comparing the features of the languages based on a framework of requirements for inter-organizational workflows. Chapter 3 focuses on the two most suitable and relevant target technologies for B2B protocols, i.e., Web Services and ebXML, and provides an in-depth comparison of the respective features of the two technologies. This comparison is further supported in Chapter 4 by an exemplary specification of a B2B protocol using each of the two technologies.

In Part II, new and improved concepts for model driven development of these two target technologies are presented. Chapter 5 defines an UML profile for modelling XML documents supporting round-trip engineering with XML Schema as target technology. Chapter 6 takes the problem of mapping between UML and XML Schema a step further in discussing approaches to extend XML Schema supporting modelling concepts not native to XML. Finally, Chapter 7 provides an outlook to open issues and future research.

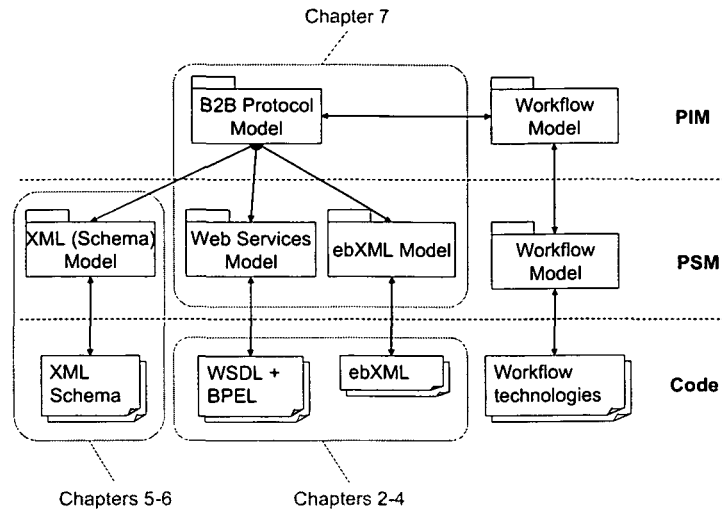


Figure 1.1: Application of MDA to inter-organizational workflows

Note that, each of the thesis' chapters is self-contained, starting with an abstract and concluding with a summary, thus representing an original contribution on its own. For this reason, some terminology used is specific to each chapter, thus documenting the evolutionary nature of this work over time. This is also the reason why the chapters can be read independent of each other. When reading them in the given order, however, a step-by-step introduction into the field is provided.

Part I

XML-based Technologies

Chapter 2

Specification of Interorganizational Workflows – A Comparison of Approaches

With the rise of the Web as the major platform for making data and services available for both, humans and applications, interorganizational workflows became a crucial issue. Several languages for the specification of interorganizational workflows have been already proposed, each of them having different origins and pursuing different goals for dealing with the unique characteristics of interorganizational workflows. This paper compares these proposals, trying to identify their strengths and shortcomings. As a pre-requisite, a framework of requirements is suggested which categorizes the major characteristics of specification languages for interorganizational workflows into different perspectives. For each of these perspectives, a set of functional requirements is proposed thereby emphasizing the difference to traditional intraorganizational workflows. On the basis of this framework, seven representative specification languages are surveyed and compared to each other.

2.1 Introduction

Workflow Management Systems are a mature technology for automating and controlling business processes [49, 42]. With the rise of the Web as the major platform for making data and services available for both, humans and applications, a new challenge has become prevalent requiring not only the support of workflows within individual organizations (called *intraorganizational workflows*), but also workflows crossing organizational boundaries referred to as *interorganizational workflows* [83, 21, 76].

Although interorganizational workflows are still very much open to research, one can identify three major characteristics which distinguish them from intraorganizational workflows and at the same time lead to several new functional requirements on specification languages, not present in the intraorganizational case. First, *interoperability* is a prerequisite for interorganizational workflows. Interoperability requires agreements on the interfaces between organizations, which provide a common understanding of the data and services exchanged. Interface standardization or interface bridging become necessary in spite of potential heterogeneity of autonomous

organizations' interfaces [96]. Second, the *autonomy* of organizations participating in an interorganizational workflow has to be considered, whereby different kinds of autonomy are relevant at different stages in the lifecycle of the workflow. These comprise design autonomy at build time, communication and execution autonomy at run time, and association autonomy at "agreement time" (cf. [74]). Finally, the *openness* of the environment leads to requirements concerning legality, trust, privacy, and security, which do not predominate in an intraorganizational environment.

For dealing with these unique characteristics, several languages for the specification of interorganizational workflows have been already proposed, each of them having different origins and pursuing different goals. This paper compares these proposals, trying to identify their strengths and shortcomings. According to that overall goal, the remainder of this paper is structured as follows. Section 2.2 presents the requirements framework along seven perspectives, emphasizing on interoperability and autonomy. Section 2.3 gives an overview of seven different specification languages and points out their distinguishing characteristics in light of the requirements framework. Section 2.4 puts the results of our comparison into perspective by presenting a classification of the languages based on the requirements framework. Section 2.5 concludes the paper by discussing the implications of our results on future work.

2.2 Framework of Requirements

In order to identify the functional requirements on languages for interorganizational workflow specification, we use a general model for the specification of workflows presented in [68]. This model comprises eight perspectives, namely functional, operational, behavioral, informational, organizational, causal, historical, and transactional. These perspectives have been derived from areas like software process modeling, organizational modeling, coordination theory, and workflow modeling [38]. As an additional perspective, we introduce *interactions*, to cope with the need for direct interactions between organizations, specifically relevant in interorganizational workflows. Note that this framework does not attempt to define a minimal set of concepts but is rather based on the union of the concepts supported by the various approaches. Figure 2.1 illustrates these perspectives and relates them to the basic building blocks of a workflow model basically using the notation of UML activity diagrams.

In the following, we will discuss the functional requirements within each of the perspectives¹, focusing especially on interoperability and autonomy issues and covering the main features of the languages compared further on.

2.2.1 Functional Perspective

The functional perspective describes *what* has to be done in a workflow, as specified by a *workflow type*. A workflow type typically includes a description of the workflow goal, of input and

¹Note that our discussion excludes both causal and historical perspective, as they are not addressed by any of the compared languages.

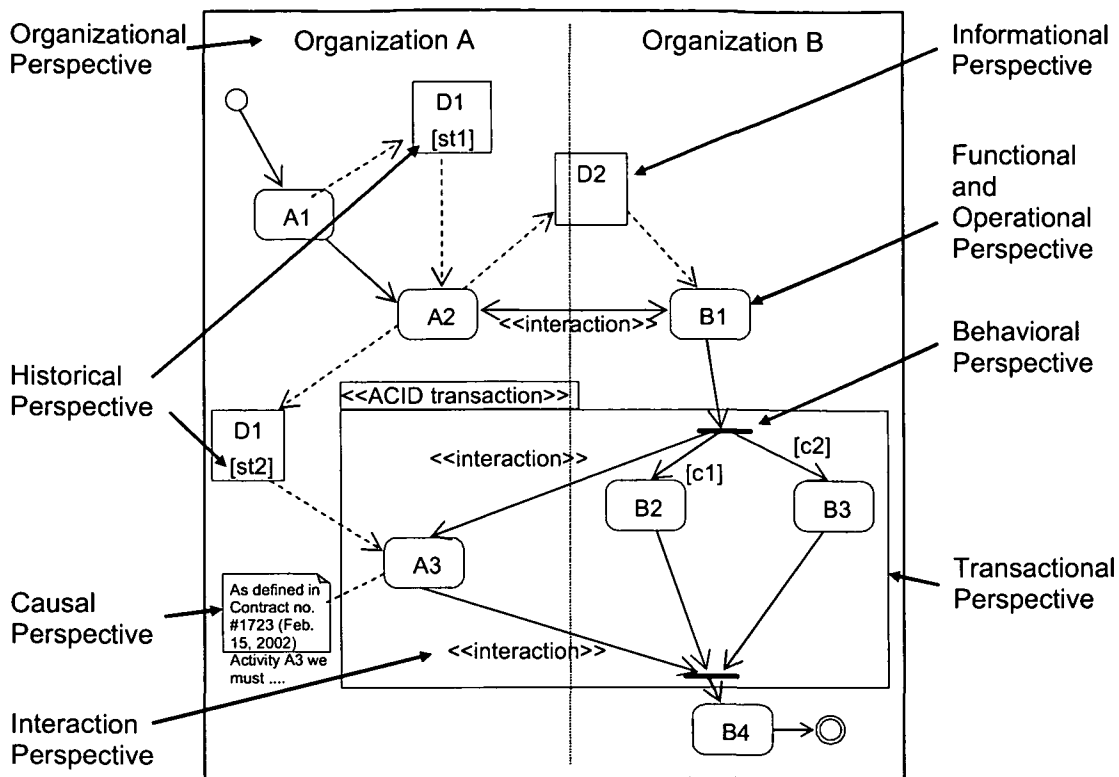


Figure 2.1: Perspectives in workflow specification

output data, additional constraints, and a decomposition into smaller units of work, which are either atomic activities, or composite subworkflows.

Interorganizational Workflow Type. A workflow type spanning multiple organizations has to make clear which activities will be performed by which of the participating organizations, in order to provide a shared understanding of the overall workflow. We call the part of an interorganizational workflow type assigned to one and the same organization *workflow fragment* [50]. Specification of workflow fragments and of interdependencies between the fragments is necessary because, unlike in intraorganizational workflows, the participants act autonomously and must coordinate themselves by means of interactions (cf. 2.2.5), rather than relying on a central workflow engine.

Information Hiding. A language for interorganizational workflow specification should be flexible enough to support both public and private workflow types (cf. [85]). A *public workflow type* is shared among collaborating organizations in order to provide a common understanding of the interorganizational workflow. It should, however, disclose only as few details as necessary of the workflow internal to the organizations, to preserve the organizations' privacy and design autonomy. A *private workflow type* is used to define the complete workflow internal to an organization, including both public visible and private activities. In order to support these requirements, a flexible information hiding mechanism is required.

Activity Semantics. For interoperability reasons, it should be possible to specify not only the decomposition of a workflow into activities, but also the semantics of these activities, e.g., by means of pre- and postconditions.

2.2.2 Operational Perspective

The operational perspective describes *how* activities are implemented.

Activity Implementation. A workflow specification language should allow to specify activity implementations, being a prerequisite for executable specifications, which in turn simplify the development of workflow systems. It should be possible, however, to exclude the implementation specification from a public workflow type, thus preserving the design autonomy of organizations.

2.2.3 Behavioral Perspective

The behavioral perspective is concerned about *when* activities have to be performed, which is defined by the control flow dependencies among activities. Specification of control flow is essential in interorganizational workflows for the coordination of workflow fragments.

Control Flow Primitives. At least the standard primitives for behavior specification of workflows should be supported. These comprise sequence, conditional execution based on transition conditions, fork and join of parallel threads, and loops based on looping conditions [17].

Timing Constraints. Constraints on duration or completion time of individual activities and of whole workflows are especially important in interorganizational workflows as a method to ensure timely execution of workflows despite execution autonomy of the participants.

Exception Handling. Different kinds of exceptions can occur and need to be handled in an interorganizational workflow. Among them are timeouts, exceptions raised by activities or communicated from an interdependent workflow fragment, and infrastructure exceptions such as communication failures. A workflow specification should define how to deal with such exceptions.

2.2.4 Informational Perspective

The informational perspective comprises data structures, and data flow between activities. In interorganizational workflows, data structures have to be specified such that both syntactic and semantic interoperability is enabled [96]. Specification of data flow must additionally consider autonomy and privacy of organizations.

Data Types. Both primitive data types and type constructors for complex data types are needed, as the data structures dealt with in interorganizational workflows are often complex business documents.

Re-useable Data Types. The ability to define re-useable libraries of data types is required, as such libraries are a prerequisite for standardization efforts.

Data Flow. A data flow specification should express which data is created and accessed by which activities, in order to facilitate both executable specifications and interoperability. If the

data flow specification also supports data transformations, the integration of heterogeneous data formats is possible.

2.2.5 Interaction Perspective

Interdependencies between workflow fragments require interactions among the participants to coordinate the execution of workflow fragments. In intraorganizational workflows, these interactions can be mediated through a central workflow engine. In interorganizational workflows, however, the participating organizations need to interact directly.

Interaction Primitives. Interaction primitives are essential as they imply the control flow and data flow between organizations or workflow fragments. Common interaction primitives, such as a request/response pair of messages, should be supported.

Interaction Implementation. A complete interaction specification must also comprise the binding of interactions to a concrete data representation, such as an XML vocabulary, and a transport protocol, such as HTTP.

Implementation Independence. Interaction primitives and the binding of primitives to a concrete interaction implementation should be separate concerns, as this allows for implementation-independent workflow types, which can be combined with different implementations, thereby increasing re-usability.

2.2.6 Organizational Perspective

The organizational perspective is concerned about *who* participates in which role in a workflow. In contrast to traditional workflows, in interorganizational workflows the participants are autonomous organizations. The difference arising is that interorganizational workflows cannot build upon a centralized organizational model, which defines the roles of and relationships between all organizations.

Roles. Support for roles is required in order to define workflow types that do not contain references to any particular organization, thus being re-useable across organizations.

Profiles. Organization profiles should be supported, i.e., a description of the capabilities of a particular organization in terms of which roles the organization can play in which workflow types. Profiles are intended to be shared with (potential) collaborating organizations, e.g., by publishing them in a registry thus facilitating future collaboration.

Agreements. Once organizations agree on collaborating in a certain interorganizational workflow for a certain time, this should be formally documented to allow, e.g., for authorization checks. Therefore, a language for specification of interorganizational workflows should be able to specify the workflow-related aspects of interorganizational agreements. Note that the process of reaching agreements is out of the scope of this paper.

Dynamic Participation. Some interorganizational workflows require a flexible decision at run time regarding the organizations that will participate in the workflow. For example, in a purchase workflow, the shipping service provider could be selected depending on the location of the customer, the quantity of ordered goods, and the time to deliver. In such cases, the workflow type must specify how to dynamically select organizations, either based directly on workflow

data, or by querying a registry. Although this requirement is not specific to interorganizational workflows, special issues arise, such as dynamic interoperability, and legal aspects.

2.2.7 Transactional Perspective

The transactional perspective describes which parts of a workflow exhibit which transactional properties. Requirements regarding specification of interorganizational transactions differ considerably from those regarding intraorganizational transactions.

Intraorganizational Transactions. Considering specification of transactions within workflow fragments, different transactional models for activities and even whole workflow fragments should be supported, ranging from models based on traditional ACID properties to extended transaction models relaxing those properties thus being suitable for long running workflows [75].

Interorganizational Transactions. For the specification of transactions spanning organization boundaries, a transaction model with loosely coupling semantics should be supported, such that both, design autonomy and execution autonomy are respected. Note that distributed transactions based on ACID or extended transaction models are not suitable, as they require the participants to take part in a two-phase-commit protocol thus leading to tight coupling [20]. In this respect, the notion of interoperable transactions has been proposed as a more suitable technique [97].

2.3 Comparison of Specification Languages

Based on our requirements framework, this section presents the results of our comparison of seven languages for interorganizational workflow specification: WSDL, WSFL, ebXML, BPML, XLANG, BPEL, WSCL, WSCI, and WPDL. The rationale behind choosing these seven languages is that they either provide a set of interesting concepts and/or are supported by a prominent consortium. A further intent was to assort a representative mix of approaches having both, different origins and different implementations or application areas. Although WSDL and WPDL are not dedicated languages for specifying interorganizational workflows, we have included them into our comparison for the following reasons. WSDL, a representative of service specification languages, constitutes the basis for many dedicated interorganizational workflow specification languages. WPDL, a representative of traditional workflow specification languages, is used as a means to highlight the differences between traditional intraorganizational workflows and interorganizational workflows. Figure 2.2 illustrates these languages, the arcs denoting either influence relationships or indicating the concrete usage of concepts.

In the following, we will give an overview of the languages and highlight their major strengths and shortcomings. An overview of the evaluation results can be found in Table 2.1. The notation used is +, /, and −, meaning that the corresponding requirement is, respectively, fulfilled, partially fulfilled, or not fulfilled at all.

Table 2.1: Overview of Evaluation Results

		Languages						Tasks			
		WSDL	WSFL	XLANG	BPML	ebXML	WSCL	WPDL	Re-usable Wf. Types	Profiles	Implementation
Functional	Interorg. Workflow Type	-	+	+	-	+	+	/		/	
	Information Hiding	/	/	/	+	/	/	/	+	+	
	Activity Semantics	/	/	/	/	+	/	/	+	+	
Operational	Activity Implementation	-	+	+	+	-	-	+			+
Behavioral	Control Flow Primitives	-	+	/	+	+	/	+		+	+
	Timing Constraints	-	-	/	+	+	-	+		/	/
	Exception Handling	-	/	+	+	+	-	-		/	+
Informational	Data Types	+	+	+	+	+	+	+	+	+	+
	Re-usable Data Types	/	/	/	/	+	/	/	+		
	Data Flow	/	+	/	+	/	/	/			+
Interaction	Interaction Primitives	+	+	+	+	+	+	-	+	+	+
	Interaction Implementation	+	+	+	-	+	-	-		+	+
	Implement. Independence	+	+	-	+	+	+	+	+		
Organizational	Roles	+	+	-	+	+	+	+	+	+	
	Profiles	/	/	/	-	+	-	-		+	
	Agreements	-	-	/	-	+	-	-			/
	Dynamic Participation	-	+	-	+	-	-	-			/
Transactional	Intraorg. Transactions	-	-	+	+	-	-	-		/	/
	Interorg. Transactions	-	-	-	/	+	-	-		/	/

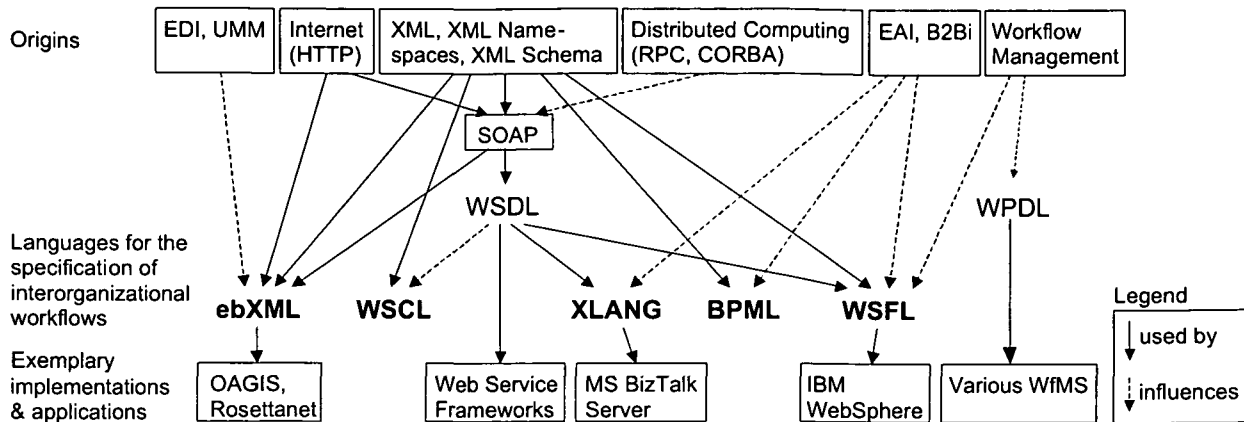


Figure 2.2: Overview of languages for interorganizational workflow specification

2.3.1 WSDL - Web Service Description Language

WSDL [91] is a language for interface specification of *web services*. A web service as specified by WSDL is a software component, accessible through the Internet. Unlike interorganizational workflows, web services are offered by individual organizations and thus do neither provide an interorganizational workflow type nor support behavior specification. The strength of WSDL is specification of interactions, and many of the languages compared herein build upon WSDL to reuse this capability. In this respect, WSDL offers two *interaction primitives*, a one-way message transmission from one workflow fragment to another, and a request/response pair of messages. Note that since WSDL specifies workflow fragments (WSDL *port type*) independent of each other, the interaction primitives come in two variants, one for the fragment initiating the interaction (*notification* and *solicit-response*), and a pendant for the fragment accepting the interaction (*one-way* and *request-response*, respectively). Concerning specification of *interaction implementation*, WSDL includes a set of protocol bindings. The SOAP binding allows to use SOAP for the specification of both, data presentation and transport protocol, which by itself provides a set of alternative choices. Additionally, a HTTP GET/POST binding and a MIME binding are directly supported by WSDL. *Interaction implementation independence* is supported by a separation of the logical specification of a workflow fragment, a WSDL *port type*, from the binding specifications and the implementation specification (*service*).

WSDL has its roots in the area of distributed computing, with ancestors such as CORBA IDL and RPC. The main differences to its ancestors are the simplicity of WSDL, and that it is based on open standards defined by the W3C, such as XML, XML Schema, XML Namespaces, and SOAP. WSDL Version 1.1 has been published as a Note by the W3C, and is broadly supported by web service implementation frameworks, such as Microsoft's .NET.

2.3.2 WSFL - Web Service Flow Language

WSFL [48] is built on top of WSDL, and extends WSDL in two orthogonal dimensions thus addressing besides interaction also most of the other perspectives. First, WSFL can be used to *refine* a WSDL service specification (`port type`) using concepts known from workflow management [49]. *Control flow* and *data flow* is specified by means of `control links` and `data links` between activities. *Activity implementations* are specified by referring to a WSDL description of a service which actually implements the activity, whereby the referred service may be provided either organization-internal (`internal`) or by another organization (`export`). The organizations providing external services are selected by so-called `locators`, which identify participating organizations either statically, or dynamically either based on workflow data or via an UDDI [79] query (cf. *dynamic participation*). The second dimension is that WSFL can be used to *compose* workflow fragments thus creating *interorganizational workflow types* in a bottom up way. The component workflow fragments may be either WSFL-refined service specifications, or pure WSDL service specifications. WSFL especially supports integration of workflow fragments with heterogeneous data structures by means of a `map` element, which uses XPath [89] expressions to specify the data transformation rules.

Version 1.0 of WSFL has been published by IBM, and is intended as contribution to a future standard in this field. WSFL is already supported by IBM products such as WebSphere Business Integrator [35].

2.3.3 XLANG

XLANG [78] is similar to WSFL, in that it allows to refine WSDL service specification with behavior, and in that it provides a means to specify compositions of WSDL services. There are, however, a number of major differences to WSFL concerning behavioral and transactional, functional, and organizational perspectives. First, XLANG uses a different approach to *behavior specification*, which is more similar to block-structured programming languages than to traditional workflow languages and provides specific support for message handling, timing, and exception handling not available in WSFL. Furthermore, XLANG supports *ACID transactions* as well as open nested transactions with compensation. The second major difference to WSFL is that XLANG focuses on specifying *public workflow types* thereby preserving the design autonomy of organizations. For instance, transition conditions and timing constraints can only be expressed by names qualified using namespaces. For the same reason, organization-internal data flow cannot be specified, and transactions are not allowed to span workflow fragments. The downside of these features is that XLANG specifications are not executable. The last difference we would like to point out is the way how WSDL specifications can be composed. An XLANG workflow fragment is defined as an extension (`XLANG:behavior`) of a WSDL service specification, which is bound to a particular network address and thereby to a specific organization. Compositions of XLANG service specifications can be defined using so-called `contracts`. Since `contracts` define the collaboration among particular organizations, they are a kind of *agreement*, whereas in WSFL compositions of service specifications are not bound to particular organizations.

The initial public draft of the specification has been published by Microsoft. XLANG is used by Microsoft's BizTalk server [55] and supported with tools for graphical modeling as well as runtime enactment.

2.3.4 BPML - Business Process Modeling Language

BPML [3] is in many respects similar to XLANG. Besides these similarities, it provides on the one hand additional concepts, such as executable specifications, transactions spanning workflow fragments, dynamic participation, and a specific information hiding mechanism, and on the other hand, lacks *interaction implementation*. Most notably, BPML supports *executable specifications* which are based on XPath, including executable transition conditions, timing constraints, and data flow specification. Furthermore, *information hiding* is best supported by a flexible visibility mechanism, which explicitly distinguishes between public workflow types (called *process abstract*) and private ones (called *process*).

Only an early draft of BPML (version 0.4) has been published by the Business Process Management Initiative (BPMI).

2.3.5 WSCL - Web Service Conversation Language

WSCL [27] is a light-weight interface specification language, with the goal "to define the minimal set of concepts necessary to specify conversations". Like XLANG, WSCL is specifically targeted at *public workflow types*. The minimalism of WSCL certainly makes the language and any implementation of it very simple, but at the same time restricts expressiveness of WSCL specifications. For instance, concerning the *organizational perspective*, WSDL limits the number of participants in an interorganizational workflow to two. Regarding the *behavioral perspective*, WSCL does not support parallel activities nor timing constraints, and transition conditions can only use the result type of a preceding activity for decisions. Despite its limitations, the simplicity of WSCL qualifies the language for a combination with WSDL in order to define stateful services [27].

WSCL has been developed by HP, derived from the Conversation Definition Language (CDL) of its now abandoned E-Speak framework. Version 1.0 has been sent to the W3C as a standardization proposal.

2.3.6 ebXML - Electronic Business XML

ebXML (<http://www.ebxml.org/>) is an initiative supported by UN/CEFACT and OASIS with the intention to create a successor to traditional EDI standards based on XML and Internet standards. A distinguishing characteristic of ebXML is that it wants to provide a complete framework for business to business transactions, and that it is based on a modeling methodology [34] for interorganizational workflows. The broad coverage of the ebXML framework is captured by a set of related specifications [28]. Specific highlights of ebXML are its support for re-useable data types, interorganizational transactions, and profiles and agreements.

Re-useable data types are addressed by the *core components* specification. A core component is a data structure with well defined semantics, which is re-useable in many domains and applications. Users of core components can adapt and compose them by means of so-called context rules and assembly rules, respectively while providing a trace back to the original semantics. The ebXML *interaction primitives*, called business transactions, are a superset of the WSDL primitives. Specifically, ebXML interaction primitives also support timing, security, and atomicity properties. Both profiles and agreements are best supported by ebXML. An ebXML CollaborationProtocolProfile (CPP) includes information identifying an organization, the workflow types and roles supported by that organization, and technical parameters concerning interactions in the supported workflow types. An ebXML CollaborationProtocolAgreement (CPA) can be derived from the intersection of two matching CPPs and includes additional information, such as status and lifetime of the agreement.

Since the ebXML project has been finished, the ebXML specifications have been adopted by various standardization efforts, either as a framework for specifying their particular vocabularies and processes, e.g., by the Open Applications Group [22], or by just using the ebXML messaging standard, like RosettaNet does. Currently, work on ebXML is continuing in follow on projects.

2.3.7 WPDL - Workflow Process Definition Language

WPDL [17] is part of a set of standards in the area of workflow management systems defined by the Workflow Management Coalition (WfMC). WPDL is intended for the exchange of workflow types between workflow management systems (WfMS), but it has not specifically designed for specification of interorganizational workflow types. Nevertheless, many concepts of WPDL can be adapted to be used in an interorganizational setting [15]. The main drawback, however, is its lack of *interaction* support. Although the necessary interactions could be derived from the control flow and data flow dependencies between workflow fragments [15], there is no standardized way for specifying interactions. Finally, it has to be noted that unlike all other languages described in this paper, WPDL is not based on XML.

2.4 Summary Of Results And Classification Of Languages

In this section, we will try to briefly summarize the results of our comparison by classifying the languages according to their suitability for three different tasks necessary when specifying interorganizational workflows in a top down manner (cf. Figure 2.3). These tasks are specification of re-usable workflow types, specification of profiles, and specification of implementation details. Each of the tasks relates to a specific subset of the requirements identified in our framework in Section 2.2, as described in Table 2.1.

