Exploring dynamic inter-organizational business process collaboration

Exploring Dynamic Inter-Organizational Business Process Collaboration

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de Rector Magnificus,
prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het
College voor Promoties in het openbaar te verdedigen op
donderdag 22 maart 2007 om 16.00 uur

door

Alexander Horst Norta

geboren te Pretoria, Zuid-Afrika

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. P.W.P.J. Grefen
en
prof.dr.ir. W.M.P. van der Aalst

Copromotor:
dr.ir. H. Eshuis

List of Abbreviations

ATC	Abstract Transaction Construct
BNM	Business Network Model
BPEL	Business Process Execution Language
BPML	Business Process Modeling Language
BTF	Business Transaction Framework
BTML	Business Transaction Model Language
CE	Conceptual External
CI	Conceptual Internal
DIBPM	Dynamic Inter−Organizational Business Process Management
eBT	eBusiness Transaction
ECML	Electronic Contracting Markup Language
eSML	eSourcing Markup Language
eSRA	eSourcing Reference Architecture
HTTP	Hypertext Transfer Protocol
IKW	Inter− and Intra−organizational Knowledge Worker
OEM	Original Equipment Manufacturer
OWL–S	Ontology Web Language **for** Services
PNK	Petri Net Kernel
PNML	Petri Net Markup Language
SOBI	Service−Oriented Business Integration
SOA	Service−oriented architecture
SOAP	Simple Object Access Protocol
SOC	Service−oriented computing
UDDI	Universal Description Discovery and Integration
UML	Unified Modelling Language
WFMS	Worklfow Management System
WISE	Workflow Based Internet Services
WS–C	Web Service Coordination
WS–CDL	Web Services Choreography Description Language
WSCI	Web Service Choreography Interface
WSCL	Web Service Composition Language
WSDL	Web Service Definition Language
WSFL	Web Services Flow Language
WS–T	Web Service Transaction
XML	Extensible Markup Language
XPDL	XML Processing Description Language
XRL	eXchangeable Routing Language
XSLT	Extensible Stylesheet Language Transformations
XTC	eXecution of Transactional Contracted electronic services

Contents

List of Figures

List of Tables

Acknowledgements

This PhD thesis was made possible with the help of my dear colleagues from the Department of Information Systems at the Faculty of Technology Management of the TU-Eindhoven. In particular, I am deeply indebted for the support I received from the CrossWork project team at the Information Systems department, namely, Paul Grefen, Rik Eshuis, Sven Till, and Jochem Vonk. Paul and his valuable feedback were the reason why this PhD thesis could be brought to a good and successful end. Rik patiently helped me during the last stages of the PhD thesis with giving important advice on the correctness of the formal chapters. Sven and Jochem were excellent partners for jointly working on case studies and they also backed me in technical matters. It was always a joy and uplifting to collaborate with them in a team.

I want to specially thank the members of the ICTA group and also the members of the BPM group under the chairmanship of Wil van der Aalst, who were an unlimited source of inspiration. Many thanks to Wil for providing very profound feedback on the Petri-net sections. Many ideas expressed in this thesis evolved in the plenty meetings and presentations that took place at our department. Special thanks to my office colleague, Anne Rozinat, who patiently and politely listened to my monologues during the final stages of the thesis. Also many thanks to Lea Kutvonen for the collaboration invitation and for allowing me to concentrate on finishing the thesis after joining her group. Additionally, I must thank Sini Ruohomaa, Janne Metso, and Toni Ruokolainen for proofreading selected chapters of this thesis, and thanks to Bram Moonen for translating the abstract into Dutch and Jaana Moonen for hosting me and my family during the defense period.

Most importantly, I have to thank my wonderful wife, Kaisa Lotta, who supported me all along the way and who took on the sacrifice of relocating from her home in Finland to Eindhoven for the duration of my PhD project, during which we became the proud parents of Elias Onni Alexander and Saga Lotta Amelie.

Last but not least, I thank my parents and my grandfather who helped and cared for me in the years preceding my move to Eindhoven. They allowed me to establish the foundation for progressing to a PhD degree. I would like to express my deepest gratitude to them for their love and support.

Abstract

In the area of business-to-business (B2B) collaboration, original equipment manufacturers (OEM) are confronted with the problem of spending considerable time and effort on coordinating suppliers across multiple tiers of their supply chain. In tightly integrated supply chains the failure of providing services and goods on time leads to interruptions of the overall production and consequently results in customer dissatisfaction.

Collaborating parties must be able to set up an inter-organizational business process by disclosing to each other only as much as necessary. At the same time, checking the correctness of an inter-organizational business process must not force the collaborating parties into revealing their process internals to each other. However, in B2B collaborations, the ability of keeping internal business activities secret is important for retaining a competitive advantage.

The described problem is investigated in this thesis by exploring the following question: How to put into effect dynamically composed inter-organizational business processes such that the correctness of the overall process is guaranteed without imposing fixed standardized routing or compromising the autonomy of one of the organizations involved?

During the setup phase of a B2B collaboration, a service consumer is interested in keeping business-process parts opaque if they are carried out in-house. On the other hand, a collaborating party must be enabled to provide a service with the flexibility of integrating back-office activities and structural constructs for the exchange of business-critical information without violating the externally promised service behavior. Moreover, the ability of selective monitoring is necessary so that a service consumer can selectively observe process-enactment progress and a service provider is protected from having to expose more enactment progress than desired.

This thesis proposes the concept of *eSourcing* for improving the coordination of service provision across several tiers of a supply chain. eSourcing allows the external harmonization of business processes and the internal integration of heterogenous system environments without requiring collaborating parties to disclose internal business details to the counterpart.

As a means of exploring eSourcing in a conceptual and technology-independent way, this thesis performs a pattern-based analysis of eSourcing features that address the issues of visibility restriction, critical business-information exchange, and selective monitoring of enactment progress. The discovered and specified eSourcing patterns are focusing, on the one hand, on the interactions between collaborating parties during the setup phase of eSourcing and, on the other hand, on the construction elements used in an eSourcing configuration. To equip the eSourcing concept with rigor, formalisms are adapted for eSourcing that allow the verification of promised service-provision adherence without requiring the disclosure of internal business-process extensions. Additionally, a method is formalized that enables a tool-based verification of the correct termination for an eSourcing configuration.

To demonstrate the feasibility of the eSourcing concept, the discovered patterns are translated into a language that is instrumental for specifying B2B service collaboration. Moreover, a reference architecture is proposed for the development of applications that support the setup and enactment of eSourcing configurations. Within the framework of the EU research project CrossWork, case studies with industry partners from the automobile industry apply the eSourcing concept.

Samenvatting

Op het gebied van business-to-business (B2B) samenwerking zijn original equipment manufacturers (OEM) genoodzaakt veel tijd en energie te besteden aan het coördineren van de activiteiten van leveranciers verspreid over verschillende onderdelen van de leveranciersketen. In een sterk geïntegreerde leveranciersketen leidt het niet tijdig leveren van goederen en diensten tot productieonderbrekingen, resulterend in een lagere klanttevredenheid.

Samenwerkende partijen moeten in staat worden gesteld om inter-organisatorische bedrijfsprocessen op te zetten waarbij zij elkaar niet meer openheid geven dan noodzakelijk. Tegelijkertijd dient het controleren van de juistheid van inter-organisatorische bedrijfsprocessen de samenwerkende partijen niet te dwingen tot het geven van onderlinge openheid over de eigen interne processen. Nochtans ,in B2B samenwerking is het van belang dat samenwerkende partijen de mogelijkheid hebben om hun interne bedrijfsvoering geheim te houden en zodoende hun concurrentievoordeel te behouden.

Het beschreven probleem wordt in dit proefschrift onderzocht door middel van de volgende vraagstelling: op welke wijze realiseren we dynamisch samengestelde inter-organisatorische bedrijfsprocessen zodanig dat we de juistheid van het totale proces kunnen garanderen, maar zonder een vaste gestandaardiseerde routing op te leggen en zonder de autonomie van de betrokken organisaties te compromitteren?

Gedurende de opzetfase van een B2B samenwerking zal een dienstenconsument onderdelen van bedrijfsprocessen geheim willen houden zolang deze in-house worden uitgevoerd. Anderzijds moet een samenwerkende partij in staat worden gesteld om diensten te verlenen met de flexibiliteit van het integreren van back-office activiteiten en het structureren van constructies voor de uitwisseling van bedrijfskritische informatie zonder de extern beloofde dienstbaarheid geweld aan te doen. Bovendien is er een noodzaak voor de mogelijkheid tot selectieve observatie, zodat de dienstenconsument de mogelijkheid heeft om selectief de voortgang van het proces-in-uitvoering te observeren, en de dienstverlener niet meer informatie over de uitvoering hoeft prijs te geven dan gewenst.

Dit proefschrift stelt het concept *eSourcing* voor als aanpak ter verbetering van de coördinatie van dienstverlening over de verschillende onderdelen van de leveranciersketen. eSourcing biedt de mogelijkheid tot externe harmonisatie van bedrijfsprocessen en de interne integratie van heterogene systeemomgevingen zonder dat de samenwerkende partijen daardoor gedwongen worden om interne bedrijfsgegevens aan de andere deelnemende partijen te openbaren.

Om eSourcing op een conceptuele, technologieonafhankelijke wijze te onderzoeken voert dit proefschrift een patroongebaseerde analyse uit op eSourcing kenmerken die betrekking hebben op beperking van de zichtbaarheid, uitwisseling van kritische bedrijfsinformatie en selectieve observatie van het uitvoeringsproces. De geïdentificeerde en gespecificeerde eSourcing patronen richten zich aan de ene kant op interactie tussen samenwerkende partijen gedurende de opzetfase van eSourcing, en aan de andere kant op de gebruikte constructie-elementen in een eSourcing-configuratie. Om het eSourcing-concept te voorzien van hardheid zijn formalismes aangepast voor eSourcing waardoor de mogelijkheid ontstaat om de nakoming van beloofde dienstverlening te verifiëren zonder dat het noodzakelijk is om interne bedrijfsprocessen te onthullen. Bovendien is een methode geformaliseerd die de mogelijkheid biedt tot een werktuiggebaseerde verificatie voor de correcte beëindiging van een eSourcing configuratie.

Om de bruikbaarheid van het eSourcing concept te demonstreren zijn de ontdekte patronen vertaald in een taal die instrumenteel is voor het specificeren van B2B dienstensamenwerking. Daarnaast is een referentiearchitectuur voorgesteld voor de ontwikkeling van applicaties die de opzet en uitvoering van eSourcing configuraties ondersteunen. Binnen het raamwerk van het EU onderzoeksproject CrossWork, wordt in case studies met industriële partners uit de automobielindustrie het eSourcing concept toegepast.

Chapter 1

Introduction

Contents

The introduction of this thesis commences with trends in business and technologies that result in the adoption of electronic support for inter-organizational business process collaboration, followed by a presentation and explanation of the overall research goal. Next, the evolution of business-process support shows which categories of business processes exist and how technology supports them intra- and inter-organizationally. After that, the research methodology of the thesis is explained, namely design-science research, accompanied by a list of research guidelines that are obeyed. Different parts of the overall research goal are underpinned with detailed research questions. The listed research steps that adhere to the research-methodology guidelines, address the research questions. Next, the structure of the thesis shows in which chapters what research steps are carried out. The expected contributions of the thesis resulting from the research work, are formulated as deliverables. A demarcation for the thesis scope points out what is not part of the research work. The chapter is concluded by a list of publications that are related to the thesis.

1.1 Electronic Business

With market liberalizations and globalization, companies experience an acceleration and decentralization of their business activities. At the same time the lifecycles of tangible and intangible products or services are shrinking and less time is available to make newly introduced products and services financially profitable as competition is fierce and market opportunities are short-lived. Products and services are composed of several nested parts that need to be obtained from collaborating enterprizes across multiple supply-chain tiers that are geographically distributed.

The changes in market conditions are complemented by a virtualization of business collaboration that is driven by new technologies like the internet, agent technology, workflows, web-services, XML-based standards for service discovery, service orchestration, and so on. This combination of pull factors resulting from new requirements caused by market changes and the push factors of newly available technologies are rapidly accelerating the trend towards a global networking of economies. For being able to participate in such a networked economy, the availability of suitable electronic collaboration concepts and supporting web-based middleware is crucial.

Adopting a process-oriented perspective is instrumental for tackling the degree of complexity in a networked economy [49]. This thesis focusses on vertical process collaboration where a client/server relationship exists between one company that sources a service and a chosen provider of core competencies. For example, in the automobile industry a dynamically out-sourced business subprocesses for the delivery of car parts is serviced by a cluster of small- and medium sized companies that are addressed by an original manufacturer like one organizations [44]. Another example is an insurance company [57] sources the provision of a damage claim assessment.

In this thesis, for addressing a process-oriented collaboration in a networked economy, the concept of *eSourcing* is proposed. It builds on the idea of having a part of the overall business process of a service consumer performed by a service provider. To handle the inherent business, conceptual, and technological complexity, a framework is adopted for eSourcing that comprises several levels to achieve a separation of concern. To achieve an inter-organizational business-process harmonization, eSourcing focusses on matching parts of the control flow of a service provider and a service consumer. Details about eSourcing are contained in following chapters.

1.2 Overall Research Question

The research problem is summed up in the following question: *How to put into effect dynamically composed inter-organizational business processes such that the correctness of the overall process is guaranteed without imposing fixed standardized routing or compromising the autonomy of one of the organizations involved?*

In Figure 1.1 the problem tackled in this thesis is considered where non matching workflows are depicted. Two organizations that want to collaborate inter-organizationally need to agree on the coordination of their respective processes, which involves an agreement of common syntax and a common semantics of the business process they want to link. Mismatching process collaboration may be caused by false expectations and misunderstandings, e.g., a deadlock because a seller only wants to ship goods after receiving a payment and the buyer only pays after receiving the goods. Traditionally such inter-organizational conflicts are solved by people who use context information and experience. In a highly dynamic and networked environment this is costly. Thus,

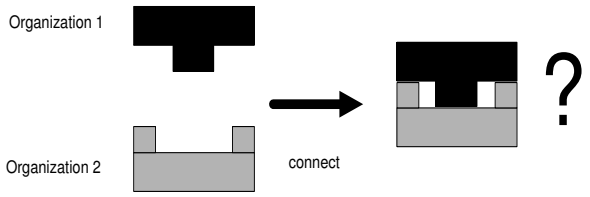

Figure 1.1: How to compose business processes inter-organizationally?

this problem of achieving matching inter-organizational business process collaboration must be addressed to successfully become electronic.

1.3 Evolution of Business-Process Support

A business process is a description of tasks with clearly defined inputs and outputs that are associated with a business activity across time and place. Business processes have a beginning and a clear end, and contained subprocess. All of them have their own goals, owners, inputs and outputs. Tasks are logical units of work and considered atomic processes that can not be further decomposed and are either carried out completely or not at all, e.g., produce an invoice, answer the phone.

There are different types of business processes [16]. *Management processes* have objectives and capital as input and must deliver business performance, e.g, profit figures. Examples of management processes are corporate governance or strategic managements. *Supporting processes* receive from the management processes the means to purchase resources, and they free up resources that are no longer required. Examples include accounting, recruitment, IT support, and so on. Finally, *operational processes* receive from the support processes resources like raw materials, personnel, components for their disposal and deliver products and services as output. From the managerial processes the operational processes receive assignments and purchasing budgets. The supporting and operational processes both report back to the managerial processes and submit their income figures.

The ability of *inter-organizationally* linking business processes is receiving increased attention in an ever more networked economy [49] as mere data exchange between companies via electronic data interchange (EDI) does not meet the demands of electronic trading relationships among businesses. EDI mainly enables bi-lateral linkages that are relationship specific and neither supports dynamic collaboration nor is it process oriented. However, there is a need to support business-process collaboration across several tiers of a supply chain that involves multiple organizations. The primary problem of inter-organizationally linking business processes is not the translation of data formats and routing. Instead, the difficulty is the bridging of the business processes embedded in an application of one collaborating party with the processes of an application located in the domain of another collaborating party [89]. The linked business processes are not defined as data but rather as tasks in workflows as the following section shows.

1.3.1 Developments in Workflow-Management Systems

Traditionally the business-process logics of a company has been hard -coded into execution applications, which is undesirable as it restricts the flexibility of an organization to adjust its information-system infrastructure to business-process changes. Instead, by introducing a workflow system [16, 69] that deals with the management of the business processes, several objectives are achieved. It allows the setup of information systems in a way that the structure of business processes is clearly reflected. If business processes change, the information system infrastructure is easily adjustable. Clearly factored out business processes may be analyzed for performance key factors such as bottle necks, throughput time, and so on. A balanced workload distribution among the resources of an organization is facilitated [16].

Companies use workflow-management systems for explicitly supporting their business processes. A workflow comprises a number of logical steps that are comparable to tasks in business processes. These logical steps are embedded in a control flow that consists of sequences, splits, and joins. A workflow-management system is a software package that is not specifically customized for supporting only one specific type of workflow. Instead, many workflows can be instantiated and enacted by one system.

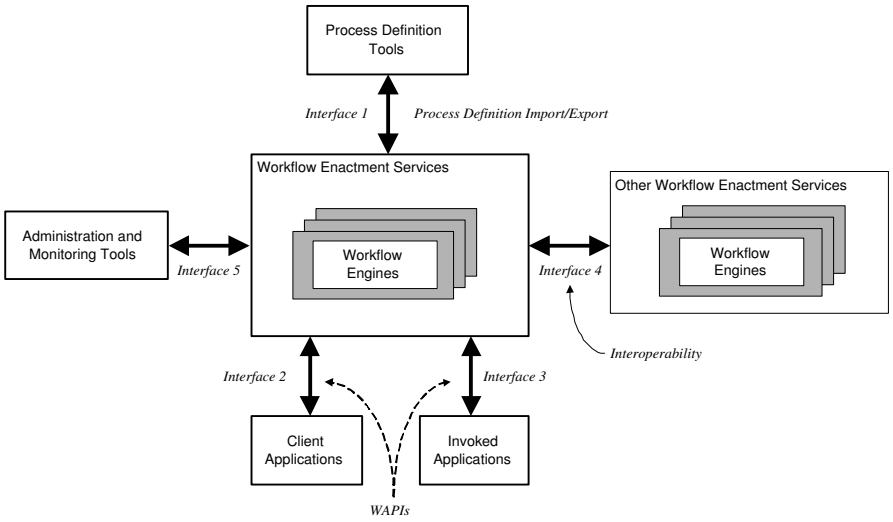

Figure 1.2: Components and interfaces of the WfMC reference model [115].

Figure 1.2 depicts a reference architecture that has been proposed by the Workflow Management Coalition (WfMC) [115]. Workflow-management systems share comparable components that are developed to different degrees. To achieve interoperability, a set of interfaces and data-interchange formats is necessary between the depicted components.

In the center of Figure 1.2 the enactment services are depicted with runtime environments that instantiate and activate workflows that are related to business processes. The enactment services interact with applications in their environment via five workflow-application programming interfaces (WAPI).

The process-definition interface allows to connects software tools with which workflows are modelled together with their related data. The resulting models are inter-

pretable by the enactment services. The client applications permit actors to interact with the enactment engine for, e.g., worklist-handling, process-instance initiation, and so on. The third interface is used for the enactment services component for invoking applications, e.g., web services, or systems for enterprize-resource planning systems, customer-relationship management.

The fourth interface in Figure 1.2 enables two or more workflow engines to communicate and interoperate so that processes can be enacted across several workflow engines. The degree of interoperability varies from simple passing of tasks to supporting a complete sharing of process definitions. Finally, the fifth interface enables workflow services to share administration and monitoring tools across heterogenous workflow products, e.g., for workflow management, auditing, resource control, process status control, and so on.

1.3.2 Developments in Service-Oriented Business Integration

The question arises how an inter-organizational integration of business processes is achievable. Emerging technologies from the domain of service-oriented computing that are part of a basic infrastructure level and a process level [52] promise to be enablers for such an objective. By using web services, it becomes possible to connect distributed business processes that belong to different companies. A *web service* is a closed, self-explaining, and modular software component that is published, found, and used on the internet. A web service provides arbitrarily complex functionality and is composed with a different application (possibly another web service) into a new system. The message exchanges between web services are usually based on an XML format [79].

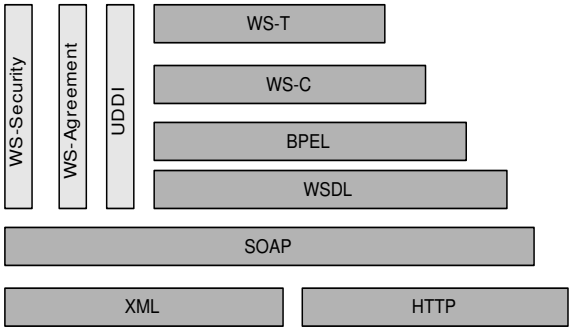

Figure 1.3: The technology stack of the service-oriented architecture [49].

Service-oriented computing (SOC) uses web services for developing loosely coupled applications where inter-system dependency is minimized. SOC relies on the technology stack of a service-oriented architecture (SOA) that is depicted in Figure 1.3. The core layers with XML, SOAP, and HTTP are well accepted of which the latter constitutes the basic communication protocol. The eXtensible Markup Language XML [86] is used for data exchange on nearly all layers of the SOA and the Simple Object Access Protocol SOAP [32] is instrumental as a messaging protocol that implements a request/response communication between connected web services.

On the higher level of the SOA in Figure 1.3, the interfaces of web services consisting of their operations and corresponding data flows are described with the Web Service Description Language WSDL [37]. To describe the execution logics of web-service based applications, an XML-based process description language is necessary. The Business Process Execution Language BPEL [40] is emerging as the dominant standard for orchestrating web services. Again on top of that, there are protocols for collaboration (WS-C) [35] and transaction management (WS-T) [34].

To the left Figure 1.3 additional services are depicted such as the Universal Description Discovery and Integration standard (UDDI) [29] to facilitate searching for web service in public directories by company name, specific service, or types of service. Further, there are standards for other aspects of collaboration, e.g., service level (agreement) management, and security.

The next section describes the research design that is used for this thesis, the research goals, and the concrete research steps that are performed for achieving the defined goals.

1.4 Research Design

In this thesis, design-science research in the domain of information systems is followed as a research methodology. The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts [54] that are broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations(implemented and prototype systems).

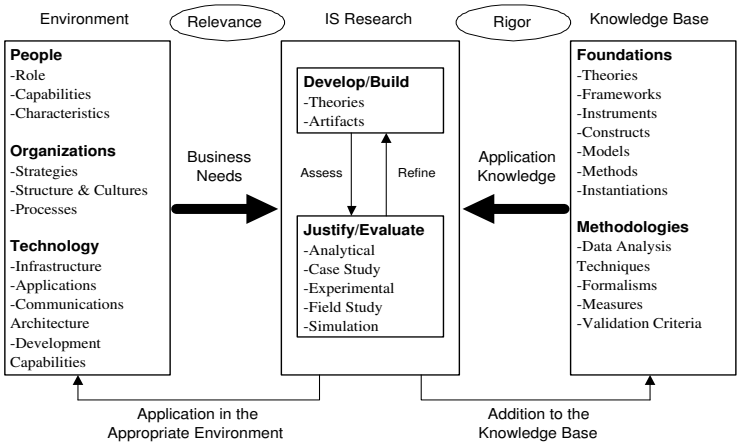

Figure 1.4: Design-science research framework for the domain of information systems [54].

In Figure 1.4, the essence of information-systems research framework is depicted. To the left, the environment pillar defines the problem space in which phenomena of interest reside consisting of people, organizations, and technology. Design-science research achieves relevance by building artifacts that address the business needs evolving from the environment.

To the right of Figure 1.4, the knowledge base delivers foundations and methodologies from and through which IS research is accomplished. Rigor is achieved by applying foundations in the develop/build phase and methodologies during the justify/evaluate phase. The results of design-science research are assessed to the business need in an appropriate environment and contribute to the content of the knowledge base for further research and practice. The next section shows how this thesis follows guidelines for conducting design-science research.

1.4.1 Research Approach

For conducting design-science research that adheres to the framework in Figure 1.4, guidelines exist [54] that are followed as a research approach in this thesis.

- *Problem Relevance*: The importance of inter-organizational electronic business for the future competitiveness of companies is explained in this introduction chapter. For developing the concept of eSourcing, the business needs of inter-organizational electronic business are relevant input.

- *Design as an Artifact*: This thesis proposes the concept of eSourcing that allows inter-organizational collaboration without directly linking the respective information infrastructure of collaborating business parties. Furthermore, eSourcing allows to safeguard business privacy and ensures flexibility for internally organizing tasks without violating the externally agreed-upon business collaboration agreements.

  The concept of eSourcing is translated into an XML-based language that allows the formulation of configurations. Additionally, a system architecture is proposed that supports the collaborative formulation of an eSourcing configuration and its enactment.

- *Research Rigor*: To understand eSourcing in further detail, a pattern-based exploration method is used that results in pattern catalogues. The advantage of using patterns is they are conceptually formulated and technology independent. A pattern meta-model establish a uniform vocabulary for specifying the pattern catalogue.

  For eSourcing exploration, workflow modelling is an integral part. Petri nets [90, 91] have been widely used to provide underlying semantics for workflow models. These formal semantics serve as an analysis technique for eSourcing that enables the development of automated verification tools.

- *Design as a Search*: The developed eSourcing concept is part of a broader lifecycle of inter-organizational electronic business collaboration. It is explored what comprises this broader notion of collaboration and how the eSourcing concept model is embedded in it.

- *Design Evaluation*: In case studies, an evaluation of the eSourcing concept is performed by using the correspondingly developed XML-based language for formulating the business collaborations as required and showing which components of an eSourcing reference architecture are important for different parts of the setup and enactment phase.

The research approach can be expressed in a concrete form through a number of specific research questions and research steps (RS) that result in answers for the research questions. The list of steps is as follows:

1.4.2 Specific Research Questions

The research questions below are aligned to the overall research question stated in Section 1.2. For every research question it is pointed out what part of the overall research question is addressed.

Q1. What existing theory provides a formal grounding for addressing the correctness of business processes that are inter-organizationally linked.
a.) What fundamental correctness criteria for business processes exist?
b.) Which formal theory provides correctness criteria for inter-organizationally linked business processes?
These questions lay the foundation for the issue of *correctness* of inter-organizational business processes that is stated in the overall research question.

Q2. What is a suitable framework for conducting dynamic inter-organizational electronic business?
a.) What are the critical issues that need to be addressed for such a framework?
b.) Which features does this framework have?
The overall research question states that inter-organizational business processes should be *composed dynamically*. Thus, such dynamic composition requires a suitable framework.

Q3. How can patterns be discovered and subsequently related to each other within the framework of eSourcing?
a.) How can a pattern concept be employed for developing a meta model that is guiding for a technology independent exploration of eSourcing?
b.) Which dimensions are specifically inherent to eSourcing and what patterns can be discovered in those perspectives?
These research questions address the demand in the overall research question that inter-organizational business process need to be *put into effect*. Thus, these questions address the issue of bridging the gap from the framework of eSourcing to its operationalization through finding a way for conducting a pattern-based exploration of eSourcing.

Q4. How can collaborating parties reach a consensus about the content of a service in an eSourcing configuration without imposing fixed standardized routing or compromising the autonomy of one of the organizations involved?
a.) What are the structural features of an eSourcing configuration?
b.) How can a service provider perform opaque service refinement without violating the service consensus?
These questions address the request of the overall research question that the inter-organizational business process must be *correct* and that *no fixed standardized routing should be imposed or the autonomy of the organizations involved be compromised*.

Q5. How should an eSourcing-formulation language and a web-based middleware be specified?

a.) How should an eSourcing-formulation language be specified that permits the verification of correct process termination?
b.) How should a web-based middleware be designed for enacting processes formulated in such a language?
c.) How should a pattern-knowledge base be designed for supporting intra- and inter-organizational knowledge workers during the setup phase of eSourcing configurations?
An eSourcing-formulation language and a web-based middleware are allow *putting into effect* an inter-organizational business process, as is requested in the overall research question.

Q6. How can the eSourcing concept of this thesis be evaluated?
While this research question does not address a specific part of the overall research question, it addresses a relevant part of the chosen design-science research method that is required to establish validity for the research conducted in this thesis.

Q7. How must the eSourcing concept be extended to fully support inter-organizational electronic business collaboration?
a.) With which mechanisms must eSourcing be extended for fully supporting inter-organizational electronic business collaboration?
b.) How can a transaction concept for eSourcing safeguard inter-organizational electronic business collaboration?
These questions address the request of the oeverall research question that inter-organizational business processes need to be composed *dynamically*.

1.4.3 Research Steps

The research steps of this thesis are presented that answer the research questions posed above. It is pointed out how the research steps relate to each other, which research questions they answer, and what design-science research guideline from Section 1.4.1 they address.

1. *Performing an exploration of existent Petri-net theory that is relevant for this thesis (RS1).*
This step answers research question Q1 by establishing a foundation of rigor for following chapters in the thesis by introducing properties of Petri-nets and a particular subclass of this formalism that is used for the workflow domain. The presented Petri-net theory is particularly important for formally defining the eSourcing concept (see RS3). This research step addresses the guideline *research rigor*.

2. *Elaboration on a general model that supports the business needs of inter-organizational electronic business collaboration (RS2).*
The model is conceptually introduced using Petry-net formalism. An exploration method is applied that determines factors related to business needs from the environment in which the eSourcing concept needs to function. This step answers research questions related to Q2. The exploration method results in a factors taxonomy that is guiding for identifying pattern specifications (see RS3). Additionally, a conceptual model is the foundation for further formal definitions (see RS4). This research step addresses the guideline *research rigor*.

3. *Identification and description of patterns for the setup and enactment phase of eSourcing (RS3).*
   A catalogue of patterns is deducted from the factors taxonomy that results in a deeper conceptual and technology independent understanding of eSourcing. The pattern meta-model used for achieving a uniform specification of the pattern catalogue is input for a component of a reference model (see RS5). This step answers research questions related to Q3. This research step addresses the guideline *research rigor*.

4. *Formally defining the properties of eSourcing using Petri-net formalism (RS4) and establishing when collaborating parties achieve a consensus about service consumption and service provision.*
   This step answers research questions related to Q4. This research step addresses the guideline *research rigor*.

5. *Identifying of the language constructs required for formulating an eSourcing configuration and of the requirements for a supporting conceptual reference architecture (RS5).*
   This step answers research questions related to Q5. This research step addresses the guideline *design as an artifact*.

6. *Evaluating the eSourcing concept, the patterns approach, the XML-based formulation language, and the corresponding reference architecture (RS6). It contains the following sub-steps:*
   A. applying the eSourcing concept for analysing business cases;
   B. applying the pattern catalogue for the design and analysis of a business case;
   C. specifying business cases using the XML-based formulation language;
   D. evaluating the application of architectural design principles in the design of the reference architecture.
   This last research step answerts research question Q6 and is relevant for satisfying the design-science guideline called *design evaluation*.

7. *Identification of a broader notion of electronic business collaboration that includes the ingredients for safeguarding such collaboration (RS7).*
   This step answers research questions related to Q7. This research step addresses the guideline *design as a search*.

Next, the structure of the thesis is presented that closely follows the list of research steps.

1.5 Structure of the Thesis

In Chapter 2 Petri-nets related theory is introduced (related to RS1). In particular a subclass of Petri-nets is covered that is used extensively in this thesis, namely workflow nets (WF-nets) that carry the property of Soundness. This property is relevant for verifying the correct termination of eSourcing configurations. The Petri-nets formalism is proposed for checking the adherence to agreed upon service provision.

In Chapter 3 the concept of eSourcing is presented (related to RS2) that is based on a three-level model to bridge the gap between the domains of collaborating parties. For the eSourcing concept the control-flow perspective is informally investigated and

the environmental business needs are explored that eSourcing must satisfy (commencement of RS3).

In Chapter 4 a pattern-based analysis is performed based on a pattern meta-model that is used for uniformly defining and relating patterns to each other. The pattern-based analysis focuses on the setup-phase of eSourcing and on construction patterns that are the building blocks of eSourcing (related to RS3).

In Chapter 5 the eSourcing concept that evolves in previous chapters is first formally defined. Secondly, it is explored which conditions need to be satisfied for a service consumer and a service provider for reaching a consensus about the structure of a service. A method is formally grounded for determining the correct termination of an eSourcing configuration and the correctness of service provision without requiring the disclosure of internal business secrets (related to RS4). Finally, a component is presented that supports the verification method (beginning of RS5).

Chapter 6 contains as a proof-of-concept an XML-based eSourcing formulation language that incorporates earlier specified construction patterns. Furthermore, a conceptual reference architecture is proposed that consists of interacting components distributed across the three-level framework. Finally, two components of the reference architecture are further explored, namely a pattern knowledge base that supports parties involved in a business collaboration, and an application that evaluates the business processes contained in an eSourcing configuration (related to RS5).

Chapter 7 presents case studies where the eSourcing concept of this thesis is applied through the pattern catalogue specified in this thesis. These patterns are used during the setup phase and consequently an analysis phase of the presented business collaborations. The first case study shows eSourcing in a multilateral situation with one service provider and several service providers and points out how the pattern catalogues are used for the design and analysis phase, and how the components of the reference architecture support the setup and enactment phase of an eSourcing configuration. The second case study is a bilateral collaboration that focusses on applying the range of modelling constructs comprising the XML-based eSourcing formulation language (related to RS6).

In Chapter 8 the eSourcing concept of this thesis is embedded in a notion of dynamic electronic business collaboration. As part of this business-collaboration notion, the ingredients of a suitable e-business transaction concept are outlined (related to RS7). In Chapter 9, a conclusion for the thesis is presented.

1.6 Contributions and Demarcations

In this section, a summary of the main contributions of the thesis is presented in the form of deliverables. Different research domains are dealing in related ways with research on inter-organizational e-business collaboration. Thus, the research work of this thesis is demarcated from other related research domains.

1.6.1 Contributions of the Thesis

As a first deliverable, a collaboration framework needs to be developed that allows different organizations to provide and consume services. This framework must take into account the business needs for dynamically composed inter-organizational business process collaboration. Collaborating parties should not be forced into directly connecting their respective information infrastructure and must be able to keep their competi-

tive advantages hidden from each other. Thus, only those business details should need to be disclosed that are necessary for establishing and carrying out a collaboration.

As a second deliverable, for offering a deep understanding on business-to-business collaboration and e-business processes, pattern catalogues for the setup and enactment phases of eSourcing configurations must be generated. These pattern catalogues need to be discovered and specified based on the environmental business needs of inter-organizational business collaboration. To achieve a uniform specification of the pattern catalogues, a pattern meta-model must be provided that shows pattern entities with relationships and attributes. By offering catalogues of conceptually formulated and technology independent patterns for the establishment of eSourcing configurations, it must become unnecessary to consistently "reinvent the wheel". Instead, based on the identification of specific situational problems, the pattern specifications should refer to constructs of knowledge that can be used for providing quick solutions.

As a third deliverable, the concept of eSourcing must be well grounded on Petri-net formalism, for which a considerable body of theory exists for the domain of business processes. By employing specific Petri-net theory, the semantics of the control-flow perspective for eSourcing configurations has to be unambiguously defined. That way collaborating parties have to be equipped with a common understanding about the control-flow constructs in eSourcing that they intend to employ. Additionally, by adopting existing Petri-nets theory for eSourcing, it has to become possible to inter-organizationally link business processes while ensuring that collaborating parties retain a business-relevant degree of internal adjustment flexibility that remains opaque to the opposing collaborating party. That way organizations must be enabled to keep their business secrets without having fixed, standardized routing imposed on themselves. By adopting clear control-flow semantics for eSourcing, it must be possible to check a configuration at the end of the setup phase for problems that make a successful enactment impossible. That way the need for expensive exception handling and compensation steps must be reduced.

As a fourth deliverable, the work presented in the thesis must facilitate the development of an inter-organizational e-business collaboration application by providing the basic functionalities and interdependencies of an eSourcing application infrastructure. A reference architecture for eSourcing applications has to comprise of components that are instrumental for the setup and enactment of eSourcing configurations. The architecture must be based on design principles that pay attention to a separation of business, conceptional, and technological concerns. As a result, collaborating parties must only have to expose as many business internals to each other as is necessary for establishing a collaboration consensus. It has to be stressed that technological details must equally remain opaque as it should not be required in an eSourcing configuration to directly link the respective legacy systems of collaborating parties.

As a sixth deliverable, for negotiating collaboration consensus with an equal set of mutually understood concepts, the thesis has to offer an XML-based formulation language that comprises constructs based on the specified eSourcing pattern catalogues. The formulation language must be the result of the setup phase of an eSourcing configuration that is enactable with software applications.

As a seventh deliverable, two components of the eSourcing reference architecture must be implemented as proof-of concept prototypes. The first prototype needs to be an enactment evaluation application for business processes that maps the control-flow constructs of the XML-based eSourcing formulation language to Petri-net constructs. With this mapping it becomes unnecessary to change the enactment engine of the evaluation application when new control-flow constructs are added to the the eSourcing

formulation language. Instead, the evaluation prototype must be adjustable by only extending the library of mapping rules. The second proof-of-concept prototype must be a pattern knowledge base that is built on top of the pattern meta-model. This knowledge base has to be instrumental for storing pattern catalogues that can grow dynamically. In this case dynamically means that users of the knowledge base must be able to propose new patterns that need to be accepted in a review process before they enter the knowledge base. With the help of this knowledge base a tool support must be provided for an accelerated setup of eSourcing configurations.

1.6.2 Demarcations

This thesis focusses on the correctness of the control-flow perspective in eSourcing. However, there are other perspectives that are also important in this context, e.g., data-flow perspective [95], resource perspective [96], transactional perspective [111]. Still, these perspectives are catered for and incorporated into the XML-based eSourcing formulation language.

For the establishment of an eSourcing configuration the employment of ontologies [36] is relevant for ensuring that opposing parties share an equal understanding about the properties of their business collaboration. Ontologies are important during the setup phase of an inter-organizational business collaboration for mutually meaningfully decomposing the goals of collaboration and performing an appropriate team formation. Furthermore, ontologies play an important role in supporting the comprehension of exchanged data between the domains of collaborating parties. Ontological issues are not the focus of this thesis as the the adoption of Petri-net formalism establishes clear control-flow semantics for eSourcing. However, it is possible to adopt ontologies for other eSourcing-related perspectives.

Business collaborations may stretch over long periods of time during which changes may be required. When changes are necessary, they should take place dynamically, i.e., the changes should take effect without having to stop an ongoing business collaboration [81]. This thesis does not investigate how such change updates can be supported without violating the correctness of an ongoing business collaboration.

It is desirable to perform an automated formation of workflows that are consequently inter-organizationally linked. For such automated formation [44] special algorithms are required that use the data-flows between collaborating business parties as a starting point for assembling control-flow patterns into workflows. However, such automated workflow formation is not explored in this thesis.

Finally, to pursue total automation in eSourcing, the adoption of agent technology is feasible. Such agents [118] establish a collaboration and negotiate business process details. During the enactment phase the inter-organizational business process coordinates agents that belong to the domains of the opposing parties. Thus, by employing agents it is possible to eliminate human involvement in eSourcing. In this thesis the pursuit of such total eSourcing automation is out of focus. In the conclusion of Chapter 9, it is discussed how the aspects listed above can be integrated in the eSourcing approach.

1.7 Research History and Publications

The contributions of this thesis that are presented in Section 1.6.1 are part of the Chapters 2-7. Many of these contributions are based on publications (conference proceed-

ings, technical reports, deliverables of the EU research project called CrossWork [2]). The work presented in Chapter 4 is to appear in a journal publication. To demonstrate the research history of this thesis, the list follows a chronology from the oldest to the latest publication. After every publication an explanation follows that points out the contribution of the thesis author. Note that the thesis author was known as Hirnschall before his marriage.

- W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In A. Banks-Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsu, editors, Proceedings of the *14^{th} International Conference on Advanced Information Systems Engineering (CAiSE'02)*, Volume 2348 of Lecture Notes in Computer Science, pages 535-552. Springer-Verlag, Berlin, 2002.

  This publication shows that the set of reduction rules presented in [97] is not complete. The thesis author participated in proposing an alternative complete algorithm that translates workflow graphs into so-called WF-nets (see Section 2.3), for which the verification tool Woflan [108] can be used. Although these contributions are not used in this thesis, much the introduced Petri-net theory is contained in Chapter 2 where the preliminaries of this thesis are presented.

- H.M.W. Verbeek, A. Hirnschall, and W.M.P. van der Aalst. XRL/Flower: Supporting Interorganizational Workflows using XRL/Petri-net Technology. In C. Bussler, R. Hull, S. McIlraith, M. Orlowska, B. Pernici, and J. Yang, editors, *Workshop on Web Services, E-Business, and the Semantic Web (CAISE'02)*, pages 93 ff. Springer-Verlag, Berlin, 2002.

  The thesis author contributed in presenting the architecture of XRL/Flower (see Section 6.5.4). XRL/Flower benefits from being both XML and Petri nets based. Standard XML tools can be deployed to parse, check, and handle XRL (eXchangeable Routing Language) documents. XRL constructs are automatically translated into Petri-net constructs. As a result, the system is easy to extend as only the translation to the Petri-net engine needs to be added for supporting a new routing primitive and the engine itself does not need to change. The Petri net representation can be analyzed with tool support.

- A. Norta. Web supported enactment of petri-net based workflows with XRL/flower. In J. Cortadella and W. Reisig, editors, Proceedings of the *25^{th} International Conference on the Application and Theory of Petri Nets 2004*, number 3099 in Lecture Notes in Computer Science, pages 494.503, Bologna, Italy, 2004. Springer Verlag, Berlin.

  While the previous workshop paper presents the architecture of XRL/flower, this publication describes the implementation of the web-based application that carries out Petri-net based workflows described with XRL. The thesis author decided that XRL/flower uses XML technology and is implemented in Java on top of the Petri-net Kernel PNK. Standard XML tools can be deployed to parse, check, and handle XRL documents. The XRL enactment application is complemented with a web server, allowing actors to interact with the system through the internet. A database allows the enactment engine and the Web server to exchange information with each other. Since XRL is instance based, a modelled workflow serves as a template that needs to be copied and may be possibly refined for enactment. In this thesis XRL/flower is used as a process validation tool.

- A. Norta. Sourcing framework, in: CrossWork: Workflow Model Elements, Work Package 2: Workflow Modelling, Task 2.1: Agent-Based Workflow Configuration, 2005. pages 14-20;

  For the EU research project CrossWork [2] the thesis author participated in the writing of deliverable documents. Much of the content of these deliverable documents is also adopted and contained in this monograph. For the CrossWork project the thesis author developed eSourcing [85] as an inter-organizational business-process collaboration concept. eSourcing is matching on an external level conceptually formulated service consuming and providing processes belonging to the domains of autonomous organizations for the formation of an inter-organizationally linked business process. For eSourcing, dynamic inter-organizational business process management (DIBPM) [50] is combined with technologies for service-oriented business integration.

- A. Norta. eSML (eSourcing Markup Language), in: R. Eshuis and I. Stalker, CrossWork: Requirements and concepts for agent-based workflow configuration, Work Package 2: Workflow Modelling, Task 2.2: Agent-Based Workflow Configuration, 2005. pp 149;

  The features of eSourcing are contained in the inter-organizational business process collaboration language eSML that uses ECML (Electronic Contracting Markup Language) [23] as a foundation and integrates XRL for specifying business process. eSML is applied in CrossWork to specify case studies for the automobile industry. Furthermore, eSML is employed for enabling the communication between components that are part of the CrossWork proof-of-concept prototype [53].

- A. Norta and P. Grefen. A Framework for Specifying Sourcing Collaborations. In Jan Ljungberg and Bo Dahlbom, editors, *14^{th} European Conference on Information Systems*: Grand Challenges, pages CD.ROM, Gothenburg, Sweden, 2006. IT-University Gothenburg.

  The thesis author consolidated the earlier published eSourcing concept for this conference contribution. The questions are answered how eSourcing manages the business, conceptual, and technological complexity of inter-organizational business process harmonization; which pre-existing formal theory is available for giving eSourcing rigor, and how the features of eSourcing can be analyzed for ensuring that the concept has relevance for industry applications.

- A. Norta and P. Grefen. Discovering Patterns for Inter-Organizational Business Collaboration in a Top-Down Way. BETA Working Paper Series, WP 163, Eindhoven University of Technology, Eindhoven, 2006.

  In this paper, the discovered features of the eSourcing concept are analyzed in a pattern-based way. The thesis author pursues an analysis and specification of eSourcing construction patterns in a top-down way. These construction patterns are the building blocks for eSourcing collaboration configurations and are contained in eSML as schema elements.

- A. Norta and P. Grefen. Developing a Reference Architecture for Inter-Organizational Business Collaboration Setup Systems. BETA Working Paper Series, WP 170, Eindhoven University of Technology, Eindhoven, 2006.

The thesis author contributed with an investigation of the characteristics of interaction between a service consumer and provider during setup time for establishing an enactable B2B collaboration. These characteristics are employed to discover interaction patterns in a dop-down way that are exemplified for the concept of eSourcing. As such, the discovered patterns form the basis for the design of a reference architecture that serves as a foundation for the design of e-collaboration setup systems.

- A. Norta and P. Grefen. A Pattern Repository for Establishing Inter-Organizational Business Processes. BETA Working Paper Series, WP 175, Eindhoven University of Technology, Eindhoven, 2006.

  In this conference publication the thesis author elaborated on a reference architecture for a pattern repository to support the effective and efficient design of inter-organizational business processes. By storing patterns in a uniform specification template of a meta model together with the technology support of individual patterns, inter- and intra-organizational knowledge workers (IKW) can quickly analyse with which intersection of pattern sets it is possible to construct intra-organizational business processes.

- A. Norta, M. Hendrix, and P. Grefen. A Pattern-Knowledge Base Supported Establishment of Inter-Organizational Business Processes. In R. Meersman, Z. Tari, and P. Herrero, editors, Proceedings of *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, Volume 4277 of Lecture Notes in Computer Science, pages 834-843. Springer-Verlag, Berlin, 2006.

  This workshop paper is a condensed version of the previous technical report about the pattern repository.

- A. Norta and P. Grefen. Discovering Patterns for Inter-Organizational Business Collaboration in a Top-Down Way. To appear in the International Journal of Cooperative Information Systems (IJCIS).

Chapter 2

Petri-net Theory

Contents

For the domain of inter-organizational business process management, collaborating parties need to have semantic clarity about the processes that have to be inter-organizationally harmonized. Additionally, semantic clarity allows the application of verification tools so that errors in an inter-organizational business process can be detected before enactment. The Petri-net formalism offers a clear semantics and theoretical research results for inter-organizational business process management. This preliminary chapter introduces existing Petri-net theory that is used in later chapters.

2.1 Introduction

For the domain of eSourcing, the control-flow perspective is the dominant perspective that has been best explored. The control-flow perspective addresses the way how tasks of a process are related to each other and orders tasks in sequences and different types of splits and joins [17]. A process may contain control-flow problems, e.g., deadlocks or livelocks, that prevent correct termination. For eSourcing, such control-flow problems are more acute as processes that terminate correctly by themselves may fail to do so when linked together [12]. Based on a correct control-flow in an inter-organizational business process, the other important perspectives like data-flow, organization, and resource usage can be constructed.

To ensure that the control-flow perspective is modelled [16] and analyzed correctly, a graph-based formalism like Petri nets [90, 91] is suitable. Here a subclass of Petri nets

is used for the domain of business-process management, namely so called *workflow nets* (WF-net) [9] that have been further explored [10, 42]. For WF-nets, the notion of *soundness* [8] exists, a property that ensures the correct termination of local workflows and which can be verified with tool support. For inter-organizational business process collaboration, WF-nets are extended into inter-organizational workflow nets (IOWF-nets) [12] for which the soundness notion is also applicable after using a flattening method for turning an IOWF-net into a WF-net.