First, the interorganizational workflow should be specified in a re-usable manner. It should neither contain details private to an organization, thus preserving autonomy, nor details specific to an organization, thus increasing reusability. Except XLANG, which lacks support for roles, all languages qualify for these requirements (cf. Table 2.1). Most notably, ebXML specifically supports re-useable information components and is therefore especially suited.

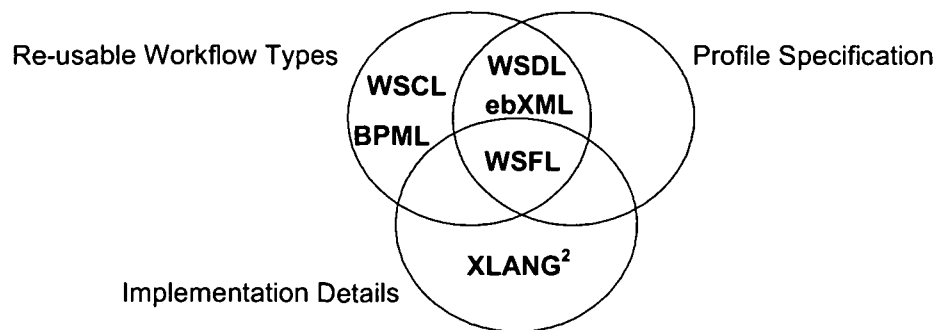


Figure 2.3: Suitability of Workflow Specification Languages

Second, as soon as a certain organization wants to announce its capability to play a specific role in an interorganizational workflow, *profile specifications* have to be added. With this, details of an organization capable of participating in the workflow are defined including the role the organization plays together with technology-specific parameters, such as network protocol and network addresses to be used for communication purposes. The qualifying languages are ebXML, and WSFL (cf. Table 1). XLANG cannot be used since it lacks support for roles, WSDL lacks control flow primitives. WSCL, BPML, and WPDL do not qualify because they neither support interaction implementation nor profiles.

Third, *implementation details* have to be added. This requires that the specification is complete, including all activities which are part of the workflow (even if they are private and not part of the profile) thus allowing for an executable specification. The only language qualifying for this task is WSFL (cf. Table 2.1). XLANG does not support executable transition conditions and intraorganizational data flow.² BPML and WPDL fail only due to the lack of interaction implementation. The remaining languages, i.e., WSDL, ebXML, and WSCL, are not intended for implementation specification, and consequently do not support most of the requirements specific to this task.

2.5 Conclusions

The comparison presented in this paper shows that no single language fulfills all requirements identified for specifying interorganizational workflows. There are two obvious approaches to cope with this problem: either some of the languages are extended such that they address all of the requirements, or the languages which are best suitable for individual tasks are combined. The first approach of extending languages would have the advantage that a uniform language could be used, thus avoiding incompatibility and integration problems. Such a “complete” language would need to be very modular in that it can be adapted to the specific requirements of each of the tasks. The second approach of combining languages would have the advantage that the

²The XLANG specification mentions executable specifications as a future extension of XLANG, although this is already supported by the BizTalk Server, a commercially available product which is based on XLANG.

existing languages and tools could be re-used. It requires, however, to make the languages compatible. While this is the case for some languages, like WSDL and WSCL, it is not for others. For instance, a combination of ebXML, which is best suited for both specification of interorganizational libraries and profiles, with one suited for implementation, such as WSFL, is tricky because the ebXML interaction patterns are not compatible with the ones used by WSFL. A more severe problem is that non of the languages suitable for implementation also support ebXML's transaction concept. Additionally, it has to be considered that an implementation language must also support the ebXML features for timing constraints and exception handling, which, e.g., is not the case with WSFL.

Our current research activities focus on employing the model driven architecture (MDA) such that, based on a platform independent model of an interorganizational workflow, it is possible to automatically derive workflow specifications expressed in the specification languages best suited for any of the different tasks.

Chapter 3

Comparing WSDL-based and ebXML-based Approaches for B2B Protocol Specification

When automating business processes spanning organizational boundaries, it is required to explicitly specify the interfaces of the cooperating software systems in order to achieve the desired properties of interoperability and loose coupling. So-called B2B protocols provide for the formal specification of relevant aspects of an interface, ranging from document types to transactions. Currently, there are two main approaches proposed for the specification of B2B protocols, the WSDL-based approach supporting Web Service languages, and the ebXML-based approach supporting languages defined along the ebXML project. Unfortunately, these approaches are not compatible, thus an organization wanting to engage in B2B collaboration needs to decide whether to embark on any of these new approaches, and which ones to use. This paper introduces a conceptual framework for B2B protocols, and based on this framework, a methodical comparison of the two approaches is provided, answering the questions of what the differences are and whether there are chances to achieve interoperability.

3.1 Introduction

The automation of business processes spanning organizational boundaries has potential. First steps towards this goal have proven highly successful, namely the use of email for communication between human agents, and the use of web applications for communication between humans and business applications published to the extranet or internet. The complete automation of business processes, however, still suffers from high implementation cost. Basically, the additional complexity is that the business applications of cooperating organizations cannot be developed independently of each other but need to be interoperable.

Interoperability of cooperating business applications which provides automation requires explicit specification of requirements and constraints on the business application's interfaces. Such specifications are referred to as B2B protocols [12], business protocols [64], public workflows

[85], or conversation processes [15]. It has to be emphasized that the specification of a B2B protocol should be separated from the specifications of workflows within organizations that support the protocol, in order to facilitate loose coupling and design autonomy of the intra-organizational workflows [13, 16].

A B2B protocol defines various aspects of an interface, such as the transport protocol, document types, security requirements, and transactional properties, to mention just a few. An organization playing a certain role in a collaboration has to support the protocol specifications in order to guarantee interoperability. If a protocol specification does not cover certain aspects, these have to be agreed on out of band by organizations willing to cooperate in order to achieve interoperability of their applications. Examples of widely used B2B protocols are EDIFACT, which covers only document types, and RosettaNet, which covers all of the above mentioned aspects.

Defining protocol specifications by means of formal languages – such as W3C’s XML Schema for the specification of document types – is beneficial in various respects. First, it enables tool support for development tasks such as consistency checks, development of data transformations, and customization of predefined protocols in a controlled manner, thus easing the adoption of a B2B protocol by an organization. Second, interpretation of the specification allows for a generic, re-useable implementation of functions such as schema validation, messaging, and security management. Finally, formal specifications which are published on the web or in specialized repositories provide the basis for automated dynamic discovery and integration with any organization supporting a matching B2B protocol.

Currently, there are two main technologies proposed for the specification of B2B protocols. Most prominently, the Web Services idea [46] subsumes a set of specification languages, with WSDL as its core and several proposed extensions, such as BPEL4WS for the specification of behavioral aspects. Although the intended application domain of Web Services is not limited to B2B protocols, B2B is considered the most prominent one. In parallel to Web Services, the ebXML initiative¹ has developed a set of standards specifically targeted at the specification of B2B protocols. Vendor support for ebXML, however, is not as strong as for Web Services. Furthermore the ebXML-based and the WSDL-based approaches are not compatible, thus an organization wanting to engage in B2B collaboration needs to decide whether to embark on any of these new technologies, and which ones to use.

There have already been efforts in comparing ebXML and Web Services. In [12], Bussler identifies the required elements of a B2B protocol, and classifies various B2B standards using the categories “business event”, “syndication”, and “supporting”. Languages for the specification of (aspects of) B2B protocols, such as ebXML and WSDL, are identified as supporting standards. No further evaluation of the classified B2B standards concerning the required protocol elements is provided. In [84], van der Aalst uses a comprehensive set of control flow and interaction patterns to evaluate the features of several languages proposed for the specification of processes, including BPEL4WS and workflow management products. The evaluation is focused on the control flow aspect, and does not include languages specific for B2B protocol specification. It is concluded that languages proposed by software vendors are often influenced by that vendors’

¹<http://www.ebxml.org/>

product interests, neglecting the real problems. In [73], Shapiro presents a detailed comparison of BPEL4WS, XPD, the WfMCs proposed standard for XML-based workflow specification languages, and BPML, a language similar in scope with BPEL. The comparison focuses on the specification of executable workflows and leaves out the concepts provided by BPEL4WS and BPML for protocol specification. Similarly, our previous work [5] provides an overview of various process specification languages including WSDL-based and ebXML-based ones, but without making a clear distinction between protocol specification and implementation specification, and without specifically highlighting the differences between WSDL-based and ebXML-based approaches. The relationship of Web Services and ebXML has also been discussed in various magazine articles. For example, [39] argues that ebXML is advantageous in typical “regulated” B2B scenarios, whereas Web Services are considered adequate for more loose collaborations without formal commitments.

This paper intends to provide further insight by presenting a methodical comparison of languages focusing on protocol specification based on a framework for the classification of protocol layers and aspects. Specifically, we aim to answer the questions of what the actual differences are between WSDL-based and ebXML-based languages, and whether there are chances to achieve interoperability. In the following section, we present the framework used to guide the comparison. Section 3.3 gives a short overview of the languages and their relationship to our framework. The detailed comparison is given in Section 3.4. Section 3.5 concludes with a summary of the comparison and an outlook to future research.

3.2 Framework for Comparison

To describe and compare the capabilities of the two approaches, this section introduces a conceptual framework to provide for common terms. The framework is based on the eCo Framework [25], which provides for a general description of e-commerce systems, and on the workflow model proposed in [68], which provides for a more specific description of workflow systems.

First, the conceptual framework is based on the eCo Framework. The latter is a layered model and can be used by businesses to define and publish descriptions about their e-commerce systems. It defines seven layers, whereby the upper three layers (i.e., the “networks”, “markets” and “business” layer) are not relevant for the description of a B2B protocol. The fourth layer, the *services layer*, is used to describe services by their interfaces which are provided and used by businesses. A service may be composed of sub-services and may invoke other services. These interactions between services are described at the *interactions layer*. It describes the types of interactions behind each service, and the types of messages which are exchanged during each interaction. A message type may contain several document types, which are described at the *documents layer*. Finally, the *information items layer* describes the types of information items that may be used in document types.

Each layer of the eCo Framework provides for the layer above and builds on the layer beneath. The layered architecture implies that an artefact defined at one layer is independent of any layer above. For example, a document type is defined independent of interactions or services it is used in. Thus it can be reused across interactions and services.

	func.	org.	info.	behav.	secur.	trans.	causal	oper.
services	X	X	X	X	–	X	–	–
interactions	X	–	X	X	X	X	–	X
documents	–	–	X	–	–	–	–	–
info. items	–	–	X	–	–	–	X	–

Table 3.1: Supported combinations of eCo layers and workflow aspects

Second, the conceptual framework is based on the workflow model proposed by Rausch-Schott [68], which describes several aspects of workflows that workflow descriptions have to cope with. While the *functional aspect* specifies what is to be executed, i.e., the semantics of a function provided by a workflow, the *operational aspect* defines how the function is implemented. The *behavioral aspect* describes how functions can be composed, e.g., as a sequence or alternative. Concentrating on data, the *informational aspect* describes data structures and data flow between functions. The *organizational aspect* describes personal and technical resources. The *transactional aspect* deals with consistency, i.e., how transactions can be used to guarantee consistent execution of functions or whole workflows. The *causal aspect* defines why a certain B2B protocol is specified in a certain way and why it is being executed. And finally, the *historical aspect* defines which data should be logged at which point in time.

While Rausch-Schott's model is intended to describe workflows that execute within a single business, all but one aspect apply for B2B protocols as well and can thus be leveraged for their characterization. The historical aspect cannot be leveraged because it describes aspects that each participating business is responsible for separately. Since B2B protocols cross business boundaries in contrast to traditional workflows, it is necessary to introduce an additional *security aspect*. It describes confidentiality, non-repudiation, integrity, authorization, and authentication.

The conceptual framework uses the layers of the eCo framework as a classification of requirements on a B2B protocol specification language, whereby each layer is refined by relevant workflow aspects. Considering relevance of combinations of aspects and layers, we consider only combinations of aspects and layers that are supported by the approaches. The respective meaning of each of these combinations has been derived from the idiosyncrasies of the ebXML-based and WSDL-based approaches and will be described along the layer-by-layer comparison of approaches in Section 3.4. Table 3.1 summarizes the supported combinations of layers and aspects.

3.3 Overview of Approaches

Each of the two approaches employs a set of specific languages for the specification of different parts of a B2B protocol. The languages employed in the WSDL-based approach have been selected from the various proposals made in the Web Services area. Since Web Service languages are developed by software vendors in loose cooperation, different options are available for certain

specification tasks. For the purpose of the comparison, we have included those languages which we consider as having the broadest support among software vendors. The languages employed in the ebXML-based approach are those developed along the ebXML project and following efforts. The comparison is based on the most recent language specifications. This section introduces the languages employed by either approach, and how these languages relate to each other and to our conceptual framework (cf. Figure 3.1).

The WSDL-based approach employs XML Schema (cf. [93, 94]) for the specification of information items. The documents layer is not supported. Interaction types are specified using WSDL (Web Service Description Language, cf. [91]) in combination with WSSP (Web Services Security Policy, cf. [36]), whereby WSSP complements WSDL in that it focuses on the security aspect. It should be noted that WSSP is in an initial public draft state, which exhibits inconsistencies. Nevertheless, it has been included in this comparison because it is the only option available for specifying the security aspect. Service types are specified using BPEL (Business Process Execution Language for Web Services, cf. [4]). Note that WSDL also supports specification of service types, but WSDL's concept of service type refers to software components, whereas BPEL specifies service types from a business case point of view, which is also the view taken in this paper.

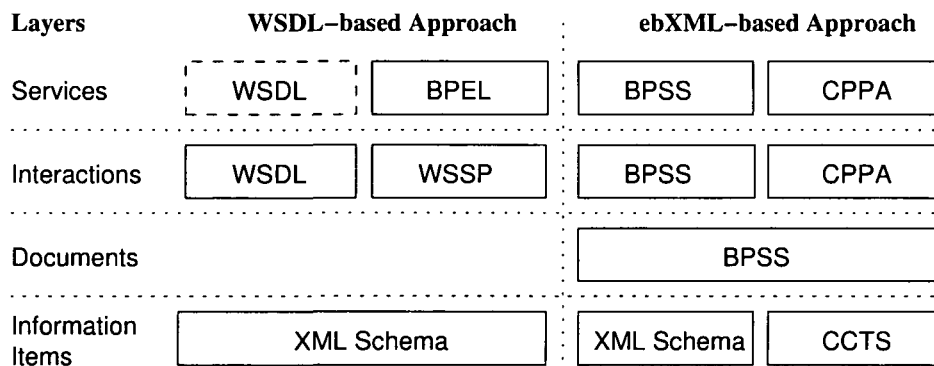


Figure 3.1: Layers of the conceptual framework and supporting languages

The ebXML-based approach also employs XML Schema for the specification of information items. Furthermore, CCTS (Core Components Technical Specification, cf. [81]) defines a methodology and language for identification of information items, which can be used in the process of defining information items. Document types are specified using BPSS (Business Process Specification Schema, cf. [82]). Interaction types are specified in terms of BPSS and CPPA (Collaboration-Protocol Profile and Agreement Specification, cf. [59]). BPSS provides for the technology- and business-independent aspects, whereas CPPA is used to supplement technology and business details. In particular, CPPA can be used to overwrite certain properties of interaction types as defined with BPSS in order to adapt them to the needs of a specific business. Service types are specified using also BPSS and CPPA. Similar to the interaction layer, CPPA can be used to adapt a service type to a specific business.

3.4 Comparison

The comparison is performed along the layers of the eCo model. Beginning at the base layer, layer by layer and aspect by aspect we will detail the conceptual framework and analyze and compare the two approaches. The approaches are described in terms of our conceptual framework, with links to the keywords of the specific languages to provide for a better understanding. Keywords are denoted in courier font using the above introduced language-specific acronyms as namespace prefix (e.g., `wsdl:message`). Note that this comparison is performed at the level of language concepts, for an example specification expressed using both approaches it is referred to Chapter 4.

3.4.1 Information Items Layer

The Information Items layer specifies re-useable data types, such as address, product code, and price, independent of their use in particular documents. Aspects of workflow modeling supported in the specification of information items are the informational and the causal aspects.

Both the WSDL-based and the ebXML-based approach support XML Schema as the preferred language for the specification of information items. We will briefly review XML Schema in the light of our conceptual framework.

The Informational Aspect is concerned with the structure and semantics of information items, including refinement of information items, composition of information items, and various constraints such as cardinality.

In this respect, XML Schema provides built-in datatypes and structuring mechanisms. XML Schema's built-in datatypes are very generic and do not provide semantics specific to the needs of B2B applications. It is therefore necessary to define more specific ones. Recently, standardization efforts in the B2B area, such as OAG² and UBL³, have begun to support XML Schema for the specification of information items, thus such standard information items can be directly employed in a WSDL-based or ebXML-based B2B protocol specification.

The Causal Aspect is concerned with the reasons behind the design of information items, i.e., the identification of influence factors such as the requirements of a specific industry or a certain country.

XML Schema does not support the causal aspect of information item specification. However, CCTS, a part of the ebXML project, addresses this aspect. CCTS defines a methodology and language for the identification and specification of so-called core components, i.e., generic information items which are independent of any particular business context such as business process, industry, and official constraints, and thus widely reusable. To make core components usable in a specific application context, they are adapted by means of restrictions and/or extensions in order to incorporate the specific requirements. As the semantics of a specific information item

²<http://www.openapplications.org/>

³<http://www.oasis-open.org/committees/ubl/>

can be derived from the semantics of the core component it is based upon, semantic expressiveness and interoperability is improved. CCTS does not specify a concrete schema language for core components and information items, which makes the methodology applicable to different technologies such as EDI and ebXML. Unfortunately, there is no standardized way to transform core components to XML Schema. As a possible solution to this problem, the UBL effort creates schemas in XML Schema for core components which have been derived from existing standard document types. UBL schemas can be directly used in both ebXML-based and WSDL-based protocol specifications.

Comparison: The two approaches are very similar as both support XML Schema. Core components are an innovative part of ebXML which could help overcome the problem of multiple competing standards by introducing common abstract information items. However, bindings of core components to specific schema definition languages are currently missing, and the core components methodology is not specific to ebXML and can be used with WSDL as well.

3.4.2 Documents Layer

Documents are containers for information items and are used to carry information in the workflow within and across businesses. Only the informational aspect is supported in the specification of document types.

In the WSDL-based approach, document types are not supported at all, meaning that message types are defined directly based on information items. In the ebXML-based approach, there is some support for document types addressing the informational aspect. A document type is specified in terms of a name and the information item contained in the document (`bpss:BusinessDocument`). The information item may be further restricted allowing for application-specific restrictions of standard information items, e.g., the status value must be “accept” (`bpss:ConditionExpression`).

Comparison: Although the WSDL-based approach does not support document types, the difference to the ebXML-based approach is small and could be overcome by using appropriately specified information items. In both approaches, documents are not considered first class objects. The approaches focus on messages instead (cf. Interactions Layer). It is therefore not possible to describe document-oriented functions such as tracking the flow of a document in a workflow.

3.4.3 Interactions Layer

An interaction is a basic exchange of messages between business partners having a certain effect. Interaction types are specified in a declarative way by means of predefined constructs supporting common interaction patterns. A typical interaction pattern is request/response, e.g., one partner sends a purchase order request, and the other responds with an acknowledgement, meaning that the order is in effect. Another typical pattern is the oneway interaction, e.g., one business partner sends a shipment notification. Several workflow aspects are supported in the specification of

an interaction type, namely the functional, informational, behavioral, security, transactional, and operational aspects.

In the WSDL-based approach, interactions have the semantics of remote procedure calls (`wsdl:operation`), i.e., the initiating partner requests some function and the responding partner performs that function and returns the result. Two kinds of interaction types are supported, namely request/response and oneway. Note that WSDL itself supports two views in the specification of interaction types, an initiator view and a responder view. When used in combination with BPEL (as described in this paper), however, only the responder view is used. Interaction types are defined in the context of an interface of a software component (`wsdl:portType`) and are thus not reusable.

In the ebXML-based approach, the guiding principle behind interactions is the so-called business transaction (`bpss:BusinessTransaction`), i.e., a request/response kind of interaction which may create or resolve a commitment between the two involved partners. Specifically, the ebXML-based approach adheres to the metamodel for business transactions defined by UMM (UN/CEFACT Modeling Methodology, cf. [80]). UMM also defines a set of so-called analysis patterns for business transactions such as Commercial Transaction and Query/Response, which can be directly used in ebXML.

The Functional Aspect defines the intention of an interaction, i.e., its goal.

In the WSDL-based approach, the functionality of an interaction is specified only in terms of a name and whether the interaction delivers a result (request/response pattern) or not (oneway pattern). As mentioned above, the functionality is defined only from the responder's view.

In the ebXML-based approach, an interaction is specified in terms of a name, informal pre- and postconditions, and whether it delivers a result. Furthermore, an interaction is decomposed into the functionality at the initiating role (`bpss:RequestingBusinessActivity`) and at the responding role (`bpss:RespondingBusinessActivity`), each of which is specified by a name.

In comparison, the ebXML-based approach provides richer support for the specification of an interaction's functionality, although not on a formal basis.

The Informational Aspect refers to the messages exchanged during an interaction. Messages are defined by a message type, which in turn specifies the documents to be included in the message.

In the WSDL-based approach, oneway interaction types comprise only one message (`wsdl:input`), whereas request/response interaction types comprise a request message (`wsdl:input`), and a number of alternative response messages (`wsdl:output` or one out of a set of named `wsdl:fault` messages). Message types are named reusable entities (`wsdl:message`). Since WSDL has no first-class concept of document type, message types are defined in terms of a list of named information items (`wsdl:part`).

In the ebXML-based approach, interaction types comprise one requesting message and optionally a number of alternative responding messages. A message type (`bpss:DocumentEnvelope`) is defined by a name, one primary document, which is defined by a document type, and additionally any number of named attachments, which can be defined

either by a document type, by a MIME type, or left unspecified. Opposed to the WSDL-based approach, message types cannot be reused across interaction types.

Overall, both approaches show only minor differences in this aspect. The WSDL-based approach provides reusable message types. The ebXML-based approach distinguishes between a primary document and attachments.

The Behavioral Aspect addresses the control flow during an interaction in terms of ordering message exchanges and of defining initiating and responding roles. Furthermore, timing and exceptions need to be considered. Typically, the behavior, i.e., the control flow of interactions is predefined and only limited means of customization are possible.

In the WSDL-based approach, the behavior of the oneway interaction type is asynchronous, i.e., the initiator sends a message to the receiver. Neither timing nor exceptions are relevant at this level of abstraction. The request/response interaction type is synchronous, i.e., the initiator sends a request message to the responder, who responds after processing the message with either a normal response or an exception message. It is not possible to specify any timing parameters.

In the ebXML-based approach, the behavior of interactions follows the UMM metamodel for business transactions, which defines the control flow as an enhanced request/response model. Basically, the initiator sends a message to the responder, the responder processes the request, and optionally sends back a response message thereby indicating success or failure of the interaction (`bpss:isPositiveResponse`). This basic model is enhanced with optional acknowledgement signals indicating receipt of the request and response messages, respectively, or indicating acceptance of the request message. Signals indicate either success or exceptional termination. A receipt acknowledgement may inform about successful schema validation (`bpss:isIntelligibleCheckRequired`), an acceptance acknowledgement informs about some further validation of the request message's content. Timeout values can be specified for both request processing (`bpss:timeToPerform`) and signalling (`bpss:timeToAcknowledgeReceipt`, `bpss:timeToAcknowledgeAcceptance`).

The interaction types supported by the WSDL-based approach match the RPC concept known from programming languages, whereas the interaction types supported by the ebXML-based approach are a superset of the WSDL ones, additionally including acknowledgement signals and timing constraints.

The Security Aspect addresses security properties of interactions, namely integrity, authenticity, and confidentiality of messages, as well as authorization and non-repudiation.

In the WSDL-based approach, only integrity, authenticity, and confidentiality of individual messages are addressed. In particular, a message type can have an attached security policy (`wssp:Policy`), which can specify integrity and authenticity (`wssp:Integrity`) and confidentiality (`wssp:Confidentiality`) requirements of selected parts of a message. Selecting parts of a message is done using XPath, which is very expressive but requires knowledge about the SOAP message format and processing model.

In the ebXML-based approach, the following security requirements can be specified for each document which is part of a message: integrity (`bpss:isTamperProof`), authenticity (`bpss:isAuthenticated`), and confidentiality (`bpss:isConfidential`). Au-

thorization (`bpss:isAuthorizationRequired`) and non-repudiation (`bpss:isNon-RepudiationRequired` and `bpss:isNonRepudiationOfReceiptRequired`, respectively) can be specified for the initiating role and the responding role.

In comparison, the WSDL-based approach supports message security in a more flexible but also more low-level way, and does not support specification of authorization and non-repudiation, the latter being also supported by the ebXML-based approach and being specifically relevant in B2B protocols.

The Transactional Aspect considers transactional properties of an interaction, such as atomicity and consistency, which are of particular importance in a distributed system without central control.

In the WSDL-based approach, transactional properties of an interaction cannot be specified explicitly. Although transaction support is addressed by several proposed protocols such as WS-Transaction⁴ and BTP⁵, the means for including them in a WSDL-based protocol specification have yet to be defined.

In the ebXML-based approach, at least the atomicity property of transactions is supported in that each interaction is considered atomic, meaning that an interaction has a defined end and in that both parties have a consistent knowledge of whether it has succeeded or failed. If it has failed, it doesn't create any commitments. It has to be mentioned that the behavior of an interaction is not sufficient to guarantee a consistent understanding about an interaction's final state, therefore a separate interaction may be necessary to notify the responder about a failure at the initiator. A detailed analysis of the differences between the ebXML behavior and two-phase distributed transaction protocols such as BTP can be found in [32]. Besides atomicity, it can be specified that an interaction must be conducted using a reliable means of communication (`bpss:isGuaranteedDeliveryRequired`), essentially regarding message exchanges as sub-transactions of an interaction.

The transactional aspect is addressed only in the ebXML-based approach in that all interactions are considered atomic. This simplifies the specification of business transactions, as these must include the specification of transactional capabilities.

The Operational Aspect considers how interactions are performed, i.e., the particular implementation-level protocols to be used for message transport and encoding, security, and transaction coordination. While in either approach some of these decisions are fix, some options can be selected in the protocol specification.

In the WSDL-based approach, several options for message transport are available, including HTTP and SOAP over HTTP (`wsdl:binding`), whereby WSDL interactions are directly mapped to HTTP interactions. Regarding message encoding (`wsdl:binding`), the available options include SOAP and SOAP with Attachments⁶. Message security is realized according to WS-Security⁷, which does not support attachments.

⁴<http://www.ibm.com/developerworks/library/ws-transpec/>

⁵<http://www.oasis-open.org/committees/tc\protect\T1\textunderscorehome.php?wg\protect\T1\textunderscoreabbrev=business-transaction>

⁶<http://www.w3.org/TR/SOAP-attachments>

⁷<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>

The ebXML-based approach defines its own messaging protocol [60]. It builds on SOAP with Attachments and provides extensions addressing reliable messaging and message security. As underlying transport protocols, HTTP, SMTP, and FTP can be used (`cppa:TransportProtocol`). The mapping between ebXML interactions and the interactions of the transport protocol can be flexibly defined, supporting both synchronous and asynchronous bindings (`cppa:CanSend`, `cppa:CanReceive`, and `cppa:syncReplyMode`). Message encoding is done using SOAP with Attachments, whereby the SOAP message is used for purposes of the ebXML messaging protocol and documents are encoded as attachments. The layout of that encoding can be specified in detail (`cppa:Packaging`). Message security is realized using S/MIME for document encryption and XML Signature for signatures carried in the SOAP message (`cppa:SenderDigitalEnvelope`, `cppa:SenderNonRepudiation`, etc.). Non-repudiation and transaction coordination are realized on top of the ebXML interaction behavior utilizing the request/response messages and the corresponding acknowledgement signals. Reliable messaging is provided by ebXML's messaging protocol and can be configured in terms of number of retries and retry interval (`cppa:ReliableMessaging`).