Besides ensuring correct termination, for inter-organizationally linked business processes it is also desirable to check how and if linked processes relate to each other. If it is assumed that linked processes belong to domains that relate in a client-server way to each other, then one serving process must must still display the agreed upon service-provision behavior despite inserted, opaque refinements. To use extensions, a special notion of inheritance is used to establish the inter-organizational business process relationship. In this chapter *projection inheritance* [14, 27] is employed for that purpose. Informally, projection inheritance allows the incorporation of a refining task in a superprocess without violating the displayed runtime behavior.

Note that this chapter comprises pre-existing research results from the Petri-net domain that are instrumental for following chapters. Each formal definition comes from [12] unless stated otherwise. The structure of this chapter is as follows. First Petri-nets are introduced in Section 2.2 followed by a presentation of WF-nets in Section 2.3, which are a subclass of Petri-nets. Additionally, the soundness property of WF-nets is explained that is relevant for verifying the correct termination of eSourcing configurations. Section 2.4 introduces IOWF-nets together with a flattening method that consequently allows the verification of soundness. Section 2.5 gives an introduction to the notion of projection inheritance that is relevant for checking the adherence to agreed upon service provision. Section 2.6 gives a brief introduction to a verification tool for checking the soundness of WF-nets and projection inheritance. Finally, Section 2.7 concludes this chapter.

2.2 Petri nets

The classical Petri net [90, 91] is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

In this chapter a variant of the classic Petri-net is defined, namely a labelled Place/Transition net. Let U be a universe of identifiers; let AL be some set of *action labels* with $\tau \in AL$ the silent action, whose role will be explained later. Let $AL_v = AL \setminus \{\tau\}$ be the set of visible labels.

Definition 1 (Labelled P/T-net). *[12] A labelled Place/Transition net, or simply P/T-net, is a tuple (P, T, L, F, ℓ) where*

1. *$P \subseteq U$ is a finite set of places;*
2. *$T \subseteq U$ is a finite set of transitions such that $P \cap T = \emptyset$;*
3. *$L \subseteq AL_v$ is a finite set of labels such that $L \cap (P \cup T) = \emptyset$;*
4. *$F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation;*

5. ℓ:$T \rightarrow L \cup \{\tau\}$ *is a labelling function.*

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. Likewise, a transition t is called an *input transition* of a place p iff there exists a directed arc from t to p. Place t is called an *output transition* of place p iff there exists a directed arc from p to t. All input nodes of a particular node constitute the *preset* and all output nodes of a particular node are called *postset*.

For the pre- and postsets an additional notation is relevant. Two auxiliary functions $\bullet\_,\_\bullet$: $(P \cup T) \rightarrow \mathcal{P}(P \cup T)$ are defined that assign to each node its preset and postset, respectively. For any node $x \in P \cup T$, $\bullet x = \{y \mid yFx\}$. To avoid confusion about which net a node belongs to, the preset and postset notation is augmented with the name of the net: Given a net N, ${}^{N}\bullet x$ is the preset of node x in N and $\bullet^{N} x$ is the postset of node x in N.

Definition 2 (Marked, labelled P/T-net). *A marked, labelled P/T-net is a pair (N, s), where N = (P, T, L, F, ℓ) is a labelled P/T-net and where s is a bag over P denoting the marking (also called state) of the net. The set of all marked, labelled P/T-nets is denoted $\mathcal{N}$.*

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., a function $s \in P \rightarrow \mathbb{N}$. A state is represented as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. This state can also be represented as follows: $p_1 + 2p_2 + p_3$. A Petri net PN and its initial marking s: $P \rightarrow \mathbb{N}$ where for each $p \in P$ there are $n \in \mathbb{N}$ tokens, are denoted by (PN, s). If confusion is possible, brackets are used to denote markings, e.g., [p_1+2p_2+p_3]. This is particularly useful for markings having only one token, e.g., [p] is the marking with just a token in place p.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

In Figure 2.1 an example of a Petri net is depicted. The circles are places, the boxes are transitions, and the black dots in the places are tokens. Marking *s'* is reachable from *s* if there is a sequence of transitions such that, starting in *s*, firing the transitions results in state *s'*. For a formal definition see [12].

The transition with the label a is enabled as all its input places contain at least one token. When the a-labelled transition fires, it consumes one token from all its input places and produces as a result the Petri net of Figure 2.1 is in the new state $1p_1$+$0p_2$+$0p_3$+$1p_4$. Next, the b-labelled transition is enabled. After the latter transition fires, it produces one token for its only output place p_3 that does not result in an enabled a-labelled transition. The reason is that place p_2 does not contain a token any more.
The following definitions represent a non-exhaustive selection of Petri-net properties that are sufficient for later sections.

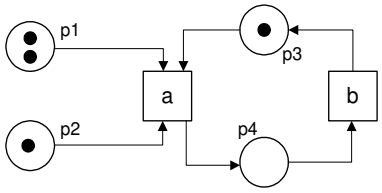

Figure 2.1: An example of a Petri net.

Definition 3 (Connectedness). *A labelled P/T-net in $\mathcal{N}$, where N=(P, T, L, F, ℓ), is weakly connected, or simply connected, iff, for every two nodes x and y in P $\cup$ T, x $(F \cup F^{-1})^*$ y. Net N is strongly connected iff, for every two nodes x and y in P $\cup$ T, x F^* y.*

The net at the top of Figure 2.2 is connected but not strongly connected because there is no directed path from *o* place to the *i* place. The bottom net of Figure 2.2 is strongly connected because of the the additional transition labelled $\bar{t}$ that connects the places with the *i* and *o* label.

Definition 4 (Live). *A marked, labelled P/T-net (N, s) is live iff, for every reachable state s' and every transition t there is a state s'' reachable from s' which enables t.*

The Petri net in Figure 2.1 is not life as the explanation to the figure shows that a state is reachable where the *a*-labelled transition is not enabled to fire.

Definition 5 (Bounded, safe). *A marked, labelled P/T-net (N, s) is bounded iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less than n. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

The example depicted in Figure 2.1 is bounded. However, due to the mentioned initial state, the net is not safe since p_1 contains 2 tokens.

For labelled P/T-nets it is important to understand when they are identical. Mathematically, two nets are identical iff all their objects are pairwise identical. For P/T-nets the notion of an isomorphism is instrumental to express equality. The following definition is based on [94].

Definition 6 (Isomorphism). *Two nets N_0 = (P_0, T_0, L_0, F_0, ℓ_0) and N_1 = (P_1, T_1, L_1, F_1, ℓ_1) are isomorphic, denoted by $N_0 \equiv N_1$ if there exist two bijections α: $P_0 \rightarrow P_1$ and β: $T_0 \rightarrow T_1$ such that for every $p \in P_0$ and $t \in T_0$,*

1. *(p,t) $\in F_0$ iff (α(p),β(t)) $\in F_1$,*
2. *(t,p) $\in F_0$ iff (β(t),α(p)) $\in F_1$,*
3. *ℓ(t)=ℓ(β(t)),*
4. *$L_0 = L_1$.*

Finally, it is possible that so-called dead transitions are contained in a P/T-net. A definition for dead transitions is given below.

Definition 7 (Dead transition). *Let (N,s) be a marked, labelled P/T-net in $\mathcal{N}$. A transition t ∈ T is dead in (N,s) if and only if there is no marking s' reachable from s, such that s enables t.*

Petri-net formalisms are suitable for specifying the control-flow of tasks in a process. However, for the domain of business processes, simple Petri-nets are not sufficient. For the domain of business-process management, a subclass of Petri-nets has been developed, namely workflow net (WF-net) [9]. The following section defines WF-nets and their properties.

2.3 Workflow nets

Workflow is the operational aspect of a work procedure: how tasks are structured, who performs them, what their relative order is, how they are synchronized, how information flows to support the tasks and how tasks are being tracked. A *WorkFlow net* (WF-net) [9, 10, 42] models the control-flow dimension of a workflow. It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation. This means that every piece of work is executed for a specific case that is also called a workflow instance. Examples of cases are handling an insurance case, an order, a tax declaration, and so on.

Definition 8 (WF-net). *[12] Let N = (P, T, L, F, ℓ) be a labelled P/T-net in $\mathcal{N}$. N is a WF-net iff the following conditions are satisfied:*

1. *Instance creation: P contains an input (source) place i ∈ P such that $\bullet i = \emptyset$;*
2. *Instance completion: P contains an output (sink) place o ∈ P such that $o\bullet = \emptyset$;*
3. *Strongly connected: $\bar{N}$ = (P, T∪{$\bar{t}$}, L, F ∪{$(o,\bar{t}), (\bar{t},i)$}, $\ell \cup \{(\bar{t},\tau)\}$) is strongly connected ($\bar{t} \notin T$);*
4. *Label use: L = rng(ℓ)\{τ};*
5. *Visible start: for any t ∈ T such that t ∈ i•: $\ell(t) \in AL_v$, i.e. $\ell(t) \neq \tau$;*
6. *Visible end: for any t ∈ T such that t ∈ •o: $\ell(t) \in AL_v$, i.e. $\ell(t) \neq \tau$.*

A WF-net has one input place (i) and one output place (o) because any case handled by the procedure represented by the WF-net is created when a token enters the place *i* and ends when a token enters place *o*, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 8 has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases. Note that transitions model tasks and places model conditions. The structure of a WF-net allows to define the following functions.

Definition 9 (source, sink). *Let N = (P, T, L, F, ℓ) be a WF-net.*

1. *$source(N)$ is the unique input place i ∈ P such that $\bullet i = \emptyset$;*
2. *$sink(N)$ is the unique output place o ∈ P such that $o\bullet = \emptyset$.*

The top P/T-net of Figure 2.2 fits the requirements of a WF-net as stated in Definition 8. To the left the input place (i) and to the right the output place (o) are depicted. Furthermore, the third condition is satisfied as all other nodes contribute to the processing of the WF-net.

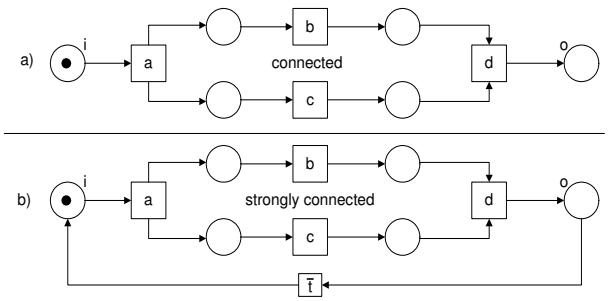

Figure 2.2: An example of a WF-net (a) and its extended shortcircuited net (b).

2.3.1 The Soundness of WF-nets

The three requirements stated in Definition 8 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement which should be satisfied:

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Looking at the WF-net of Figure 2.2, this requirement is fulfilled. The requirement mentioned at the end of the previous subsection corresponds to the so-called *soundness property* [8].

Definition 10 (Soundness). *A WF-net N = (P, T, L, F, ℓ) with source(N) = i and sink(N) = o is weakly sound iff the following conditions are satisfied:*

(i) safeness: (N, [i]) is safe;

(ii) proper completion: for any marking s reachable from [i], $o \in s$ implies s = [o];

(iii) completion option: for any marking s reachable from [i], [o] is reachable from s.

N is said to be strongly sound, or simply sound, if and only if, in addition there are no dead transitions, i.e., (N, [i]) contains no dead transitions.

The set of all strongly sound WF-nets is denoted as $\mathcal{W}$. The first condition of Definition 10 states that a sound WF-net is safe. The second condition focuses on the completion of a WF-net. If a marking in o is reached, all places are empty with the exception of place o that must contain one token. Finally, the third condition states that from the initial marking i that activates a case, it is always possible to reach the marking with one token in place o that results in a successful termination. The fourth condition about dead transitions that defines strong soundness, states that for each transition there is an execution sequence activating this transition. To show the relation between strong and weak soundness, we use a β operator [12] that removes all dead transitions and corresponding places from the net.

Definition 11 (Removing dead transitions: β). *Let (N, s) be a marked, labelled P/T-net in $\mathcal{N}$, with N = (P, T, L, F, ℓ) and a set of dead transitions D $\subseteq$ T such that T $\setminus$ D does not contain dead transitions. β is a function such that it maps marked P/T-nets onto P/T nets: β(N, s) = (P', T', L', F', ℓ') with T' = T$\setminus$D, P' = {p $\in$ P | ($\bullet$p $\cup$ p$\bullet$) $\not\subseteq$ D}, F' = F $\cap$ ((P' $\times$ T') $\cup$ (T' $\times$ P')), dom(ℓ') = T', for t $\in$ T':ℓ'(t) = ℓ(t), and L' = rng(ℓ')$\setminus${τ}. If N is a WF-net with source place i, then β can also be applied without explicitly stating the initial marking, i.e., β(N) = β(N, [i]).*

The theorem below states that a weakly sound WF-net can be transformed into a strongly sound WF-net by using the β operator for removing dead transitions.

Theorem 1. *[12] Let N be a weakly sound WF-net. β(N) is strongly sound.*

The results show that we can focus on strongly sound WF-nets in the sequel and therefore the term "soundness" is used for referring to strongly sound WF-nets, unless stated otherwise. Note that the soundness property relates to the dynamics of a WF-net. Given a WF-net N = (P, T, L, F, ℓ), one wants to decide whether N is sound. In [9] it is shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, an extended net $\overline{N} = (\overline{P}, \overline{T}, \overline{L}, \overline{F}, \overline{\ell})$ is defined that is the P/T-net obtained by adding an extra transition $\bar{t}$ (see Definition 8) which connects o and i. Such an extended net is called the *short-circuited* net of N that allows for the formulation of the following theorem.

Theorem 2. *[12] A WF-net N is sound iff $(\overline{N}, [i])$ is live and safe.*

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness. For the verification of complex WF-nets it is desirable to have tool support (see Section 2.6).

For the area of intra-organizational business processes WF-nets are of significant value. With increasing size of a process models it becomes hard to detect control-flow problems as deadlocks manually. When process models are enacted by workflow systems, contained mistakes result in loss of time and money. By using WF-net formalisms, it is possible to automate during design time the verification of soundness. Next, the theory of WF-nets is extended to an inter-organizational level.

2.4 Inter-Organizational Workflow nets

The previous section lays a theoretical foundation for a internal business processes in isolation. For inter-organizational business process collaboration this theory needs to be extended. Therefore, this section begins with the definition of IOWF-nets [12] that connects several WF-nets through channels.

Definition 12 (IOWF-net). *[12] An inter-organizational workflow net (IOWF-net) is a tuple (C, n, N_0, N_1,..., N_{n-1}, L, G) where:*

1. *C $\subseteq$ U is a finite set of channels,*
2. *N_0, N_1,..., N_{n-1} are n WF-nets such that*
   (a) ($\forall$ k: 0 $\leq$ k < n: N_k = (P_k, T_k, L_k, F_k, ℓ_k),
   (b) ($\forall$ k, l: 0 $\leq$ k < l < n: ($P_k \cup T_k \cup L_k$) $\cap$ ($P_l \cup T_l \cup L_l$) = $\emptyset$), and
   (c) ($\forall$ k: 0 $\leq$ k < n: ($P_k \cup T_k \cup L_k$) $\cap$ C = $\emptyset$)
3. *L = ($\cup$ k: 0 $\leq$ k < n: L_k) is the union of labels, and*

4. *$G \subseteq (C \times L) \cup (L \times C)$ is a set of directed arcs, called the channel flow relation.*

An IOWF-net comprises a set WF-nets interconnected by a set of channels C, a set of labels L, and a channel flow relation G. An example of an IOWF-net is given in Figure 2.3 where two WF-nets are connected inter-organizationally. The respective WF-nets are enclosed in boxes and their connections are realized with channels between them. The channels exist of interface channels that are depicted in Figure 2.3 with grey shaded places. In this example, each channel place is connected to two labels belonging to each respective connected WF-net. The labels are part of the transitions and each channel has a direction from one WF-net to the other one.

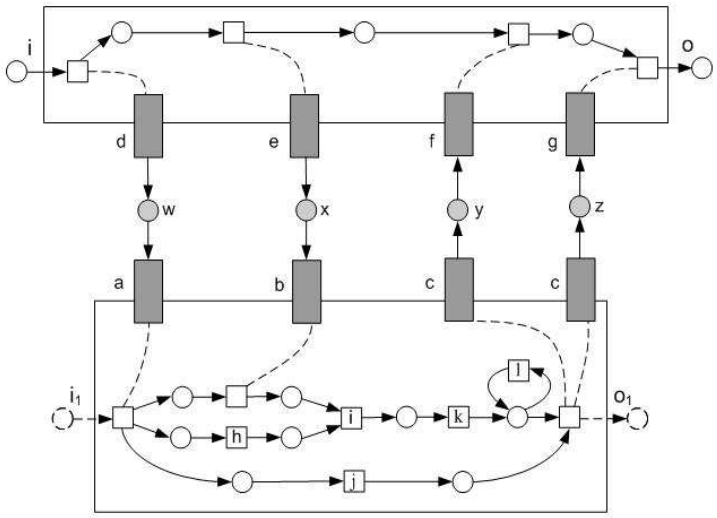

Figure 2.3: An IOWF-net.

For IOWF-nets a flattening method is available to create a labelled P/T-net. The flattening method takes the union of all WF-nets, adds a place for each channel, connects transitions to the added places as specified by G, and removes unnecessary source and sink places. The following definition describes the *flat* function [12] for transforming IOWF-nets into labelled P/T-nets.

Definition 13 (*flat*(Q)). *Let $Q = (C, n, N_0, N_1, ..., N_{n-1}, L, G)$ be an IOWF-net. $N = (P, T, L, F, \ell)$ is the flattened P/T-net such that:*

1. *$P = C \cup (\cup k: 0 \leq k < n: P_k)$,*
2. *$P_i = \{source(N_k) \mid 0 \leq k < n\}$,*
3. *$P_o = \{sink(N_k) \mid 0 \leq k < n\}$,*
4. *$T = (\cup k: 0 \leq k < n: T_k)$,*
5. *$L = (\cup k: 0 \leq k < n: L_k)$,*
6. *$\ell = (\cup k: 0 \leq k < n: \ell_k)$, and*

7. $F = (\cup k\colon 0 \leq k < n\colon F_k) \cup \{(p, t) \in P \times T \mid (p, \ell(t)) \in G\} \cup \{(t, p) \in T \times P \mid (\ell(t), p) \in G\}$.

Let N' = (P', T, L, F', ℓ) be the labelled P/T net obtained by removing all places $X = \{p \in P_i \mid {}^N\bullet(p\bullet^N) \neq \{p\}\} \cup \{p \in P_o \mid ({}^N\bullet p)\bullet^N \neq \{p\}\}$, *i.e., P' = P\X and* $F' = F \cap ((P' \times T) \cup (T \times P'))$. flat*(Q) = N' is the flattened IOWF-net.*

According to the definition of *flat*, the source place *source*(N_k) is removed if and only if there is a transition which consumes tokens from *source*(N_k) and at least one other place, i.e., only source places which serve as the only input place for all connected transitions are kept. Similarly, sink place *sink*(N_k) is removed if and only if there is a transition which produces tokens for *sink*(N_k) and at least one other place. In [12] more information about the activation safeness, soundness, and self triggering of flattened IOWF-nets is contained.

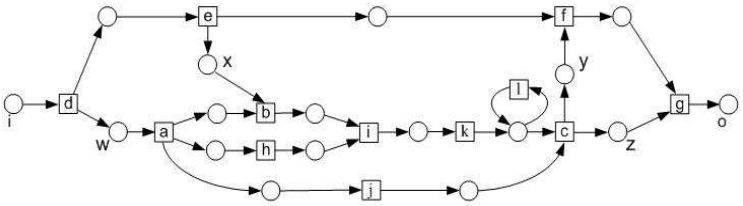

Figure 2.4: The flattened IOWF-net.

Figure 2.4 shows the IOWF-net of Figure 5.7 after the *flat* function is applied. Note that the dashed source and sink place of the bottom WF-net in Figure 2.3 are removed. The transitions of both WF-nets that are connected to labels in Figure 2.3 are now connected by places in Figure 2.4 that represent channels in Figure 2.3.

2.4.1 The Soundness of IOWF-nets

A subflow in an IOWF-net is activated if at least one of the places in the subflow is marked (except the source and sink place). Here a subflow is one WF-net that is part of an IOWF-net. A subflow might be activated again without being deactivated first, which may lead to anomalies in an IOWF-net. To detect this, the notion of activation soundness [12] is used.

Definition 14 (Activation safeness). *Let (N, s) be a marked, labelled P/T-net in* $\mathcal{N}$, *where N = (P, T, L, F, ℓ). A subset of places P' $\subseteq$ P is activation safe in (N, s) if and only if for any reachable state s' any transition* $t \in \bullet P' \backslash P'\bullet$, *and any place* $p \in P'$: *if s' enables t, then s'(p) = 0.*

A set of places *P'* is activation safe if all transitions producing tokens for *P'* but not consuming tokens from *P'* are not enabled as long as there are tokens in *P'*. A subflow that is part of an IOWF-net is not activated multiple times if and only if the places of each subflow are activation safe. Next, the notion of soundness for IOWF-nets is defined [12].

Definition 15 (Soundness of IOWF-nets). *Let Q = (C, n,* N_0, N_1,..., N_{n-1}, *L, G) be an IOWF-net and let N = (P, T, L, F, ℓ) be the corresponding flattened net without dead transitions, i.e., N =* β*(flat(Q)). Q is sound if and only if:*

1. *($\forall k$: $0 \leq k \leq n$: $N_k \in \mathcal{W}$), i.e., all subflows are sound,*
2. *$N \in \mathcal{W}$, i.e., the flattened IOWF-net is a sound WF-net, and*
3. *($\forall k$: $0 \leq k \leq n$: $P_k \backslash$\{source(N_k), sink(N_k)\} is activation safe in (N, [i])).*

The first condition states that sound WF-nets compose a sound IOWF-net. According to the second requirement, the flattened IOWF-net is also a sound WF-net. Note that a flattened IOWF-net is without dead transitions, i.e., the dead transitions are removed using β. The third requirement ensures activation safeness.

When business processes need to be related inter-organizationally, it is desirable to establish a relationship that can be analyzed and checked for correctness. The following section proposes such an inheritance relationship.

2.5 A Notion of Business-Process Inheritance

To expresses a client-server relationship between an original equipment manufacturer and suppliers, a special notion of business-process inheritance is used in this section, namely the notion of *projection inheritance*[14, 27] that can informally described as follows:

> *For two workflow process definitions A and B, where B contains all transitions in A and some additional ones, if it is not possible to distinguish between the behavior of A and B, when the effects of the transitions that are in B but not in A are hidden (ignored), then B is a subclass of A under projection inheritance*

Before, projection inheritance can be defined formally, first an equivalence relation needs to be specified. This equivalence is based on the idea that a superprocess and a refined subprocess have the same (observable) behavior. Concretely, *branching bisimilarity* [48] is such an equivalence.

The notion of a *silent action* is pivotal for branching bisimilarity that can be used to hide labels. Silent actions are result from an abstraction that is defined as follows:

Definition 16 (Abstraction). *Let N = (P, T, L, F, ℓ_0) be a labelled P/T-net. For any $I \subseteq AL_v$, the abstraction operator τ_I is a function that renames all transition labels in I to the silent action τ. Formally τ_I(N) = (P, T, L, F, ℓ_1) such that, for any $t \in T$, $\ell_0(t) \in I$ implies $\ell_1(t) = \tau$ and $\ell_0(t) \notin I$ implies $\ell_1(t) = \ell_0(t)$.*

Silent actions can not be observed and are denoted with the label τ, i.e., only transitions in a Petri net with a label different than τ are observable. Such a single label suffices as all internal actions are equal in the sense that can not be observed by the collaborating counterpart. In Figure 2.5, the depicted WF-net contains τ-labels for both transitions of the parallel branch, which means these two transitions are silent actions.