Both approaches use the same core technologies such as HTTP, SOAP, and XML Signature, however, differently. In particular, the ebXML-based approach defines its own SOAP extensions regarding reliable messaging and message security and supports both synchronous and asynchronous bindings to lower-level transport protocols.

Comparison:

Besides the operational differences, the most important differences between the two approaches are at the conceptual level as described. The WSDL-based approach provides only simple, generic interaction types suitable for many domains. On the contrary, in the ebXML-based approach interaction types are bundled with features such as non-repudiation and transactions, which make them specifically useful in B2B applications.

Considering interoperability, it is basically possible to express interaction types from the WSDL-based approach in terms of the ebXML-based approach, not taking into account the implied atomicity of ebXML interactions and the operational differences. Vice versa, two different approaches exist. First, interoperability can be achieved by extending WSDL interaction types with more specific features. Since this requires adaptation of existing specifications and tools of the WSDL-based approach, it is not considered viable. Second, ebXML interaction types could be expressed using existing and/or forthcoming behavior, security, and transaction features of the WSDL-based approach (cf. Services Layer). This approach allows the combination of features in very flexible ways, e.g. one could realize a transaction which involves several interactions and spans multiple business partners, which is not possible in the ebXML-based approach. The downside is that the resulting specification is much more complex and that the semantics of ebXML's business transactions is not captured explicitly. Besides conceptual interoperability as discussed above, operational interoperability could be achieved by adaptation mechanisms to translate between the different technologies, however, having conceptual interoperability as a prerequisite.

3.4.4 Services Layer

A service is the work done by a business (the service provider) that benefits another (the service consumer). For the specification of a service type, the supported workflow aspects are the functional, organizational, behavioral, informational, and transactional ones, as discussed below.

In the WSDL-based approach, service types are *unilateral*, i.e., the service functionality, behavior, etc. are specified from the service provider's point of view. In particular, service types are specified using BPEL, which builds upon WSDL interaction types and provides for the specification of so-called abstract processes (`bpel:process`). BPEL also provides concepts for the specification of so-called executable processes, which define the workflow realizing a service type, however, these are out of scope of a B2B protocol.

In the ebXML-based approach, two kinds of service types are distinguished. *Bilateral* service types are restricted to exactly two roles, i.e., service provider and service consumer (`bpss:BinaryCollaboration`). *Multilateral* service types involving many roles can be specified as a composition of bilateral service types (`bpss:MultiPartyCollaboration`). Each of these two kinds of service types address all but the informational workflow aspect.

The Functional Aspect is concerned with the work provided by the service type and its functional decomposition. Regarding decomposition, a service type can be decomposed into sub-services and ultimately into interactions as defined in the interactions layer.

In the WSDL-based approach, the functionality of a service type is specified in terms of a name and the decomposition into interactions (`bpel:invoke` and `bpel:receive/bpel:reply` for used and provided functionality, respectively). Through appropriate combination of these constructs, execution dependencies between interactions can be defined in flexible ways.

In the ebXML-based approach, bilateral service types are specified in terms of a name, informal pre- and postconditions, and the decomposition into bilateral sub-services (`bpss:CollaborationActivity`) and interactions (`bpss:BusinessTransaction-Activity`). Multilateral service types are specified in terms of a name and the decomposition into bilateral sub-services. Execution dependencies between interactions can be defined in both service types, in particular nesting of interactions is supported (`bpss:onInitiation`).

Comparing both approaches, ebXML provides a richer model supporting pre- and postconditions and recursive composition of service types, although only for bilateral service types.

The Organizational Aspect addresses the roles of businesses involved in a service type, and the authorizations and obligations of each role. Furthermore, a role's agent selection policy defines how a particular business playing that role is identified in an actual service instance.

In the WSDL-based approach, a service type specifies one primary role (`bpel:process`) and a number of secondary roles (`bpel:partner`). Only the primary role's functional, behavioral, and informational properties can be specified explicitly, whereas the corresponding properties of secondary roles are left undefined except for the compatibility requirements imposed by their relationship with the primary role. Agent selection policies are supported by means of a specific data type (`bpel:serviceReference`) which can be used in conjunction with a

data flow specification (`bpel:assign`) to define possible businesses playing a certain role, allowing to determine it dynamically at run time.

In the ebXML-based approach, bilateral service types define two roles (`bpss:Role`), which are associated with the initiating and responding roles of the interactions that constitute the service (`bpss:fromRole` and `bpss:toRole`, respectively), thereby implying the authorizations and obligations of each of the two roles in terms of functional, behavioral, informational, and transactional aspects. Multilateral service types define multiple roles (`bpss:BusinessPartnerRole`), whereby the relationship between each pair of roles is defined in terms of a bilateral sub-service. A multilateral service type can furthermore specify the coordination obligations of a role which is involved in multiple bilateral sub-services using nesting of interactions (see functional aspect). Agent selection policies are not supported in the ebXML-based approach.

In comparison, the ebXML-based approach basically supports binary relationships, which can be considered as closer related to agreements or contracts between collaborating partners, whereas the WSDL-based approach focuses on the specification of individual roles or endpoints of relationships rather than relationships, which can be considered as closer related to the implementation of the workflow supporting the service. Furthermore, the two approaches have different limitations regarding the specification of roles and agent selection policies.

The Informational Aspect is concerned with protocol relevant data used in a service type, which is defined by variables, their data types, the data flow, and message correlation, i.e., the association of messages to service instances.

In the WSDL-based approach, data used in a service type is defined local to the primary role, in terms of variables (`bpel:variable`), data types (`wsdl:message`), the data flow between variables (`bpel:assign`), and the data flow between variables and interactions (`bpel:inputVariable` and `bpel:outputVariable`). Protocol relevant data is explicitly identified using XPath expressions applied to the contents of variables (`bpel:property`). The data flow of protocol relevant data must be completely specified, whereas the flow of other application data can be specified only partially. Message correlation is defined based on a subset of the protocol relevant data which identifies a service instance in the context of an interaction (`bpel:correlationSet`).

In the ebXML-based approach neither variables nor data flow can be specified. Furthermore, message correlation does not need to be defined explicitly in the service type as it is handled by the underlying run time infrastructure.

Since the ebXML-based approach does not address the informational aspect. The WSDL-based approach on the contrary provides support, especially the specification of message correlation provides independence from operational level support for long-running sessions. Furthermore, since the complete specification of data flow is an important prerequisite for the specification of executable processes, the step from service types to executable processes is smaller than in the ebXML-based approach.

The Behavioral Aspect describes the dynamics of a service type in terms of states and the control flow between them, including conditions, timing, and exception handling.

In the WSDL-based approach, behavior of a service type is described as local to the primary role, and only the states and control flow of the primary role are explicitly defined. Behavior is specified primarily in a block-structured way using atomic and composite states. Atomic states are requested interactions (`bpel:invoke`), the begin and end of provided interactions (`bpel:receive` and `bpel:reply`), and service-internal data flow (`bpel:assign`). Composite states consist of nested atomic or composite states and are interruptible (`bpel:scope`). Control flow between states can be specified as sequential (`bpel:sequence`, `bpel:link`), parallel (`bpel:flow`), conditional (`bpel:switch`), repetitive (`bpel:while`), and as triggered by a timeout or a message receipt (`bpel:pick`, `bpel:eventHandler`). Conditions and temporal expressions are defined using XPath. Regarding exception handling, an exception can be thrown by a failed interaction or explicitly (`bpel:throw`); once thrown the containing state is interrupted and a corresponding exception handler associated with the state is activated (`bpel:catch`).

In the ebXML-based approach, the behavior of bilateral service types is defined in terms of the states and the control flow of the relationship as a whole, i.e., without taking into account that each of the role playing actors must manage its own state and control flow. The concepts provided for behavior specification are similar to those provided by UML activity diagrams. In particular, states are defined as either interactions or sub-services, both of which cannot be interrupted. Control flow between states can be specified as sequential (`bpss:Transition`), parallel (`bpss:Fork` and `bpss:Join`), conditional (`bpss:Transition` with condition), repetitive (by a cyclic graph of transitions), nested (`bpss:onInitiation`) and as triggered by a timeout of an interaction or a sub-service (`bpss:timeToPerform`). Conditions can be expressed using XPath, time durations can be either defined as constant or left undefined. Regarding exception handling, exceptions can be thrown by a failed interaction, a timeout, or explicitly by means of specific terminal states (`bpss:Failure`). Exceptional control flow is defined based on appropriate control flow conditions referring to success or failure of the state preceding a transition (`bpss:conditionGuard`). The behavior of multilateral service types is specified differently in that no global synchronized state is assumed. The behavior interrelating different sub-services is specified local to the role involved in these sub-services. In particular, the control flow between certain states of interrelated sub-services can be specified as sequential or nested (`bpss:Transition`, `bpss:onInitiation`).

In the behavioral aspect, the approaches show another big conceptual difference by specifying behavior based on a bilateral synchronized state vs. local to individual roles. Bilateral synchronized state as provided by the ebXML-based approach greatly simplifies the behavior specification as there is, e.g., no need for message triggered behavior, whereas a control flow specification for individual roles may provide higher flexibility. Regarding the control flow concepts provided, ebXML provides fewer and simpler concepts, which helps keeping simple specifications simple, whereas the WSDL-based approach better addresses exception handling, event handling, and timing.

The Transactional Aspect considers the transactional properties of a service type, such as atomicity, and the specific means to achieve them, such as compensation handlers.

In the WSDL-based approach, the transactional aspect is considered to some extent in that

a mechanism supporting the specification of transaction compensation in open nested transactions is provided (`bpel:compensationHandler`). It is, however, not possible to explicitly specify the transactional properties of a service type.

In the ebXML-based approach, a simple solution based on the atomicity property of interactions is provided in that also all bilateral service types are defined as being atomic units of work, but multilateral service types do not exhibit transactional properties. Regarding compensation, no specific concepts are supported, therefore compensating behavior must be specified using control flow mechanisms. Besides atomicity, it can be specified that an interaction has legal consequences if completed successfully (`bpss:isLegallyBinding`), which is, in some sense, a transactional property.

The ebXML-based approach offers a complete but simple solution to transaction specification in that all bilateral service types are atomic per definition, whereas the WSDL-based approach provides part of a sophisticated solution, i.e., a compensation handling mechanism, but transactional properties cannot be specified as such. Perhaps this will change in future versions when distributed transaction protocols suitable for the WSDL-based approach are available, which would enable complex distributed transactions.

Comparison:

Two general differences can be observed between the WSDL-based and the ebXML-based approach, namely unilateral vs. bilateral specification of service types, and powerful but complex vs. simplified constructs regarding the behavioral, informational, and transactional aspects.

Regarding interoperability, in general it is not possible to express WSDL-based service specifications in terms of ebXML-based ones due to ebXML's lack of expressive power in the informational and behavioral aspects. Translating ebXML-based service specifications to WSDL-based ones is possible to some extent. Such a translation would include generating unilateral specifications of bi- and multilateral ones, and generating generic data flow and message correlation specifications. The binary and multiparty relationships as well as the semantics of transactionality and business transactions, however, cannot be translated.

3.5 Summary and Outlook

We have introduced a conceptual framework for the analysis of B2B protocols, based on existing frameworks from the areas of B2B protocol specification and workflow management, respectively. Using this framework, two major approaches for specifying B2B protocols, the WSDL-based approach and the ebXML-based approach, have been analyzed and compared.

The results of the comparison show that the difference between the two approaches at the base layers of the eCo framework, i.e., information items and documents, are quite small, whereas at the higher layers, i.e., interactions and services, the approaches provide different concepts. The main differences between the two approaches are summarized in Table 3.2. As can be seen, the ebXML-based approach provides richer concepts at the interactions layer allowing for a declarative specification of many interaction characteristics relevant to business transactions. Based on this rich interaction concept, the concepts provided at the services layer are simple while being

Layers	WSDL-based Approach	ebXML-based Approach
services	unilateral, expressive (data flow, message correlation, agent selection, exceptions, events, and compensation)	primarily bilateral, simple (synchronized state), transaction atomicity
interactions	asynchronous (oneway) and synchronous (request/response) interactions	request/response with acknowledgements supporting non-repudiation and transaction atomicity, timing, synchronous and asynchronous binding, and reliable messaging
documents	not supported	rudimentary support only
info. items	none	core components methodology (not exclusive to this approach)

Table 3.2: Main distinguishing characteristics of the two approaches

sufficiently expressive. In contrast, the WSDL-based approach supports only basic interaction concepts but offers powerful concepts at the services layer. Resulting from the differences, there is no direct interoperability between the WSDL-based approach and the ebXML-based approach, neither conceptually nor operationally.

Finally, answering the question of which of the approaches to use, the ebXML-based approach is favorable for its closer alignment with the B2B domain, because it provides more specific concepts and is therefore much simpler to use while being expressive enough to cover typical B2B applications. The WSDL-based approach, on the other hand, is favorable for its much stronger vendor support and tool availability. Furthermore, it is closer aligned with existing software components which need to be integrated in the implementation of a B2B protocol in an organization.

To get best of both approaches, one could use the ebXML-based approach for the specification of a B2B protocol, and automatically generate a WSDL-based specification suitable for implementation. This would require to define a transformation from ebXML-based to WSDL-based specifications. Since the WSDL-based specification would not capture all concerns covered in the ebXML-based specification, e.g., non-repudiation, the B2B protocol would comprise both specifications. Going one step further, one could use a conceptual modeling language such as UML for the design of B2B protocols independent of the idiosyncrasies of particular specification languages, and automatically generate WSDL-based and/or ebXML-based specifications out of the UML models following the model-driven architecture approach⁸. This would require to define a UML profile for B2B protocol specification and corresponding mappings to WSDL and ebXML. Such a UML profile would likely be based on the conceptual framework used in this paper, and on already existing work such as the UMM [80] and OMG's UML Profile for Enterprise Distributed Object Computing [62]. Elaborating these ideas, as well as completing

⁸<http://www.omg.org/mda/>

the conceptual framework with the business and market layers of the eCo framework, is subject of ongoing work.

Chapter 4

Example B2B Protocol Specification Using WSDL and ebXML

The automation of business processes spanning organizations requires the formal specification of so-called B2B protocols, which define the interfaces of cooperating business software systems thereby enabling interoperability and loose coupling. Currently, there are two main approaches proposed for the specification of B2B protocols, the WSDL-based approach supporting Web Service languages, and the ebXML-based approach supporting languages defined along the ebXML project. This paper describes an example B2B protocol specification using either approach based on a common scenario, in order to provide an understanding of the respective approaches and their differences at the source code level.

4.1 Introduction

Automation of collaboration between organizations requires an explicit specification of requirements and constraints on the business application's interfaces to enable interoperability. Such specifications are referred to as B2B protocols [12], business protocols [64], public workflows [85], or conversation processes [15]. A B2B protocol defines various aspects of an interface, such as the transport protocol, document types, security requirements, and transactional properties, to mention just a few. Examples of widely used B2B protocols are EDIFACT, which covers only document types, and RosettaNet, which covers all of the above mentioned aspects. It has to be emphasized that the specification of a B2B protocol should be separated from the specifications of workflows within organizations that support the protocol, in order to facilitate loose coupling and design autonomy of the intra-organizational workflows [13, 16].

Defining protocol specifications by means of formal languages – such as W3C's XML Schema for the specification of document types – is beneficial in various respects. First, it enables tool support for development tasks such as consistency checks, development of data transformations, and customization of predefined protocols in a controlled manner, thus easing the adoption of a B2B protocol by an organization. Second, interpretation of the specification allows for a generic, re-useable implementation of functions such as schema validation, messag-

ing, and security management. Finally, formal specifications which are published on the web or in specialized repositories provide the basis for automated dynamic discovery and integration with any organization supporting a matching B2B protocol.

Currently, there are two main technologies proposed for the specification of B2B protocols. Most prominently, the Web Services idea [46] subsumes a set of specification languages, with WSDL as its core and several proposed extensions, such as BPEL4WS or WSCI for the specification of behavioral aspects. Although the intended application domain of Web Services is not limited to B2B protocols, B2B is considered the most prominent one. In parallel to Web Services, the ebXML initiative¹ has developed a set of standards specifically targeted at the specification of B2B protocols. Vendor support for ebXML, however, is not as strong as for Web Services. Furthermore the ebXML-based and the WSDL-based approaches are not compatible, thus an organization wanting to engage in B2B collaboration needs to decide whether to embark on any of these new technologies, and which ones to use.

This paper describes an example B2B protocol specification of a common example using either approach in order to provide an understanding of the respective approaches and their differences at the source code level. The example business process involves two organizations, a buyer and a seller, and two interactions, goods ordering and notifying shipment. The process proceeds in two steps. First, the buyer orders the goods from the seller, who in turn acknowledges the order. Second, if the order acknowledgement was positive, and when the ordered goods become ready to be shipped, the seller notifies the buyer about that event.

To make the the two example specifications more comparable, we employ the eCo architecture [25] for structuring the description of the example specifications. The eCo architecture is a layered model for descriptions about e-commerce systems. It defines seven layers, whereby the upper three layers (i.e., the “networks”, “markets” and “business” layer) are out of scope of the example business process. The fourth layer, the *services layer*, is used to describe services by their interfaces which are provided and used by businesses. A service may be composed of sub-services and may invoke other services. These interactions between services are described at the *interactions layer*. It describes the types of interactions behind each service, and the types of messages which are exchanged during each interaction. A message type may contain several document types, which are described at the *documents layer*. Finally, the *information items layer* describes the types of information items that may be used in document types.

The examples are explained layer by layer in a bottom-up way, starting at the information items layer, because of the layered architecture implying that an artefact defined at one layer is independent of any layer above. In particular, we will discuss for each layer what needs to be specified and how it can be done using the WSDL-based and the ebXML-based approach, respectively, by presenting excerpts from source code and corresponding explanations. In the remainder of this section, we present a short overview of the languages employed in the two approaches. In the following sections, the example specifications are presented layer by layer. The paper concludes with a discussion of the results.

¹<http://www.ebxml.org/>

4.1.1 Overview of Approaches

Each of the two approaches employs a set of specific languages for the specification of different parts of a B2B protocol. The languages employed in the WSDL-based approach have been selected from the various proposals made in the Web Services area. Since Web Service languages are developed by software vendors in loose cooperation, different options are available for certain specification tasks. In this example, we have used those languages which we consider as having the broadest support among software vendors. The languages employed in the ebXML-based approach are those developed along the ebXML project and following efforts. The comparison is based on the most recent language specifications. This section introduces the languages employed by either approach, and how these languages relate to each other and to the eCo architecture (cf. Figure 4.1).

The WSDL-based approach employs XML Schema (cf. [93, 94]) for the specification of information items. The documents layer is not supported. Interaction types are specified using WSDL (Web Service Description Language, cf. [91]) in combination with WSSP (Web Services Security Policy, cf. [36]), whereby WSSP complements WSDL in that it focuses on the security aspect. It should be noted that WSSP is in an initial public draft state, which exhibits inconsistencies. Nevertheless, it has been included in this comparison because it is the only option available for specifying the security aspect. Service types are specified using BPEL (Business Process Execution Language for Web Services, cf. [4]). Note that WSDL also supports specification of service types, but WSDL's concept of service type refers to software components, whereas BPEL specifies service types from a business case point of view, which is also the view taken in this paper.

Layers	WSDL-based Approach		ebXML-based Approach	
Services	WSDL	BPEL	BPSS	CPPA
Interactions	WSDL	WSSP	BPSS	CPPA
Documents			BPSS	
Information Items	XML Schema		XML Schema	CCTS

Figure 4.1: eCo layers and supporting languages

The ebXML-based approach also employs XML Schema for the specification of information items. Furthermore, CCTS (Core Components Technical Specification, cf. [81]) defines a methodology and language for identification of information items, which can be used in the process of defining information items. Document types are specified using BPSS (Business Process Specification Schema, cf. [82]). Interaction types are specified in terms of BPSS and CPPA (Collaboration-Protocol Profile and Agreement Specification, cf. [59]). BPSS provides for the

technology- and business-independent aspects, whereas CPPA is used to supplement technology and business details. In particular, CPPA can be used to overwrite certain properties of interaction types as defined with BPSS in order to adapt them to the needs of a specific business. Service types are specified using also BPSS and CPPA. Similar to the interaction layer, CPPA can be used to adapt a service type to a specific business.

4.2 Information Items Layer

The information items needed in our business process include simple information items such as order id, and issue date, and also complex information items such as order lines, and the complete order. Information items are intended to be reused in different applications in order to facilitate integration and interoperability. Consequently, existing information items should be reused as much as possible.

4.2.1 WSDL-based Approach

In the WSDL-based approach, information items are specified using XML Schema. See the ebXML-based approach below for an information item specification based on XML Schema, which can also be used in the WSDL-based approach.

4.2.2 ebXML-based Approach

In the ebXML-based approach, information items are preferably specified using XML Schema. Furthermore, a methodology for finding existing information items and defining new ones is provided by CCTS. The basic concept of this methodology are the so-called core components, i.e., generic information items which are independent of any particular business context such as business process, industry, and official constraints, and thus widely reusable. To make core components usable in a specific application context, they are adapted by means of restrictions and/or extensions in order to incorporate the specific requirements. Adapted core components are called business information entity. CCTS is primarily intended to be applied by standardization efforts.

For instance, the UBL effort² creates a library of core components based on existing standard document types (xCBL³), and also provides schemas in XML Schema for these core components. Among the information items defined by UBL is `Order`, which is also usable in our example business process. The XML Schema fragment below shows part of the definition of the UBL `Order` business information entity. As can be seen, an UBL schema provides documentation linking back to the core components implemented by the schema by means of the elements `ccts:ABIE` and `ccts:BBIE` denoting aggregate and basic business information entities, respectively. Such linking information facilitates deriving the semantics of the XML elements specified in the schema.

²<http://www.oasis-open.org/committees/ubl/>

³<http://www.xcbl.org/>

Information item specification using XML Schema as done in UBL⁴

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:bie="urn:oasis:names:tc:ubl:Order:1.0:0.70.xsd"
  xmlns:ccts="urn:oasis:names:tc:ubl:CoreComponentParameters..."
  ...>
  ...
  <xsd:element name="Order" type="OrderType" id="UBL000001"/>
  <xsd:complexType name="OrderType" id="UBL000001">
    <xsd:annotation>
      <xsd:documentation>
        <ccts:ABIE dictionaryEntryName="Order. Details"
          definition="information directly relating to
            the order."
          objectClassTerm="Order"
          propertyTerm="Details"
          representationTerm="Details"/>
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="OrderID" id="UBL000002">
        <xsd:annotation>
          <xsd:documentation>
            <ccts:BBIE dictionaryEntryName="Order. Identifier"
              definition="The OrderId element is a
                unique number assigned to the Order in
                respect to the parties assigning the
                number."
              objectClassTerm="Order"
              propertyTerm="Identification"
              representationTerm="Identifier"/>
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

4.3 Documents Layer

The document types used in our business process are order, order acceptance, order denial, and shipment notice.

⁴Excerpt from UBL_Library_0p70_Order.xsd downloaded from <http://oasis-open.org/committees/ubl/lcsc/0p70/UBL0p70.zip>

4.3.1 WSDL-based Approach

In the WSDL-based approach, specification of document types is not supported at all, meaning that interaction types are defined directly based on information items representing documents, such as the Order element shown above.

4.3.2 ebXML-based Approach

In the ebXML-based approach, document types need to be explicitly specified in terms of a name and the information item contained in the document. The information item may be further restricted allowing for application-specific restrictions of standard information items.

In the fragment of a BPSS below (element `bpss:ProcessSpecification`), a document named `PurchaseOrder` is specified (element `bpss:BusinessDocument`) in terms of a Reference to the `Order` information item as described in the information items layer. Furthermore, it is specified that the order document will contain pricing information in US dollars (element `bpss:ConditionExpression`).

Document type specification using BPSS

```
<bpss:ProcessSpecification ...>
  <bpss:BusinessDocument name="PurchaseOrder"
    nameID="PurchaseOrder_BD"
    specificationLocation="UBL_Library_0p70_Order.xsd" />
    specificationElement="Order" >
    <bpss:ConditionExpression expressionLanguage="XPath"
      expression="/Order/OrderPricingCurrencyCode = 'USD' " />
    ...
  </bpss:BusinessDocument>
  ...
</bpss:ProcessSpecification>
```

4.4 Interactions Layer

The interaction types used in our business process are order and shipment notification. The order interaction type involves the buyer sending an order to the seller and the seller processing the request and responding with either an order confirmation or an order denial. The shipment notification interaction type involves the seller sending a shipment notice to the buyer.

4.4.1 WSDL-based Approach

In the WSDL-based approach, interactions are specified using WSDL and WSSP. WSDL supports message types as first class entities, interaction types are defined based upon message types. We will first describe the specification of message types using WSDL in combination with WSSP, and, based on that, describe the specification of interaction types using WSDL.

The fragment of a WSDL schema shown below (element `wSDL:definitions`) defines a message type named `PurchaseOrder` in terms of its data structure and security policies (element `wSDL:message`). The message type comprises one message part named `order`, which in turn is defined by the XML Schema element specification `ubl:Order` as shown in the information items layer. The relationship to the specific schema is established via the `wSDL:import` element. Note that the original UBL schema doesn't define a target namespace for the `Order` element. For the purpose of this example, we assume that the element is defined in some namespace, as WSDL doesn't support unqualified references. As an extension to the WSDL message type, WSSP is used to specify a security policy named `tns:IntegrityPolicy` and attach it to the `PurchaseOrder` message type (attribute `wsp:PolicyRefs`). The `tns:IntegrityPolicy` defines that the message body will be signed using a certain algorithm and that an X509 certificate of the signer's identity is to be included. Note that the specification of the parts of the message requiring to be signed (element `MessageParts`) is an XPath expression evaluated in the context of a SOAP message's root element `SOAP:Envelope`. Therefore, this security policy depends on SOAP being used.

Message type specified using WSDL and WSSP

```
<wSDL:definitions ...>
  ...
  <wSDL:import namespace="urn:oasis:names:tc:ubl:Order:1.0:0.70"
    location="UBL_Library_0p70_Order__NS.xsd" />
  ...
  <wSDL:message name="PurchaseOrder"
    wsp:PolicyRefs="tns:IntegrityPolicy">
    <wSDL:part name="order"
      element="ubl:Order" />
  </wSDL:message>
  ...

  <wsp:UsingPolicy wSDL:Required="true"/>

  <wsp:Policy ... Name="IntegrityPolicy">
    <wssp:Integrity wsp:Usage="wsp:Required">
      <wssp:Algorithm Type="wssp:AlgCanonicalization"
        URI="http://www.w3.org/Signature/Drafts/xml-exc-c14n"/>
      <wssp:Algorithm Type="wssp:AlgSignature"
        URI="http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
      <wssp:SecurityToken>
        <wssp:TokenType>wssp:X509v3</wssp:TokenType>
      </wssp:SecurityToken>
    </wssp:Integrity>
  </wsp:Policy>

```



```
</wssp:SecurityToken>
<MessageParts
  Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
  wsp:Body() </MessageParts>
</wssp:Integrity>
</wsp:Policy>
...
```

Regarding interaction types, WSDL supports two kinds of interaction types, namely request/response and oneway. The order interaction type can be specified using a request/response kind of interaction type, whereas the shipment notification can be specified using a oneway kind.

In WSDL, interaction types are defined in the context of an interface of a software component (`wSDL:portType`). Two views are supported in the specification of interaction types, an initiator view and a responder view. When used in combination with BPEL, as described in this paper, however, only the responder view is used. In our example, this means that the interaction types have to be grouped into two port types. One for the interaction types responded to by the buyer, i.e., the shipment notification interaction type, and another port type for the interaction types responded to by seller, i.e., the order interaction type.