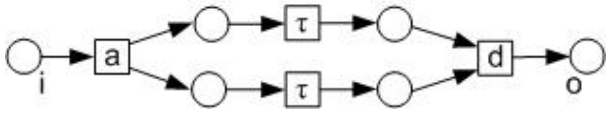

Figure 2.5: Silent actions in a WF-net.

Two marked, labelled P/T-nets are called *branching bisimilar*, denoted $p \sim_b$, iff their observable behaviors coincide, i.e., abstracting from silent actions. For a formal definition it is referred to [12]. Branching bisimilarity is an equivalence relation on $\mathcal{N}$, i.e., $\sim_b$ is reflexive, symmetric, and transitive (see [26]). Branching bisimilarity is used in the following definitions.

Definition 17 (Behavioral equivalence of WF-nets). *For any two WF-nets N_0 and N_1 in $\mathcal{W}$, $N_0 \cong N_1$ iff (N_0,[i]) $\sim_b$ (N_1,[i]).*

After clarifying the notion of behavioral equivalence the initially presented notion of projection inheritance above can be defined formally. For that purpose the abstraction operator τ_I of Definition 16 is useful for hiding labels. The definition of projection inheritance is presented as follows.

Definition 18 (Projection inheritance). *For any two weakly sound WF-nets N_0 and N_1 in $\mathcal{W}$, N_1 is a subclass of N_0 under projection inheritance, denoted $N_1 \leq_{pj} N_0$, iff there is an $I \subseteq AL_v$ such that ($\tau_I(N_1)$,[i]) $\sim_b$ (N_0,[i]).*

In Figure 2.6 projection inheritance is illustrated by means of an example. The top of the figure shows a WF-net where all transitions carry visible labels. The WF-net at the bottom of Figure 2.6 shows three additional transitions. Thus, when the labels of these transitions are renamed into silent actions, their effect is not visible.

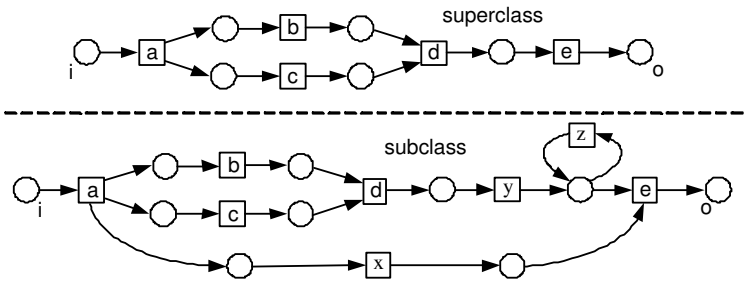

Figure 2.6: A projection-inheritance example.

The question arises how the two processes of Figure 2.6 relate to each other. Both WF-nets are identical with the exception of additional transitions with the labels *x*, *y*, *z* and their corresponding input- and output places in the bottom net. These transitions are inserted into the subclass process in according to refinement patterns. The thee refinement patterns are explained by means of the example the Figure 2.6 explains.

- Firstly, the transition carrying label *x* represents an inserted *parallel branch* that connect the *a*- and *e*-labelled transitions. These transitions are also contained in the superclass process. Note that a parallel branch can connect two τ-labelled transitions.
- Secondly, the transition carrying a label *y* represents an *inserted transition* that extends a sequence of nodes.
- The transition with label *z* is part of an inserted *loop construct* that start and ends with the same place.

In [27] details are contained about the three mentioned projection-inheritance preserving refinement patterns in the subprocess of Figure 2.5. Finally, according to Definition 18, the bottom WF-net of Figure is a subclass according to projection inheritance compared to the top WF-net.

It was stated earlier that business-process models of increasing size are difficult to check manually for contained mistakes such as deadlock. When business processes are linked inter-organizationally, this problem becomes even more acute as the linked processes may result in deadlocks or livelocks although the separate processes terminate correctly. Manual checks are additionally impaired as organizations that own business processes are reluctant to disclose all their internal business details. Thus, on the one hand the availability of tool support saves time and money by automating the process verification, on the other hand it also helps to keep business internals hidden when such tools are available as intermediary services between collaborating parties that are operated by trusted third parties, e.g., an e-notary. The following section introduces a tool for verifying process models.

2.6 A Verification Tool

As a means of verification, Woflan [108] has been developed, a tool which analyzes workflow process definitions specified in terms of Petri nets. There is a need for such a tool, to verify the correctness of workflow process definitions. Serious errors may remain undetected if process models are checked manually. This means that an erroneous process model may be carried out by an application, i.e., a workflow management system, thus causing dramatic problems for an organization.

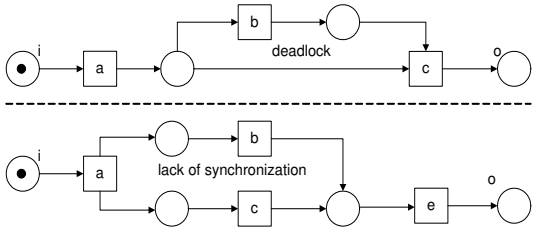

Figure 2.7: Violations of soundness.

If an intra-organizational business-process model is not sound, Woflan guides the user in finding and correcting the error. For WF-nets typical detected errors are a *deadlock* or a *lack of synchronization*. In the top of Figure 2.7, soundness is violated because *c* is dead. The reason is that no state can occur during carrying out the proces where both input places of the *c*-labelled transition contain at least one token. The bottom process of Figure 2.7 shows a lack of synchronization caused by an AND-split being complemented by an OR-join. As a result more than one token remains in the *o*-labelled place after executing that process.

When processes are matched for inter-organizational business collaboration, Woflan supports the checking of projection inheritance. In Figure 2.8, a variation of the subclass from Figure 2.5 is depicted. When the earlier superprocess and the subprocess from Figure 2.8 are checked by Woflan, a violation of projection inheritance is detected. The reason is the different refinement compared to the subprocess in Figure 2.5

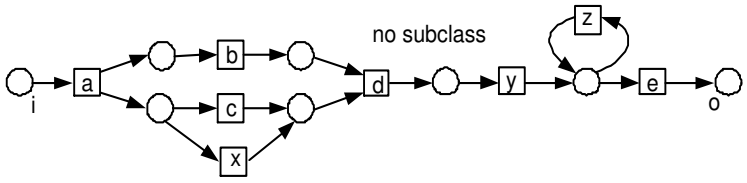

Figure 2.8: A violation of projection inheritance for the superclass of Figure 2.5.

with the *x*-labelled transition that is part of an OR-split and competes for the token with the *c*-labelled transition. Thus, when the *x*-labelled transition fires, the perceived behavior differs from the superprocess in Figure 2.8 as the *c*-labelled transition does not need to fire.

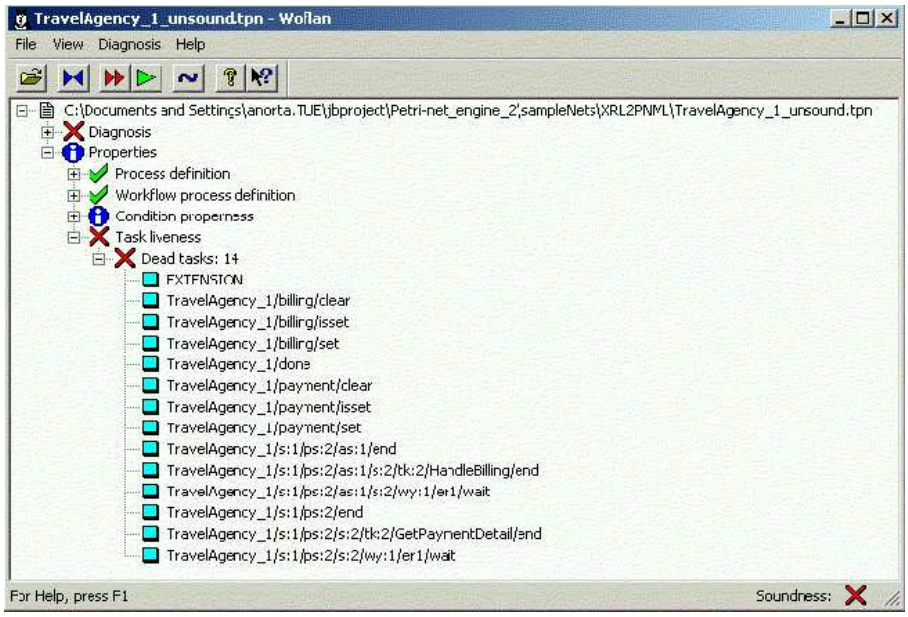

Figure 2.9: A screenshot of Woflan.

A screenshot of Woflan is depicted in Figure 2.9 where the results from diagnosing a WF-net are displayed. In this case Woflan has detected dead tasks, i.e., tasks that never get enabled. These tasks are listed in the graphical user interface of Figure 2.9. Thus, a process modeler can use this information for detecting which nodes are involved in the control-flow error.

2.7 Conclusion

This preliminary chapter presents the use of the Petri-net formalism for inter-organizational business process collaboration. A subclass of Petri-nets, namely WF-nets, is well established for modelling intra-organizational processes. Since for WF-nets the notion of

soundness exists, it can be used for verifying the correct termination of a modelled process before enactment. This is of importance as manually checking complex processes for errors like deadlocks or lack of synchronization is difficult.

For inter-organizational process collaboration, the IOWF-net formalism and corresponding soundness is presented. Furthermore, the use of a matching formalism is proposed, namely projection inheritance, which allows establishing a perceived behavior-relationship between a superprocess and a subprocess. Transitions that extend the subprocess are hidden and the perceived process behavior still needs to be equivalent compared to the superprocess. Projection inheritance and soundness can be verified by the Woflan tool, which offers a graphical user interface for supporting business-process modelers.

Chapter 3

The Perspective of eSourcing

Contents

Dynamic inter-organizational business process management (DIBPM) combines service-oriented business integration (SOBI) and workflow management as a promising approach for supporting commercial business-to-business (B2B) activities over web-based infrastructures. SOBI applies concepts from the field of service-oriented computing in the domain of dynamic business collaboration. There is scope for SOBI technologies to improve the support of B2B collaboration where dynamic matching of structures of service consuming and service providing processes is performed. Collaborating parties want to control how much process detail they expose and which parts of them are monitorable. SOBI technology should offer rigor that permits verification of desirable features before enactment, e.g., correct termination. Furthermore, current SOBI technologies lack concepts which are useful for specifying and implementing B2B collaborations. Hence, several related critical issues are explored in this chapter. Firstly, how to manage the inherent conceptual, business, and technological complexity of such business collaboration. Secondly, the issue is addressed of laying a foundation for verifying control-flow adherence and correct termination of coupled business processes. These requirements need to be guiding the development of specification languages of inter-organizational business processes and related middleware that enact them in a web-based way. Exploring these critical issues leads to the proposal of eSourcing.

For tackling the business-, conceptual-, and technological complexity of dynamically matching a service consuming and a service providing process in eSourcing a three-level framework is employed. Furthermore, eSourcing offers rigor by utilizing WF-net theory that results in improved control over inter-organizational business process structure. Finally, the issue of application suitability is tackled by discovering inherent eSourcing features that permit the positioning of Sourcing configurations in differing perspectives.

3.1 Introduction

The way companies collaborate with each other is experiencing significant changes. Employing information infrastructures in novel ways enables new ways of B2B collaboration. A promising approach for B2B collaboration is the coupling of workflow management with SOBI. This framework of DIBPM [50] offers a new model for addressing the need of organizations for dynamically bringing together a service consumer and a service provider over web-based infrastructures where the service is a business process. In this chapter the term *dynamic* refers to automatically integrated business relations that are forged between business parties by matching structures of respective processes.

The setup of such B2B collaboration is a client-server relationship where one party offers a service that is integrated into the process of a consumer. Thus, parties that wish to engage dynamically disclose process details to each other while they keep many details hidden to safeguard their competitive advantages. Though a service provider has to adhere to the requirements agreed with the service consumer, the provider still needs flexibility for extending and adjusting the service provisioning to internal needs that remain opaque to the consumer, e.g., to perform back-office tasks.

More recently, in the EU research project CrossWork [2], the objective is pursued to develop automated mechanisms for allowing dynamic workflow formation and enactment, enabling tight coupling and strong synergies between different organizations. The CrossWork architecture involves the development of ontologies for goal decomposition and team formation, followed by an inter-organizational business-process setup-, verification-, and enactment-environment that integrates legacy systems. Furthermore, this architecture is complimented by visualization tools for the setup and enactment phase of an eSourcing configuration.

An integrating concept for DIBPM is missing that exploits existing research results. *eSourcing* is proposed building on the idea of having a part of the overall business process of a service consumer performed by a service provider. To handle the inherent business, conceptual, and technological complexity, a framework is adopted that comprises several levels to achieve a separation of concerns. Furthermore, the concept of eSourcing is rigorous as it builds on Petri-net theory as presented in Chapter 2. With respect to the issue of suitability of eSourcing applications for the requirements of dynamic business collaboration, a top-down exploration approach is chosen for discovering and exploring relevant characteristics of eSourcing. Various degrees of enactment progress monitoring of a service consumer are resulting from differing ways of linking nodes of the consumer process and provider process. Dedicated constructs in eSourcing lay the foundation for inter-organizational data exchange. Finally, different degrees of collaboration visibility are discovered that permit collaborating parties to keep certain business activities hidden from each other.

The structure of this chapter is as follows. First, the context of eSourcing is given in Section 3.2 where the use of a three-level model is proposed, followed by a definition

of the nature of eSourcing in Section 3.3. After presenting an exemplary eSourcing configuration in Section 3.3.1, the control-flow properties of eSourcing are defined in Section 3.3.2 and verification criteria of eSourcing are discussed in Section 3.3.3. Next, the eSourcing perspective is positioned and investigated in correlation to DIBPM in Section 3.4. To investigate eSourcing in a top-down way, multi-dimensional, logical spaces are presented for interaction patterns that occur during the setup phase in Section 3.4.1 and for construction-elements of eSourcing configurations in Section 3.4.2. The values on the space axis are described in detail in these sections. Finally, subsequent research areas related to eSourcing are given in Section 3.5 followed by a conclusion in Section 3.6.

3.2 Tackling Complexity

A model is required to manage the complexity resulting from matching and subsequently enacting inter-organizational processes. A definition of DIBPM is given as follows [50]:

> *A dynamic inter-organizational business process is formed dynamically by the (automatic) integration of the subprocesses of the involved organizations. Here dynamically means that during process enactment collaborator organizations are found by searching business process market places and the subprocesses are integrated with the running process.*

Related issues to DIBPM are the definition and identification of processes, the way compatible business partners find each other efficiently, the dynamic establishment of inter-organizational processes, and the setup and coupling for process enactment. With respect to business-processes integration in DIBPM that is based on matching specific characteristics, various approaches are possible. The simplest matching approach is by name, which is only applicable in very simple or highly standardized cases. Attribute-based matching is performed by comparing values of attributes of services that are standardized within a specific domain, such that there are no semantics conflicts. Examples for attributes are the name of a service, the price in business-oriented matching, transactional properties like the presence of a particular compensation mechanism, or QoS dimensions such as service availability. Semantics-based business process matching is an extension of the attribute-based approach where attributes are compared based on ontologies that are realized with semantic web technology such as OWL-S [74]. Finally, the eSourcing concept represents a structure-based approach of business-process matching that focuses on the structure (or behavior) of the process itself. In order to manage such complex issues, a three-level framework [51] is a suitable model.

The bottom level of Figure 3.1 shows the internal level. Using an internal level in the domains of a service consumer and provider caters towards a heterogenous system environment. Often organizations support their business processes by containing them in a hard-coded way in legacy systems. Examples of such legacy systems are applications for enterprize-resource planning, databases, accounting systems, applications for human-resource management, and so on. If the business processes of an organization are known and modelled, they are directly enactable by process management applications, e.g., by intra-organizational workflow management systems. Companies are reluctant to directly link their internal-level legacy systems inter-organizationally to safeguard their information infrastructure and the fear of disclosing business internals that result in a loss of competitive advantages.

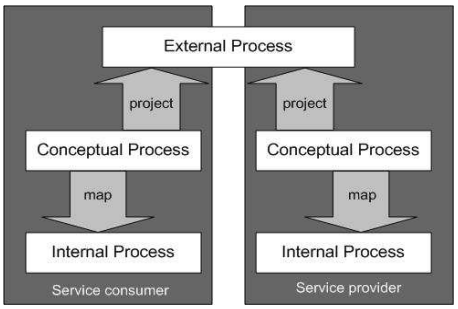

Figure 3.1: A three-level business process framework.

At the conceptual level, the business processes are designed independent from infrastructure and collaboration specifics. Conceptual processes are mapped to their respective internal level for enactment. If the conceptual-level processes are supported by service oriented architecture, their enactment allows the orchestration of web-service wrapped legacy systems of the internal level. A language used on the conceptual level should have clearly defined semantics so that collaborating parties can us a common denominator for inter-organizational business process harmonization.

The external level stretches across the domains of the process initiator and responder. Parts of the conceptual processes are projected to the external level and compared by the collaborating parties. That way the parties investigate the demands of service consumption and the ability of service provision. Since not the entire conceptual-level process must be project to the external level, an organization can determine which business internals should remain hidden from the counterpart. The process-based collaboration is automated and dynamically forged.

After proposing a three-level framework to tackle the issue of complexity in DIBPM, the following section defines eSourcing, gives an example, and presents important control-flow properties, followed by a subsection about quality features of eSourcing.

3.3 eSourcing and Control-Flow

eSourcing embedded in DIBPM is essential for the automatic external-level harmonization of parties that wish to collaborate. Harmonization refers to the external-level structural matching of business processes, i.e., the control-flow properties of the externalized processes are compared. The following definition of eSourcing is used:

> *In the context of DIBPM, eSourcing is a framework for harmonizing on an external level the intra-organizational business processes of a service consuming and service providing organizations into a B2B supply-chain collaboration. Important elements of eSourcing are the support of different visibility levels of corporate process details for the collaborating counterpart and mechanisms for service monitoring and information exchange.*

The next section presents eSourcing in a configuration example that focuses on the control-flow perspective and is modelled using labelled Petri nets as introduced in Chapter 2. The control flow perspective focuses on the way tasks are ordered, either

in a sequence or in parallel [18, 19]. Note that Chapter 5 provides definitions for this informal discussion of eSourcing control-flow properties.

3.3.1 An Exemplary eSourcing Configuration

eSourcing is applicable in many market configurations. For example, a travel agency offers complex travelling products consisting of parts that it sources in from other, specialized tourism-industry companies.

Figure 3.2 depicts the in-house process of the travel agency on the conceptual level. Passive nodes labelled with *i* and *o* are the unique input and output places of a demarcated area of the conceptual-level process. This demarcated area is a subnet of the in-house process that is termed *sphere* and destined for externalization.

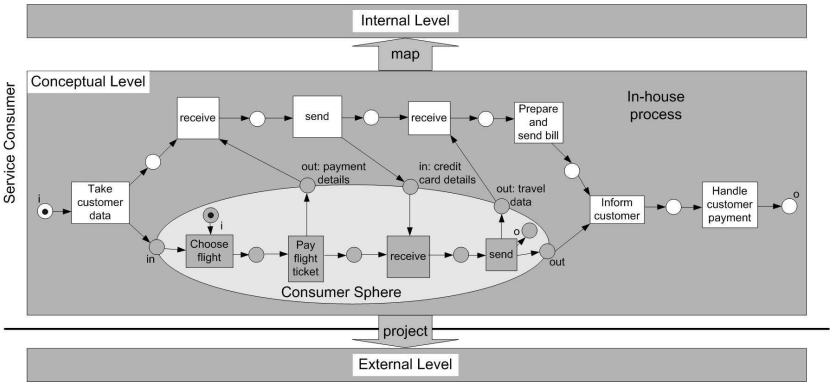

Figure 3.2: A conceptual-level process of a service consumer.

Next, the content of Figure 3.2 is explained. Following the three-level business process framework of Figure 3.1, the consumer's in-house process is mapped to the internal-level. Regarding the conceptual level process, after starting the process by taking a customer's details, the particular travelling wishes are acquired. Next, a parallel split is modelled where one branch contains a consumer sphere that is depicted with a grey shaded ellipse.

In the consumer sphere the travel agency allocates a flight ticket for the customer while billing matters are prepared and exchanged with the in-house process. Eventually, the parallel branches are joined by a node that results in informing the customer about all flight details followed by the handling of customer payment. The exchange direction between the consumer sphere and the rest of the in-house process is recognizable by the *in* and *out* labels of interface places, which are a labelling notation for a subset of places in spheres. Thus, the subset of places with *in*-labels is denoted as *in*-labelled interface places and the subset of places with *out*-labels is denoted as *out*-labelled interface places.

The other parallel branch handles exchanges of business critical information with the consumer sphere. Note that the consumer sphere is externalized to a different organization that carries the sphere out as a service provider. Thus, the modelled exchanges result during enactment in an exchange of business information between the opposing collaboration domains.

Booking a flight ticket is not core business of the travel agency. Thus, the travel agency assigns the consumer sphere to a separate organization that functions as a service provider. Relating this situation to Figure 3.3, the consumer sphere is projected entirely to the external level turning it into the consumer's contractual sphere. The eSourcing counterpart responds with projecting a structurally equivalent provider's contractual level to the external level. Consensus is achieved when the respective contractual spheres are equal as depicted in Figure 3.3.

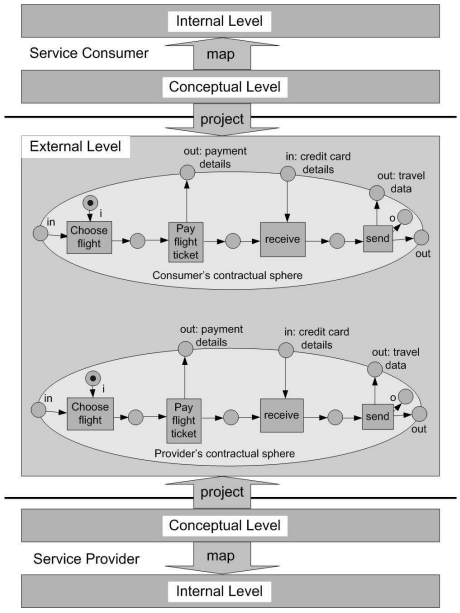

Figure 3.3: An external level of an eSourcing configuration.

In compliance with the three-level model of Figure 3.1, the service provider also has a conceptual level. In Figure 3.4 the conceptual-level process is depicted, which is a refined sphere in correlation to the contractual sphere that is termed provider sphere. Active nodes with labels equally contained in the provider's contractual sphere are visualized using broader lining. Figure 3.4 shows an internal level on which the provider sphere of the conceptual level is mapped.

Next, the provider sphere is explained. An active node for choosing a flight ticket is followed by an added parallel branch in which the flight ticket is booked and the billing organized. These additional active nodes carry labels that do not exist in the corresponding contractual spheres. Thus, the service consumer is not aware of this refinement. After the provider sends the payment data, further refining nodes for posting the flight ticket and checking payment are contained in the provider sphere. The sequence continues with two consecutive nodes, one for receiving credit card details and a following send node for exchanging travel data with the service consumer's in-house process. Note that data flow in eSourcing is not the focus of this chapter. However, explicit send and receive nodes in combination with *in* and *out*-labelled passive interface nodes lay the foundation for developing sophisticated inter-organizational data flow for eSourcing instances.

Once enactment of the provider sphere is completed, an active node in the in-house

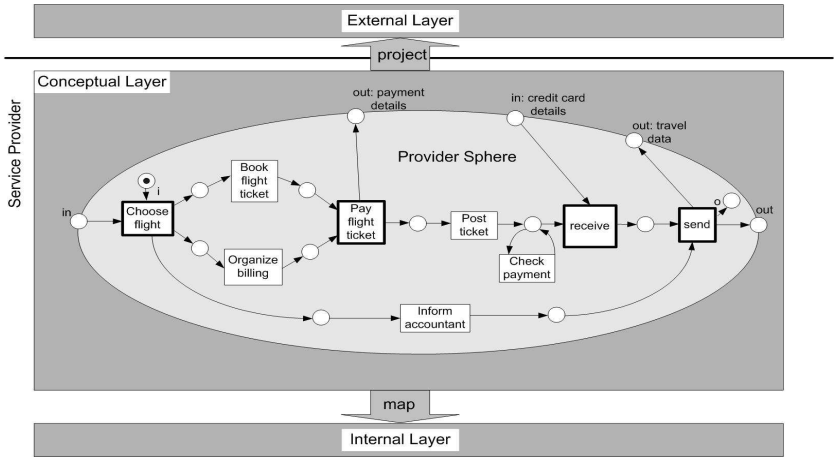

Figure 3.4: A conceptual-level process of a service provider.

process is enabled for informing the customer. Since this active node is outside of the consumer sphere, the service provider is not aware of such a step. The in-house process concludes with handling customer payment. After that the enactment of the eSourcing configuration is completed.

The figures of this subsection comprise an eSourcing configuration that requires further explanations of the control-flow properties. Thus, the next section defines those properties followed by a subsection about a desirable requirement of eSourcing. Note that Chapter 5 contains the formal definitions for eSourcing properties.

3.3.2 Control-Flow Properties of eSourcing

Figure 3.2 shows *in* and *out*-labelled passive nodes that are so-called interface places. They connect active nodes that are located in a consumer sphere and the rest of the in-house process. The labels specify the nature of exchange between the in-house process and a consumer sphere. Exchange can only occur after a consumer sphere has begun with enactment being enabled by a token placed in an *in*-labelled interface place. When a token is placed into an *in* or *out*-labelled interface place from an active input node, an exchange is attempted between the domains of a service consumer and provider.

Taking into account the Petri-net theory of Chapter 2, the conceptual-level process of Figure 3.2, so the in-house process with the contained consumer sphere, is a sound WF-net. The in-house process has a unique input place with one token, no nodes are dangling, and there is a unique output place where only one token is left once enactment has completed. The consumer sphere of the service consumer in Figure 3.2 is a subnet contained in the in-house process. All nodes belonging to a consumer sphere are depicted as grey shaded when located as a subnet in the in-house process of a service consumer. A consumer sphere has an input place labelled i and an output place labelled o. All nodes belonging to the sphere are connected. When a consumer sphere is enabled, a token is put into an *in*-labelled interface place produced from an active input node not belonging to the consumer sphere. After its enactment, only one token is left in its unique output place and one *out*-labelled interface place enabling one or many active nodes from outside of the consumer sphere belonging to the in-house

process.

By using control-flow constructs for emulating business-data exchange, soundness can be verified before an actual exchange between eSourcing domains takes place during enactment. In this context, the term exchange is synonymous with data flow between a sphere and an in-house process. Enactment abnormalities like deadlocks may occur when the information flow deviates from control-flow that is otherwise sound. However, many data-flow problems occurring during enactment are avoidable when it is assumed that data ideally flows along the control flow of a process. That way data flow is emulated during build time with control-flow elements that include explicit control-flow constructs like *in* and *out*-labelled interface places.

Neither the consumer sphere nor the provider sphere of a service provider are WF-net. As required, starting from the *i*-labelled input place the *o*-labelled output place is eventually reached once enactment is completed and all active nodes contribute to the processing of a case. However, the interface places of a provider sphere represent additional input and output places that are not permitted according to the definition of a WF-net. Looking at the consumer sphere in isolation and at the provider sphere, they are WF-nets when the interface places are removed. Thus, it is necessary to ensure in an eSourcing configuration that the interface places are empty when the in-house process terminates.

Comparing the consumer sphere and the provider sphere, one notes the labels of active nodes are identically contained in both spheres. However, labels in a provider sphere must not always be in the consumer. On the other hand a service provider is not aware of active-node labels from the domain of a consumer that are not projected to the external level. Likewise, a service consumer is not aware of labels contained in a provider sphere that are not projected to the external level, as depicted in Figure 3.3.

3.3.3 Verification Criteria for eSourcing

Respecting control flow, the question arises how a service consumer ensures that a provider adheres to the contractual sphere in an eSourcing configuration while not imposing fixed routing in the provider's domain. Therefore a notion of *inheritance* is required that focusses on the dynamics rather than data and/or signatures of methods. Four notions of such dynamic inheritance have been identified in [13, 26] addressing the usual aspects of substitutability, subclassing, and subtyping. Substitutability refers to replacing the superclass with a subclass without breaking the system. Subclassing asks if the subclass can use the implementation of the superclass. Finally, subtyping means a subclass can use or conform to the interface of the superclass. The four notions of inheritance are inspired by a mixture of these aspects. For eSourcing a restriction to so-called *projection inheritance* (see Definition 18 in Section 2.5) is considered. Informally, projection inheritance allows the incorporation of a refining task in a superprocess without violating the displayed runtime behavior. Mapped on the eSourcing configuration of the figures in Section 3.3.2, hidden steps are active nodes of a provider sphere where their labels are not contained in the corresponding contractual sphere. When a consumer's sphere is enacted and the service consumer only perceives the effects of active nodes with labels that are also contained in the consumer's sphere, the provider sphere is a subclass.

An evaluation is achieved by introducing the procedure of *collapsing*. Figure 3.5 shows at the top the service consumer's and provider's conceptual-level processes. A consumer sphere is contained in the in-house process of the service consumer. Interface places permit exchanges between the in-house process and the consumer sphere.

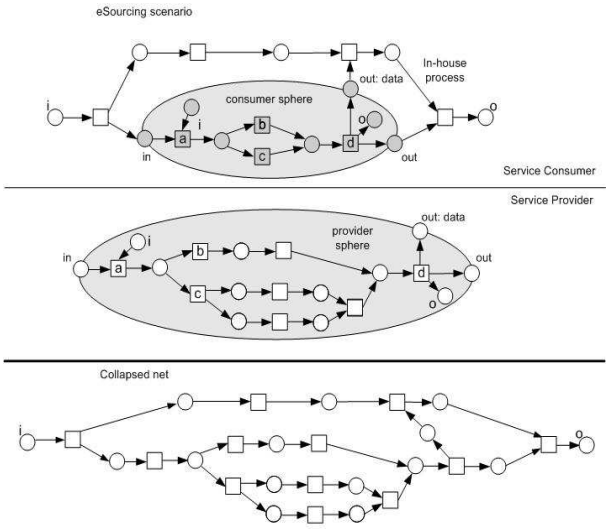

Figure 3.5: Collapsing an eSourcing configuration.

Likewise, the corresponding provider sphere has interface places. The collapsed net is shown at the bottom of Figure 3.5.

Compared to the top depiction, the bottom process shows that the consumer sphere is removed and replaced with the provider sphere in the in-house process. As a result, the collapsed net must be a sound WF-net. Projection inheritance is adhered to if the collapsed net containing the provider sphere is a subclass net of the consumer's in-house process. However, in this case the collapsed net is not a projection-inheritance subclass of the consumer's in-house process. While the soundness of an eSourcing configuration is a required minimum that must always be adhered to with respect to control flow, projection-inheritance is not always compulsory. For example, if a service consumer and a service provider decide to only agree on disclosing the interfaces of their service collaboration, projection inheritance can not be requested.

If an eSourcing configuration is not sound, the enactment of a corresponding eSourcing configuration fails because of control-flow abnormalities, e.g., a contained deadlock. On the other hand, when the full content of the consumer's conceptual-level sphere is projected to the external level, the provider must demonstrate projection-inheritance adherence in its provider sphere. In the example of Section 3.3.1 this situation is given. Flexibility is given as the provider is permitted to add refinement nodes that do not violate projection inheritance. The topic of control-flow verification is formally analyzed in further detail in Chapter 5.

After proposing the three-layer framework for complexity management and informally discussing the control-flow perspective in eSourcing, the third relevant issue of investigation in this chapter is an analysis of eSourcing features. They are instrumental for the development of suitable applications, which meet the needs of collaborating business parties.

3.4 Suitability Features of eSourcing

As the previous section demonstrates, control-flow is one fundamental perspective of inter-organizational business-process collaboration. In this section other related perspectives are discussed. Informally, a perspective is a particular angle from which a certain domain is regarded. Figure 3.6 relates eSourcing to other essential perspectives of inter-organizational business-process collaboration. To the very left and right of the figure, factory symbols represent a service consumer and provider where internal and conceptual-level processes are located. eSourcing rests on other relevant perspectives depicted as pillars in the center of Figure 3.6 where external-level processes are located. The listed pillar-perspectives are considered the most significant for DIBPM without claiming completeness.

The importance of the control-flow pillar is identified in the previous section. In an eSourcing configuration, data flow is essential as input and output of information during the enactment of service provision steps. Data flow focusses on the various ways in which data is represented and utilized in business processes [95]. The provision and consumption of services involves human or non-human resources, e.g., machines, production material, office space, and so on. Thus, the resource pillar deals with the way how the involvement of such resources is represented and utilized [96]. Finally, the enactment of an eSourcing configuration must offer a degree of certainty. By including a transaction pillar, enacted eSourcing steps are secured and exceptional situations are handled and compensated if required.

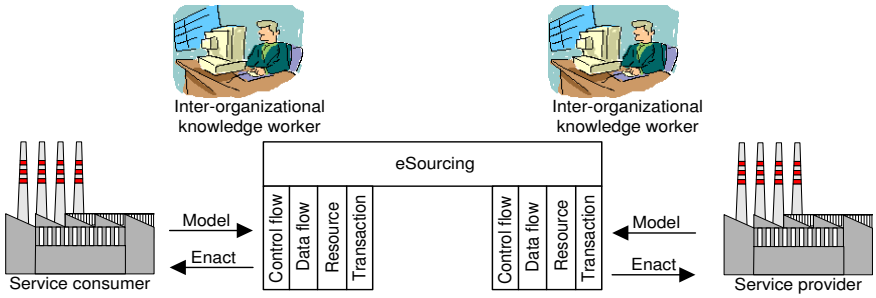

Figure 3.6: Relating perspectives for DIBPM.

For eSourcing the objective is pursued to realize expressive and suitable applications that support collaborating organizations in setting up and enacting eSourcing configurations in an automated way. Here, the setup phase focuses on specifying an inter-organizational business process template and linking the information infrastructure of the respective collaborating parties according to the three-level framework [51], i.e., the legacy systems on the respective internal levels are not directly linked with each other. For the enactment phase an instance of a specified inter-organizational business process template is used to orchestrate the collaboration of a service consumer and a service provider.

A well-structured approach is needed to explore the features of the perspectives contained in Figure 3.6 in order to create a foundation for developing a suitable language for eSourcing. The problem must be tackled of having to deal with a high degree of complexity that results from a heterogenous system environment of collaborating business parties.

On top of Figure 3.6 two IKWs are depicted that each belong to a collaborating organization. The IKWs support collaborating organizations in submitting process models to an eSourcing configuration that involves the depicted perspectives. Although the creation of eSourcing configurations should be carried out fully automated, it is realistic that IKWs are involved for the foreseeable future. For carrying out their work effectively and efficiently the support of a knowledge base that contains perspective-specific patterns for intra- and inter-organizational business process management is important.

To explore the eSourcing perspective for its setup phase and for its construction elements, the following method is chosen. DIBPM contains several feature dimensions in the form of axes that create a multi-dimensional space. On every axis, dimension values are located that detail the DIBPM feature an axis represents. By taking a subset of axes, a logical space is created that represents a particular DIBPM perspective.

The remainder of this section comprises examples of the explained eSourcing-exploration method. Two different multidimensional spaces are presented that emphasize separate concerns of an eSourcing configuration. First, in Section 3.4.1 a two-dimensional space guides the exploration of the interactions of collaborating parties during the setup phase in eSourcing. Secondly, starting with Section 3.4.2 the second three-dimensional space focusses on features from which construction elements are deductable. These construction elements are building blocks of an eSourcing configuration.

3.4.1 eSourcing-Interaction Dimensions

When intra-organizational business processes are linked in a B2B supply chain, the setup phase requires an additional perspective that focusses on the way how business processes need to be linked inter-organizationally. Thus, the interaction perspective that is the focus of this section, deals with the the setup phase of establishing an inter-organizational business process between a service consuming and a service providing organization. For the interaction perspective a definition is proposed:

> *The interaction perspective is focussing on the way a service consuming and a service providing party interact with each other during the setup phase of a B2B supply chain with the objective of aligning their respective intra-organizational business processes on an inter-organizational level.*

Based on CrossWork [2] case studies, two feature dimensions for the interaction perspective emerged. As Figure 3.7 shows, these dimensions exist in the form of axes that create a two-dimensional space. On every axis the dimension values are located.
Figure 3.7 shows the dimensions for the interaction perspective. One dimension is called *assignment*, which is focussing on the way a service provider is determined for an eSourcing configuration. The values on the *assignment* dimension state to which degree the collaborating parties know at the beginning of an interaction, that they collaborate with each other during enactment time of an inter-organizational business process configuration.

- *Static* assignment means the collaborating parties already know before setup time they surely collaborate with each other.
- On the other hand, *dynamic* assignment means the collaborating parties bid in an anonymous market for service provision and/or consumption and only towards the end it is clear who collaborates during enactment.

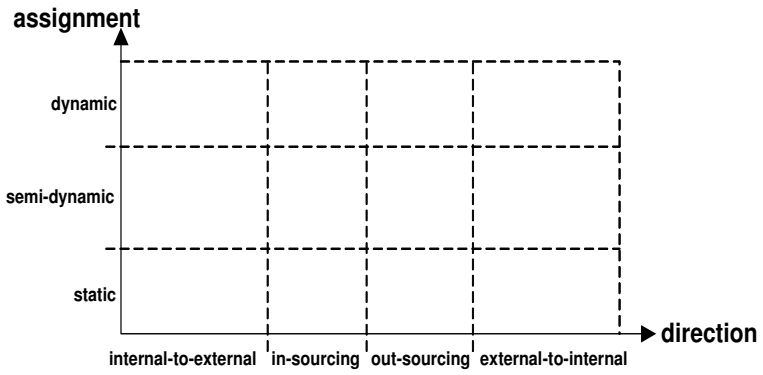

Figure 3.7: Dimensions and values of interaction patterns.

- In the *semi-dynamic* case, the number of collaborating parties that engage in a setup phase is limited at the beginning and therefor known. However, only at the end of the setup phase it is clear who collaborates.

The other dimension depicted in Figure 3.7 is named *direction* and focusses on the degree of external-level harmonization of inter-organizational business process collaboration. Thus, this dimension describes the dependencies between the processes on the conceptual- and the external level of an eSourcing configuration.

- *Internal-to-external* assignment direction means the collaborating parties have internal processes that are only harmonized externally at the end of their setup interaction.
- Likewise, the assignment direction *in-sourcing* means a service provider has a service that is subsequently integrated into the process of a service consumer. Thus, external harmonization is only performed at a later stage.
- *Out-sourcing* is similar to in-sourcing with respect to harmonization. However, now the consumer starts the interaction with externalizing a service demand first.
- Finally, *the external-to-internal* dimension means that external harmonization is the starting point of interaction and the collaborating parties set up internal processes at a later stage just before enactment starts.

For every setup phase of a B2B collaboration only one assignment and one direction value are combinable to describe the nature of interaction between collaborating parties. Thus, following Figure 3.7, there are 12 combinations possible. In Section 4.5 detailed examples of interaction patterns are given.

3.4.2 eSourcing-Construction Dimensions

The cube depicted in Figure 3.8, is created by three axes representing different eSourcing dimensions on which values are positioned. The created multi-dimensional space is instrumental for deducting eSourcing-construction elements.

The axis of the multi-dimensional space of Figure 3.8 carry the dimensions called contractual visibility, monitorability, and conjoinment. Based on experiences from

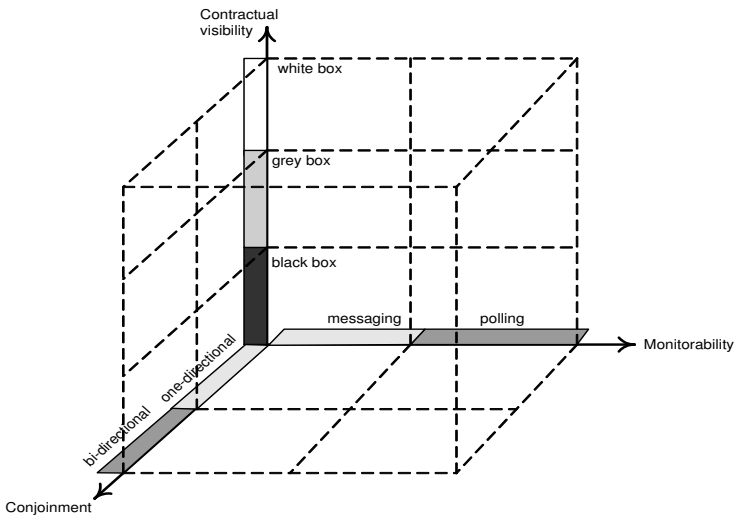

Figure 3.8: Dimensions and values of the eSourcing perspective.

modelling eSourcing configurations, further refining values depicted on the first dimension have exactly one pattern assigned. The other values positioned on the latter two dimensions have multiple patterns assigned.

3.4.3 Detailing the eSourcing-Construction Dimensions

The cube dimensions and values of Figure 3.8 are described as follows:

- **Contractual Visibility:** is focussing on how much process detail is disclosed to a collaborating counterpart. In Figure 3.3 a consensus exists since both spheres are similar. Three values are located on the contractual-visibility dimension of Figure 3.8. The values are regarding the amount of nodes from the conceptual level that are projected to the contractual sphere of the external level. First, a *white-box* value means that all nodes of a consumer sphere are contained in the contractual sphere of the external level. In case of a *black-box* value only the interfaces of a consumer sphere are projected to the contractual level. Finally, the *grey-box* value means the interface places and a subset of the nodes and arcs of the consumer sphere are projected to the external level's contractual sphere.

  Patterns covering different levels of visibility in eSourcing configuration are mentioned in relation with dynamic service outsourcing [51]. However, the listed patterns are all values of a control-flow dimension and are not separated into the axis contractual visibility and monitorability as in this chapter. For example, the black box pattern roughly corresponds to black-box contractual visibility in the sense that a consumer has no information about how the service is executed by the provider. The glass-box pattern roughly corresponds to the white-box contractual visibility as the consumer is aware of internal states of the outsourced process. Since it is not clear to which extent internal states are disclosed, the glass-box pattern can also be interpreted as similar to the grey box. The half-open box and the open box patterns are for service synchronization where the

first pattern only synchronizes from service consumer to provider and the latter pattern allows for synchronization in both directions. While nothing equivalent is contained in the dimension contractual visibility, these patterns are assignable to the dimension described below.

- **Conjoinment:** focusses on the exchange of business-relevant information between the domains of the service consumer and service provider. Consequently, the consumer sphere and the provider sphere contain equal conjoinment constructs. *One-directional* conjoining implies that there is one *out* or *in*-labelled interface place present that is either complemented by an active send node or an active receive node respectively. Those active send and receive nodes are contained in the spheres of an eSourcing configuration and handle the exchange between the domains of a service consumer and provider. *Bi-directional* conjoining is initiated by a sending active node to the domain of the eSourcing counterpart that returns the communication exchange immediately to the initiating eSourcing party.

- **Monitorability:** covers the way how nodes in a consumer's and provider's contractual spheres of the external level are linked with each other. Available linking values are *messaging* and *polling*. The nodes of the contractual spheres are connected to nodes in the corresponding spheres in the respective conceptual levels. The degree of monitorability of service provision for a service consumeris increased by the amount of nodes that are linked with monitorbility constructs. At a minimum all interface places of both contractual spheres need to be linked with each other to transitively relate the consumer sphere and provider sphere with each other. Additional passive and active nodes of the contractual spheres may be linked. *Messaging* can be applied to linked passive and active nodes contained in both contractual spheres. If two passive nodes are linked in a messaging way, the node experiencing a change in number of tokens signals that information to the linked destination node. When active nodes are linked in a messaging way, the executing node messages this event to the linked active node. The classifier termed *polling* links nodes of the service consumer's and provider's contractual sphere in a way where one node periodically checks whether the linked node has changed. When passive nodes are linked in a polling way, one node checks if the number of contained tokens has altered. Subsequently, the change is mirrored by the polling node. If two active nodes are linked in a polling way, one node periodically checks if the linked node has experienced an execution event. Subsequently, the polling node reflects the execution event.

If the dimension values of Figure 3.8 are used as a taxonomy for the specification of patterns, the following statements hold: For creating an instance of an eSourcing configuration, exactly one pattern belonging to a value of contractual visibility must be chosen. Regarding the monitorability dimension, at least two patterns have to be used, either from the value messaging or polling. That way the service provider's provider sphere is connected to the consumer sphere via the external level. The minimum amount of connected nodes are the respective *i* and *o*-labelled interface places of the spheres located on the external and conceptual levels. If increased monitorability is desired by the consumer and granted by the provider, multiple patterns from both values are optionally used to link more nodes contained in the spheres. Finally, the conjoinment dimension is optional and not required for the creation of an eSourcing configuration.

3.5 Related Research

With respect to research projects, the WISE project [22, 68] resulted in a software platform for process-based B2B electronic commerce that focusses on support for a network of small and medium sized enterprizes. In WISE a virtual business process consists of a number of black-box services that are linked in a workflow process [22]. A service is offered by an involved organization and can be a business process that is controlled by a local workflow management system. Although the WISE project succeeds in orchestrating workflows of different collaborating organizations, it does not cater for a flexible degree of mutual visibility of business-process details. Furthermore, collaborating parties can not negotiate how much may be observed during the enactment phase.

In the CrossFlow project [93] inter-organizational business process collaboration was investigated. In the context of this project, the formation of virtual enterprizes is realized by dynamically out-sourcing a part of the service consumer's process to a service provider. A service matchmaker matches a service offering and a service request. The service provider has adjustment flexibility as nodes of the assigned process can be internally refined on a lower process level. However, that way the service consumer can not ensure the lower-level refinement still results in an adherence to the agreed upon service provision. CrossFlow has an external level that spans across organizational domains where the process specification is part of a contract specification. The workflow specification language of the workflow management system IBM MQSeries Workflow [5] forms the internal process level.

The prototypical implementation called Nehemiah [101] is an inter-organizational workflow management system that combines private workflows of collaborating parties with a so-called joint coalition workflow. The prototype supports synchronous and asynchronous communication between collaborating parties while offering the protection of respective workflow details. Nehemiah uses task specialization as a mechanism to ensure the protection of private workflow views. However, just as in the case of CrossFlow, it is hard to verify whether the lower-level refinement adhere to the agreed upon service provision.

The integrated EU research project ATHENA [1] investigates enterprize inter-operability in a holistic way that is guided by user requirements. It is the strategic objective of the project to enable networked businesses and governments. In the area of inter-organizational business process modelling, a business-level framework is proposed with a B2B process spanning across organizations that is complemented by a public process and a private process in the collaborating domains. This business level is complimented with a technical and execution level that is responsible for enactment. For business-process modelling an extension of event-process chain (EPC) formalism with so-called process-modules is proposed to achieve process abstraction.

3.6 Conclusion

Three crucial issues for the evolvement of eSourcing are investigated, namely complexity management, control-flow rigor, and suitability features for applications that support the establishment and enactment of eSourcing configurations.

eSourcing uses a three-level business process framework to manage the conceptual, business, and technological complexity involved in inter-organizational business process collaboration. This chapter informally defines eSourcing with its control-flow

properties and demonstrates how the structural matching of service-requesting and service-providing processes is supported. Furthermore, a collapsing method is presented for checking during build time the adherence of eSourcing parties to process-behavior requirements and whether the enactment of an eSourcing configuration can successfully terminate.