The WSDL fragment below shows the specification of the `Order` interaction type (element `wSDL:operation`) as part of the seller's port type (element `portType`). It is a request/response kind of interaction type, as indicated by the `wSDL:input` element defining the request message, followed by a `wSDL:output` element defining the normal response message and a `wSDL:fault` element indicating a potential error response message.

The operational details of the interaction types in a port type are specified in a so-called binding. Operational details include the transport protocol to be used, and the message encoding format. In the WSDL fragment below, a binding of the seller's port is shown (element `wSDL:binding`), which specifies that SOAP over HTTP is used as transport protocol (element `soap:binding`). Furthermore, the fragment shows the message encoding specification for the request message, defining it as a SOAP message with the order part contained as an element of the SOAP body encoded in literal form, i.e., as defined by the XML schema.

Interaction type specified using WSDL

```
<wsdl:portType name="Seller_PT">
  <wsdl:operation name="Order">
    <wsdl:input message="tns:PurchaseOrder"/>
    <wsdl:output message="tns:PurchaseOrderAcceptance"/>
    <wsdl:fault name="denied" message="tns:PurchaseOrderDenial"/>
  </wsdl:operation>
  ...

<wsdl:binding name="Seller_SOAP_PT" type="tns:Seller_PT">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <wsdl:operation name="Order">
    <wsdl:input>
      <soap:body parts="order" use="literal"/>
    </wsdl:input>
    <wsdl:output ...>
    <wsdl:fault ...>
  </wsdl:operation>
  ...

</wsdl:definitions>
```

4.4.2 ebXML-based Approach

In the ebXML-based approach, BPSS and CPPA are used in combination for the specification of interaction types. BPSS provides for the specification of abstract service types, and CPPA provides for the specification of operational details as well as configuration parameters relevant to a particular business or a particular agreement between businesses. In the following, we will first describe the use of BPSS for the specification interaction types, and then the use of CPPA.

BPSS interaction types are specified based on the metamodel for business transactions defined by UMM (UN/CEFACT Modeling Methodology, cf. [80]). UMM also defines a set of so-called analysis patterns for business transactions such as Commercial Transaction and Query/Response, which can be directly implemented in BPSS. In the BPSS fragment below, the order interaction type is defined (element `bpss:BusinessTransaction`) according to the “commercial transaction” pattern as defined by UMM⁵ with the corresponding security, timing, and reliability constraints. The specification is divided into the requesting role (`bpss:RequestingBusinessActivity`) and the responding role (`bpss:RespondingBusinessActivity`). For each of the roles, the interaction parameters are specified, as well as the request and response messages. The request message is

⁵Neither UMM nor ebXML define possible values of the attribute `pattern`. The URI used in the example is fictitious.

defined as comprising one primary document `PurchaseOrder` with certain security properties and no attachments (element `bpss:DocumentEnvelope`). Two alternative response messages are defined, one (`PurchaseOrderAcceptance`) indicating success, the other (`PurchaseOrderDenial`) indicating failure (attribute `isPositiveResponse`).

Interaction type specification using BPSS

```
<bpss:ProcessSpecification ...>
  ...
  <bpss:BusinessTransaction name="PurchaseOrderTransaction"
    nameID="PurchaseOrderTransaction_BT"
    pattern="http://ebxml.org/patt/CommercialTransaction"
    isGuaranteedDeliveryRequired="true">
    <bpss:RequestingBusinessActivity name="Offer"
      isAuthorizationRequired="true"
      isIntelligibleCheckRequired="true"
      isNonRepudiationReceiptRequired="true"
      isNonRepudiationRequired="true"
      timeToAcknowledgeAcceptance="PT6H"
      retryCount="3"
      timeToAcknowledgeReceipt="PT2H">
      <bpss:DocumentEnvelope businessDocument="PurchaseOrder"
        businessDocumentIDRef="PurchaseOrder_BD"
        isAuthenticated="true"
        isConfidential="false"
        isTamperProof="true" />
    </bpss:RequestingBusinessActivity>
    <bpss:RespondingBusinessActivity name="Accept"
      isAuthorizationRequired="true"
      isIntelligibleCheckRequired="true"
      isNonRepudiationRequired="true"
      timeToAcknowledgeReceipt="PT2H">
      <bpss:DocumentEnvelope
        businessDocument="PurchaseOrderAcceptance"
        isPositiveResponse="true"
        ... />
      <bpss:DocumentEnvelope
        businessDocument="PurchaseOrderDenial"
        isPositiveResponse="false"
        ... />
    </bpss:RespondingBusinessActivity>
  </bpss:BusinessTransaction>
  ...
```

The operational details are specified using CPPA in a so-called collaboration protocol profile (element `cppa:CollaborationProtocolProfile`), fragments of which are shown

below. CPPA does not adhere to the eCo layered architecture as operational details can only be specified in the context of a particular business or in the context of a particular agreement between businesses. Unlike with abstract interaction types, the specification of operational details of a service type is not reusable across businesses. In the following, the example specification of the seller is presented along five main components called service binding, delivery channel, transport protocol configuration, ebXML messaging protocol configuration, and message packaging specification.

The service binding (element `cppa:ServiceBinding`) specifies the operational details for each of the interactions in a service type (the service type is referenced via the `cppa:ProcessSpecification` element, its description can be found in Section 4.5). Note that this is another difference to the eCo architecture as operational details are not specified per interaction type but only per usage of an interaction type in a particular service type and in the context of a particular business. The `cppa:Role` element identifies the role within the service type, in our case the seller. Since the operational details are specified in the context of the seller, the responder's view is taken for the order interaction and the initiator's view is taken for the notify shipment interaction. The specification deals with individual incoming and outgoing messages, for example in the order interaction, there is an incoming request message (element `cppa:CanReceive`), an outgoing receipt acknowledgement message (element `cppa:CanSend`, not shown in full detail), and so on. Note that the acknowledgement message is implicitly specified by the BPSS interaction types but needs to be explicitly specified at this level. The relationship of the individual messages to BPSS interactions is defined via the `cppa:ActionContext` element. The actual operational details, i.e., the specification of the delivery channel and the message packaging to be used, are given by the element `cppa:ChannelId` the attribute `cppa:packageId`, respectively. Furthermore, nesting of `CanSend` and `CanReceive` elements could be used to express synchronous message exchanges. In our example, however, all request, acknowledgement, and response messages are exchanged asynchronously. Finally, the settings of the BPSS interaction type regarding security and timing could be overwritten, but in our example they remain unchanged (element `tp:BusinessTransactionCharacteristics`).

Service binding specification using CPPA

```
<cppa:CollaborationProtocolProfile ...>
  <cppa:PartyInfo partyName="aSeller" ...>
    ...
    <cppa:CollaborationRole>
      <cppa:ProcessSpecification cppa:version="1.0"
        cppa:name="test"
        xlink:href="test.bpss.xml"
        cppa:uuid="urn:icann:buyer.com:bpid:test$1.0" >
        <cppa:Role cppa:name="test"
          xlink:href="test.bpss.xml#SellerId" />
      ...
    <cppa:ServiceBinding>
```

```

<cppa:Service> urn:icann:buyer.com:bpid:test$1.0
</cppa:Service>
<cppa:CanSend>...</cppa:CanSend>
...
<!-- asynchronous binding:
      CanReceive independent from CanSend -->
<cppa:CanReceive>
  <cppa:ThisPartyActionBinding
    cppa:id="Seller_ReceiveOffer"
    cppa:action="Offer"
    cppa:packageId="Seller_OrderRequestPackage">
  <cppa:BusinessTransactionCharacteristics />
  <cppa:ActionContext
    cppa:binaryCollaboration="DropOrder"
    cppa:businessTransactionActivity="Order"
    cppa:requestOrResponseAction="Offer"/>
    <cppa:ChannelId>Seller_asyncChannel</cppa:ChannelId>
  </cppa:ThisPartyActionBinding>
</cppa:CanReceive>
...

```

The delivery channel specifies the endpoint where the seller can be reached, in terms of the transport protocol and messaging protocol. Different to the WSDL-based approach, the specification is not separated into business-specific and business-independent aspects. As the focus of this example is B2B protocol specification, we omit the business-specific details such as security certificates identifying a particular business.

A delivery channel is basically specified by referencing a transport protocol configuration (attribute `cppa:transportId`) and a messaging protocol configuration (attribute `cppa:docExchangeId`).

Endpoint specification using CPPA

```

<cppa:DeliveryChannel cppa:channelId="Seller_asyncChannel"
  cppa:transportId="Seller_transport"
  cppa:docExchangeId="Seller_docExchange">
  <cppa:MessagingCharacteristics />
</cppa:DeliveryChannel>

```

The transport protocol configuration specified the transport protocol(s) available for sending and/or receiving messages (element `cppa:Transport`). In our example, we basically specify that HTTPS with basic or digest authentication is to be used as transport protocol (element `tp:TransportSender`). Note that the security configuration parameters such as `ClientCertificateRef` and `ServerSecurityDetailsRef` refer to business-specific settings not included in this example.

Endpoint transport protocol specification using CPPA

```

<cppa:Transport cppa:transportId="Seller_transport">
  <cppa:TransportSender>
    <cppa:TransportProtocol cppa:version="1.1"> HTTP
    </cppa:TransportProtocol>
    <cppa:AccessAuthentication> basic
    </cppa:AccessAuthentication>
    <cppa:AccessAuthentication> digest
    </cppa:AccessAuthentication>
    <cppa:TransportClientSecurity>
      <cppa:TransportSecurityProtocol cppa:version="3.0"> SSL
      </cppa:TransportSecurityProtocol>
      <cppa:ClientCertificateRef
        cppa:certId="Seller_ClientCert" />
      <cppa:ServerSecurityDetailsRef
        cppa:securityId="Seller_SecurityDetails" />
    </cppa:TransportClientSecurity>
  </cppa:TransportSender>
  <cppa:TransportReceiver>
    ...

```

The messaging protocol configuration shown below specifies the parameters for ebXML messaging protocol, including reliable messaging parameters, and signature and encryption algorithms (element `cppa:DocExchange`).

Endpoint messaging configuration specification using CPPA

```

<cppa:DocExchange cppa:docExchangeId="Seller_docExchange">
  <cppa:ebXMLSenderBinding cppa:version="2.0">
    <cppa:ReliableMessaging>
      <cppa:Retries>3</cppa:Retries>
      <cppa:RetryInterval>PT2H</cppa:RetryInterval>
      <cppa:MessageOrderSemantics> Guaranteed
      </cppa:MessageOrderSemantics>
    </cppa:ReliableMessaging>
    <cppa:PersistDuration>P1D</cppa:PersistDuration>
    <cppa:SenderNonRepudiation>
      <cppa:NonRepudiationProtocol>
        http://www.w3.org/2000/09/xmldsig#
      </cppa:NonRepudiationProtocol>
      <cppa:HashFunction>
        http://www.w3.org/2000/09/xmldsig#sha1
      </cppa:HashFunction>
      <cppa:SignatureAlgorithm>

```

```

    http://www.w3.org/2000/09/xmldsig#dsa-sha1
  </cppa:SignatureAlgorithm>
  <cppa:SigningCertificateRef
    cppa:certId="Seller_SigningCert" />
</cppa:SenderNonRepudiation>
<cppa:SenderDigitalEnvelope>
  <cppa:DigitalEnvelopeProtocol cppa:version="2.0"> S/MIME
  </cppa:DigitalEnvelopeProtocol>
  <cppa:EncryptionAlgorithm> DES-CBC
  </cppa:EncryptionAlgorithm>
  <cppa:EncryptionSecurityDetailsRef
    cppa:securityId="Seller_SecurityDetails" />
</cppa:SenderDigitalEnvelope>
  ...
<cppa:ebXMLReceiverBinding cppa:version="2.0">
  ...

```

Finally, the message packaging specification shown below defines the message layout and MIME parameters for individual documents such as the order (element `cppa:SimplePart`) and for the whole messages such as an order request message (element `cppa:Packaging`).

Message packaging specification using CPPA

```

<cppa:SimplePart cppa:id="Seller_OrderRequest"
  cppa:mimetype="application/xml">
  <cppa:NamespaceSupported
    cppa:location="UBL_Library_0p70_Order.xsd"
    cppa:version="0.7">
    urn:oasis:names:tc:ubl:Order:1.0:0.70
  </cppa:NamespaceSupported>
</cppa:SimplePart>
  ...
<cppa:Packaging cppa:id="Seller_OrderRequestPackage">
  <cppa:ProcessingCapabilities cppa:parse="true"
    cppa:generate="true"/>
  <cppa:CompositeList>
    <cppa:Composite cppa:id="Seller_RequestMsg"
      cppa:mimetype="multipart/related"
      cppa:mimeparameters="type=text/xml">
      <cppa:Constituent cppa:idref="Seller_MsgHdr"/>
      <cppa:Constituent cppa:idref="Seller_OrderRequest"/>
    </cppa:Composite>
  </cppa:CompositeList>
</cppa:Packaging>
  ...

```

4.5 Services Layer

The service type specifies the whole business process, based upon the interaction types specified in the previous section. In our example, two roles need to be specified, i.e., buyer and seller. The control flow has to be specified such that it starts with the buyer ordering goods from the seller, who in turn acknowledges the order. If the order acknowledgement was positive, and when the ordered goods become ready to be shipped, the seller notifies the buyer about that event.

4.5.1 WSDL-based Approach

The WSDL-based approach employs BPEL for the specification of business processes. A BPEL process is always local to one business role, and it specifies the behavior of that role and the interactions with its partners. In our example, we will show the specification of the seller's process. For purpose of readability, the description is split up into several parts. First, a BPEL-specific extension to the WSDL specification is provided as a prerequisite to the BPEL process specification. The BPEL process specification itself is presented in the parts process header, receive order, reply acceptance (the latter two realizing the order interaction), and notify shipment (realizing that interaction).

BPEL is based on WSDL-specified interaction types, but requires two extensions to the WSDL specification. First, so-called service links have to be defined, which specify the relationship between WSDL port types and the roles in a BPEL process in terms of uses and provides relationships. In our example, the buyer provides the port type `Buyer_PT` and uses another port type `Seller_PT` provided by the seller (element `slnk:serviceLinkType`). Second, so-called message properties have to be defined, which provide names to be used in BPEL process specifications for accessing message data, and mappings to actual message content. Message properties are primarily used for message correlation, i.e., the association of messages to process instances. In our example, process instances are identified using the buyer's order id. Therefore, a corresponding property is declared (element `bpel:property`), as well as a mapping to the `OrderId` element of the order document (element `bpel:propertyAlias`).

Service link and message properties extending the WSDL specification

```
<wsdl:definitions ...>
  ...

  <!-- relationship among partners' interfaces -->
  <slnk:serviceLinkType name="BuyerSellerLink">
    <slnk:role name="buyer">
      <slnk:portType name="buy:Buyer_PT"/>
    </slnk:role>
    <slnk:role name="seller">
      <slnk:portType name="sell:Seller_PT"/>
    </slnk:role>
  </slnk:serviceLinkType>
```



```

<!-- transparent access to message properties -->
<bpel:property name="tns:buyerOrderId"
               type="dn:OrderId"/>
<bpel:propertyAlias propertyName="tns:buyerOrderId"
                    messageType="tns:PurchaseOrder"
                    part="order"
                    query="Order/OrderId"/>
...
</wsdl:definitions>

```

The actual BPEL specification of the sellers process (element process) is a so-called abstract process (attribute `abstractProcess="yes"`), meaning that it does not provide an executable process but just the protocol which has to be followed by an implementation. Before the actual process specification, it defines the interacting partners, the process-relevant data, and the properties to be used for message correlation. In our example, there is only one interacting partner, the buyer (element `bpel:partner`). The process-relevant data are basically the messages exchanged with the buyer, such as the order, the order acceptance, etc. The message types specified in WSDL can be directly used for variable specification (element `bpel:variables`). Regarding message correlation, the above defined property for the buyer's order id will be used for message correlation (element `bpel:correlationSet`).

Specificaiton of the seller's process using BPEL, part 1: header

```

<bpel:process name="SellerOrderProcessing"
              targetNamespace="http://seller.example.com/schemas/order"
              abstractProcess="yes" ... ">

  <bpel:partners>
    <bpel:partner name="buyer"
                  serviceLinkType="tns:BuyerSellerLink"
                  myRole="seller"
                  partnerRole="buyer"/>
  </bpel:partners>

  <bpel:variables>
    <bpel:variable name="BuyersOrder"
                  messageType="pt:PurchaseOrder" />
    <bpel:variable name="OrderAcknowledgement"
                  messageType="pt:PurchaseOrderAcceptance" />
    ...
  </bpel:variables>

  <bpel:correlationSets>
    <bpel:correlationSet name="orderCorrelation"
                        properties="pt:buyerOrderId"/>
  </bpel:correlationSets>

```

The actual process is a sequence of order interaction and notify shipment interaction (element `bpel:sequence`). The order interaction is defined within a so-called scope (element `bpel:scope`) defining a time limit of 24 hours for the interaction (element `bpel:onAlarm`) as well as the possibility for the buyer to interrupt the interaction (element `bpel:onMessage`). Within this scope, the behavior of the order interaction is defined as a sequence of receiving the order from the seller (element `bpel:receive`), and replying (see next part). This process is actually triggered by an order received from a buyer (attribute `createInstance="yes"`). Furthermore, the buyer's order id contained in the order message will be used for identification of the newly created process instance (element `bpel:correlation`). Note that this requires the buyer's order id to be unique across all process instances.

Specifcaiton of the seller's process using BPEL, part 2: receive order

```

<!-- default faultHandler and compensationHandler used -->

<!-- sequence of order and notify shipment -->
<bpel:sequence>

  <!-- scope of order interaction -->
  <bpel:scope>

    <bpel:eventHandlers>
      <!-- event handler providing timeout -->
      <bpel:onAlarm for="PT24H">
        <bpel:throw faultName="tns:Timeout" />
      </bpel:onAlarm>
      <!-- event handler providing interaction termination -->
      <bpel:onMessage partner="buyer"
        portType="pt:Seller_PT"
        operation="NotificationOfFailure"
        variable="FailureNotification">
        <bpel:correlation set="orderCorrelation">
          <bpel:throw faultName="tns:BuyerNotificationOfFailure"/>
        </bpel:onMessage>
      ...
    </bpel:eventHandlers>

    <bpel:sequence>
      <bpel:receive partner="buyer"
        portType="pt:Seller_PT"
        operation="Order"
        variable="BuyersOrder"
        createInstance="yes">
      <bpel:correlations>
        <bpel:correlation set="orderCorrelation"

```

```

        initiation="yes"/>
    </bpel:correlations>
</bpel:receive>

```

Continuing the order interaction, the order request is followed by a reply, which may either be an acceptance or a denial. A switch-statement (element `bpel:switch`) is used to denote the two alternatives. BPEL requires to specify the conditions under which either case is selected. In our example – as in most protocol specifications – we do not want this decision to be specified in the protocol but in the implementation process. Therefore, a condition based on an uninitialized variable is used (attribute `condition`), making the condition expression non-deterministic. In case the order is accepted, an acceptance message is sent to the buyer, thereby completing the order interaction. The process includes a data flow specification (element `bpel:copy`), modeling that the acceptance message will include the same buyer's order id as the request message. This buyer's order id is again used for message correlation, in this case to identify the interaction instance which should be completed. In case the order is not accepted, a denial message is replied and a fault is thrown to terminate the process (element `bpel:otherwise`).

Specification of the seller's process using BPEL, part 3: reply to order

```

<bpel:switch>
    <bpel:case condition="getVariableProperty('OrderState',
        'isOkay')">
        <bpel:sequence>
            <bpel:assign>
                <bpel:copy>
                    <bpel:from variable="BuyersOrder"
                        property="tns:BuyerOrderId" />
                    <bpel:to variable="OrderAcknowledgement"
                        property="tns:BuyerOrderId" />
                </bpel:copy>
            </bpel:assign>
            <bpel:reply partner="buyer"
                portType="pt:Seller_PT"
                operation="Order"
                variable="OrderAcknowledgement" >
                <bpel:correlations>
                    <bpel:correlation set="orderCorrelation" />
                </bpel:correlations>
            </bpel:reply>
        </bpel:sequence>
    </bpel:case>

    <bpel:otherwise>
        ...

```

```

    </bpel:otherwise>
  </bpel:switch>
</bpel:sequence>
</bpel:scope>

```

The last part of the process specifies the shipment notification interaction (element `bpel:invoke`). For interactions initiated by the process, there is no need to specify request processing and response generation, therefore the specification is much simpler. Note that between the order interaction and the notify shipment interaction, some internal processing at the seller will happen, which is not modelled in this abstract process specification.

Specificaiton of the seller's process using BPEL, part 4: notify shipment

```

  <bpel:invoke partner="buyer"
    portType="ip:Buyer_PT"
    operation="Shipping"
    outputContainer="ShipmentNotice">
    <bpel:correlations>
      <bpel:correlation set="orderCorrelation" />
    </bpel:correlations>
  </bpel:invoke>
</bpel:sequence>

</bpel:process>

```

4.5.2 ebXML-based Approach

In the ebXML-based approach, the business process between two interacting businesses is specified using so-called binary collaborations (element `bpss:BinaryCollaboration`). A binary collaboration defines two business roles participating in the process, the interactions used in the process, and the control flow among businesses and interactions. In our example, there are two roles, buyer and seller (element `bpss:Role`), whereby the buyer initiates the process (attribute `initiatingRole="BuyerId"`). The process starts with the order interaction (element `bpss:Start`) and, if that succeeds, continues to the shipment notification interaction (element `bpss:Transition`), and finally ends (element `bpss:Success`). In case of a failure of either interaction, the process ends in an error state (element `bpss:Failure`). The interactions (elements `bpss:BusinessTransactionActivity`) are defined based on the reusable interaction types (attribute `businessTransaction`), additionally assigning the process roles with the initiator and responder roles of the interaction (attributes `fromRole` and `toRole`, respectively) and providing an execution deadline (attribute `timeToPerform`).

Service type specification using BPSS

```

<bpps:ProcessSpecification name="test"
  uuid="urn:icann:buyer.com:bpid:test$1.0" version="1.0" ...>
  ...

  <bpps:BinaryCollaboration name="DropOrder"
    initiatingRole="BuyerId">
    <bpps:Role name="Buyer" nameID="BuyerId"/>
    <bpps:Role name="Seller" nameID="SellerId"/>
    <bpps:Start toBusinessState="Order"
      toBusinessStateIDRef="Order_BTA"/>
    <bpps:BusinessTransactionActivity name="Order"
      fromRole="Buyer"
      toRole="Seller"
      businessTransaction="PurchaseOrderTransaction"
      timeToPerform="PT24H"/>
    <bpps:BusinessTransactionActivity name="ShipmentNotice"
      fromRole="Seller"
      toRole="Buyer"
      businessTransaction="AdvanceShipmentNotice"
      timeToPerform="PT24H"/>
    <bpps:Success fromBusinessState="ShipmentNotice"
      conditionGuard="Success"/>
    <bpps:Failure fromBusinessState="Order"
      conditionGuard="Failure"/>
    <bpps:Failure fromBusinessState="ShipmentNotice"
      conditionGuard="Failure"/>
    <bpps:Transition fromBusinessState="Order"
      toBusinessState="ShipmentNotice"
      conditionGuard="Success">
    </bpps:Transition>
  </bpps:BinaryCollaboration>

</bpps:ProcessSpecification>

```

4.6 Discussion

Summarizing the differences between the two approaches, it becomes obvious that the ebXML-based approach involves a lot of complexity in the interactions layer as compared to the WSDL-based approach. Specifically the specification of operational details using CPPA becomes quite complex and verbose. On the other hand, at the services layer the ebXML-based specification is much simpler and more readable as compared to the WSDL-based approach. Part of this difference stems from the fact that in the ebXML-based approach concepts such as deadlines for

interactions are built-in, whereas in the WSDL-based approach they must be explicitly specified in order to get a comparable specification. A conclusion which can be drawn from this observation is that the WSDL-based approach is preferable in applications which can not benefit from the built-in features of the ebXML interaction types. Consequently, typical “EDI-style” business processes are preferably specified using the ebXML-based approach.

Commonalities among the two approaches can be found mostly in the lower eCo layers. Specifically, the information items layer provides a common base. The documents layer is of little or no relevance to both approaches. Also the interactions layer shows some commonalities, since SOAP messaging is used in both approaches, including a common notion of endpoint. The ebXML-based approach, however, relies on its own messaging protocol, which is based on top of SOAP. And at the higher layers, different concepts are used in the two approaches.

Regarding the eCo architecture used for structuring the description, it does not fit exactly with the structuring of the languages. Specifically in the ebXML-based approach the specification of operational details of interaction types using CPPA does not fit into the eCo layers. There are also some misfits in the WSDL-based approach, e.g., interaction types cannot be specified independently of port types, but port types are closer related to the services layer rather than to the interactions layer.

Finally, we would like to mention that proper tool support is an important prerequisite for the practicability of either of the two approaches for creating and managing B2B protocol specifications.

Part II

Model Driven Development

Chapter 5

Representing XML Schema with UML

There is a need to integrate XML schemas, i.e., schemas written in XML Schema, into UML-based software development processes. Not only the production of XML schemas out of UML models is required, but even more the integration of given XML schemas as input into the development process. In the model driven architecture, a two step integration is assumed, comprising a platform specific model and a platform independent model. This paper addresses the problem of automatically creating a platform specific model for XML schemas. A UML profile and transformation rules from XML Schema to the UML profile are defined, supporting creation of a platform specific UML model that is as concise and semantically expressive as possible without losing XML Schema information.

5.1 Introduction

UML is being used as de-facto standard for software development, including web applications that exchange XML documents. Therefore a need arises to integrate XML schemas, i.e., schemas written in XML Schema, into UML-based software development processes. Not only the production of XML schemas out of UML models is required, but even more the integration of XML schemas as input into the development process, because standard data structures and document types are part of the requirements.

In the model driven architecture [57], a two step integration is assumed, comprising a platform specific model which abstracts from implementation language details, and a platform independent model which abstracts from technology details. For the platform independent model, plain UML is applied, whereas for the platform specific model, UML tailored to the target technology is employed. A UML profile for XML Schema as possible target technology is the main contribution of this paper.

The problem of automatically transforming XML schemas into a platform specific model hasn't been addressed in enough detail yet. In particular, we are looking for a semantically equivalent representation of an XML schema in UML supporting a bijective mapping between both representations. A solution to this problem has to address the whole range of XML Schema concepts, such that any XML schema can be expressed in UML. The goal is to support round-

trip engineering, i.e., transformation from XML Schema to UML and back again without loss of schema information. Furthermore, a solution should maximize understandability of semantic concepts by users knowledgeable of UML but not XML Schema. Semantic equivalence is even more important when the UML models are to be used for application code generation, as it will happen in a model-driven development process.

The transformation problem is a matter of both syntax and semantics. The syntactical aspect can be solved easily - at least in theory -, since an XML-based syntax is defined for both languages. The semantical differences are more difficult to solve, since UML class diagrams aim primarily at a conceptual level, dealing with classes, generalizations, and associations, whereas XML Schema is at a technology-specific level, dealing with elements, attributes, and the grammatical structure of elements. Although a rough equivalence between some concepts of both languages can be easily found, e.g., the correspondence between XML Schema complex types and UML classes, there are also semantical and syntactical heterogeneities. The most important stumbling blocks when transforming from XML Schema to UML are that XML Schema supports specification of content grammars, first class attributes and elements, and identity constraints.

Existing work on representing XML Schema in UML has emerged from approaches to platform specific modeling in UML and transforming these models to XML Schema, with the recognized need for UML extensions to specify XML Schema peculiarities. [11] is the first approach of this kind to modeling XML schemas using UML. Although based on a predecessor to XML Schema, it introduces UML extensions addressing modelling of elements and attributes, model groups, and enumerations that can also be found in following approaches. [14] describes an approach based on XMI rules for transforming UML to XML Schema. [14] also defines a UML profile which addresses most XML Schema concepts, except of simple content complex types, global elements and attributes, and identity constraints. Regarding semantic equivalence, the profile has some weaknesses in its representation of model groups, i.e., sequence, choice, and all. Based on the profile defined in [14], a two-way transformation between XML Schema and UML has been implemented in the commercially available tool "hypermodel"¹. [65] has addressed some of the weaknesses of [14], addressing representation of enumerations and other restriction constraints, and of list and union type constructors, although the latter doesn't conform to UML. [24] (in german, based on [18]) also defines a profile similar to that in [14], with some enhancements regarding simple types and notations. [69] points out the importance of separating the conceptual schema, i.e., the platform independent model, from the logical schema, i.e., the platform specific model, a separation that is not considered in the other approaches. In [69], the logical schema is a direct, one-to-one representation of the XML schema in terms of a UML profile. The profile² covers almost all concepts of XML Schema, but several of its representations are not UML conform. Related work on mapping conceptual models expressed in UML or EER to XML Schema or DTD, has identified various options for transforming conceptual-level concepts to XML Schema concepts [14, 18, 26, 47, 65]. Most of these transformations are, however, not unambiguously applicable in the reverse direction and would thus only be useful

¹<http://xmlmodeling.com/hyperModel/>

²A complete description can be found at <http://titanium.dstc.edu.au/papers/xml-schema-profile.pdf>

in an interactive transformation process, requiring a user's knowledge of the XML schema to be transformed to UML.

Our solution follows [69] in that it also aims at a one-to-one representation of XML schemas in an UML profile. It builds on the existing UML profiles for XML Schema, improving and extending them where necessary, and also taking into account results of UML to XML Schema mapping approaches in order to attain the requirements identified above. An overview of the profile is given in the next section, discussing suitable representations of individual XML Schema concepts in UML. Relationships to related work as well as example diagrams are included in the respective subsections. The examples are mostly taken from standard XML schemas for business documents³. Section 5.3 contains a more precise specification of the profile in terms of the proposed UML metamodel extensions. The paper concludes with a comparison of existing UML profiles and an outlook to future work.

5.2 Transformation Rules and UML Profile

Three design goals have guided the design of the profile and transformation rules. First, it must be possible to represent any XML schema in UML, i.e., there must be a representation for each relevant XML Schema concept, in order to *facilitate round-trip engineering* without loss of schema information.⁴ Second, a representation of an XML schema has to be such that if the profile specific stereotypes are omitted, the result should - to the extent possible - convey the same meaning, in order to *facilitate understanding by non-XML Schema experts* and to support interoperability with tools not aware of the profile. This goal is also in line with the capability of UML stereotypes, which can only extend but not modify the semantics of UML concepts. Finally, the number of *UML constructs* necessary to represent a certain XML schema should be *minimal*, to improve readability. This goal can be achieved in some situations where UML concepts are more expressive than XML Schema concepts, allowing to represent certain patterns of XML Schema concepts using only one UML concept.

The description of the transformation rules and UML extensions is organized along the major XML Schema concepts, i.e., schema, complex types, simple types, elements, attributes, model groups (i.e., complex content), identity constraints, group definitions, annotations, and notations. The relationships to previous work are considered individually per transformation rule, indicating whether rules have already been proposed or whether there are alternative proposals. In some cases, alternative representations are possible, each having different implications. In these cases, selection criteria and compatibility with other alternatives have to be considered.

The description is given at a level of detail that provides an overview of the proposed profile and related work. A specification of the individual stereotypes, their properties and constraints can be found in Section 5.3.

³Open Application Group Integration Specification, OAGIS 8.0; see <http://www.openapplications.org/oagis>

⁴Peculiarities of the XML syntax of XML Schema, such as namespace prefixes, are not considered relevant. The transformation rules are based on the abstract syntax of XML Schema as specified in [93]

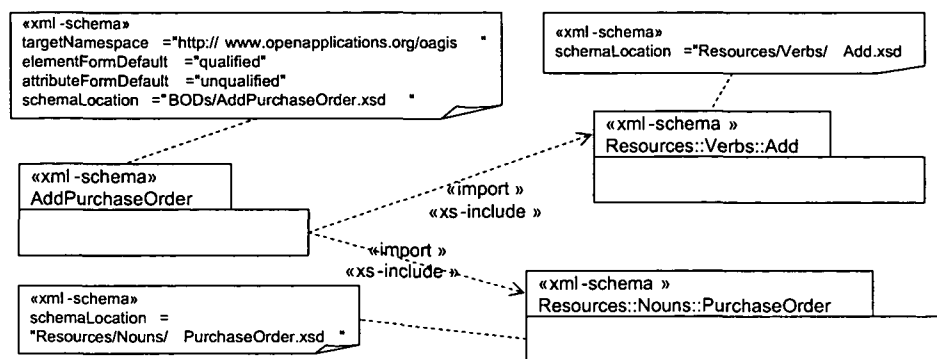


Figure 5.1: Representation of the AddPurchaseOrder schema and its dependencies

5.2.1 Schema Document

We propose to represent schema documents and dependencies between them (i.e., import, include, and redefine) as follows:

SC Represent every schema document as a package with stereotype `«xml-schema»` [11, 14, 24, 69]. Properties of this stereotype are used to represent the target namespace, version, and location information (cf. Figure 5.1). Schema dependencies are represented as dependencies [69] with stereotypes `«xs-import»`, `«xs-include»`, and `«xs-redefine»`. In the context of UML 2.0 we propose to use dependencies `Permission` and `PackageImport` as stereotype base for representing import and include/redefine, respectively.

When representing the target namespace as package name, as proposed by [11], it is not possible to represent multiple schema documents having the same target namespace.

5.2.2 Complex Type Definition

Literature proposes a common and straightforward way to represent complex types in UML, which we adopt:

CT1 Represent every global complex type as class with the type's name and stereotype `«complexType»` [14, 65]. See the left side of Figure 5.2 for an example.

CT2 Represent every local complex type as class with stereotype `«complexType»` [24, 69, 65] nested into its containing class [65]. The class' name is formed by the name of the element defining the complex type [24]. An alternative approach in [69] encodes the containment hierarchy in the class' name, conveying less semantics than the nesting solution.

Complex types have various properties that apply for both global and local ones:

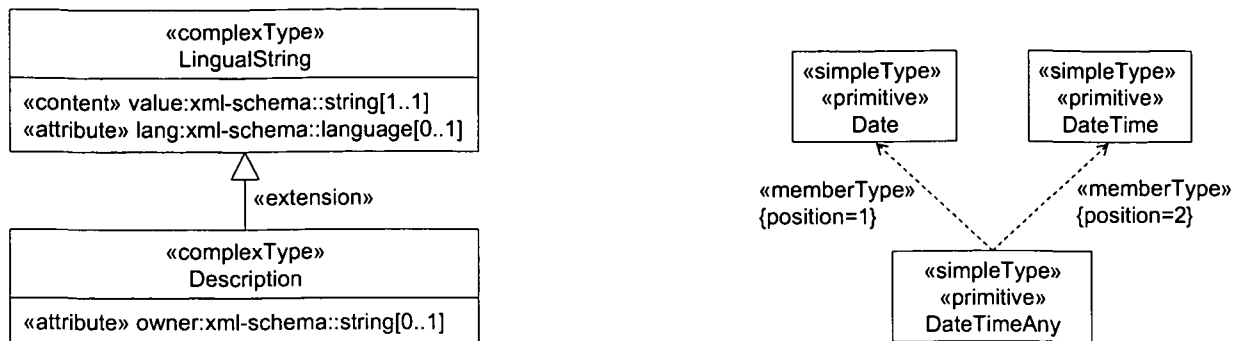


Figure 5.2: Representation of the `LingualString` and `Description` complex types (left) and representation of a simple type constructed by union (right)

- A complex type's content model is represented implicitly, as described by the representation of simple content below and representation of model groups in Section 5.2.6, except of mixed content, which is represented as property of stereotype «`complexType`» [24].
- An abstract complex type is represented as an abstract class.
- Derivation by extension and restriction are represented as generalizations with stereotypes «`extension`» and «`restriction`», as common to most approaches [14, 69, 65].
- Derivation and substitution constraints are represented as UML constraint, e.g., {final="restriction"}. Another option would be to use stereotype properties [69]. However, since these constraints are also meaningful for platform independent models, we favor the constraint representation.
- Simple content is represented as an attribute with stereotype «`content`», which has the type of the simple content and is named `value` as in the Value design pattern (cf. Figure 5.2). Although this representation has not yet been proposed in literature, we think it adheres to semantics and pragmatics of UML as well as of other implementation languages such as Java and SQL. In [69] it has been proposed to represent simple content as a generalization relationship to the simple content type, which is not UML conform in case that simple types are represented as datatype and not as class.

5.2.3 Simple Type Definition

An XML Schema simple type can be naturally represented as an UML datatype. Since UML distinguishes between enumerations and other kinds of datatypes, two different representations are proposed:

ST1 Represent every simple type that includes an enumeration constraint as an enumeration [65] with stereotype «`simpleType`». For an example see the left part of Figure 5.3.

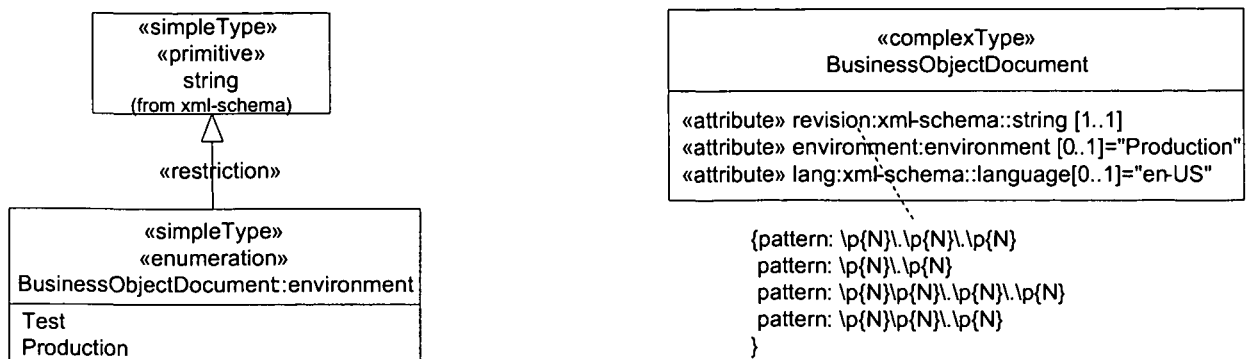


Figure 5.3: Representation of a simple type local to `BusinessObjectDocument` (left) and representation of a simple type restriction according to ST3 (right)

ST2 Represent every simple type that does not include an enumeration constraint as primitive datatype with stereotype `«simpleType»`. In literature such simple types are not represented as datatypes but as stereotyped classes [14, 24, 69, 65], a solution which is semantically less precise.

Local simple types are represented like local complex types, i.e., nested into the containing class, with the restriction that nesting into a containing datatype is not possible. The name of the datatype is formed by the name of the attribute/element defining the simple type, e.g., `BusinessObjectDocument::revision` in Figure 5.3.

Other constraints than enumerations, such as range and pattern, are represented as UML constraints [69, 65]. An alternative representation would be as properties of stereotypes [14, 24], however, we consider UML constraints as the more appropriate representation, as the constraints specify the datatype and standard UML.

Derivation by restriction in simple type construction is represented by a generalization with stereotype `«restriction»`, a solution also found in previous work [69, 14, 65]. Note that in a generalization between two enumerations in UML the special enumeration is only allowed to add enumeration values opposed to XML Schema. Therefore the proposed solution is not fully UML compliant, even if a stereotype is used to point out the difference. A UML compliant but less readable solution would be to use a stereotyped dependency.

List construction is represented as a dependency with stereotype `«itemType»` to the list item datatype. Other solutions found in literature comprise a stereotype's property which refers to the list item datatype [24], and a template instantiation of a predefined list template [65]. While the stereotype representation does not carry UML semantics, the template instantiation representation would be more expressive than our dependency. Nevertheless, the dependency representation is similar in style to the representation of the union constructor, making it easier to understand.

Union construction is represented as a dependency with stereotype `«memberType»`, which has a property position to define the order of member types. This is favorable to Provost [65] who

proposes to represent a union type by multiple generalizations connected by an {xor} constraint, which does not conform to UML semantics. An example is given in the right side of Figure 5.2.

In certain cases, representation of local simple types can be further simplified in that it is not necessary to represent them as an explicit datatype. We have identified two simplification rules addressing common cases:

ST3 Merge the representation of a local simple type that is the type of a UML attribute, with that attribute, as follows: a) Merge a restriction type such that the attribute type represents the restriction's base type, and restriction constraints are directly attached to the attribute. See the right part of Figure 5.3 for an example. This rule is not applicable to restrictions defining enumeration constraints. b) Merge a list type such that the attribute type represents the list item type, and attribute multiplicity becomes 0..*. c) Merge a restriction type that restricts a list type such that the attribute type represents the list item type, attribute multiplicity represents length constraints. This rule is not applicable to restrictions defining enumeration or pattern constraints. d) Merge a union type such that the attribute becomes a derived union, and for each union member type introduce an appropriately typed attribute which subsets the union attribute. **ST3** can be applied recursively on the member type attributes. This rule is specific to UML 2.0. e) Merge a restriction type that restricts a union type as described in d), with an attached constraint to represent any pattern constraints. This rule is not applicable to restrictions defining enumeration constraints.

ST4 Merge the representation of a list or union type that is defined local to a restriction type with the representation of that restriction type, such that the dependencies representing item type and member type, respectively, are directly attached to the restriction type.

No specific stereotypes need to be introduced to distinguish simplified representations from **ST1** and **ST2**.

5.2.4 Element Declaration and Usage

With XML Schema one can separate an element declaration from its use by using global element declarations, a possibility not provided by UML.

Therefore we consider the two cases of local and global elements separately. Local elements, i.e., an element declaration and usage in-place, are commonly represented in two similar ways:

EL1 Represent a local element as an association role with stereotype «element» of an association between the class representing the containing model group and the class representing the element's type, and with appropriate multiplicity. The nesting direction is indicated by an aggregation [24]. For an example see **DataArea** in Figure 5.4. To represent nesting by navigation directions as proposed in [14] is unfavorable, because in XML a nesting is navigable in both directions.

EL2 Represent a local element as an attribute with stereotype «element» of the class representing the containing model group, with appropriate type and multiplicity.

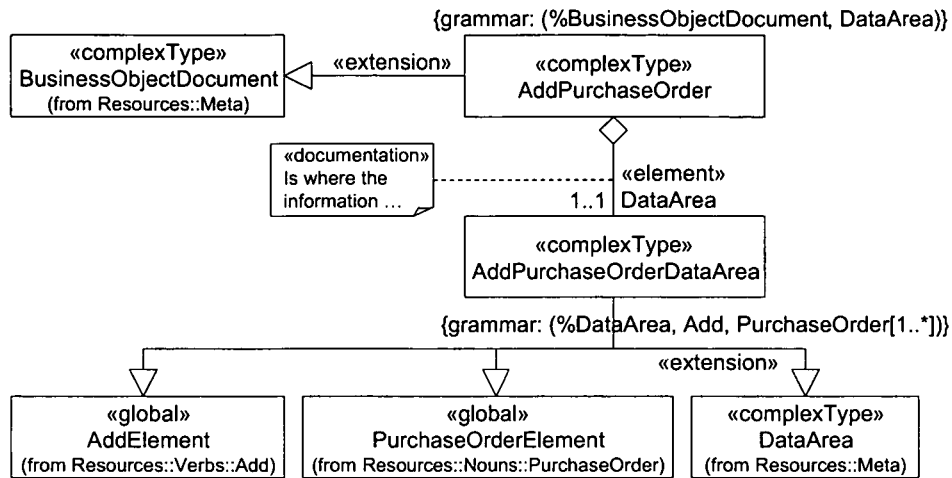


Figure 5.4: Declaration of complex types `AddPurchaseOrder` and `AddPurchaseOrderDataArea`

Note that both representations are semantically equivalent in UML, however, we propose to use EL1 in case of the element's type being a complex type and EL2 in case the element's type being a simple type for two reasons. First, this is a common UML modelling style, and second, EL2 supports the representation of a default value, which may be defined for simple types but not for complex ones.

For global elements we propose pattern EG:

EG Represent every global element declaration like a local element declaration, i.e., as an association with a stereotyped role or an attribute, with default multiplicity `0..*`. Since global elements are declared outside the scope of a containing model group, a class with stereotype `«global»` is introduced which contains the association or attribute. The class is named after the element with postfix `Element`, e.g., `AddElement` (cf. left of Figure 5.5).

Element usage is represented by a generalization relationship from the using class to the class containing the representation of the global element. Furthermore, in the using class the inherited attribute may need to be redefined or the inherited association may need to be specialized to specify a non-default multiplicity.

We further propose to represent substitution group participation in EG as a generalization with stereotype `«substitution»` to the class that represents the substitution group's head element and redefinition of the attribute or specialization of the association representing the head element (cf. left part of Figure 5.5). Note that attribute redefinition involves renaming of the attribute, which is possible in UML 2.0 but not in earlier versions.

Our pattern EG is different from previously proposed global element representations in [69]. There, every global element declaration is represented as a stereotyped class which does not contain any attribute. The type of the element, either simple or complex, is defined via a stereotyped

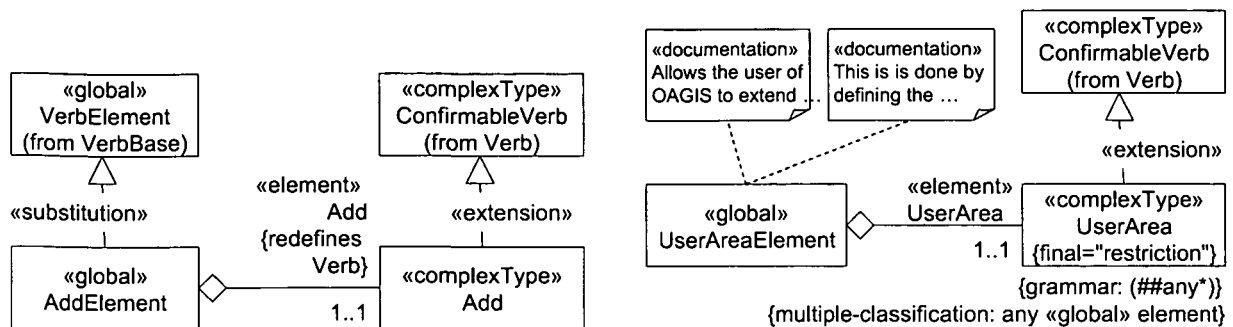


Figure 5.5: Declaration of global element Add (left) and declaration of UserArea element and complex type (right)

dependency. An element's membership in a substitution group is represented by a property of the class's stereotype. An element usage is represented by a stereotyped attribute with the attribute name referring to the global attribute and with appropriate multiplicity.

We favor our pattern EG over global element representations in literature, because they do not conform to UML by expressing an element, a concept which does not exist in UML, as a class. Moreover, EG has the advantage that it is consistent with patterns for local element declaration and usage EL1 and EL2. It comes, however, at the cost of a more verbose representation.

We propose, as literature does, to represent a nillable constraint as a constraint [24, 69]. The same holds for other constraints associated with XML Schema elements, i.e., for substitution group exclusion and disallowed substitutions.

Regarding the representation of an element wildcard, only one pattern has been proposed in literature [14, 24, 69]. It represents an element wildcard as a stereotyped class, with stereotype properties namespace and processContents having the corresponding XML schema values as default.

Because the representation for element wildcards proposed in literature does not conform to UML, it is difficult to use its semantics in a PIM. Thus, we propose another representation, which is based on pattern EG:

EW Represent every element wildcard as a proprietary multiple classification constraint named {multiple-classification}, indicating that an instance of the class the constraint is attached to can be an instance of other classes representing global elements as well (cf. right part of Figure 5.5). Thus the occurrence of a global element at the instance level is represented in terms of multiple classification without the need for explicit generalizations at the model level. This representation can be seen as a shorthand to have generalizations to all classes in specified packages representing global elements. The pattern can be used in combination with MG2, and with MG1 if a dummy class is introduced to hold the multiple classification constraint.

5.2.5 Attribute Declaration and Usage

Like with elements, we distinguish between the representation of local attributes and of global attributes. Local attributes map naturally to UML attributes, but some specific considerations can be taken into account:

- AL Represent every local attribute as an attribute with stereotype `<<attribute>>` of the class representing the complex type or group containing the attribute [14, 69, 65, 24], with multiplicity `0..1` or `1..1` and default value. Fixed value attributes can be represented by the corresponding UML constraint `{read only}` [24, 69]⁵. Standard datatypes that are in plural form, i.e., `IDREFS`, `NMTOKENS`, and `ENTITIES`, can be represented in singular form with multiplicity `0..*` [24]. The pattern cannot distinguish between an empty attribute and an absent attribute and is therefore not strictly equivalent.

Declaration and use of global attributes is more difficult to represent. We do not know of any published approach that supports such a representation, therefore we propose the following, similar in style to EG:

- AG Represent every global attribute declaration like a local attribute, i.e., as an attribute with stereotype `<<attribute>>`, with default multiplicity `0..1`. Since global attributes are declared outside the scope of a containing complex type, a class with stereotype `<<global>>` is introduced which contains the attribute. The class is named after the attribute with postfix `Attribute`, e.g., `nameAttribute`.

Attribute usage is represented by a generalization relationship from the using class to the class containing the representation of the global attribute. In the using class the inherited attribute may be redefined to specify non-default multiplicities and a default value.

Regarding the representation of attribute wildcards, only one pattern has been proposed in literature. It represents an attribute wildcard as a stereotyped dummy attribute [24], a solution that fails to satisfy the second design goal. Thus, we propose another representation which is based on pattern AG and follows the style of EW:

- AW Represent every attribute wildcard as a proprietary multiple classification constraint, indicating that an instance of the class the constraint is attached to can be an instance of other classes representing global attributes as well. This representation can be seen as a shorthand to have generalizations to all classes in specified packages representing global attributes.

5.2.6 Model Group

In UML there is no concept directly similar to the one of a model group, i.e., `sequence`, `choice`, or `all`. The following pattern has been proposed in literature:

⁵[24] uses proprietary `{frozen}`

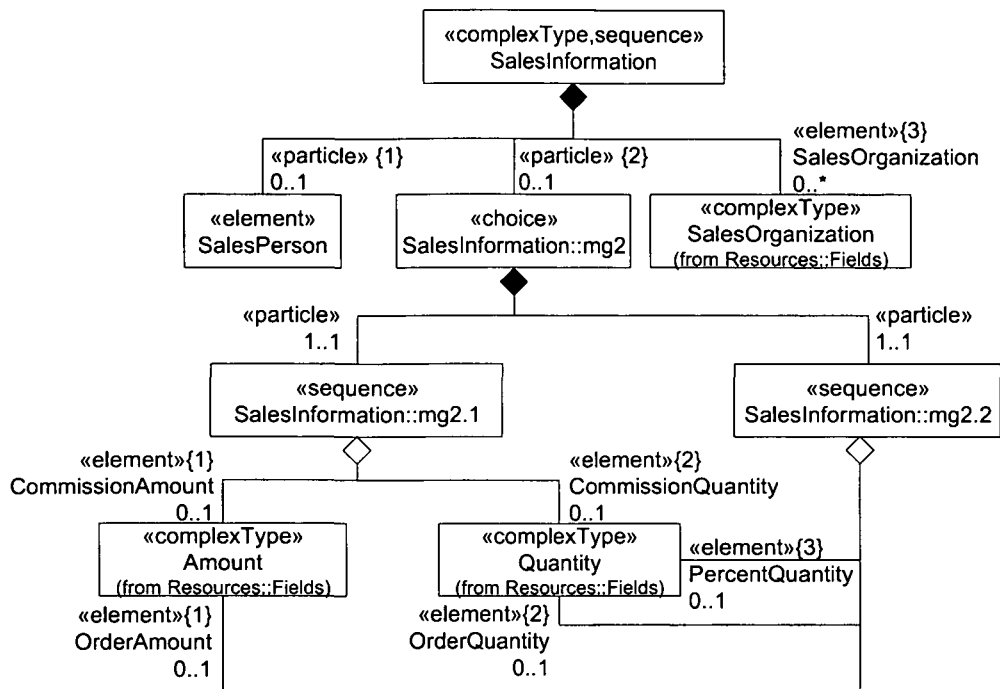


Figure 5.6: Complex type SalesInformation’s model group representation using MG1

MG1 Represent a model group as a class with one of the stereotypes «sequence», «choice», and «all» to represent the respective model group compositor [14, 24, 69]. The name of the class is mg followed by a hierarchical number. A nesting of model groups is represented by a composition association with appropriate multiplicity, such that the syntax tree of XML Schema model groups is visually represented (cf. Figure 5.6). Since model groups are private to their container, we propose to nest them into the class containing the model group tree. The order of model group particles, i.e., the order of elements, element wildcards, element group references, and nested model groups is represented by a stereotype property.

Two simplifications are available for MG1. First, the root model group of a complex type or group definition can be represented by the class representing the complex type or group definition (cf. Section 5.2.8 for group definitions) [14, 69] An example using this simplification is SalesInformation shown in Figure 5.6 into which the root model group has been merged. Second, to denote the order of model group particles the order of attributes in the class representing the model group is used [69]. This approach requires, however, that all kinds of particles are represented as attributes. Since for some kinds of particles, e.g., group references, an attribute representation is not available, this simplification is not always applicable.

Representing a model group as stereotyped class by MG1 is well suited if the model group represents domain semantics. However, often this is not the case, for instance a nested choice usually does not express an application domain concept but a disjunction constraint. Therefore, we propose a novel alternative representation which models the grammatical structure in a visu-

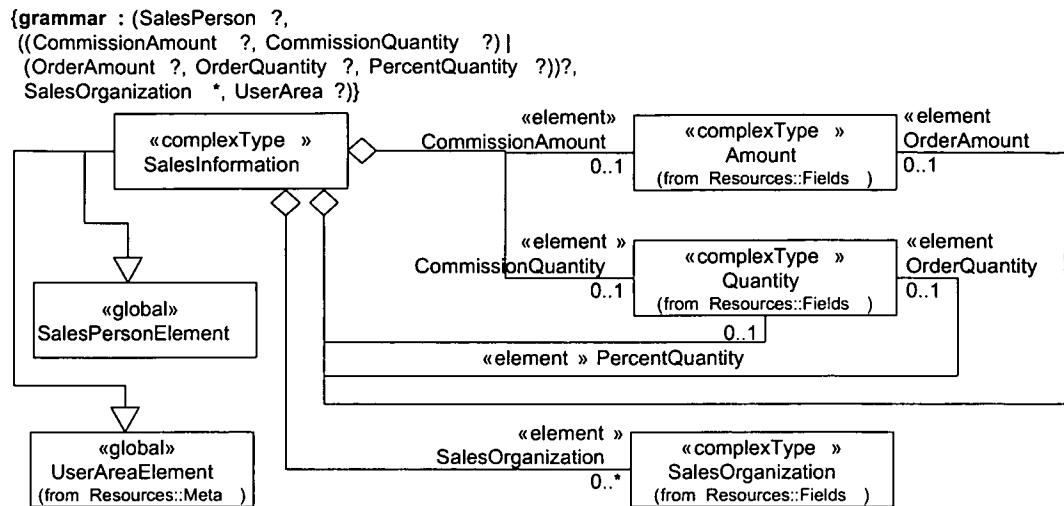


Figure 5.7: Complex type SalesInformation’s model group representation using MG2

ally less dominant way:

MG2 Represent the grammar expressed by a model group tree as constraint, using a textual notation which covers both hierarchical structuring and ordering (cf. Figure 5.7). We propose to use the grammar language of DTDs, as it is concise and well known. It is extended to support user-defined occurrence ranges, wildcards, model group references, and the `all` compositor, e.g., as in [98]. Different to MG1, nested model groups are not represented individually but by a single constraint attached to the class representing the complex type or group definition containing the model group tree. This also means that element uses are represented outside their containing model group, directly in the context of the containing complex type or group definition. Therefore, multiplicities need to be modified accordingly, e.g., an element use with multiplicity 1..1 contained in a choice model group with multiplicity 2 results in overall multiplicity 0..2 for the attribute or association role representing that element use.