To succeed in exploring suitability features for eSourcing, the establishment of multi-dimensional spaces is pursued for exploring the setup phase and structural elements of eSourcing configurations. These multi-dimensional spaces permit the positioning of individual eSourcing configurations and axis values of the multi-dimensional eSourcing spaces are elaborated upon. The following chapter demonstrates how the multi-dimensional spaces creates a taxonomy for top-down patterns discovery. These patterns will be instrumental for developing suitable and expressive web service composition languages and their corresponding enactment applications for formulating and enacting dynamic inter-organizational business processes.

Chapter 4

The Nature of Patterns in the Context of DIBPM

Contents

In Chapter 3, a top-down method for the exploration of eSourcing was presented. Multi-dimensional spaces are given for exploring interaction features during the setup phase of eSourcing configurations and for exploring features of eSourcing-construction elements. The objective of this chapter is to explore the eSourcing features in further detail in a technology independent and conceptual way. To achieve the objective, the eSourcing features of Chapter 3 are input for discovering patterns. Beforehand it is important to understand the nature of patterns in the context of dynamic inter-organizational business process management (DIBPM). Thus, a pattern meta-model is proposed that serves the purpose of establishing a soft ontology to achieve a uniform understanding of pattern entities and their relationships. Secondly, the pattern meta-model is the foundation for the development of a knowledge base that is populated with DIBPM-related patterns. The knowledge base is intended to support intra- and inter-organizational knowledge workers (IKWs) that search for patterns of different DIBPM

perspectives. Finally, the discovered eSourcing patterns of this chapter are examples to populate such a knowledge base .

4.1 Introduction

To establish intra- and inter-organizational business processes efficiently and effectively in DIBPM, the use of patterns is recommendable. Corporations typically have an information infrastructure consisting of a heterogenous system environment supporting their business processes. The situation turns even more complex when the business processes of collaborating parties are linked. By checking which patterns the respective heterogenous system environments support, a common denominator of collaboration is detected. Control-flow patterns [17, 18, 19] have been specified after investigating several intra-organizational workflow management systems. Furthermore, patterns for intra-organizational data-flow and resource management [95, 96] have been discovered and specified. More recently service-interaction patterns [25] have been specified for the coordination of collaborating processes that are distributed in different web services.

In the domain of SOBI, web service composition languages (WSCL) have emerged for supporting process specifications, e.g., BPEL, BPML [33, 40] and so on. Such languages formulate the orchestration of services in a workflow, creating a complex service that carries out activities. The referenced pattern specifications and WSCLs show that a rich amount of results exist that are relevant for DIBPM. For example, many e-business related patterns are textually available online [6] for the perspectives of business, integration, composite, custom design, application and runtime. For inter- and intra-organizational knowledge workers (IKWs) who are exposed to business, technological, and conceptual complexity, such patterns promise a meaningful support for effectively and efficiently establishing inter-organizational business processes with the help of SOBI technology.

IKWs organize the business processes in-house and establish business process links for B2B activities. They manage the heterogenous system infrastructure that supports such business processes. However, the pattern specifications of various perspectives that IKWs need to employ differ in specification terminology. For example, the service-interaction patterns [25] specify issues and design choices while the control-flow patterns [19] define an implementation specification. It has not been investigated how the different specification terms can be harmonized.

It is desirable to store all the pattern related data uniformly in one knowledge base and make it accessible for IKWs with tool support. Existing pattern repositories are static in content and limited to either one or a couple of perspectives. However, for IKWs it is desirable to have a repository available that is interactive and dynamically growing in perspectives and content. The repository should store knowledge about how patterns relate to each other within the same perspective and across different perspectives. This chapter fills the gap by presenting a pattern meta-model that allows dynamic growth in content by permitting the admission of new patterns that may belong to newly introduced perspectives.

The structure of this chapter is as follows. First, Section 4.2 gives an overview of a pattern meta-model that is used for uniformly storing and relating patterns to each other. In Section 4.4, the eSourcing setup-phase patterns are specified based on the dimensions of Figure 3.7 and in Section 4.5 the pattern specifications are presented based on the dimensions depicted in Figure 3.8. Next, Section 4.3 discusses pattern

catalogues that are relevant for dynamic inter-organizational business collaboration and Section 4.6 concludes this chapter.

4.2 The Pattern Meta-Model

A pattern meta-model must be able to accommodate for new patterns belonging to newly introduced perspectives that are relevant for DIBPM. It is predictable that the meta model needs to capture many patterns and an IKW should be able to quickly find them based on characterizing search options. As inter-organizational business processes are supported by a heterogenous system environment, a meta model needs to capture technology support.

Such information is useful for IKWs to determine with which commonly supported patterns inter-organizational collaboration can be established. Finally, besides IKWs, different types of users of a pattern meta-model based repository exist, e.g., an administrator, users who submit patterns, pattern reviewers, and so forth. A meta model must capture information of different user types for managing access rights to pattern information. The following subsection uses UML notation [46] to first group the pattern meta-model into packages and relates them. Next, the content of the packages are presented.

4.2.1 Meta-Model Packages

The left side of Figure 4.1 depicts a model of packages that are related to each other. These packages encapsulate classes [46] that are explained in following sections. The center of the package-model is named `Pattern`, which contains all classes that capture information for specifying a patterns. In the `Taxonomy` package, classes are contained that capture information about DIBPM perspectives. This package contains classes that create a taxonomy into which patterns can be embedded. The `Support` package encapsulates classes for managing information about technologies that support patterns. Finally, the `User Management` package captures information of different users of the pattern repository, e.g., administrator, reviewer, pattern submitter, and so on.

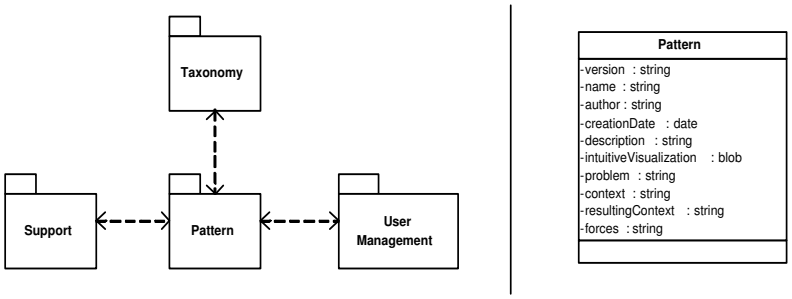

Figure 4.1: Meta-model packages with their dependencies and the `Pattern` class.

On the right side of Figure 4.1 the core class of the Pattern package is depicted, which is also named `Pattern`. The attributes of this class form the main description template of a pattern specification. A pattern has a `version` and a `name` that should be

meaningful. Furthermore, a pattern has an `author` and a `creationDate` for every version. The `description` of a pattern mentions the inherent pattern properties and describes the relationship between them. Furthermore, the `intuitiveVisualization` contains a model that helps to support the comprehensibility of the pattern description. The `problem` of a pattern is a statement describing the context of pattern application. In this context conflicting environmental objectives and their constraints are described. The application of a pattern in that context should result in an alignment of the given objectives. Next, the `context` states a precondition, which is the initial configuration of a system before the pattern is applied to it. On the other hand, the `resultingContext` describes the postcondition and possible side-effects of pattern application. Finally, the `forces` describe trade-offs, goals and constraints, motivating factors and concerns for pattern application that may prevent reaching the described postcondition.

After presenting an overview of the pattern meta-model, the following subsections present detailed models of the packages depicted in Figure 4.1. The detailing classes of package `Pattern` are presented as a white box. When the packages `Taxonomy` and `Support` are presented in detail, their references to classes in the `Pattern` package are depicted explicitly.

4.2.2 The Pattern-Taxonomy Model

As the first package that is presented in detail, Figure 4.2 depicts the classes of the `Taxonomy` package. Creating a taxonomy is relevant for ordering patterns and relating them to each other. Additionally, a taxonomy helps to find patterns that are stored in the repository.

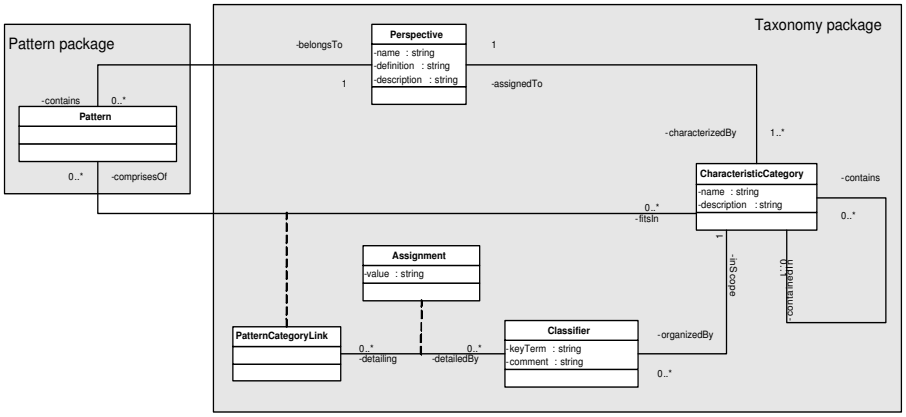

Figure 4.2: Detailed class model of the `Taxonomy` package.

The introduction of this chapter references patterns belonging to DIBPM-relevant perspectives, e.g., *control-flow*, *data-flow*, *resource*. Thus, in the depiction of package Taxonomy in Figure 4.2 the class named `Perspective` is central. Informally, a perspective can be seen as a particular angle from which a certain domain is perceived. To the left of Figure 4.2 the relationship to the `Pattern` package is depicted. It shows that a pattern always only belongs to one perspective while a perspective possibly references many patterns.

Publications of pattern specifications contain groupings of patterns that share characteristics. For example, the *white-box*, *black-box*, and *grey-box* patterns [85, 84] of the eSourcing construction dimension (see Section 3.4.2 and Section 4.5) are in one group with the characteristic *contractual visibility*. If a considerable amount of patterns is in the repository, a more detailed grouping of patterns is sensible to allow IKWs speedy pattern discovery. Therefore, Figure 4.2 shows a class called `Characteristic Category` that is assigned to one perspective. To permit recursive groupings of patterns, `CharacteristicCategory` instances may contain each other. For example, in the control-flow perspective the patterns are grouped in six characteristic categories [17], e.g., structural patterns and cancellation patterns.

Although Figure 4.2 depicts a reference between the `CharacteristicCategory` and a `Pattern` by using an association class called `PatternCategoryLink`, a further refinement of the taxonomy with additional classes is realized. Thus, a class `Classifier` organizes a `CharacteristicCategory` with refining keywords that are commented for clear comprehension. For example, the eSourcing category called *contractual visibility* [85, 84] (see Section 4.5.1) is refined by the category keyword *projection*. This keyword indicates how much process content is projected to an external level where the collaborating counterpart can perceive it. Finally, the class `Assignment` is completing the taxonomy creation by allowing the assignment of a value to a pattern classifier that belongs to a special category. For example, the black-box pattern of the characteristic category with the value *contractual visibility* has the assignment value *none*.

4.2.3 Pattern-Related Properties

The next package focusses on properties around class `Pattern`. As Figure 4.3 shows, a pattern can have one or several `Alias` instances associated, e.g., the control-flow pattern called sequence is also known as *sequential routing* or *serial routing*.

Patterns can relate to each other in different ways. The package depicted in Figure 4.3 includes relationships where patterns are either `Similar` to each other or one pattern is a `Generalization` of another one. For example, it can be argued that the control-flow patterns *parallel split* and *synchronization* are similar as they complement each other where the first pattern creates parallel branches of execution and the latter pattern is required to join them. On the other hand, the *discriminator* pattern is a generalization of the *n-out-of-m-join* pattern [17]. Both patterns perform a merge of many executing paths and execute the subsequent activity only once. However, while the first pattern performs no synchronization, the latter pattern is more flexible as it is possible to set the number of parallel branches that need to be synchronized.

There are two different ways how a `Solution` may relate to a pattern. Firstly, a pattern tackles a particular problem for which one or many `Solution` instances are applicable. On the other hand a solution is only assigned to one pattern problem. Secondly, it may require the application of several patterns to achieve one greater solution. Therefore a second relationship arc is depicted in Figure 4.3 where a solution may use several patterns. In this context a solution contains static relationships and dynamic rules describing how to realize a desired outcome. For example, for the eSourcing category contractual visibility [85, 84] the *problem* attribute (see Section 4.5.1) states the service consumer is interested in using service provision. However, the consumer does not mind how the provision is carried out as long as the specified exchanges are performed correctly. The applied pattern is called *black box* from the *contractual visibility* category and is characterized by a disclosure of business-process interfaces only

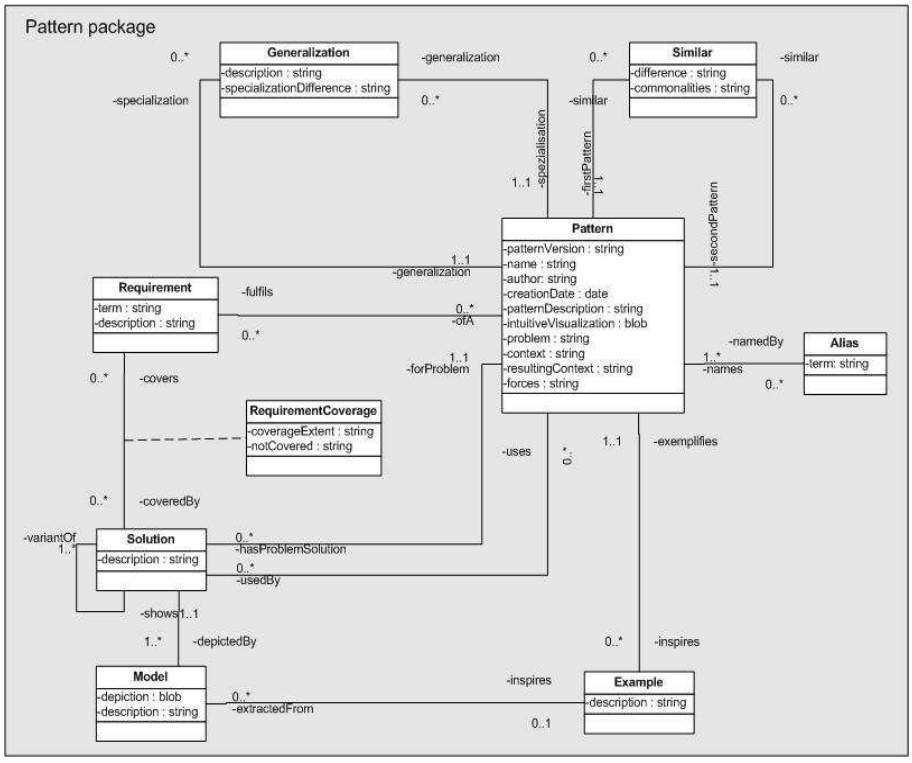

Figure 4.3: Detailed class model of the `Pattern` package.

without revealing the rest of internal process content. A solution that tackles the pattern problem is the application of a three-level framework [51] where it is possible to disclose on an external layer less than the full content of an internally given process on an internal layer.

The pattern and solutions are additionally linked by the `Requirement` class. Thus, a pattern must fulfill some requirements so that a solution becomes applicable. It represents a statement about the demanded pattern function and performance with respect to quantitative and qualitative features. For example, a requirement term can be *system-interoperability support*. A complementary description may mention that a solution must pay attention that collaborating parties do not want to disclose all their system details to each other.

The association class `RequirementCoverage` allows textual statements expressing to which extent a solution covers a requirement or not. For example, system-interoperability support is fully covered by a solution that represents a three-layer framework with internal process levels and conceptual process levels for the respective collaborating parties, and one external level for process merging [51]. That way collaborating parties have the chance to hide internal details from each other while only needing to disclose what is necessary to perform process merging on an external level.

Two more classes are depicted in Figure 4.3, namely the classes `Example` and `Model`. While one example only belongs to one particular pattern, a pattern may inspire several examples. An example is a textual description of a concrete instance in

a real-world setting where a pattern is used. Alternatively, it is also possible to give an abstract example that is based on a formal model,e.g., Petri nets.

While patterns are conceptually formulated, IKWs need to evaluate which patterns are applicable for their own system setup. Thus, the meta model should also cater for the management of technology-support information. In the next subsection a package is presented with classes for capturing such information.

4.2.4 Capturing Pattern-Support

In Figure 4.4, a package called `Support` is depicted containing classes for capturing technology information of patterns. On top of Figure 4.4 the `Pattern` package is shown with the subset of contained classes that have relationships to classes from the `Support` package. Accordingly, a pattern example is visualized by an `Illustration` instance, a model is expressed in an instance of a `Language` subclass, and a pattern is supported by an `Artifact`. In the latter case an artifact can be of a complex nature, e.g., a software system that supports a standard language.

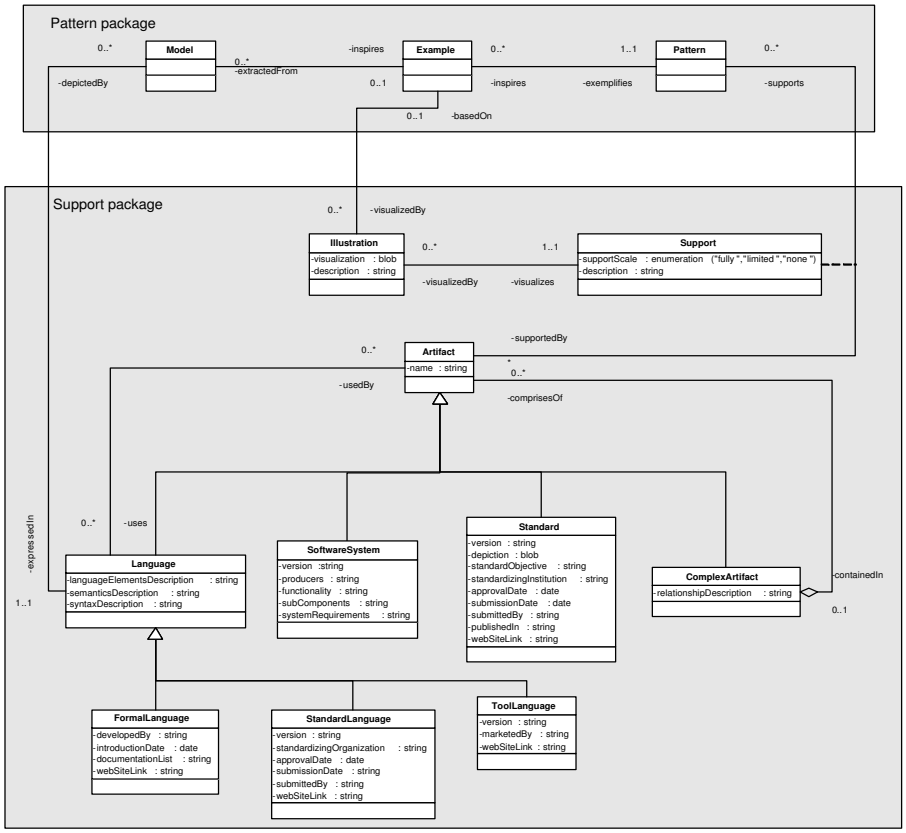

Figure 4.4: Detailed class model of the `Support` package.

Instances of the class `Illustration` contain a screenshot with textual description that shows how an artifact supports a pattern, e.g., a modelling element description. This screenshot is related to an instance of the association class `Support` that links a

pattern instance with a particular technology, indicating to which degree the the pattern is supported, i.e., either fully, limited, or not at all.

A `Model` can be represented in a language that consists of several elements. One hierarchy level lower, Figure 4.4 depicts several subclasses. An instance of `Formal Language` is, e.g., the Petri-net markup language PNML [65, 113] that is based on Petri-net theory. Instances of `StandardLanguage` refer to languages that are, e.g., XML based and pursue the objective of supporting the modelling of business processes that are carried out with the help of web service orchestration. Examples of such XML-based standards are BPEL [40] or WSDL [37]. Instances of `ToolLanguage` are related to some software system, e.g., the workflow management system Staffware [102]. Next, an instance of `Software System` is an aligned set of programs and application software that perform a specific function directly for the user. Finally, a `Standard` is a prescription or regulation that is fixed by approved institutions. Standards achieve the unification of artifacts that are published in journals. Standards in the ICT domain are necessary for interworking, portability, and reusability. They may be de facto standards for various communities, or officially recognized national or international standards. The next section discusses related pattern specifications that are candidates for filling pattern knowledge base that is built using the pattern meta-model.

4.3 Patterns in other perspectives

Referencing patterns, Gamma et al. [47] first catalogued systematically some 23 design patterns that describe the smallest recurring interactions in object-oriented systems. Those patterns are formulated in a uniform specification template and grouped into categories. For the domain of intra-organizational business process collaboration patterns were discovered in various perspectives.

In the area of control flow, a set of patterns was generated [17, 18, 19] by investigating several intra-organizational workflow systems for commonalities. The resulting patterns are grouped into different categories. Basic patterns contain a sequence, basic splits and joins, and an exclusive split of parallel branches and their simple merge. Further patterns are grouped into the categories advanced branching and synchronization, structural patterns, patterns involving multiple instances, state-based patterns, and cancellation patterns. The resulting pattern catalog is for the evaluation [15, 116] of WSCLs.

Following a similar approach as in the control-flow perspective, data-flow patterns [95] are grouped into various characteristics categories. One category is focuses on different visibility levels of data elements by various components of a workflow system. The category called data interaction focusses on the way in which data is communicated between active elements within a workflow. Next, data-transfer patterns focus on the way data elements are transferred between workflow components and additionally describe mechanisms for passing data elements across the interfaces of workflow components. Patterns for data-based routing deal with the way data elements can influence the control-flow perspective.

Patterns for the resource perspective [96] are aligned to a the lifecycle of a work item. A work item is created and either offered to a single or multiple resources. Alternatively a work item can be allocated to a single resource before it is started. Once a work item is started it can be temporarily suspended by a system or it may fail. Eventually a work item completes. The transitions between those life-cycle stages of a work item either involve a workflow system or a resource. Characteristic categories for the

resource perspective are deducted from those life-cycle transitions and group specified patterns.

More recently so-called service interaction patterns [25] are specified for the coordination of collaborating processes that are distributed in different, combined web services. Again, the patterns are categorized according to several dimensions. Based on the number of parties involved, an exchange between services is either bilateral or multilateral. The interaction between services is either of the nature single or multi transmission. Finally, if the bilateral interaction between services is of the nature two ways, a round-trip interaction means the receiver of a response must be equal to the sender. Alternatively a routed interaction takes place.

The next section presents the interaction patterns that occur between collaborating organizations during the setup phase of an eSourcing configuration. The pattern specifications are deducted from the interaction dimensions presented in Section 3.4.1 in a top down way. With respect to the interaction patterns of Section 4.4, for the examples of patterns, sequence diagrams are used that propose a possible interaction during the setup phase of an eSourcing configuration. However, for sake of brevity completeness of the interactions is not claimed. Furthermore, chunks of the overall interaction sequences are grouped into phases that are reused as interaction blocks in following sequence diagrams. In order to keep the figures brief, those reused interaction blocks abstract from the message exchanges and merely contain activation bars to depict which parties and applications are active in that particular interaction block. The labels of the interaction blocks are unique throughout this section. For example, the interaction example of Figure 4.5 depicts the interaction block labelled *contracting* as a white box. Thus, in the first occurrence all message exchanges between the parties involved in contracting are depicted. In contrast, the interaction example of Figure 4.6 depicts the contracting interaction block as a black box, hiding for sake of brevity the identical message exchanges shown in Figure 4.5.

4.4 Interaction Patterns of the eSourcing Setup Phase

This section explains first the assignment dimension of Figure 3.7 with the corresponding dimension values. Next, for each dimension value one pattern is deduced that is specified using the description template mentioned above. The same approach is used in Section 4.4.2 where the direction dimension is explained and corresponding patterns are specified for each dimension value. Each specification of Section 4.4.1 and Section 4.4.2 gives two examples, one abstract example and a real-life example.

4.4.1 Assignment-Dimension Patterns

During the setup time of an inter-organizational business process collaboration, a service consumer may be confronted with different numbers of potential providers that are either known and predetermined beforehand or not. On the one hand a service provider can already be chosen by the consumer before setup time, which is termed *static assignment*. The opposite extreme is *dynamic assignment* where a potentially unlimited number of service providers engage in a competition for involvement in an eSourcing configuration. Finally, between those extremes the pattern for *semi-dynamic assignment* specifies that a predefined number of service providers engage in a competition at setup time.

Pattern 1 (Static Assignment)
Problem: Due to market pressures a company is involved in optimized production cycles of complex industrial goods with very small order numbers. To reach the objective, it is critical that a significant part of the overall production needs to origin from an external source that can guarantee high quality and time precision.

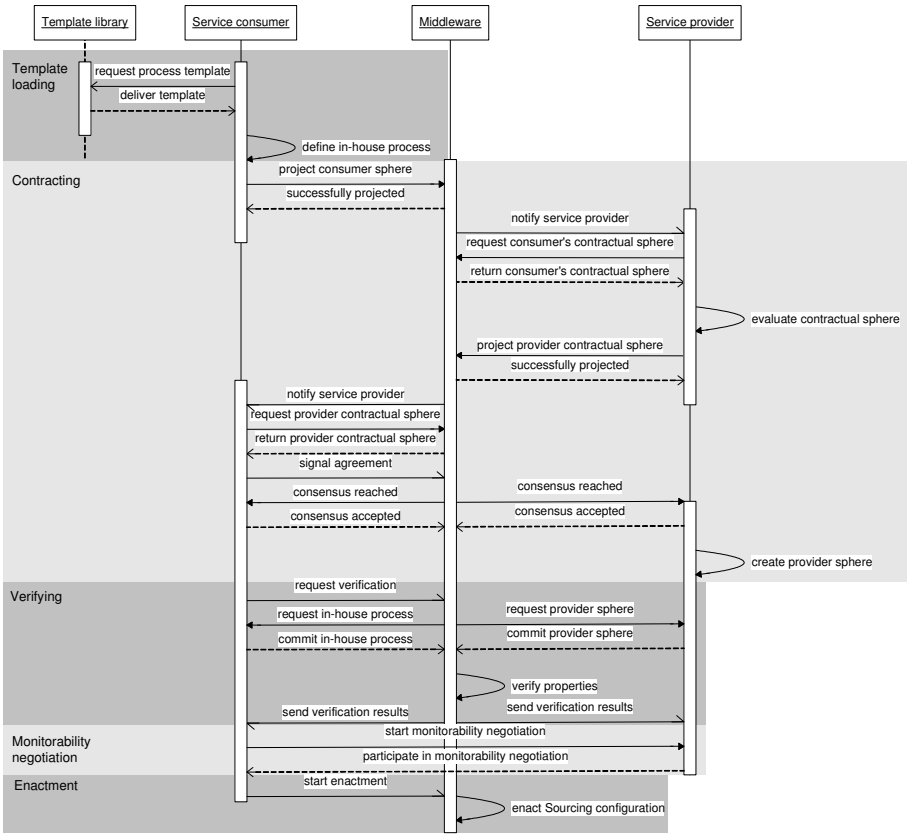

Figure 4.5: An interaction-sequence example for static service assignment.

Description: Before the setup phase of a B2B collaboration begins, a service consumer and a service provider limit their collaboration choice to one candidate. The provision candidate must be able to guarantee the consumer the capability of offering an agreed upon service. The sourced service is formulated as a template and externalized to initialize the setup interaction between collaborating parties with the objective to receive a service of predictable time precision and agreed upon quality.
Forces: Achieving tight integration between a service consumer and a provider might fail for different reasons. For example, if a provider is not capable of performing the agreed upon service, it fails to be a credible candidate. Furthermore, failing technological integration attempts between provider and consumer are a potential obstacle. The provider might not be able to offer requested quality standards of services.

Examples:

- A truck manufacturer has a competitive advantage by delivering according to customer specifications within 17 working days. Such prompt delivery is only assured when the truck manufacturer consumes clearly defined, external services that are reliably available. Therefore, there exists one specially prepared supplier who is capable of performing a mirroring of externalized consumer-processes. Since the chosen supplier happens to possess crucial production know how without which the truck manufacturer isn't able to reach his deadlines, the consumer is interested in a very tight supply-chain integration.

- An abstract example of static assignment that focusses on eSourcing is given in Figure 4.5. The a priori tight integration efforts between service consumer and provider are abstracted from. Instead the sequence of interaction starts with a service consumer loading the predefined template resulting from earlier integration efforts. The template represents a consumer sphere that is integrated into the consumer's in-house process. Next, the consumer projects its sphere to the middleware situated between the consumer and the service provider. Consequently, the service provider is informed by the middleware about the committed consumer's contractual sphere. The provider responds with a sphere projection to the middleware. If the respective contractual spheres don't match, the projection procedures need to be repeated until the respective contractual spheres match and a consensus is created. The contracting phase of interacting is ended after the middleware informs the collaborating parties about the shared consensus and when the provider has created its provider sphere. Next, the service consumer requests the middleware to perform a verification of control-flow and data-flow properties of the eSourcing configuration. In order to keep each other's processes secret from each other, the collaborating parties supply the in-house process and the provider sphere to the middleware that performs all checks. The results are sent out to the collaborating parties. If the verification succeeds, the collaborating parties may engage in negotiating the monitorability aspect of the eSourcing configuration. Finally, the service provider sends a message to the middleware that enactment should commence.

Pattern 2 (Dynamic Assignment)
Problem: For a part of the overall in-house process, an organization intends to externalize this process part to find an eSourcing counterpart. However, it is not clear who the best counterpart is. Thus, the service-externalizing organization wants to see the prospective candidates engage in a competition for service collaboration.
Description: An organization externalizes its service to a public broker that functions as an anonymous process-market place for the purpose of engaging in a business collaboration. Other organizations may evaluate the published process and decide to engage in a bidding for service. The service-externalizing organization chooses the offer and rejects all other.
Forces: The publicly available process broker needs to be equipped with a directory that allows service consumers to correctly file their process according to attributes like, e.g., the type of industry, geographic location, language, and so on. With the help of a search engine, potential service providers must be able to find those filed processes. Furthermore, it must be ensured that bids and eventually bid acceptance and bid rejections can be communicated between the potential service providers and the consumer.

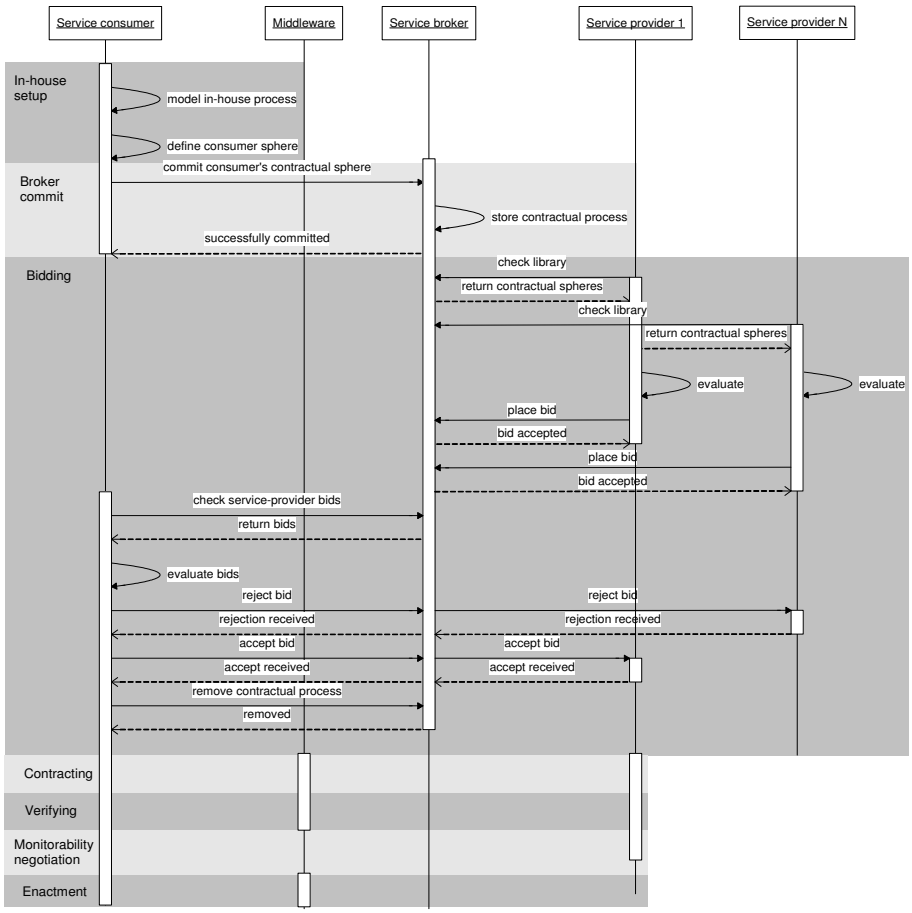

Figure 4.6: An interaction-sequence example for dynamic service assignment.

If the consumer happens to be confronted with a big number of provision bids, tool support must permit the evaluation of those bids in an efficient and effective way.
Examples:

- A travel agency carries out the booking of a trip for a customer who needs to travel to a capital city. The overall booking comprises a flight ticket, a hotel, and renting a car. However, the hotel booking is carried out by a different company that is actually located in the capital city and therefore has the appropriate expertise. Thus, the travel agency submits the part of the in-house process to a public broker as a request for service provision. Many potential service providers start bidding and the travel agency chooses the best deal and rejects all other offers.

- In Figure 4.6 a possible setup phase is depicted that uses a dynamic-assignment pattern for creating an eSourcing configuration. First, the service consumer creates an in-house process and defines a consumer sphere in it. Next, the consumer sphere is projected to a publicly available service broker that can be searched by potential service providers. Thus, interested parties place bids for service provisions that are evaluated by the service consumer. The latter party chooses a

service provider that represents the best deal and rejects all other offers. After that, interaction blocks termed contracting, verifying, monitorability negotiation, and enactment follow that contain interaction sequences similar to Figure 4.5.

Pattern 3 (Semi-Dynamic Assignment)
Problem: An organization is involved in tight supply-chain integration where parts of an overall service are sourced. While it is important that a service is sourced, it is not clear which organization can offer guaranteed collaboration.

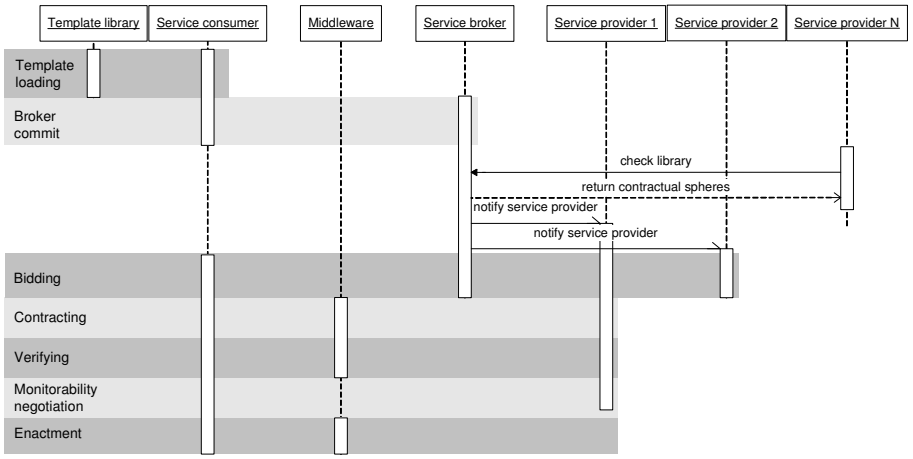

Figure 4.7: An interaction-sequence example for semi-dynamic service assignment.

Description: Before the setup time of a service exchange begins, the number of organizations that are willing to collaborate is predetermined. When the service to be sourced is externalized to a broker, only the collaboration candidates may evaluate and engage in a bidding for the service. Any other organization that is not a predetermined candidate, is excluded. The service-externalizing organization chooses the offer and rejects all other.
Forces: With respect to the tight integration of service providers, the forces of Pattern 1 apply. For using a publicly visible process broker and managing a bidding procedure, the forces of Pattern 2 are applicable. The requirement of a notification system is specific for this semi-dynamic assignment pattern. Such a notification system is activated when the template process is committed to the public process broker by the service consumer. Since the template contains the set of predetermined service providers, the broker should notify those providers so that they may engage in the bidding.
Examples:

- An insurance company evaluates claims resulting from car crashes. A part of the in-house process is concerned with evaluating the damage. However, the insurance company doesn't consider such an assessment its core business and therefore has external companies providing such assessment services. Those companies form a pool of competing service providers that are shared between a number of different insurance companies. Thus, it is never clear which company is available for performing a car-damage assessment.
- An abstract example geared towards the creation of an eSourcing configuration is given in Figure 4.7. First, a service consumer loads a template that contains

definitions referring to a set of tightly integrated service providers. After that the consumer commits its sphere to a publicly available service broker. A *service provider N* is browsing the service broker and receives several processes in return. However, processes where the predetermined service-provider candidates are defined are not returned. Instead the service broker sends out notifications to those providers defined in the process. The notified providers engage in a bidding procedure after which a contracting phase follows with the providers that were chosen by the service consumer. Afterwards the service consumer requests a verification of the preliminarily defined eSourcing configuration. If the verification concludes with an approval, the monitoring of process nodes is negotiated. Eventually the service consumer requests from the middleware to enact the ready eSourcing configuration.

After specifying assignment patterns, the following section comprises patterns deduced from the direction dimension of Figure 3.7. As in the case of the assignment dimension, each direction value is translated into one pattern specification.

4.4.2 Direction-Dimension Patterns

As Figure 3.7 shows, there are four options in the direction dimension of the interaction perspective. Out-sourcing is characterized by a service consumer which initiates the setup procedure of an eSourcing configuration by determining which part of the in-house process is published externally. In the case of in-sourcing, the service provider starts the setup phase with publishing a subset of process properties externally in a broker. For both directions of interaction it is possible that the opposing collaborating party is unknown at the beginning of the setup face of an eSourcing configuration. For the following two direction options it is assumed the collaborating parties are determined at the beginning of the setup face. When an external-to-internal interaction is chosen, both collaborating parties start with negotiating the shared, external process before adding their respective internal processes. Finally, during the internal-to-external interaction both collaborating parties have internal processes defined that need to be harmonized on an external level between the collaborating domains.

Pattern 4 (Out-Sourcing)
Problem: A company has a part of its in-house business process that is detrimental to its overall competitive advantage. Thus, the company knows that a subcontractor could carry out the process in a better way.
Description: A part of an organization's in-house process that should be carried out by a third party, is demarcated into a subprocess. Next, the subprocess is taken over by an organization that agrees with offering the service. In the domain of the service provider further refinement of the service may take place that remains opaque to the service consumer. The subprocess in the domain of the service consumer and the refined process in the domain of the provider are linked with each other and the service consumer starts with enactment of the created inter-organizational configuration.
Forces: Since the assignment patterns of Subsection 4.4.1 are variants of the out-sourcing pattern, all forces mentioned in the assignment patterns also apply for this direction pattern.
Example: All examples mentioned for the assignment patterns of Subsection 4.4.1 are variants of the out-sourcing direction pattern.

Pattern 5 (In-Sourcing)
Problem: A company has serendipitously discovered a process innovation that poses a competitive advantage. However, it is initially not clear which companies the potential purchasers of the process innovation are.

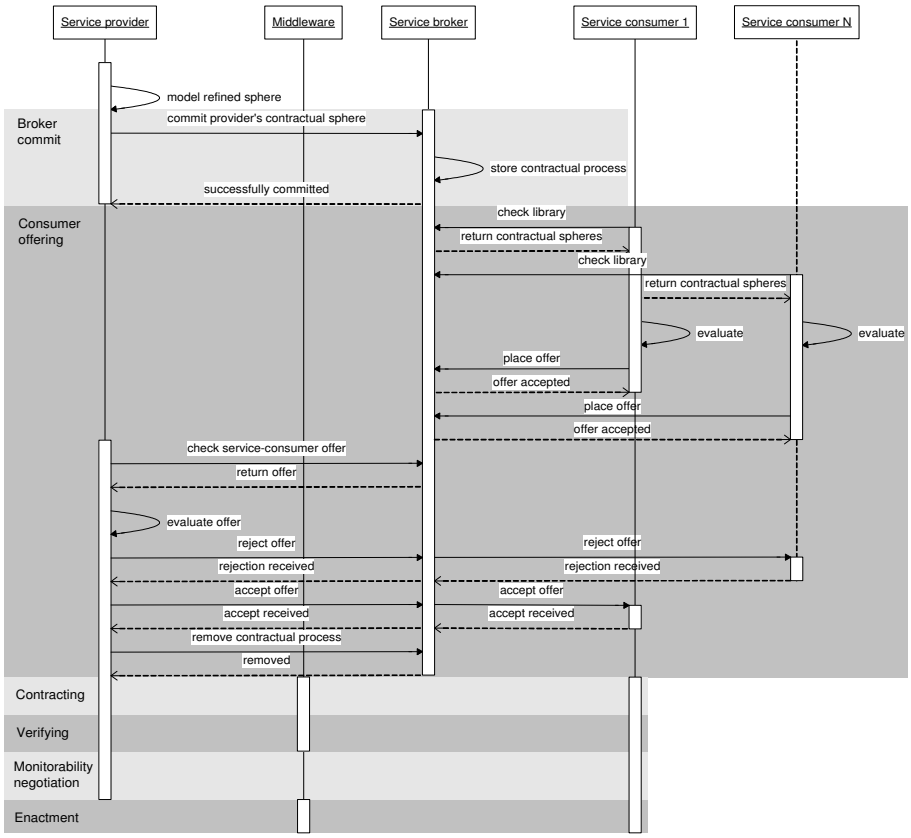

Figure 4.8: An interaction-sequence example for in-sourcing.

Description: The service provider sets up a process in its own domain and subsequently exposes a subset of the process details publicly. Compared to the exposed version, the internal process contains additional refinement steps that remain opaque. Next, an interested service consumer adopts the exposed process and integrates it in the in-house process.
Forces: The service consumer may find it impossible to integrate the exposed process of the provider. In that case extra negotiation about the exposed process content needs to unfold between the collaborating parties. The situation can occur that potential consumers are not aware of the offered service and as a result no collaboration comes into existence.
Examples:

- A firm has set up a service where databases of many airline companies are queried for last minute offers that are available for a reduced price. The search

is embedded in a bigger service where the booking and payment of the flight are automatically handled. Travel agencies that sell tailored holiday trips to end customers may integrate the flight-booking service into their booking process that also incorporates insurance purchases, bookings of hotels, cars, and so on.

- In Figure 4.8 an in-sourcing example is depicted that focusses on the eSourcing concept of Chapter 3. First, a service provider models a provider sphere internally. A subset of this provider sphere is turned into a provider's contractual sphere and committed to a publicly available service broker. Next, interested service consumers check the committed contractual process and respond with placing offers. The service provider evaluates the offers, accepts the best, and rejects all other offers from the remaining consumers. Once two collaborating parties are allocated, the remainder of Figure 4.8 is described by interaction blocks that are detailed in the assignment patterns of Section 4.4.1. Thus, the collaborating parties engage in concrete contracting, followed by a verification phase. After negotiating which process steps need to be monitored, the resulting eSourcing configuration is enacted.

Pattern 6 (External-to-Internal)
Problem: Two parties want to establish a B2B supply chain without existing constraints resulting from historically grown business processes in their own domains.
Description: The service consuming and providing organizations start with negotiating process properties on an external level. When they have reached consensus, both parties take over the publicly agreed upon process for their internal domains. In the domain of the service consumer the adopted process becomes a subnet of a bigger in-house process, while in the service provider's domain further refining process steps are inserted.
Forces: A service broker can not be employed if there is no published service offer available in the beginning. Thus, it might be a problem to find a collaborating party. Furthermore, integrating the externally agreed process might fail because of an inadequacy of internal resources, e.g., the organizational structure turns out to mismatch, no employees have the appropriate skills for carrying out certain tasks, required production resources might be lacking, and so on.
Examples:

- An IT-company intends to develop an innovative software package for a leading telecom enterprize. For this purpose personnel with state-of-the-art skills is hired. Still, a part of this new software package for the telecom enterprize can't be developed internally. Instead, a subcontractor that has been founded very recently is used for this externalized part of the overall software.

- In Figure 4.9 an example of external-to-internal sourcing is depicted. Since there is no initial contractual process available to publish in a public service broker, collaborating parties need to find each other in a different way. The example in Figure 4.9 suggests that a service consumer directly contacts a suitable provider to engage in contractual negotiation on an external level. Thus, the collaborating parties propose to each other their respective contractual spheres until a consensus is reached. Next, the contractual spheres are backed by processes in the domains of the service consumer and provider. The first party integrates the sphere in the in-house process, and the latter party extends the contractual sphere to a refined process. The following phases are depicted by interaction

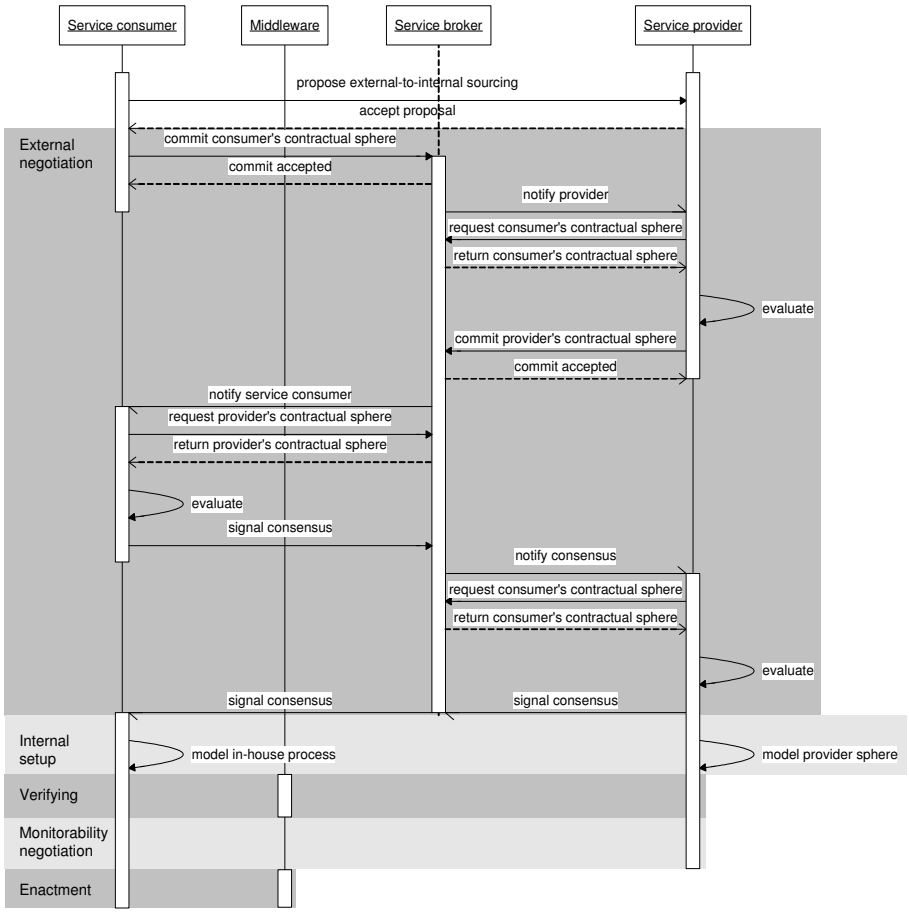

Figure 4.9: An interaction-sequence example for external-to-internal sourcing.

blocks that have been detailed in previous pattern examples. Thus, the service consumer initiates a verification phase that is followed by a negotiation phase for determining which process steps need to be monitored. Finally, the resulting eSourcing configuration is enacted.

Pattern 7 (Internal-to-External)
Problem: Collaborating parties have historically grown, stabile business processes in their respective domains. These processes need to be harmonized on an external level across organizational boundaries.
Description: Both the service consumer and provider have established business processes in their domains. The consuming organization considers a part of its in-house process to not be core business. On an external level, the consumer and provider engage in negotiating a consensus process that accommodates their already existent respective internal processes.
Forces: The internal processes of the service consumer and provider may proof to differ so extensively from each other that it is not possible to find a consensus on an external level.