Choosing between MG1 and MG2 is a matter of application semantics and is left to the user. However, assuming that a relevant domain concept is expressed either as an individual complex type or as a named model group, we strongly propose to use MG2 as the default representation. For a comparison when the same XML schema is expressed using the two patterns see Figure 5.6 and Figure 5.7. Note that there are further implications if the resulting class structure is used as basis for code generation. If MG1 is chosen, the simplified version of it should be used whenever applicable.

5.2.7 Identity Constraint Definition

Representation of identity constraints has not been addressed in previous work. The inverse of representing an ER identity constraint as an XML Schema key, e.g., as proposed in [69],

is not sufficient because key constraints in XML Schema are always defined in the context of an element, the so called “scope”. Opposite thereto, approaches such as [69] assume global identification among all instances of a class. Therefore we propose to represent key constraints as simplified OCL constraints:

KY1 Represent a key constraint as a constraint in the form of **{key "name": for all $selector_1, selector_2, \dots: field_1, field_2, \dots$ }** attached to the class containing the representation of the element that is the key’s scope. In the constraint, *name* represents the key’s name and $selector_i$ represents the key’s selector in OCL notation. A selector starts with the element that is the key’s scope. Multiple selectors represent a union operation in the original XPath expression. We assume proprietary OCL operations *all-element-children*, *all-element-children-in-namespace(namespace)*, and *all-element-descendants* to represent XPath wildcard steps “.//”, “*”, and “NCName : *”, respectively. A typical use of wildcard steps is to handle elements in substitution groups, however, in this case the selector can be represented without the proprietary OCL operations due to use of pattern EG. Finally, $field_i$ represent the key’s fields in OCL notation. The semantics of the key constraint is as follows:

context *ScopeClass*

def: $name_keys = selector_1 \rightarrow collect(Tuple\{f_1 = field_1, \dots\}) \rightarrow union(selector_2 \rightarrow collect\dots)$

inv: $name_keys \rightarrow isUnique()$

inv: $name_keys \rightarrow forAll(f_1 \rightarrow notEmpty() \text{ and } f_2 \rightarrow notEmpty() \text{ and } \dots)$

In situations where scope and selected class are defined in the same context, e.g., as in Figure 5.8, it is more intuitive to attach the key constraint to the selected class. Such a representation is only applicable, however, in case the selector unambiguously identifies a class, i.e., it must not contain union or wildcard steps. For these cases, we propose the following alternative representation:

KY2 Represent a key constraint whose selector does not contain union and wildcard steps as constraint **{key "name": in context *selector*: $field_1, field_2, \dots$ }** attached to the class selected by the selector. Semantically, this constraint is equivalent to KY1.

Uniqueness constraints can be represented similar to key constraints, using constraint **unique:** Semantics is similar to that of the key constraints except that the second invariant requiring that all fields are present is omitted.

Referential integrity constraints (*keyref*) can also be represented in a similar manner by constraint **{keyref "name": for all $selector_1, selector_2, \dots: field_1, field_2, \dots \rightarrow keyname$ }** attached to the class containing the representation of the element that forms the *keyref*’s scope. The relationship to the key constraint is represented as dependency as well as by *keyname*.

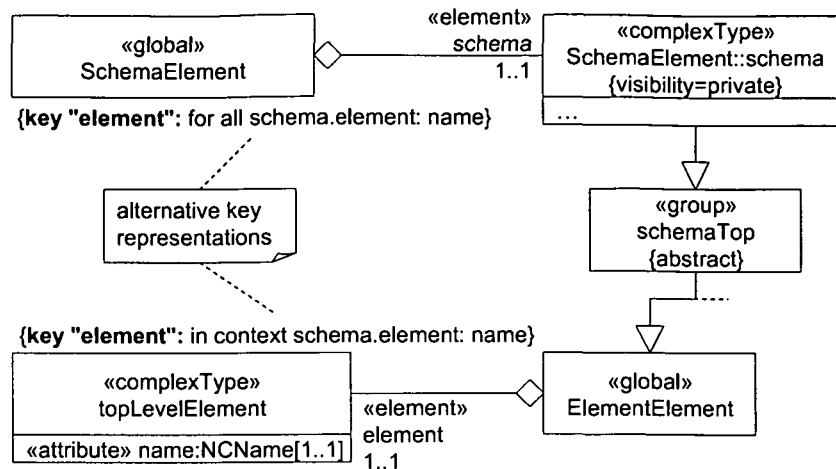


Figure 5.8: Key element (from the schema for XML Schema)

5.2.8 Group Definition and Reference

We distinguish between representation of named element groups and named attribute groups.

GA Represent every attribute group as an abstract class with stereotype `«group»` containing attributes represented using AL [69]. References to the attribute group are represented as a generalization. Note that this has not been proposed in literature yet.

An alternative representation for a group reference as a stereotyped attribute named like the referenced attribute group has been proposed in [69]. This representation, however, does not conform to UML semantics.

Because there exist two representations for model groups MG1 and MG2, two variants exist for the representation of element groups:

GE1 Represent an element group as a class with stereotype `«group»` and represent every reference to the group as a composition according to MG1. Different to MG1, the class is named after the group and is not nested into a containing class.

GE2 Represent an element group as an abstract class with stereotype `«group»`, named after the group, and represent the group according to MG2. A reference to the group is represented as generalization from the class representing the using model group. The latter's class grammar constraint is modified to define position and occurrence of the used model group. Therefore, it is necessary that the using model group is also represented using MG1.

5.2.9 Annotations

Representation of annotations is straightforward:

AN Represent every annotation as a set of comments with stereotypes `<<applInfo>>` and `<<documentation>>`, distinguishing between application and user information. The comments are associated with the model element representing the annotated XML Schema artifact.

Previous approaches did not take into account the differentiation between application and user information and the handling of multiple annotations [69].

5.2.10 Notation

A notation in XML usually specifies a named reference to an application. When an element's content is defined to have a notation's format by using an attribute with the notation's name as its value, the application referenced by the notation is usually capable of processing the notation.

NO Represent every notation declaration as a literal with stereotype `<<notation>>` in an enumeration named `Notations` with stereotype `<<notations>>`. The enumeration comprises all notations of a schema and all notations of included schemas via generalizations to their `Notations` enumerations. A notation attribute's simple type is derived by restriction from the `Notations` enumeration by using `ST1`.

Previous approaches represented notations as a stereotyped class [24], not considering the semantics of notations.

5.3 UML Profile Implementation

This section provides implementation details for extending the UML metamodel with a profile for XML Schema, i.e., the stereotypes defined, the UML metamodel elements they are based on, the stereotype's properties, and the constraints involved. The stereotypes are mutually exclusive, i.e., it is not possible to extend the same metamodel element with more than one of the stereotypes defined by this profile.

Note that both metamodel and model-level constraints are involved. Metamodel constraints are those imposed by a stereotype on all model elements which carry this stereotype, ensuring that an UML model conforming to the profile can be mapped to an XML schema. For instance, an attribute stereotyped as `<<element>>` must not define a default value. Model-level constraints are constraints that may be attached to individual model elements. The profile does not define specific constraints but rather which kinds of constraints are supported, e.g., to classes stereotyped as `<<complexType>>`, a "key" constraint may be attached.

The description of the stereotypes is organized along the UML concepts extended, i.e., package, class, property, generalization, datatype, and comment. Regarding terminology, we use the shorthand "a `<<foo>>`" meaning "a metamodel element stereotyped as `<<foo>>`".

5.3.1 Package Stereotypes

The UML **Package** is extended to support modelling of XML Schema documents. In addition, the **Permission** dependency and the **PackageImport** relationship are extended to support modelling of interdependencies between XML Schema documents (cf. Fig. 5.9). These extensions are intended to be used in a package diagram, showing the relationships between one or more root schema documents and their component schema documents.

The stereotype `«xml-schema»` implements the attributes of the `schema` element. In addition to these, `schemaLocation` is also implemented as an attribute of the schema document rather than as an attribute of the import or redefine dependencies, in order to be able to record the location of the root schema document.

`«xml-schema»` imposes the following restrictions on the use of a package:

- Nesting of packages within the stereotyped package is not allowed.
- Only elements with one of the stereotypes defined in this profile are allowed as contents of the package, except of **Generalization** and **Constraint**, which are also allowed without stereotype.

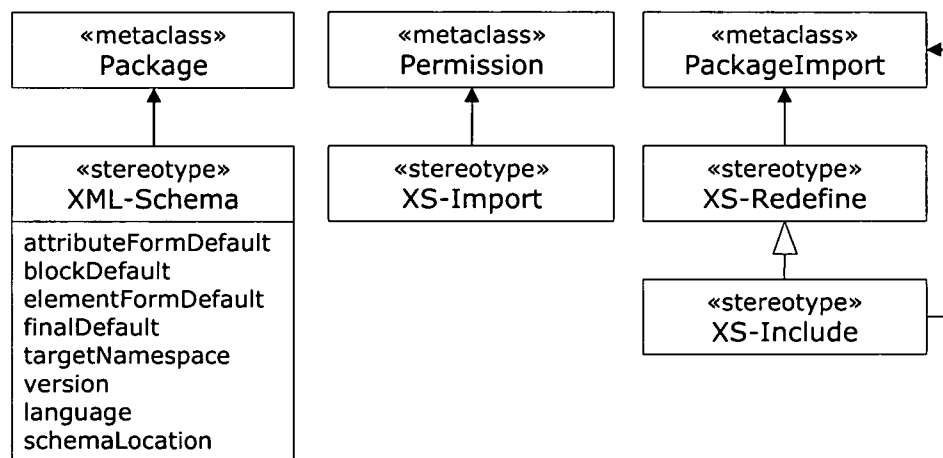


Figure 5.9: Package and related stereotypes

Regarding schema dependencies, `«xs-import»` has been implemented as stereotype of the **Permission** dependency, since both concepts have the same semantics. `«xs-include»` and `«xs-redefine»` have been implemented a stereotypes of `«PackageImport»` with some additional constraints. Since the difference between `«xs-include»` and `«xs-redefine»` is that the first imposes some more specific constraints, the stereotypes have been implemented in a generalization relationship.

The stereotypes place the following constraints on the UML metamodel:

- Both client and supplier of an `«xs-import»` must be an `«xml-schema»`.

- Both `importingNamespace` and `importedPackage` of a `«xs-redefine»` must be an `«xml-schema»`.
- The importing package of a `«xs-redefine»` must not contain an element with the same name and the same type as the one imported except if that element has a generalization relationship to the imported element of the same name.
- The importing package of a `«xs-include»` must not contain an element with the same name and the same type as the one imported, without exception (refines the precedent constraint).

5.3.2 Class Stereotypes

Class is extended to support modelling of complex types (`«complexType»`), global properties (`«global»`), groups (`«group»`), and attribute groups (`«attributeGroup»`), as shown in Fig. 5.10.

Stereotype `«complexType»` defines derived boolean attributes describing the kind of complex type represented by the class. Attribute `local` is true if the complex type is nested within another complex type. `empty` is true if the complex type does not define any properties of stereotype `«element»` or `«simpleContent»`. `simpleContent` is true if the complex type defines a property of stereotype `«simpleContent»`. `complexContent` is true if the complex type defines some properties of stereotype `«element»`. Note that all of the class' inherited properties must be considered, too.

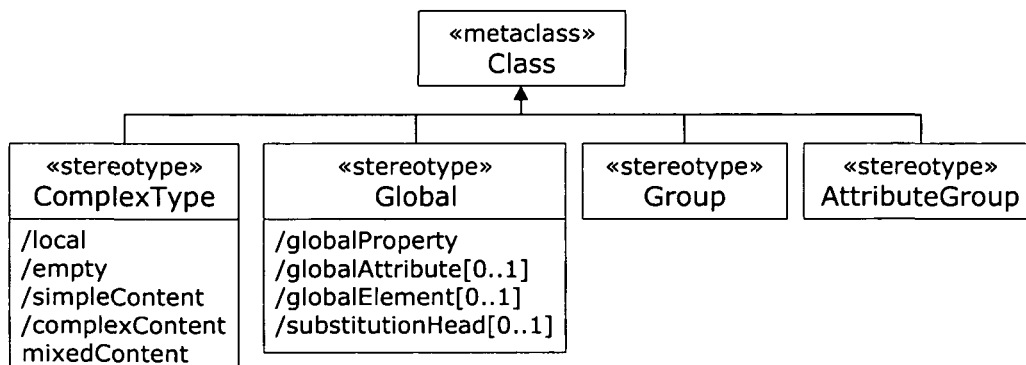


Figure 5.10: Class stereotypes

Several kinds of constraints may be attached to a class stereotyped as `«complexType»`:

- A “grammar” constraint, defining the order and multiplicity of elements as specified in the previous section. The constraint may redefine the multiplicities of the element's properties defined in the complex type. However, minimum and maximum multiplicities may only be lowered.

- A “multiple-classification” constraint, which may optionally define the namespace(s) as defined in the XML Schema any construct. An instance of a such constrained class may also be an instance of the classes contained in the namespace(s) as defined in the constraint. Note that classes with this constraint are the only ones allowing multiple classification. The constraint specification must be an Expression (explicit syntax tree) with an operator corresponding to the XML Schema wildcards (any, any-except, any-out-of) and with operands “namespace” and “processing control”.
- A “substitution” constraint defining values for `restrictionFinal`, `extensionFinal`, `restrictionBlock`, and `extensionBlock`. If both `restrictionFinal` and `extensionFinal` are set to true, then the class must be defined as leaf.
- Constraints “key”, “keyref”, and “unique”, as specified in the previous section.

The following metamodel constraints apply to classes stereotyped as `<<complexType>>`:

- The class may only have properties with one of the `<<xs-property>>` stereotypes, i.e., `<<attribute>>`, `<<element>>`, or `<<simpleContent>>` (cf. Section 5.3.3).
- The class may have at most one generalization of stereotype `<<extension>>` or `<<restriction>>`.
- The class may have generalizations without stereotypes, with the general class being stereotyped as `<<globalProperty>>`, `<<group>>`, or `<<attributeGroup>>`.
- The class may contain nested classes stereotyped `<<complexType>>` and nested datatypes stereotyped `<<simpleType>>`.

Stereotype `<<global>>` extends class to be used as container for a global property, i.e., a global element or a global attribute (cf. Section 5.3.3 for elements and attributes). The stereotype defines some derived attributes supporting access to the global property: `globalProperty` points to the property, independent of it being an `<<element>>` or `<<attribute>>`. `globalElement` points to that property in case it is an `<<element>>`, otherwise it is empty. Similarly `globalAttribute` in case the class defines an `<<attribute>>`. `substitutionHead` is a shortcut to the `<<global>>` class forming the root of the substitution hierarchy, if any.

A `<<global>>` class may have these constraints attached:

- A “final” constraint, defining boolean values for extension and/or restriction. If both are true, the class must be leaf.
- The constraints “key”, “keyref”, and “unique” can be attached to a class stereotyped as `<<global>>`, in the same way as to a `<<complexType>>`.

The following metamodel constraints restrict usage of classes stereotyped as `<<global>>`:

- The class must define exactly one property, stereotyped either as `<<element>>` or as `<<attribute>>`. If that property is an `<<element>>`, its multiplicity must be `0..*`, otherwise `0..1`.
- The class' name should be constructed from the property's name with 'Property' appended.
- The class may have at most one generalization, stereotyped as `<<substitution>>`, implementing substitution group affiliation. The special class must redefine the property of the general class.

Stereotypes `<<group>>` and `<<attributeGroup>>` extend class with semantics specific to element and attribute groups, respectively. Some of the constraints defined for `<<complexType>>` may also be attached to a class stereotyped by `<<group>>` or `<<attributeGroup>>`:

- Constraints "grammar", "key", "keyref", and "unique" may be attached to a `<<group>>`.
- Both `<<group>>` and `<<attributeGroup>>` may have a "multiple-classification" constraint attached.

The following metamodel constraints apply to a class with one the group stereotypes:

- The class must be abstract.
- A `<<group>>` may define only properties of stereotype `<<element>>`.
- An `<<attributeGroup>>` may define only properties of stereotype `<<attribute>>`.

5.3.3 Property Stereotypes

Properties are used to represent elements, attributes, and simple content, as defined by the stereotypes `<<element>>`, `<<attribute>>`, and `<<simpleContent>>` (cf. Fig. 5.11). Note that in UML 2.0, properties serve as both attributes and association ends. Therefore the stereotypes implement both representation patterns EL1 and EL2. The stereotypes' property qualified defines whether the element is namespace qualified or not.

Properties stereotyped as `<<element>>` may have one of the following constraints attached:

- A "nillable" constraint, declaring that the property may hold nil values.
- A "block" constraint, defining valued for extension, restriction, and substitution, implementing disallowed substitutions and substitution group exclusion, respectively.

Properties stereotyped as `<<attribute>>` may have restriction constraints attached, as specified in Section 5.3.5, in order to support representation patterns ST3 and ST4. Furthermore, a stereotyped property must adhere to the following metamodel constraints:

- An `<<attribute>>` property which is read-only must also provide a default value, representing the XML Schema value constraint "fixed".
- A `<<simpleContent>>` property must have multiplicity `0..1` and no default value.

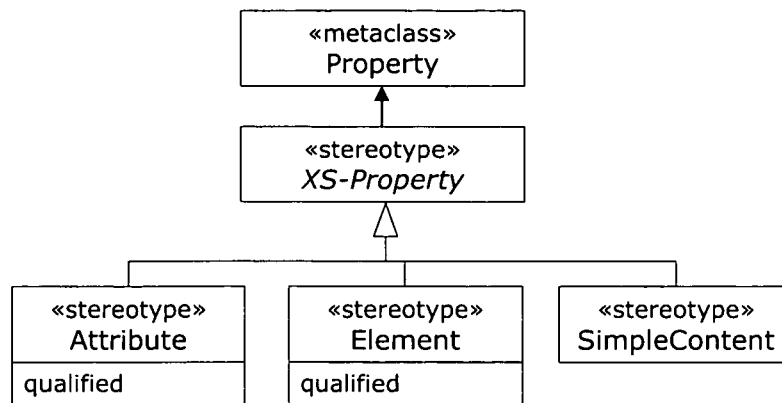


Figure 5.11: Property stereotypes

5.3.4 Generalization Stereotypes

Generalization is extended for defining relationships among complex types (`«extension»`, `«restriction»`), among simple types (`«restriction»`), and among global elements (`«substitution»`), as illustrated in Fig. 5.12. Furthermore, Generalization, without stereotypes, is used for specifying usage of global properties and of groups.

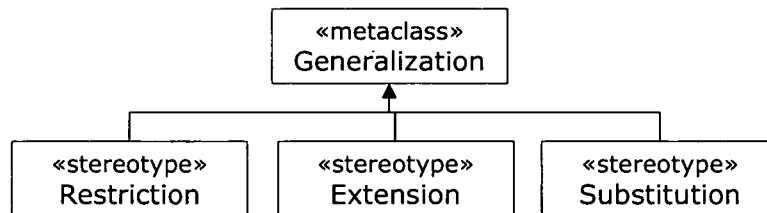


Figure 5.12: Generalization stereotypes

Stereotype `«restriction»` imposes the following metamodel constraints on the usage of a generalization:

- The general classifier must be stereotyped either as `«complexType»` or as `«simpleType»`.
- The special classifier must be of the same stereotype as the general one.
- If the general classifier is a `«complexType»` which defines a constraint `restrictionBlock`, then the generalization must be not substitutable.

The following metamodel constraints are imposed by stereotype `«extension»`:

- Both general and special classifier must be stereotyped as `«complexType»`.
- If the general classifier defines a constraint `extensionBlock`, then the generalization must be not substitutable.

The following metamodel constraints apply to a `<<substitution>>` generalization:

- The generalization must be substitutable.
- Both general and special class must be stereotyped as `<<global>>`.

5.3.5 Datatype Stereotypes

UML Datatype and its subclasses Primitive and Enumeration are extended (cf. Fig. 5.13) for usage in four different ways. Primitive datatypes with stereotype `<<simpleType>>` are used to represent XML Schema's built-in datatypes. Enumerations with stereotype `<<simpleType>>` are used to represent simple types with enumeration constraints. Datatypes with stereotype `<<simpleType>>` are used to represent other simple types. Finally, enumerations with stereotype `<<notations>>` are used to represent notation elements. Furthermore, specific dependencies are introduced to support representation of local datatypes (`<<local>>`), and of list (`<<itemType>>`) and union (`<<memberType>>`) type constructors (cf. Fig. 5.14).

The stereotype `<<simpleType>>` defines two derived attributes simplifying access to properties of XML Schema simple types modelled as datatypes. `<<variety>>` denotes the kind of simple type, which is implied by the way the simple type is constructed. If the simple type has a restriction generalization, then its variety is the same as the one of the general simple type. The root of the restriction hierarchy, which is the XML Schema simple ur-type, has variety "atomic". Otherwise, it is "list" if the simple type is client of an `<<itemType>>` dependency, or "union" if the simple type is client of a `<<memberType>>` dependency. `local` is true if the simple type is a local anonymous simple type, which can be derived from the datatype being the client of a `<<local>>` dependency.

A datatype stereotyped as `<<simpleType>>` can be furthermore specified by attaching one of the following constraints:

- A "final" constraint, defining boolean values for `restriction`, `list`, and `union`.
- If the simple type is derived by restriction, then some of the following restriction constraints may be attached: `whiteSpace`, `pattern`, `length`, `minLength`, `maxLength`, `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive`, `totalDigits`, and `fractionDigits`. Each of them defines a constraint value, and optionally whether that value is fixed or may be overridden in specialized simple types.

The following metamodel constraints apply to a datatype stereotyped as `<<simpleType>>`:

- The datatype must have either exactly one generalization of stereotype `<<restriction>>`, or exactly one dependency of stereotype `<<itemType>>`, or one or more dependencies of stereotype `<<memberType>>`.
- If the datatype is a primitive datatype, then the simple type must be one of the XML Schema primitives.

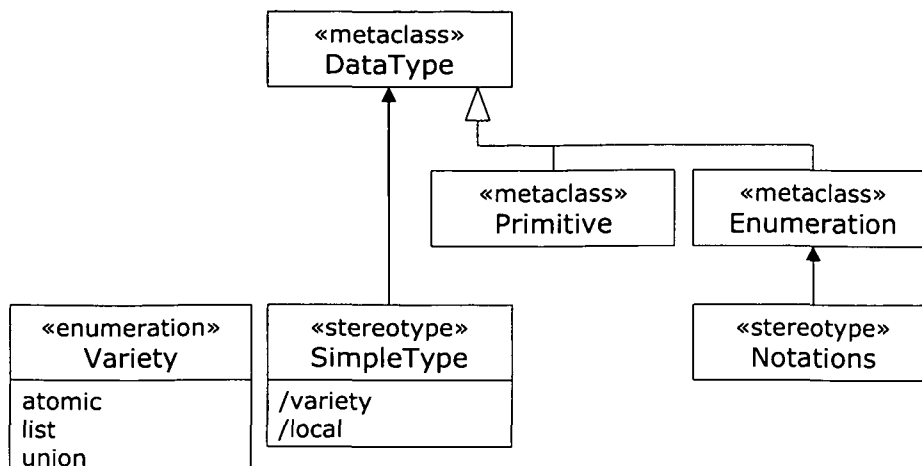


Figure 5.13: Datatype Stereotypes

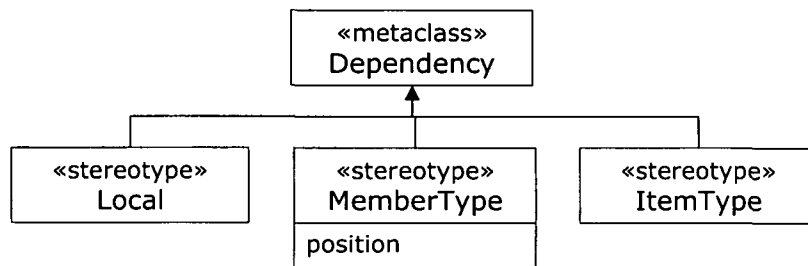


Figure 5.14: Datatype-related dependency stereotypes

- If the datatype is an enumeration, it must have a generalization of stereotype `«restriction»` and all enumeration literals of the enumeration must be instance specifications of the general datatype.
- If a “final” constraint for `restriction` is attached, the datatype must be a leaf datatype.

The dependency stereotypes `«local»`, `«itemType»`, and `«memberType»` imply the following metamodel constraints:

- A `«local»` dependency must have a `«simpleType»` as both client and supplier.
- An `«itemType»` dependency must have a client of stereotype `«simpleType»` with `variety=“list”`, and a supplier of stereotype `«simpleType»` with `variety` either “atomic” or “union”.
- A `«memberType»` dependency must have a client of stereotype `«simpleType»` with `variety=“union”` and a supplier of stereotype `«simpleType»` (any variety).

Stereotype `«notations»` refines enumeration with the semantics and syntactic constraints of XML Schema notations. The following metamodel constraints apply to an enumeration stereotyped as `«notations»`:

- The enumeration must be named “Notations”, and it must be the only one with this stereotype contained in an `«xml-schema»`.
- The enumeration must have generalization relationships to all notations enumerations in all `«xml-schema»` packages included by the containing package.
- The enumeration literals must all be Expression elements of the form “notation(*public, system*)”.

5.3.6 Comment Stereotypes

Comments are extended to support the different aspects of XML Schema’s annotation concept (cf. Fig. 5.15). In particular, stereotypes are defined supporting representation of user documentation (`«documentation»`), application data (`«applInfo»`), and for capturing XML namespace prefixes (`«xmlns»`).

The properties of stereotypes `«documentation»` and `«applInfo»` conform to the attributes of the corresponding XML Schema elements. Since the contents of these XML elements may contain references to namespace prefixes defined outside these elements, the definition of namespace prefixes must be captured separately, in order to enable parsing of the comment’s contents without the context of the complete XML Schema document. For this purpose, stereotype `«xmlns»` is provided. Depending on the model element a comment containing namespace declarations is attached, different scoping rules apply. An `«xmlns»` attached to an `«xml-schema»` package defines namespace prefixes common to all comments in that package. An `«xmlns»` applied to an element which has also `«documentation»` and `«applInfo»` comments attached, defines the namespace prefixes for these comments. An `«xmlns»` attached to a `«documentation»` or `«applInfo»` comment defines namespace prefixes for that particular comment.

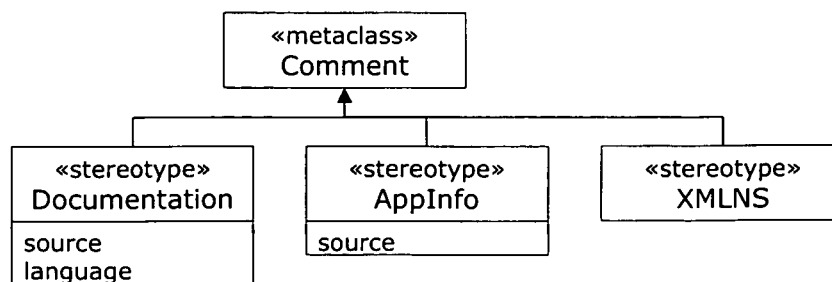


Figure 5.15: Comment stereotypes

The following metamodel constraints apply:

- Constraints stereotyped as «documentation» or «applInfo» may be applied to any element with a stereotype defined in this profile, including packages, but except of comments.
- «xmlns» may also be applied to any of these elements, but including comments as well.

5.4 Comparison and Outlook

We have shown how to represent XML schemas in UML, such that the representation expresses the semantics of XML Schema in terms of UML as far as possible, and such that the representation is minimized in terms of UML constructs. The representation is based on a UML profile to convey the semantics of XML Schema's concepts going beyond UML.

A comparison of the features of our approach to existing UML profiles is provided in Table 5.1, organized along the various representation patterns as identified in this paper. As can be seen, the main contributions of our approach are solutions to represent model groups (MG) as well as global elements (EG, EW) and global attributes (AG, AW) in a way more compliant to UML semantics, to represent identity constraints (KY), and to represent simple types in a more concise, UML like way (ST3-4). We further expect that the transformation approach presented in this paper improves understandability and integration of XML schemas in UML-based software development processes.

Evaluation of the profile transformation rules has been done based on the UML and XML Schema specifications, and by application to example schemas. Ongoing work includes the implementation of a prototype supporting the transformation rules, based on the XMI representation of UML, and the use of automatically generated UML models as basis for application development. In particular, we want to extend the UML representation of an XML schema with behavior that is capable of performing run time processing, i.e., reading, writing, and verifying XML instance documents matching the respective XML schema.

Table 5.1: Comparison of UML profiles by representation patterns

	SC	CT		ST				EL		EG	EW
		1	2	1	2	3	4	1	2		
[14]	+	+		/	-			-	+		-
[24]			-	-	-			+	+		-
[69]	+	+	-	/	-			+	+	/	-
[65]		+	+	+	-			+	+		
OA	+	+	+	+	+	+	+	+	+	+	+

	AL	AG	AW	MG		KY		GA	GE		AN	NO
				1	2	1	2		1	2		
[14]	+			+								
[24]	+		/	+								-
[69]	+			+			-	-	+		-	
[65]	+			-								
OA	+	+	+	+	+	+	+	+	+	+	+	+

Legend: + ... good support
 - ... weak support (incomplete)
 / ... violation of UML semantics
 space ... not supported
 OA ... our approach

Chapter 6

Approaches to Extending XML Schema

Tailored schema languages define domain concepts thus semantics once and for all across schemas. For instance, the Active XML Schema approach defines active behavior within XML schemas along metadata, and stores traces of active behavior such as occurred events within XML documents along data. The semantic expressiveness of XML Schema, the schema language recommended by the W3C, however, is not sufficient to define the active semantics of Active XML Schema concepts. The contribution of this paper is to identify, explore, and evaluate approaches to implementing the tailored schema language Active XML Schema with XML Schema, discussing the trade-off between semantic expressiveness and interoperability. Assuming that Active XML Schema may be seen as representative for tailored schema languages, the findings of this paper can be applied for arbitrary tailored schema languages.

6.1 Introduction

Tailored schema languages define domain concepts thus semantics once and for all across schemas. In relational databases for example, the schema language defines concepts such as tables and foreign keys, constituting modelling primitives for database schemas. Applications exhibiting event driven, active behavior are another example where the use of a dedicated schema language is favorable. Such a tailored schema language defines the semantics of active concepts such as event-condition-action (ECA) rules and event types independent of individual application schemas.

The UML meets the need for supporting tailored schema languages through its extension mechanisms. Profiles are a light-weight approach to extending and refining UML in terms of constraints and stereotypes. Furthermore, based on MOF, i.e., UML's metamodelling infrastructure, new languages can be defined independent of UML but based on a common metamodel. Both of these extension approaches can be used in model driven development. As different trade-offs regarding expressive power and conformance to UML have to be made, the selection of an extension approach depends on the requirements of the tailored schema language [71].

As the usage of XML increases, also the need for tailored XML schema languages, which go beyond the semantic expressiveness of XML Schema [93, 94], arises. This goes in parallel with

the emerging practice to define an XML syntax both for schemas and instances (e.g., as RDF [88, 95] does).

For instance, the Active XML Schema approach [72] provides an XML syntax both for schemas and instances. Its schema language¹ allows to define circumstances having *intensional aspects* and/or *extensional aspects*. While the former refers to circumstances that only affect a schema, such as ECA rules, the latter refers to circumstances that only affect instances of a schema, such as the structure of occurred events.

Using XML for schemas and instances instead of using other data formats is beneficial with respect to *interoperability*, *openness*, and *integration*. This means that schemas and instances described with XML syntax are accessible under various platforms and environments, they can be extended by employing XML namespaces [87], and they can be integrated with other XML standards such as XLink [92], XSLT [90], and RDF. However, defining XML syntaxes for both schemas and instances is not advantageous at first, since it adds an extra layer of complexity when defining metaschemas that needs to be handled properly.

The contribution of this paper is to identify, explore, and evaluate approaches to implementing tailored metaschemas with XML Schema. In particular, four approaches with distinct characteristics are presented. They are explored and applied to Active XML Schema, giving insight into their respective implications. Furthermore, the approaches are evaluated with respect to criteria that have been identified to be relevant for the quality of a tailored metaschema's implementation.

Since Active XML Schema comprises concepts that have intensional and extensional aspects, it can be assumed to be a representative for tailored metaschemas. Thus the paper generalizes statements about Active XML Schema to statements about tailored metaschemas. However, keep in mind that the findings presented in this paper, except for the approaches and evaluation criteria themselves, are based on experiences in implementing the metaschema of Active XML Schema.

The paper is organized as follows. After giving a brief overview of Active XML Schema in Section 6.2, Section 6.3 defines criteria by which the presented approaches are evaluated. Section 6.4 presents the four approaches and ranks them according to the criteria. Section 6.5 summarizes the approaches by directly comparing them to each other and discusses examples from theory and practice. Section 6.6 finally concludes the paper.

6.2 An Overview of Active XML Schema

Active XML Schema and related approaches (e.g., [1, 9, 10]) have been proposed for enriching XML with active behavior as known from active databases [63]. They specify active behavior by ECA rules, which allow to automatically perform an action as reaction to an occurred event if a given condition applies. Active XML Schema has several unique features, which are discussed in the following as far as necessary to understand the examples throughout the paper.

First, *active behavior* is defined at the schema level so that it comes with a schema and is used for all its instances. Furthermore, a mechanism is provided to define *passive behavior*, i.e.,

¹For the purpose of readability, we use the term metaschema instead of schema language throughout the rest of the paper. If we talk about a schema expressed in XML Schema, we concisely call it XML schema.

operations on documents. Thus a so-called active document type defines instances' structure as well as passive and active behavior.

Second, events are *first class objects*, which have an *event type* describing their internal structure. Event types are classified into different kinds of events. Among them *operation events* are important in this paper's context, they occur upon executions of operations. Per default an operation event type is defined for each operation defined with a document type. Events are collected in *event classes*, where each class collects events of an event type, called its member type.

Third, *distributed events* are provided by employing the publish/subscribe model from event based systems. This enables active XML documents to import event classes from remote documents. The import of an event class is defined at the schema level, whereby the remote document from which the event class is imported is represented by a typed *proxy* property, which is bound to an actual document at the instance level. A *remote event* is an event that occurs in a remote document. When imported into a local document it is wrapped in a local, so called *imported event* which provides two additional time stamps to keep track when the remote event was published (called publication time) and when it was delivered to the importing document (called delivery time). This allows to detect exceptional situations, where event delivery was significantly delayed.

Example 1. A job agency provides an Active XML schema defining active document type `j:JobAnnounce` and a document `academicJobs.xml` having that type, which comprises a list of current job offers. A new job offer is announced by an invocation of operation `announce(j:Job)`, which is defined by the active document type. It adds new job offer `j` at the end of the list. A university's science faculty posts, as a courtesy to its staff and students, relevant job offers supplied by the job agency at its document `science.xml` of active document type `u:Faculty`. The latter imports event class `announce`, using proxy `jobSite` to refer to the document from which the remote event class is imported. The proxy's value is bound in instance `science.xml` to document `academicJobs.xml`. Announced job offers are now locally available within a faculty's page in the form of events contained in the imported event class.

6.3 Evaluation Criteria

This section presents criteria to determine the quality of the different approaches. These quality criteria, which we have identified to be relevant in the narrow context of implementing tailored metaschemas, are related to quality factors proposed in literature [8, 37] to facilitate a better understanding of their implications. This is only done informally since it is not the focus of this paper.

Semantic expressiveness describes how much semantics is expressed formally and concisely by a schema. Since semantics is defined by the tailored metaschema, a schema expressed therein is most expressive. When using XML Schema instead, semantics of the tailored metaschema need to be mapped to XML Schema. Because usually not all semantics can be mapped, schemas expressed in XML Schema are less expressive. The more semantics is explicit in a schema, the better the schema can be verified against an explicit metaschema such that errors can be detected at design time. Semantic expressiveness influences quality factors such as understandability,

maintainability, testability, and reusability.

Schema interoperability in general describes the ability of a system to exchange schemas with other systems and interpret them. Interoperability also affects design time of a system in that it allows to reuse schema components from interoperable schemas and to reuse software components implemented for interoperable schemas and metaschemas. In the paper's specific case schema interoperability describes the ability of standard XML software to process schemas that have been created following each of the presented approaches. Thus it directly affects the extent of reusing standard XML software when writing applications that process such schemas. This criteria influences quality factors such as interoperability, flexibility, and portability.

Locality of change describes the self-containedness of a schema such that a change in one schema component does not require subsequent changes in dependent schema components in the same or other schemas. It is negatively affected by redundantly modelled schema components (i.e., components that model the same circumstance by different concepts) and non-atomic schema components (i.e., a circumstance is modelled by several dependent components). Locality of change is influenced by two aspects: First by the schema's environment (i.e., the employed metaschema and its imposed usage), and second, by the design of a given schema. Because the second aspect is application specific and thus independent of the presented approaches, we focus on the first aspect. Locality of change influences quality factors such as understandability, maintainability, and integrity.

6.4 Approaches

This section describes four approaches to implementing a tailored metaschema and ranks them with respect to the evaluation criteria presented in Section 6.3.

To talk about the various levels of tailoring, we adopt a four layer metadata architecture as for example proposed by OMG's Meta Object Facility [61]. The layers comprise the instance layer (short "M0") for data, the schema layer ("M1") for metadata describing data, the metaschema layer ("M2") describing metadata, and the meta-metaschema layer ("M3") describing meta-metadata. Between elements of the various layers instance-of relationships exist in the sense that an element of M_i is an instance of an element of $M_{(i+1)}$ with the exception that depending on the underlying conceptual model an element of M3 can be seen as an instance of itself.

6.4.1 Proprietary Schema Approach

This approach expresses schemas directly in terms of the tailored metaschema, constituting the most "natural" approach with respect to schema design. As shown on the left of Figure 6.1, schema s (an XML document) is created by instantiating tailored metaschema m (e.g., an XML schema). Instance data in turn is created by proprietarily instantiating schema s .

For intensional aspects, i.e., aspects that apply for all instances but have no corresponding materialization at the instance level, it is not necessary to consider an XML syntax for M0.

Example 2. The tailored metaschema and the exemplary proprietary schema below show

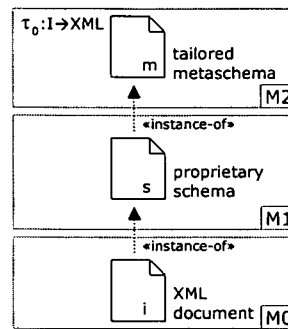


Figure 6.1: Proprietary Schema approach

how rules are defined and expressed by element `actm:rule`. A rule is identified by its name and is defined upon an event class (attributes `name` and `on`), it comprises a condition and an action (elements `condition` and `action`). Since ECA rules only have intensional aspects, no XML syntax needs to be considered for M0.

(M2) Tailored metaschema with target namespace `actm`:

```
<xs:element name="rule" ..> ..
  <xs:sequence>
    <xs:element name="condition" ../>
    <xs:element name="action" ../>
  </xs:sequence>
  <xs:attribute name="on" type="xs:NMTOKEN" ../> ..
  <xs:attribute name="name" type="xs:NMTOKEN" ../> ..
</xs:element>
```

(M1) Exemplary proprietary schema:

```
<rule on="jobSite.announce" name="announceJobRule" ..>
  <condition>..</condition>
  <action>..</action>
</rule>
```

For extensional aspects it is necessary to define so-called *instance transformation function* $\tau_0: I \rightarrow XML$. It defines how a proprietary instance is transformed into an XML document. Moreover an inverse function τ_0^{-1} must exist to transform an XML document back into a proprietary instance. Note that τ_0 is defined at M2, i.e., independent of a particular schema. Therefore it can be reused across applications.

Example 3. The tailored metaschema below defines the import of an event class by element `actm:importEventClass`. The exemplary proprietary schema imports event class `announce` from a remote document represented by proxy `jobSite`. Finally, the imported event class and events contained therein materialize at the instance level.

(M2) Tailored metaschema with target namespace actm:

```

01 <xs:element name="importEventClass" ..>
02   <xs:complexType>
03     <xs:attribute name="name" type="xs:QName" ../>
04     <xs:attribute name="proxy" type="xs:QName" ../>
05     <xs:attribute name="remoteEvtCsName" type="xs:QName" ../>
06     <xs:attribute name="hasMemberType" type="xs:QName" ../> ..
07   </xs:complexType>
08 </xs:element>

```

(M1) Exemplary proprietary schema:

```

09 <actm:importEventClass name="jobSite.announce"
10   proxy="jobSite" remoteEvtCsName="announce" .. />

```

(M0) Exemplary instance:

```

11 <jobSite.announce>
12   <event id="i19" deliveryTime="2003-03-01T12:14.00.00" ..>
13     <remoteEvent id="e47" occurrenceTime="2003-05-01T12:13:14.15" .. > ..
14     </remoteEvent>
15   </event>
16   <event id="i20" ..> .. </event> ..
17 </jobSite.announce>

```

Semantic expressiveness is high because using a tailored metaschema makes all semantics explicit at the schema level. However, since these schemas are expressed in a proprietary format, *schema interoperability* is low because standard XML software cannot interpret the proprietary schema. This also affects implementation aspects, since standard XML software cannot be reused to validate instance documents against the proprietary schema. *Locality of change* is high, because the tailored metaschema does not impose redundant or non-atomic schema components.

6.4.2 Side by Side Approach

This approach is similar to the Proprietary Schema approach in that it uses an explicit tailored metaschema to define schemas. However, an XML schema is provided in addition, which does not replace the proprietary one but stands side by side to it. Likewise, instance transformation function τ_0 is still used to serialize instances as XML.

The transformation of a proprietary schema into an XML schema is defined at M2 by so-called *schema transformation function* $\tau_1: S \rightarrow XSD$ and applied at M1 as depicted in Figure 6.2. Function τ_1 can be derived from τ_0 , since the latter defines the structure of XML documents implicitly. While τ_0 is used at runtime, i.e., when documents are processed, τ_1 is used at design time, i.e., when schemas are created. Because only extensional aspects of the proprietary schema are transformed to an XML schema, τ_1 is *partial*.

Example 4. The result of transforming extensional aspects of proprietary schema s (cf. Example 3) to XML schema s' by τ_1 is shown below. It defines the imported event class as element `announce`, which contains a sequence of `event` elements, each representing an imported event and in turn containing the wrapped remote event as element `remoteEvent`.

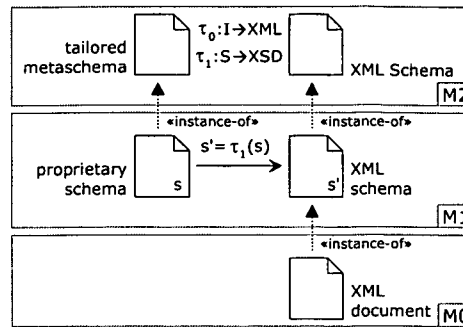


Figure 6.2: Side by Side approach

(M1) Exemplary XML schema:

```

01 <xs:element name="jobSite.announce" actm:eventClass="jobSite.announce">
02   <xs:complexType><xs:sequence>
03     <xs:element name="event" ..>
04       <xs:complexType><xs:sequence>
05         <xs:element name="remoteEvent" ..> .. </xs:element>
06       </xs:sequence></xs:complexType>
07     <xs:attribute name="id" type="xs:ID" ../>
08     <xs:attribute name="deliveryTime" ../>
09     <xs:attribute name="publicationTime" ../>
10     <xs:attribute name="deliveryTime" ../>
11   </xs:element>
12 </xs:sequence></xs:complexType>
13 </xs:element>

```

Still using a tailored metaschema to model schemas results in high *semantic expressiveness*. But in contrast to the Proprietary Schema Approach, this approach provides an XML schema for extensional aspects, resulting in higher *schema interoperability*. Thus standard XML software can be used to validate instance documents at the cost of implementing τ_1 to transform schemas. Implementation of applications is supported by providing explicit links from components in s' to components in s (cf. attribute `actm:eventClass` in Example 4). Having two schemas expressing the same circumstances redundantly by components in terms of different metaschemas makes it necessary to keep them synchronized. Thus *locality of change* is low.

6.4.3 Framework Approach

This approach uses only an XML schema that expresses all circumstances formerly modelled by the proprietary schema as shown in Figure 6.3. Thus it eliminates the need for proprietary schema s , transformation τ_1 , and synchronization of s with s' .

Since intensional aspects are orthogonal to XML Schema, they can be expressed easily using XML Schema extension mechanisms (annotations and foreign attributes). Expressing exten-

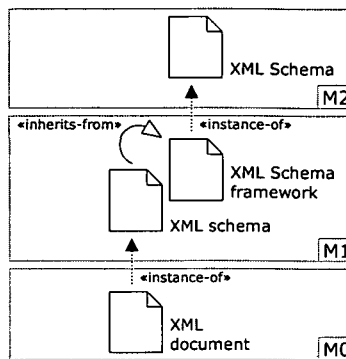


Figure 6.3: Framework approach

sional aspects is more complicated as they must be expressed solely with concepts provided by XML Schema.

The framework concept as known from object-oriented programming [40] can help in this situation. A framework is a means to provide a base schema common to all applications, along with conventions for its adaption and usage in the design of particular schemas. XML Schema provides a set of concepts that can be employed in framework design, such as abstract types, type derivation, abstract elements, and substitution groups (see [52] for a brief overview). Therefore an XML Schema framework comprises a set of reusable and/or specializable elements and types, which form the base schema, and a set of informal conventions describing their reuse and specialization.

Example 5. The Active XML Schema framework below defines the structure of event classes which are represented by `actf:eventClass` elements that are of abstract type `actf:TEventClass`. Moreover, abstract type `actf:TEventType` describes events, which comprise an identifier (attribute `id`), occurrence time (attribute `occurrenceTime`), and status (attribute `status`). Type `actf:TEventType` is directly or indirectly extended by specialized event types which are provided by the framework for all kinds of events (e.g., `actf:TOperationEvtTp` for operation events and `actf:TImportedEvtTp` for imported events). Finally, in addition to event types, reusable event classes are provided by the framework (e.g., `actf:TEvtCs_ImportedEvtTp` is a special event class having event elements of type `actf:TImportedEvtTp`, a specialized `actf:TEventType`).

(M1) XML Schema framework with target namespace `actf`:

```

01 <!-- Abstract base event type and class -->
02 <xs:element name="eventClass" type="actf:TEventClass"/>
03 <xs:complexType name="TEventClass" abstract="true"/>
04 <xs:complexType name="TEventType" abstract="true">
05   <xs:attribute name="id" type="xs:ID" ../>
06   <xs:attribute name="occurrenceTime" type="xs:dateTime" ../>
07   <xs:attribute name="status" type="actf:TEvtStatus" ../>
08 </xs:complexType>
09 <!-- Event type and class for operation events -->
10 <xs:complexType name="TOperationEvtTp" abstract="true"> ..

```



```

12 <xs:extension base="actf:TEventType"><xs:sequence>
13   <xs:element name="return" nillable="true"> .. </xs:element>
14   <xs:element name="diff" nillable="true" minOccurs="0"> .. </xs:element>
15 </xs:sequence></xs:extension> ..
16 </xs:complexType>
17 <xs:complexType name="TEvtCs_OperationEvtTp"> ..
18   <xs:extension base="actf:TEventClass"><xs:sequence>
19     <xs:element name="event" type="actf:TOperationEvtTp" ../>
20   </xs:sequence></xs:extension> ..
21 </xs:complexType>
22 <!-- Event type and class for imported events ->
23 <xs:complexType name="TImportedEvtTp" abstract="true"> ..
24   <xs:extension base="actf:TEventType"><xs:sequence>
25     <xs:element name="remoteEvent" type="actf:TEventType"/>
26   </xs:sequence> .. </xs:extension> ..
27 </xs:complexType>
28 <xs:complexType name="TEvtCs_ImportedEvtTp"> ..
29   <xs:extension base="actf:TEventClass"><xs:sequence>
30     <xs:element name="event" type="actf:TImportedEvtTp" ../>
31   </xs:sequence></xs:extension> ..
32 </xs:complexType>

```

To some extent, conventions defining the reuse and specialization of schema components provided by an XML Schema framework can be enforced by mechanisms of XML Schema. For example, an abstract type must be specialized before it can be used, or an abstract substitution group's head has to be substituted by an element of an appropriate type. Unfortunately, in many cases these mechanisms are not sufficient to enforce a correct usage of the framework. Therefore, schema designers must know and follow informal conventions regarding the use of framework components.

Example 6. Modelling the exemplary schema based on the XML framework requires the definition of the following (as shown below). First, event type `j:TImported_TExecAnnounceEvtTp` is defined for imported events, which is done by extending event type `actf:TImportedEvtTp`. Since element `actf:remoteEvent` cannot be refined by the exemplary schema (because it is not in the framework's namespace), an annotation is provided that indicates that such elements (which represent remote events) shall be of type `j:TExecAnnounceEvtTp` in instance documents. Second, a corresponding event class (element `j:jobSite.announce`) is defined by making it part of the substitution group headed by `actf:eventClass`. The type of the event class's `actf:event` elements is defined in an annotation as `j:TImported_TExecAnnounceEvtTp`, due to the same reasons as with the event type for imported events. Proxy `j:jobSite`, which is an intensional aspect, is defined within an annotation. Note that the use of the annotation as well as the definition of parallel type hierarchies comprising event types and event classes are informal conventions, i.e., not enforceable by the XML Schema framework.

(M1) Exemplary XML schema with target namespace *u*:

```

01 <!-- Imported event type -->
02 <xs:complexType name="TImported_TExecAnnounceEvtTp"> ..
03   <xs:extension base="actf:TImportedEvtTp">
04     <xs:annotation><xs:appinfo>
05       <actm:remoteEvtTp="j:TExecAnnounceEvtTp"/>
06     </xs:annotation></xs:appinfo>
07   </xs:extension> ..
08 </xs:complexType>
09 <!-- Imported event class -->
10 <xs:element name="jobSite.announce"
11   type="actf:TEvtCs_ImportedEvtTp" substitutionGroup="actf:eventClass">
12   <xs:annotation><xs:appinfo>
13     <actm:proxy name="j:jobSite" forDocType="j:JobAnnounce" type="single"/ >
14     <actm:hasMemberType="j:TImported_TExecAnnounceEvtTp"/>
15   </xs:appinfo></xs:annotation> ..
16 </xs:element>

```

Regarding the characteristics of the framework approach, most notably is the lack of *semantic expressiveness* of extensional aspects. This is exemplified by comparing the import of an event class by the proprietary schema shown in Example 3 with the above schema. Furthermore, informal conventions that must be followed when using a framework severely impact semantic expressiveness. On the positive side, *schema interoperability* is high since schemas (and frameworks) are expressed solely in XML Schema. *Locality of change* is medium since the framework may impose modifications of multiple schema components in order to achieve the modification of a single circumstance.

6.4.4 Specialized XML Schema Approach

This approach extends XML Schema with new concepts of the tailored metaschema as shown in Figure 6.4 opposed to the framework approach, which expresses new concepts of the tailored metaschema by XML Schema concepts. To relate new concepts to XML Schema concepts the mechanisms provided by XML Schema itself are used, because XML Schema at M2 is defined by an XML schema, (cf. [93]), which in turn assumes XML Schema at M3 (as one can see, XML Schema is meta-circularly defined [51]). Thus plenty of possibilities exist to relate concepts, such as element composition, type composition, or type derivation. Also redefinition as shown in [66] is an option. Note, however, while [66] focusses on restricting XML Schema, this approach focusses on extending it.

Depending on whether XML Schema is to be extended by intensional or extensional aspects, different procedures are followed. An extension with intensional aspects is simply a matter of adding new concepts to XML Schema without the need to relate them to existing concepts. On the contrary, extensional aspects *must* be defined as specializations of existing concepts such as elements and attributes, in order to inherit the extensional semantics of those concepts. Inten-

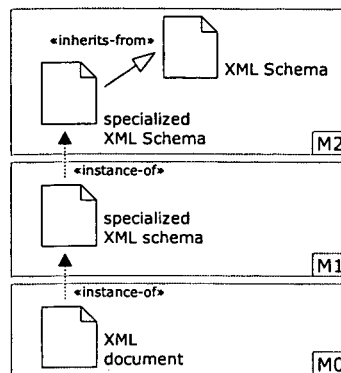


Figure 6.4: Specialized XML Schema approach

sional aspects thus have no standard meaning, i.e., they can be safely ignored by standard XML Schema validators.

This approach has the advantage that an XML Schema validator can interpret specialized XML schemas, because it is possible to derive the basic meaning of a schema component of a specialized concept from the XML Schema concept it is based on. Or in different terms, it is possible to perform a *downcast* according to the principle of type substitutability. Unfortunately, standard XML Schema validators currently do not provide for a plug-able XML Schema necessary for a downcast.

Example 7. In XML Schema, group `xs:schemaTop` defines the content of element `xs:schema`, the document element of every XML schema. The group defines a choice of elements `xs:element`, `xs:attribute`, and others. It is redefined² by the specialized XML Schema to include elements `actm:rule` and `actm:importEventClass`. Because a rule has only intensional aspects, element `actm:rule` can be declared by referencing the respective element declaration of the tailored metaschema depicted in Example 2. Because an imported event class has extensional aspects, element `actm:importEventClass` is indirectly derived from `xs:topLevelElement`, the type of a global element declaration in XML Schema. The two schema documents with different namespaces shown below form the specialized XML schema.

```

(M2) Specialized XML Schema with targetNamespace xs:
01 <xs:redefine schemaLocation="XMLSchema.xsd">
02   <xs:group name="schemaTop"><xs:choice>
03     <xs:group ref="xs:schemaTop"/>
04     <xs:element ref="actm:rule"/>
05     <xs:element ref="actm:importEventClass"/> ..
06   </xs:choice></xs:group>
07 </xs:redefine>
08 <xs:complexType name="actm.EventSequence"> ..
09   <xs:restriction base="xs:topLevelElement"> .. </xs:restriction>
  
```

²The redefinition of `xs:schemaTop` is a group redefinition that contains a reference to itself. Thus, it is semantically equivalent to a derivation by extension, being applied to an element group instead of a complex type.

```
10 </xs:complexType>
```

(M2) *Specialized XML Schema with targetNamespace actm:*

```
01 <xs:element name="importedEventClass"> ..
02   <xs:extension base="xs:actm.EventSequence"> ..
03     <xs:attribute name="proxy" type="xs:QName" use="required"/>
04     <xs:attribute name="remoteEvtCsName" type="xs:QName" use="required"/>
05     <xs:attribute name="exported" type="xs:boolean" use="required"/>
06   </xs:extension>
07 </xs:element>
```

On the negative side, the power of a downcast is very limited compared to an explicitly defined schema transformation τ_1 . In particular, a specialized concept can not arbitrarily modify extensional semantics of its base concept. For instance, the extensional semantics of a specialized element is always limited to that of exactly one element. Therefore it is not possible to define a particular composition of elements by means of one specialized element.

Overall *semantic expressiveness* is medium, whereby the semantic expressiveness of intensional and extensional aspects differ. It is high for intensional aspects because they are expressed in terms of their unconstrained metaschema. It is medium for extensional aspects, because their metaschema is constrained by the concepts of XML Schema. If standard XML Schema validators provide for a plug-able XML Schema, *schema interoperability* will be high since they can interpret specialized XML Schemas. Unfortunately, in practice this is not yet the case causing low interoperability. Concerning *locality of change* it is advantageous that only one metaschema is employed. However, one concept of the tailored metaschema is possibly expressed by several concepts of XML Schema, producing several dependent schema components. Therefore locality of change is medium.

6.5 Comparison and Related Work

When comparing the approaches' characteristics summarized in Table 6.1, it gets evident that there is a *tradeoff* between semantic expressiveness and schema interoperability. The Proprietary Schema approach defines new concepts at M2 not defined by XML Schema and thus imposes proprietary schemas at M1, resulting in high expressiveness but low interoperability. The Side by Side Approach tries to overcome this by defining a transformation from new concepts to XML Schema concepts at M2 and applying it to proprietary schemas at M1. This increases interoperability, however, at the cost of locality of change. The Framework approach goes one step further by expressing new concepts by XML Schema concepts at M1, having positive effects on interoperability and locality of change, but negative effects on expressiveness. Finally, the Specialized XML Schema approach directly extends XML Schema with the new concepts at M2. It suffers from the lack of support by existing XML Schema validators and the constraints of the underlying XML Schema.

Since there is no single best approach, one has to choose the most appropriate one based on given requirements. In case schemas will change often, locality of change is the primary criterion

Table 6.1: Characteristics of the presented approaches

Criteria		Proprietary Schema	Side by Side	Framework Specialized XML Schema	
Semantic siveness	expres-	high	high	low	med.
Schema ability	interoper-	low	med.	high	high [†]
Locality of change		high	low	med.	med.

[†] Assuming standard XML validators provide for a plug-able XML Schema

with the Proprietary Schema approach being favorable. In case instance documents have to be shared with other partners, schema interoperability is the primary criterion with the Proprietary Schema Approach falling behind. In practice it may be beneficial to use a mixed approach (for an example see Section 6.6).

Examples for approaches employing the *Proprietary Schema approach* are RDF and JavaBeans Persistence [77]. The RDF standard defines RDF's tailored metaschema and a syntax of RDF instances as XML by an EBNF grammar. This grammar can be seen as a declarative specification of τ_0 . In addition, the RDF Schema (RDFS) standard defines a proprietary schema language for RDF. Going beyond XML, JavaBeans Persistence provides for serialization of JavaBean objects as XML documents. It realizes τ_0 by a dedicated Java class. Here, the Java language constitutes the tailored metaschema.

Among approaches following the *Side by Side approach* are [44, 56, 67]. [44] describes transforming OIL [33] ontologies to XML schemas by textually describing τ_1 that transforms OIL concepts to XML Schema concepts. Independent of XML, Microsoft's ADO.NET DataSet [56] implements among others a generic mapping between the relational model (constituting the tailored metaschema) and XML. It allows to read relational data and to write XML data with its XML schema and vice versa, thus implementing τ_0 , τ_1 , and their inverse. The Side by Side approach has been also extensively explored in [67] recently, yielding an abstract algebra for model mapping [6] (the notion of "model" in [6] corresponds to "schema" in our approach).

The *Framework approach* has not been employed in the XML field yet. We suspect that a major reason for not employing it is that frameworks usually evolve from simple XML schemas instead of being created from scratch by implementing a tailored metaschema. Going beyond XML, an example of fostering the Framework approach is UML with its extension mechanisms [70]. Thus, instead of extending the UML metaschema at M2 (called metamodel in UML), the extension mechanisms provide a means to customize UML at M1. Extension mechanisms are the main concepts to build reusable frameworks, called profiles in UML. Another example of providing a new semantic concept at M1 is the role pattern [3, 41]. For example, it has been

implemented in Smalltalk in terms of a predefined framework [30].

The *Specialized XML Schema approach*, as the Framework approach, has not been employed in the XML field yet. A major reason could be not wanting to lose interoperability. However, tailoring metaschemas is well known in the non-XML literature. So-called open data models have been proposed in the past (e.g., [43, 53]), which consist of a few built-in concepts but which can be extended by additional modeling concepts at M2 for specific application needs.

6.6 Conclusion

In this paper four different approaches to implementing tailored XML metaschemas in terms of XML Schema have been discussed and compared concerning the trade-off between semantic expressiveness and interoperability.

When implementing Active XML Schema we decided to follow a mixed approach, mixing Side by Side and Framework approach. We employed the Side by Side approach to provide maximum semantic expressiveness for human modelers. Extensional aspects of proprietary schemas are transformed to XML schemas adhering to a dedicated framework. Thus we are fully interoperable and able to reuse standard XML software. Employing both approaches in combination minimizes the required transformation functionality that has to be provided by τ_1 . Since Active XML schemas are assumed not to change often, low locality of change is not considered a problem.

After all, we would highly welcome efforts to foster the Specialized XML Schema approach. Most important, standard XML validators should provide for a plug-able XML Schema so that they can perform a downcast, and XML Schema should provide additional mechanisms to define extensional aspects of new concepts as specializations of XML Schema concepts more easily.

Chapter 7

Outlook

The work presented in this thesis leaves several issues open for further research. First, extension of the coverage of the profile from the data aspect to also include behavior. Second, implementation of the profiles, and evaluation of the presented modelling concepts in their practical application. Third, and most visionary, extensions to the modelling concepts regarding support of model synchronization as well as support for integration of heterogeneous workflows.

- The profile developed in Chapter 5 covers only the data aspect of inter-organizational workflows. Complete modelling of B2B protocols, however, also requires consideration of behavior and other aspects, as discussed in chapters 2 and 3. Of particular interest are UML profiles for platform-specific modelling of B2B protocols supporting the main target technologies, BPEL and ebXML. Furthermore, to achieve independence of these technologies, a platform-independent UML profile is needed which abstracts away the peculiarities of these technologies, thus leading to a higher-level model.
- The practical implementation of profiles and corresponding model transformations is not as straight forward as it may appear at first. The MDA is not just a matter of modelling methods and techniques, but essentially of modelling and model management tools. Tool support for MDA, however, is still immature, providing very basic functionality for the definition of user-defined profiles and the application of these profiles only. More sophisticated features, such as model transformations, consistency checks of models, and code generation are mostly limited to built-in functionality. More open tools are required supporting the specification of user-defined profiles and transformations such as those presented in this thesis.
- Evaluation of the modelling techniques and concepts proposed in this thesis in practical applications is required to determine their suitability. As already noted above, the power of model driven development lies in its potential of automation and tool support. Therefore, an evaluation must consider the different factors in model driven development independently, i.e., suitability and expressiveness of modelling concepts, quality and extensibility of model transformation rules, features and usability of model development tools, etc.

On the other hand, the value of a model driven development approach to realizing inter-organizational workflows can only be determined from the combination of these factors.

- Considering the relationships between platform-independent and platform-specific models, two extensions to the current profiles are suggested. First, based on OMG's forthcoming model transformation language (cf. [29]), the UML profiles can be extended with model transformation capabilities, providing for automatic transition between platform-independent and platform-specific models in both directions. Second, in situations where there are design decisions involved requiring user's expertise, there is a need for supporting a kind of loose coupling between platform-independent and platform-specific models. Mapping rules could be used in consistency checks as well as in interactive design support, providing help in most typical cases. The work on transformation patterns presented in Chapter 5 is a starting point in defining such mapping rules. Further research needs to be done considering application of such mapping rules in behavior modelling as well. Furthermore, an extensible specification formalism for mapping rules is needed, such that users can extend rules for particular application areas.
- An anticipated advantage of the MDA specifically relevant to development and maintenance of inter-organizational workflows is its capability to integrate heterogeneous systems [57]. It is envisaged that systems integration is much more feasible on a platform-independent level than on an implementation technology level, as, first, heterogeneity of technology is not considered, and second, higher-level modelling concepts are considered easier to integrate. Integration of B2B protocols and workflows at a conceptual level has already been tackled in research (cf. [85, 54, 7]). Open issues are the application of these results in the context of UML, in particular, based on UML extensions for B2B protocols such as those defined in this thesis, and other extensions intended for modelling intra-organizational workflows and systems integration such as OMG's UML profile for enterprise distributed object computing (EDOC). Furthermore, it would be interesting to exploit the MDA capability of automatically generating bridges between heterogeneous technologies, e.g., for realizing bridges between ebXML and Web Service based systems.

Chapter 8

Acknowledgements

In acknowledging that I would not have been able to carry out this work on my own, I mention only the most important people who have contributed to this achievement.

First, I would like to thank Gerti Kappel and Werner Retschitzegger, my advisors. Gerti for setting high demands and providing room to fulfill them. Werner for being the most encouraging and motivating teacher I know of.

Also, I would like to thank Martin Bernauer for his inspiring ways of being a working colleague, office mate, contributor to some parts of this work, and partner for many interesting and sometimes fruitful discussions.

Finally, I thank Marianne and Karl Kramler, my parents, who have supported me both materially and morally throughout my studies.

List of Figures

1.1	Application of MDA to inter-organizational workflows	4
2.1	Perspectives in workflow specification	8
2.2	Overview of languages for interorganizational workflow specification	13
2.3	Suitability of Workflow Specification Languages	17
3.1	Layers of the conceptual framework and supporting languages	23
4.1	eCo layers and supporting languages	38
5.1	Representation of the AddPurchaseOrder schema and its dependencies	61
5.2	Representation of the LingualString and Description complex types (left) and representation of a simple type constructed by union (right)	62
5.3	Representation of a simple type local to BusinessObjectDocument (left) and representation of a simple type restriction according to ST3 (right)	63
5.4	Declaration of complex types AddPurchaseOrder and AddPurchaseOrderDataArea	65
5.5	Declaration of global element Add (left) and declaration of UserArea element and complex type (right)	66
5.6	Complex type SalesInformation's model group representation using MG1	68
5.7	Complex type SalesInformation's model group representation using MG2	69
5.8	Key element (from the schema for XML Schema)	71
5.9	Package and related stereotypes	73
5.10	Class stereotypes	74
5.11	Property stereotypes	77
5.12	Generalization stereotypes	77
5.13	Datatype Stereotypes	79
5.14	Datatype-related dependency stereotypes	79
5.15	Comment stereotypes	80
6.1	Proprietary Schema approach	87
6.2	Side by Side approach	89
6.3	Framework approach	90
6.4	Specialized XML Schema approach	93

List of Tables

2.1	Overview of Evaluation Results	12
3.1	Supported combinations of eCo layers and workflow aspects	22
3.2	Main distinguishing characteristics of the two approaches	34
5.1	Comparison of UML profiles by representation patterns	82
6.1	Characteristics of the presented approaches	95

Bibliography

- [1] J. Bailey, A. Poulouvasilis, and P. T. Wood. An Event-Condition-Action Language for XML. In *Proceedings of the 11th International Conference on World Wide Web (WWW11)*, Honolulu, USA, pages 486–495. ACM Press, 2002.
- [2] A. Basu and A. Kumar. Research Commentary: Workflow Management Issues in e-Business. *Information Systems Research*, 13(1):1–14, March 2002.
- [3] D. Bäumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle, and H. Züllighoven. Framework Development for Large Systems. *Communications of the ACM (CACM)*, 40(10):52–59, October 1997.
- [4] BEA, IBM, Microsoft, SAP, and Siebel. Business Process Execution Language for Web Services, Version 1.1. <http://ifr.sap.com/bpel4ws/BPELV1-1May52003Final.pdf>, May 2003.
- [5] M. Bernauer, G. Kappel, G. Kramler, and W. Retschitzegger. Specification of Interorganizational Workflows - A Comparison of Approaches. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, 2003.
- [6] P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A Vision of Management of Complex Models. *SIGMOD Record*, 29(4):55–63, 2000.
- [7] P. Bichler, G. Preuner, and M. Schrefl. Workflow Transparency. In A. Olivé and J.A. Pastor, editors, *Proceedings of the 9th International Conference on Advanced Information Systems Engineering (CAiSE '97)*, volume 1250 of *Lecture Notes in Computer Science*, pages 423–436. Springer Verlag, June 1997.
- [8] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering, San Francisco, United States*, pages 592–605, 1976.
- [9] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, USA*, 2002.
- [10] A. Bonifati, S. Ceri, and S. Paraboschi. Active Rules for XML: A New Paradigm for E-Services. *The VLDB Journal*, 10(1):39–47, 2001.

- [11] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. UML for XML Schema Mapping Specification. Rational White Paper, December 1999.
- [12] C. Bussler. B2B Protocol Standards and their Role in Semantic B2B Integration Engines. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, volume 24, pages 3–11. IEEE, 2001.
- [13] C. Bussler. Modeling and Executing Semantic B2B Integration. In *Proceedings of the 12th Int'l Workshop on Research Issues in Data Engineering: Engineering e-Commerce/e-Business Systems (RIDE'02)*. IEEE, 2002.
- [14] D. Carlson. *Modeling XML Applications with UML*. Addison-Wesley, 2001.
- [15] Q. Chen, U. Dayal, and M. Hsu. Conceptual Modeling for Collaborative E-business Processes. In *Conceptual Modeling - ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2001.
- [16] Q. Chen and M. Hsu. CPM Revisited - An Architecture Comparison. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 72–90. Springer, 2002.
- [17] Workflow Management Coalition. Interface 1: Process Definition Interchange - Process Model, Doc. No. WfMC TC 1016-P, Version 1.1 (Official Release), October 1999.
- [18] R. Conrad, D. Scheffner, and J. C. Freytag. XML Conceptual Modeling Using UML. In *19th International Conference on Conceptual Modeling (ER), Salt Lake City, Utah, USA*, volume 1920 of *Springer LNCS*, pages 558–571, 2000.
- [19] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, October 2003.
- [20] A. Dan and F. Parr. An Object Implementation of Network Centric Business Service Applications (NCBSAs): Conversational Service Transactions, Service Monitor and an Application Style. In *Business Object Workshop, OOPSLA'97*, 1997.
- [21] U. Dayal, M. Hsu, and R. Ladin. Business Process Coordination - State of the Art, Trends, and Open Issues. In *Proc. of the 27th VLDB Conference (VLDB2001)*, pages 3–13, 2001.
- [22] J. Dubray. OAGIS Implementation Using the ebXML CPP, CPA, and BPSS specifications v1.0. Open Applications Group (<http://www.openapplications.org/>), 2001.
- [23] A. Dussart, B. Aubert, and M. Party. An Evaluation of Inter-Organizational Workflow Modelling Formalisms. Centre Interuniversitaire de Recherche en Analyse des Organisations (CIRANO), <http://ideas.repec.org/p/cir/cirwor/2002s-64.html>, 2002.

- [24] R. Eckstein and S. Eckstein. *XML und Datenmodellierung*. dpunkt.verlag, 2004.
- [25] eCo Working Group. eCo Architecture for Electronic Commerce Interoperability. <http://eco.commerce.net/rsrc/eCoSpec.pdf>, June 1999.
- [26] R. Elmasri, Y. Wu, B. Hojabri, C. Li, and J. Fu. Conceptual Modeling for Customized XML Schemas. In *21st International Conference on Conceptual Modeling (ER), Tampere, Finland*, volume 2503 of *Springer LNCS*, pages 429–443. Springer, 2002.
- [27] A. Banjeri et al. Web Services Conversation Language (WSCL) 1.0. W3C Note, March 2002.
- [28] D.A. Chappel et al. *Professional ebXML Foundations*. Wrox Press Ltd., UK, 2001.
- [29] T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of OMG MOF 2.0 Query/View/Transformations Submissions and Recommendations towards the final Standard.
- [30] G. Gottlob, M. Schrefl, and B. Röck. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems (TOIS)*, 14(3):268–296, 1996.
- [31] The Object Management Group. Model Driven Architecture. <http://www.omg.org/mda>, 2004.
- [32] B. Haugen and T. Fletcher. Multi-Party Electronic Business Transactions, Version 1.1. <http://www.supplychainlinks.com/MultiPartyBusinessTransactions.PDF>, December 2002.
- [33] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. The Ontology Inference Layer OIL. Technical Report IR-479, Vrije Universiteit Amsterdam, 2000.
- [34] C. Huemer. Defining Electronic Data Interchange Transactions with UML. In *Proceedings of the 34th Hawaiian International Conference on System Sciences (HICSS-34)*, January 2001.
- [35] IBM. WebSphere Business Integrator. <http://www-3.ibm.com/software/webserver/btobintegrator/>, 2002.
- [36] IBM, Microsoft, RSA, and VeriSign. Web Services Security Policy Language (WS-SecurityPolicy). <http://www.verisign.com/wss/WS-SecurityPolicy.pdf>, December 2002.
- [37] ISO. International Standard ISO/IEC 9126, Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use, 1991.
- [38] S. Jablonski. *Workflow-Management-Systeme: Modellierung und Architektur*. Thomson Publishing, 1995.

- [39] D. E. Jenz. The 'big boys' unite forces - What does it mean for you? <http://www.webservices.org/index.php/article/articleview/633/1/4/>, September 2002.
- [40] Ralph E. Johnson. Frameworks = (Components + Patterns). *Communications of the ACM (CACM)*, 40(10):39–42, 1997.
- [41] G. Kappel, W. Retschitzegger, and W. Schwinger. A Comparison of Role Mechanisms in Object-Oriented Modeling. In *Proceedings Modellierung '98*, pages 105–109, 1998.
- [42] G. Kappel, S. Rausch Schott, and W. Retschitzegger. A Framework for Workflow Management Systems Based on Objects, Rules and Roles. *ACM Computing Surveys Electronic Symposium on Object-Oriented Application Frameworks*, 2000.
- [43] W. Klas and M. Schrefl. *Metaclasses and Their Applications – Data Model Tailoring and Database Integration*, volume 943 of *Springer LNCS*. Springer-Verlag, 1995.
- [44] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The Relation between Ontologies and Schema-Languages: Translating OIL-specifications in XML-Schema. In *Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence (ECAI), Berlin, Germany*, 2000.
- [45] G. Kramler and W. Retschitzegger. Towards Intelligent Support of Workflows. In *Proceedings of Americas Conference on Information Systems (AMCIS 2000)*, pages 581–585. Online publication, August 2000.
- [46] H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). <http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, May 2001.
- [47] T. Krumbein and T. Kudrass. Rule-Based Generation of XML Schemas from UML Class Diagrams. In *In Proceedings of the XML Days at Berlin, Workshop on Web Databases (WebDB)*, pages 213–227, 2003.
- [48] F. Leymann. Web Services Flow Language (WSFL 1.0). <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, May 2001.
- [49] F. Leymann and D. Roller. *Production Workflow: concepts and techniques*. Prentice-Hall, 2000.
- [50] F. Lindert and W. Deiters. Modelling inter-organizational processes with process model fragments. In *GI WS Informatik 99*, October 1999.
- [51] P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings on the International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Orlando, Florida*, 1987.

- [52] A. Malik. Create Flexible and Extensible XML Schemas. <http://www-106.ibm.com/developerworks/library/x-flexschema/>, November 2002.
- [53] F. Manola and S. Heiler. A 'RISC' Object Model for Object System Interoperation: Concepts and Applications. Technical Report TR-0231-08-93-165, GTE Laboratories Incorporated, August 1993.
- [54] A. Martens. *Verteilte Geschaefstprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. PhD thesis, Humboldt-Universitaet zu Berlin, 2004.
- [55] B. Mehta, M. Levy, G.M.T. Andrews, B. Beckman, J. Klein, and A. Mital. BizTalk Server 2000 Business Process Orchestration. *IEEE Data Engineering Bulletin*, 24(1), 2001.
- [56] Microsoft. XML and the DataSet. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconxmldataset.asp>, 2001.
- [57] J. Miller and J. Mukerji (eds.). The MDA Guide, Version 1.0.1. OMG Document omg/2003-06-01, <http://www.omg.org/docs/omg/03-06-01.pdf>, 2003.
- [58] P. Muth, J. Weissenfels, and G. Weikum. What Workflow Technology Can Do For Electronic Commerce. In *Proceedings of the Euro-Med Net '98 Conference, Electronic Commerce Track*, March 1998.
- [59] OASIS. ebXML Collaboration-Protocol Profile and Agreement Specification, Version 2.0. <http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf>, September 2002.
- [60] OASIS. ebXML Message Service Specification, Version 2.0. <http://www.ebxml.org/specs/ebMS2.pdf>, April 2002.
- [61] OMG. OMG Meta Object Facility (MOF) Specification. OMG Document formal/2000-04-03, <http://www.omg.org/technology/documents/formal/mof.htm>, March 2000.
- [62] OMG. UML Profile for Enterprise Distributed Object Computing Specification. OMG Adopted Specification ptc/2002-02-05, <http://www.omg.org/cgi-bin/doc?ptc/2002-02-05>, February 2002.
- [63] N. Paton and O. Diaz. Active Database Systems. *ACM Computing Surveys*, 31(1):63–103, March 1999.
- [64] C. Peltz. Web services orchestration – A review of emerging technologies, tools, and standards. http://devresource.hp.com/drc/technical_white_papers/WSOrch/WS-Orchestration.pdf, January 2003.

- [65] W. Provost. UML For W3C XML Schema Design. http://www.xml.com/lpt/a/2002/08/07/wxs_uml.html, August 2002.
- [66] W. Provost. Working with a Metaschema. <http://www.xml.com/lpt/a/2002/10/02/metaschema.html>, October 2002.
- [67] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.
- [68] S. Rausch-Schott. *TriGS_{flow} – Workflow Management Based on Active Object-Oriented Database Systems and Extended Transaction Mechanisms*. PhD thesis, University of Linz, 1997.
- [69] N. Routledge, L. Bird, and A. Goodchild. UML and XML Schema. In *13th Australian Database Conference (ADC2002)*, pages 157–166. ACS, 2002.
- [70] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.
- [71] A. Schleicher and B. Westfechtel. Beyond Stereotyping: Metamodeling Approaches for the UML. In *Proc. 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, 2001.
- [72] M. Schrefl and M. Bernauer. Active XML Schemas. In *Proceedings of the Workshop on Conceptual Modeling Approaches for e-Business (eCOMO) at the International Conference on Conceptual Modeling (ER), Yokohama, Japan*, volume 2465 of LNCS. Springer, 2001.
- [73] R. Shapiro. A Comparison of XPDL, BPML, and BPEL4WS. http://www.ebpml.org/A_Comparison_of_XPDL_and_BPML_BPEL.doc, August 2002.
- [74] A.P. Shet and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), 1990.
- [75] A.P. Shet and M. Rusinkiewicz. On Transactional Workflows. *Data Engineering Bulletin*, 16(2), 1993.
- [76] A. Sladek and A. Wolski. Modeling Inter-Organizational Workflows. In *Proc. Int'l Symposium on Applied Corporate Computing (ISACC'96)*, pages 13–22, October 1996.
- [77] Sun. JSR 57 Long-Term Persistence for the JavaBeans™ Specification. <http://www.jcp.org/en/jsr/detail?id=57>, dec 2002.
- [78] S. Thatte. XLANG - Web Services for Business Process Design, Draft Specification. Microsoft, http://www.godotnet.com/team/xml_wsspecs/xlang-c/default.htm, May 2001.

- [79] uddi.org. UDDI (Universal Description, Discovery and Integration) Technical White Paper. <http://www.uddi.org>, September 2000.
- [80] UN/CEFACT. UN/CEFACT Modeling Methodology (N090 of TMWG). <http://www.unece.org/cefact/docum/download/01bp\protect\T1\textunderscoren090.zip>, 2001.
- [81] UN/CEFACT. ebXML Core Components Technical Specification, Version 1.90. <http://xml.coverpages.org/CCTSv190-2002.pdf>, December 2002.
- [82] UN/CEFACT and OASIS. ebXML Business Process Specification Schema, Version 1.01. <http://www.ebxml.org/specs/ebBPSS.pdf>, May 2001.
- [83] W. M. P. van der Aalst. Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow. *Information Systems*, 24(8):639–671, 1999.
- [84] W. M. P. van der Aalst. Don't go with the flow: Web services composition standards exposed. In *IEEE Intelligent Systems*. IEEE, 2003.
- [85] W. M. P. van der Aalst and M. Weske. The P2P Approach to Interorganizational Workflows. In *Advanced Information Systems Engineering (CAiSE 2001)*, volume 2068 of *Lecture Notes in Computer Science*. Springer, 2001.
- [86] W.M.P. van der Aalst. Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. *Information and Management*, 73(2):67–75, March 2000.
- [87] W3C. Namespaces in XML, W3C Recommendation. <http://www.w3.org/TR/REC-xml-names>, 1999.
- [88] W3C. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. <http://www.w3.org/TR/REC-rdf-syntax>, 1999.
- [89] W3C. XML Path Language (XPath), W3C Recommendation. <http://www.w3.org/TR/xpath>, November 1999.
- [90] W3C. XSL Transformations (XSLT) Version 1.0, W3C Recommendation. <http://www.w3.org/TR/xslt>, November 1999.
- [91] W3C. Web Services Description Language (WSDL) 1.1, W3C Note. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, March 2001.
- [92] W3C. XML Linking Language (XLink) Version 1.0, W3C Recommendation. <http://www.w3.org/TR/xlink/>, June 2001.
- [93] W3C. XML Schema Part 1: Structures, W3C Recommendation. <http://www.w3.org/TR/xmlschema-1>, May 2001.

- [94] W3C. XML Schema Part 2: Datatypes, W3C Recommendation. <http://www.w3.org/TR/xmlschema-2/>, May 2001.
- [95] W3C. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft. <http://www.w3.org/TR/rdf-schema>, January 2003.
- [96] P. Wegner. Interoperability. *ACM Computing Survey*, 28(1):285–287, 1996.
- [97] H. Weigand and A.H.H. Ngu. Flexible specification of interoperable transactions. *Data & Knowledge Engineering*, 25, 1998.
- [98] E. Wilde. A Compact Syntax for XML Schema. <http://www.xml.com/lpt/a/2003/08/27/xscs.html>, 2003.
- [99] J.L. Zhao. Interorganizational Workflow and E-Commerce Applications. Presentation at HICSS-35, http://attila.stevens-tech.edu/sigpam/publications/tutorial/HICSS-35/Interorganizational_Workflow.pdf, January 2002.
- [100] M. Zismann. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, 1977.

Curriculum Vitae

Address	Gerhard Kramler Feldsdorf 9 4201 Gramastetten, Austria kramler@big.tuwien.ac.at
Date of Birth	March 29th, 1973
Education	July 2001 to June 2004 Ph.D. studies in Computer Science at the Johannes Kepler University Linz and at the Vienna University of Technology October 1992 to July 2000 M.S. studies in Computer Science at the Johannes Kepler University Linz

Job Experience

Since December 2002
Faculty member of the
Institute of Software Technology and Interactive Systems
at the Vienna University of Technology

July 2001 to December 2002
Faculty member of the
Institute of Applied Computer Science
at the Johannes Kepler University Linz

July 2000 to July 2001
Conducted an industrial project
funded by Siemens Österreich
development of an embedded workflow management system

Publications

Please see:
<http://www.big.tuwien.ac.at/research/publications/>