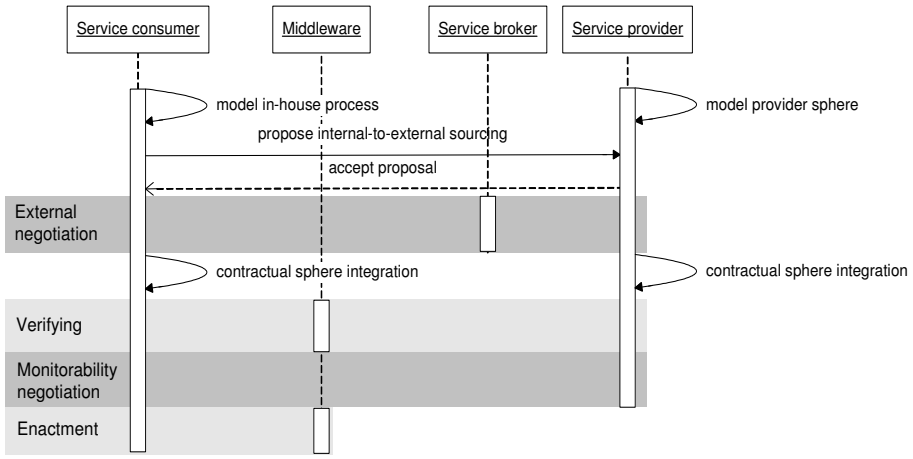

Figure 4.10: An interaction-sequence example for internal-to-external sourcing.

Examples:

- A market-share dominating OEM is a producer of high-precision track maintenance machines. The OEM feels no necessity to change the production and business processes. The suppliers of the OEM are equally not prepared to change their historically grown production and business processes. After investing in information technology and techniques the OEM and its suppliers begin to establish business collaborations electronically.
- An abstract example is depicted in Figure 4.10 as a sequence diagram. First the collaborating parties model their internal processes before they engage with each other proposing to commence with contractual negotiations on an external level. The external negotiations are depicted as an interaction block that is detailed in Figure 4.9. Once a consensus is reached, the contractual processes are integrated in the processes of the respective domains. The remaining interaction shows several interaction blocks for verification, monitorability negotiations, and the eventual enactment of the ready eSourcing configuration.

After presenting the interaction patterns for the setup phase, the next section comprises the construction-elements patterns of eSourcing configurations. These patterns are deducted in a top-down way from the dimensions depicted in Figure 3.8.

4.5 eSourcing-Construction Patterns

The first dimension of *contractual visibility* addresses the issue of keeping business secrets by disclosing different amounts of internal process details to the collaborating counterpart. Thus, construction-dimension values of Section 3.4.2 are called white box, grey box, and black box and are presented in Section 4.5.1. The *monitorability* dimension is focusing on linking equivalent nodes of the consumer sphere and the provider sphere so that their enactment is coordinated and that it is possible for one party to observe in a flexible way the enactment progress of the eSourcing counterpart. Given values in Figure 3.8 are messaging and polling from which patterns are

deducted in Section 4.5.2. Finally, the *conjoinment* dimension addresses the issue of modelling the exchange of commercially relevant information between the collaborating domains while ensuring correct termination. Such information exchange is either one-directional or bi-directional and Section 4.5.3 specifies patterns that are deducted from the given values.

4.5.1 Contractual Visibility

The contractual visibility is variable depending on the amount of process content that is projected from the consumer sphere and the provider sphere to the contractual spheres. On the one hand the consumer sphere and the contractual spheres contain identical nodes and control-flow constructs. Such an example is depicted in Figure 3.2 and Figure 3.3. On the other hand only the interfaces of the consumer sphere are projected to the contractual sphere, which grants the service provider no visibility of further process details in the consumer sphere. Finally, a third option of contractual visibility is located between the mentioned extremes where the consumer sphere's interfaces and a subset of the remaining process content is projected to the contractual sphere. Corresponding to the values on the contractual-visibility axis of Figure 3.8, the emerging patterns for contractual visibility are called *black box*, *grey box*, and *white box*.

The figures of the contractual-visibility patterns only contain two levels, namely the service provider's and consumer's conceptual level, and the external level. The reason is that the consumer's and provider's contractual spheres always need to be similar for representing a consensus. Thus, depicting both contractual spheres does not add to the specification of contractual-visibility patterns. Furthermore, in the consumer's conceptual level only the consumer sphere is depicted and the rest of the in-house process omitted as the content of the in-house process has no influence on the type of contractual-visibility pattern. Finally, when a transition in the provider sphere of the service provider is depicted with a τ label in a provider sphere, it means that an abstraction has taken place (see Definition 16 of Section 2.5). Thus, the τ-labelled transition is an additionally inserted refinement the service consumer is not aware of. During the *setup phase* the service consumer is not aware because a τ-labelled transition is not projected to the external level and during *enactment* the service consumer is not aware since the effects of enacting a τ-labelled transitions are hidden from perception of the collaborating counterpart.

There are further τ labels in the domain of the service consumer. The transitions of the in-house process that are located outside of consumer spheres experience an abstraction as they are no candidates for projection to the consumer's contractual sphere of the external level. As a result the service provider is not aware of them during the setup phase and the enactment of an eSourcing configuration. However, in the depictions of the contractual-visibility patterns, only consumer spheres are contained and the rest of the in-house process with τ-labelled transitions is omitted as it doesn't add to the pattern depictions.

Pattern 1 (Black Box)
Problem: The service consumer is interested in using service provision. However, the consumer does not mind how the provision is carried out as long as the specified exchanges are performed correctly.
Description: In the case of black-box visibility, only the interfaces of the consumer sphere are focused on. These interface places must be equally contained with identical labels in the consumer sphere and the provider sphere. It is not permitted for any

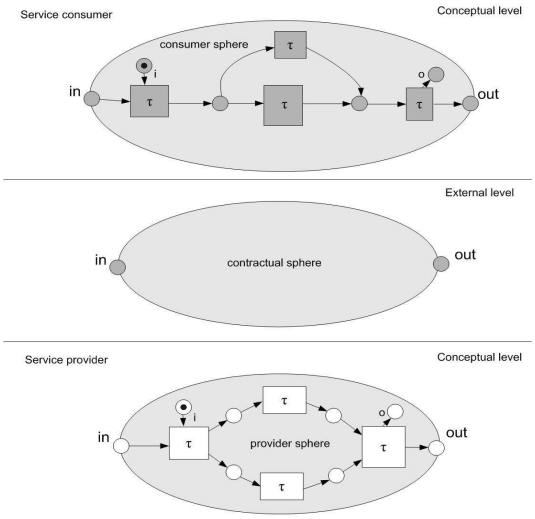

Figure 4.11: Black-box pattern example.

sphere of the external level or the respective conceptual levels to have a deviating set of interface places or differing labels. The opposing eSourcing parties are not aware of other details in the consumer sphere and the provider sphere since the contractual spheres only contain interface places with their labels. Thus, the provider sphere may otherwise completely deviate from the consumer sphere provided the similar interface places are serviced correctly according to their labels.
Forces: When only the interfaces of a contractual sphere are given, the service provider might struggle to fill the provider sphere with process content. During the enactment of an eSourcing configuration the consumer might experience service provision that is counter productive.
Examples:

- A travel agency organizes journeys abroad for arbitrary customers. Such a travel package consists of booking a trip, finding a hotel, renting a car, and so on. The travel agency does not specialize on negotiating and ordering affordable hotel bookings. As a result such a service is sourced in from service brokers where providers are registered who know the market and have special agreements with hotels in proximity. Since the concrete procedures of finding and booking most suitable hotels differs from country to country, the travel agency merely discloses the interfaces for starting and ending the hotel-booking service.

- In Figure 4.11 an abstract example of a black-box pattern is presented. The *in* and *out*-labelled interface places are fully disclosed in the contractual sphere. Consequently these interface places are also part of the provider sphere. However, it is depicted that all transitions contained in the consumer sphere and the provider sphere have a τ label, which means the eSourcing counterparts are not aware of the other's process content. Additionally, Figure 4.11 depicts that control-flow constructs in both conceptual-level spheres are different. These differences do not result in soundness problems as long as the interface places are adhered

to. The reason is that the consumer sphere and the provider sphere are WF-nets when the *in* and *out*-labelled interface places are removed.

Pattern 2 (White Box)
Problem: The service consumer demands an externalized service where the provider must strictly adhere to the requirements defined in the consumer sphere. Despite having strict service requirements imposed, the provider should still have the flexibility to adjust his service provision to internal requirements. However, these internal adjustments should remain hidden from the service consumer.

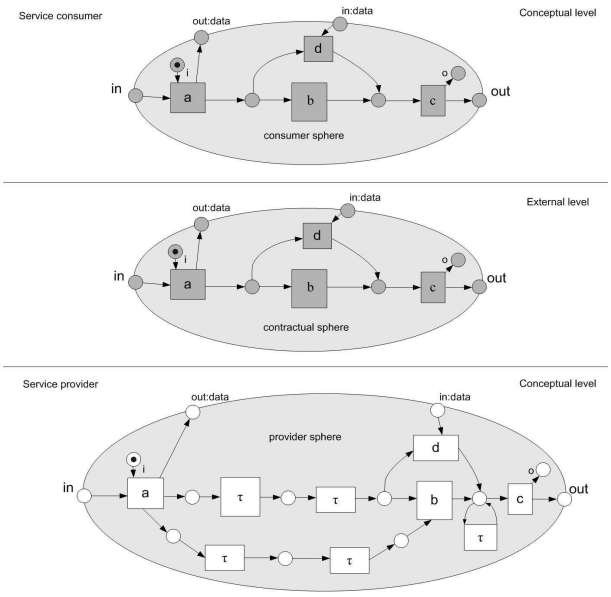

Figure 4.12: White-box and a grey-box pattern example.

Description: A consumer sphere is fully projected to the external level of an eSourcing configuration. All labels are visible to the provider as they are fully projected to the consumer's contractual sphere. As a result the nodes and labels of the contractual sphere are all present in the provider sphere. However, it is possible for the service provider to insert additional transitions in the provider sphere without violating the agreed upon service behavior.
Forces: If projection inheritance [11] is used for provider sphere, it is not trivial for a service provider to adhere as the correct application of refinement rules requires deep knowledge of modelling formalisms when no tool support is available. However, with tool support projection inheritance can be verified by replacing the consumer sphere of the in-house process with the provider sphere. Projection inheritance is given when the resulting net is a subclass of the in-house process.
Examples:

- For a so-called original equipment manufacturer (OEM) in the automobile industry it is important to reduce the production time per truck to 14 working days. In order to achieve this objective, the OEM pursues a tight integration of suppliers. Given the complexity of producing a truck within the market-dictated

time budget, the OEM demands that providers precisely mirror processes that are outsourced to avoid supply problems.

- Figure 4.12 depicts that all nodes, control-flow constructs, and labels of the consumer and contractual sphere are similar and no deviations are contained. While the provider sphere at the bottom of Figure 4.12 contains all elements of the contractual sphere, many additional elements are depicted with transitions containing a τ label. This means the service consumer is not aware of these additional transitions during build and run time.

Pattern 3 (Grey Box)
Problem: The service consumer does not mind in large parts how service provision is realized. However, the consumer wants to ensure that particular steps contained in the provided service are mandatory and carried out in certain control-flows.
Description: The pattern does not permit deviating interface places in the consumer sphere, the contractual sphere, and the provider sphere. Furthermore, a subset of nodes different from interface places contained in the consumer sphere and provider sphere is projected to the contractual sphere. The resulting consumer sphere is a connected graph.
Examples:

- An OEM in the automobile industry demands from a provider the delivery of leather coated car seats. However, the OEM insists the provider must purchase the leather from a special certified seller following an agreed upon flow of tasks.

- An example of grey-box contractual visibility is depicted in Figure 4.12. The provider sphere of the service consumer depicts interface places with labels that are identical compared to the consumer sphere and the contractual sphere. The contractual sphere shows a connected graph that contains a subset of nodes of the consumer sphere. The provider sphere is a refinement of the contractual sphere, as the inserted τ-labelled transition shows.

In the next section patterns belonging to values of the dimension called monitorability are presented. These monitorability patterns ensure different levels of enactment awareness for the service consumer.

4.5.2 Monitorability

Revisiting the initial eSourcing example of Section 3.3.1, there is no linking depicted between the contractual sphere, provider sphere, and the contractual spheres of the respective eSourcing domains. However, during enactment of an eSourcing configuration, the collaborating parties must be able to coordinate and overview in a flexible way the service provision and consumption. Thus, the monitorability patterns of this section offer ways of establishing such links between spheres in different domains.

Communication across organizational domains may take place in two ways. Accordingly, in Figure 3.8 two values are contained in the dimension called monitorability, namely polling and messaging. Based on those generalized communication concepts, patterns for eSourcing are deducted. In the case of value *polling* one eSourcing domain periodically asks if a change has taken place to a linked node of the opposing eSourcing domain. Detected changes are duplicated by the linked node in the polling eSourcing domain. The second monitorability classifier called *messaging* reverses the

signalling direction. When a linked node experiences a change, a signal is sent to the other linked node in the opposing eSourcing domain. Investigating such opposing monitorability patterns is relevant to cater for a heterogenous enactment environment that does not take into account similar monitorability options in the opposing eSourcing domains.

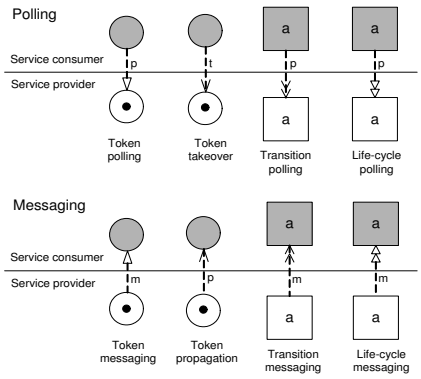

Figure 4.13: Linking options for pursuing run-time visibility.

The degree of monitorability for a service consumer during the enactment of an eSourcing configuration is determined by the level of contractual visibility (see Section 4.5.1). Only nodes from the consumer sphere and the provider sphere of an eSourcing configuration that are projected to the respective contractual spheres on the external level are considered for monitorability. Figure 4.13 depicts different messaging and polling patterns for linking such passive and transitions without claiming completeness. However, before messaging-monitorability patterns are specified in the following subsection, an investigation of issues related to the monitorability of equally labelled transitions with a lifecycle is carried out

Introduction of Life-Cycle Monitorablity

For applying patterns of life-cycle monitorability, it is necessary to explore whether it is possible to achieve a mapping of life-cycle stages that equally labelled transitions use in opposing eSourcing domains. For presenting an analysis of the problem in this subsection, a life cycle of a transition is refined to a labelled WF-net when an isolated transition has only one input place and one output place. As Figure 4.14 shows, there is only one life-cycle transition serving as an output node of that transition's input place and only one life-cycle transition serving as an input node of the transition's passive output node. Those life-cycle transitions propel a transition's life-cycle states, which are represented by labelled places.

If a transition in a process has multiple input places, they are at the same time multiple input places of the first labelled life-cycle transition that is starting to propel the life cycle. Equally, when the transition with a life cycle has multiple output places, they are at the same time the output places of the last unique, labelled life-cycle transition. In Figure 4.14 the respective transitions' life cycles are WF-nets as both have one input place and one output place. The transitions are depicted by grey shaded boxes. All other nodes are connected, i.e., they have at least one input and one output arc. In both

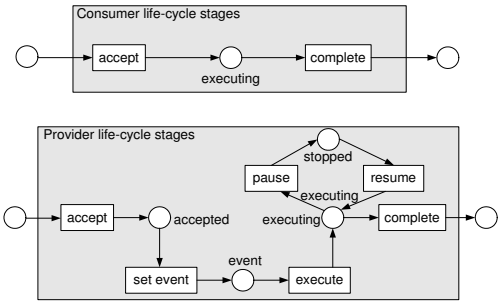

Figure 4.14: Mapping of life-cycle stages.

cases the life-cycle nets terminate after enactment so that only one token is left in their output places. Furthermore, Figure 4.14 contains a loop for pausing and resuming a transition that is in the life-cycle stage *executing*. The transition contains an additional life-cycle transition and state for event setting, which is not present in the consumer's life cycle. When a transition is accepted by a provider, this may serve as an event that other transitions might need to wait for before they can be enabled.

Revisiting the notion of projection inheritance in Section 2.5, the life-cycle of the provider's isolated transition in Figure 4.14 is a subclass of the consumer's transition life cycle. Thus, in this case, when all transitions in a provider sphere and an in-house process are replaced with life-cycle transitions and life-cycle places then an in-house process containing a provider sphere instead of the consumer sphere is still a sound WF-net. Next, the patterns depicted in Figure 4.13 are specified in detail starting with patterns for the monitorability value called *messaging*. For all specified monitorability patterns the examples are of an abstract nature and adjusted to purposeful applications in an eSourcing configuration.

Pattern 4 (Token Propagation)

Problem: During the enactment of an eSourcing configuration, tokens enter *out*-labelled interface places in the provider sphere that should trigger a message exchange with the in-house process. Since all places of the provider sphere must be empty after enactment, these tokens should be removed while the exchange between the provider sphere and the in-house process takes place. Additionally, the final token left in the *out*-labelled interface place of the provider sphere needs to be removed to complete enactment.

Description: This pattern links two equally labelled places from spheres that are not part of the same level in an eSourcing configuration. Thus, linked are two *out*-labelled interface places of the provider sphere, contractual sphere, and the consumer sphere. The propagation starts when a token arrives in the linked source place. In that case the token is passed on as a message to the linked target place in the different level. As a result the source place has that token removed and placed in the target place.

Forces: A token already resides in the *out*-labelled interface place of the consumer sphere before token propagation takes place. Thus, this token may be ahead of the provider sphere's *out*-labelled interface place and as a result it still takes more time until a token-propagation message is triggered. Such an 'early' token can't result in

enabling a transition.

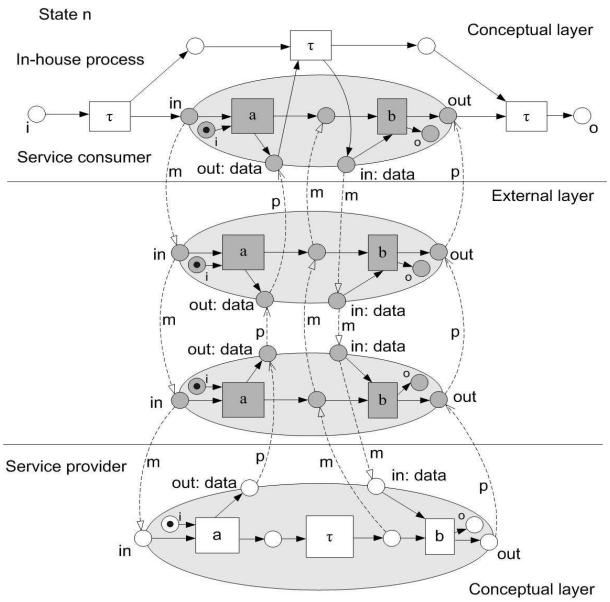

Figure 4.15: Example of token propagation and token messaging.

Examples:

- For *out*-labelled interface places, the eSourcing configuration depicted in Figure 4.15 shows an example. When a token enters the *out*-labelled interface place of the provider sphere, a propagation to the consumer sphere takes place. As a result, the enactment of the provider sphere is terminated and the rest of the service consumer's in-house process is carried out.

- The *out*-labelled interface place of the provider sphere in Figure 4.15 is connected with a token-propagation arc. When a token is placed in the interface place, a message is sent to the equally labelled interface place of the consumer sphere. As a result the token is passed on across the eSourcing domains from one interface place to the other.

Pattern 5 (Token Messaging)
Problem: During the enactment of service provision, the consumer wants to observe certain provision states. Thus, state changes that occur in the provider sphere of the provider domain should be mirrored by the consumer sphere in the opposing domain.
Description: The token-messaging mechanism is triggered when a source place experiences a change in the contained amount of tokens. A message delivers the new number of contained tokens to the target place. The target place evaluates if its contained token amount deviates from the number delivered in the received message. If the number deviates, the tokens contained in the target place are synchronized while the amount of tokens in the source place remain unchanged.

Examples:

- Figure 4.15 depicts a token-messaging example for *in*-labelled interface places. The token-messaging direction must lead from the consumer sphere to the provider sphere. The reason for this linking direction is that an active input node belonging to the service consumer's in-house process puts a token into the consumer sphere. Consequently, the enactment of the consumer sphere is started. In order to start the enactment of the provider sphere, token messaging is required to put a token into the *in*-labelled interface place of the provider sphere. After token messaging, the input places of both spheres contain a token and the enactment of the eSourcing configuration may carry on.

- To support monitorability for the service consumer, places in spheres that are not interface places can be linked in the direction from the provider domain to the consumer domain. Once the enactment of service provision is started, state changes are taking place in the provider sphere that should be monitorable in the consumer sphere. Thus, token messaging from the provider domain to the consumer results in having state changes of service provision followed for permitting consumer monitoring. In Figure 4.15 corresponding examples are depicted.

- Interface places with an *in* label have an active input node from the in-house process outside of the consumer sphere. As Figure 4.15 shows, such a token enables the active receive nodes in the consumer sphere and the provider sphere. Therefore, token messaging is applicable in such a case with a linking direction from the consumer sphere to the provider sphere.

Pattern 6 (Transition Messaging)
Problem: A transition does not contain a lower-level life cycle. An equivalently labelled transition in the consumer domain only has to be enacted when the linked transition in the provider's provider sphere has fired.
Description: While a linked source transition fires, a message is sent to the equally labelled target transition. If the target transition is enabled, i.e., has a token in all its input places, the transition fires once the message from the equally labelled source transition arrives. Consequently, the target transition produces a token for all its output places.
Examples:

- An example of transition messaging is depicted in Figure 4.16 in connection with transitions that have life-cycles. When enabled, the consumer's life-cycle transitions can fire in synch with an equally labelled life-cycle transition of a service provider. The life-cycle transitions labelled *accept* and *complete* equally occur in transitions of the service consumer and provider domain. Thus, these two nodes are suitable for linking with transition-messaging arcs directed from the service provider to the consumer.

- Active nodes without a life cycle can be linked with a transition-messaging arc. These nodes must have equal labels and be mutually known, i.e., be part of the respective contractual spheres on the external level of an eSourcing configuration. Then the linking direction is from the domain of the service provider to the domain of the consumer.

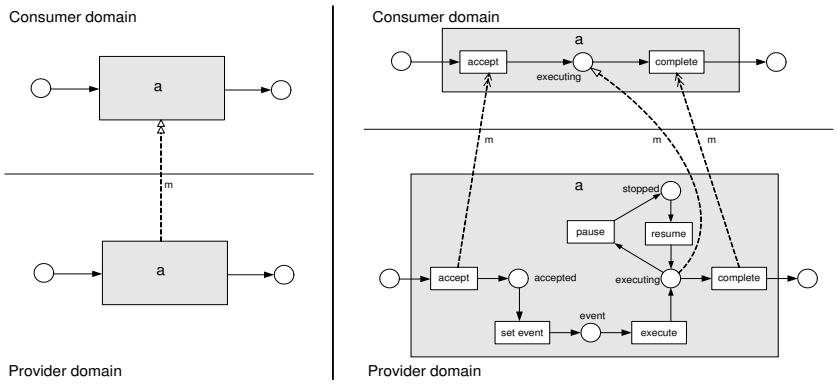

Figure 4.16: Life-cycle messaging as a black box and white box.

Pattern 7 (Life-Cycle Messaging)
Problem: During the enactment of transitions carrying labels that are equally located in the domains of the respective eSourcing parties, the life-cycle changes need to be communicated from the domain of the service provider to the consumer's domain.
Description: This pattern assumes the life-cycle of transitions that are linked for monitoring consist of lower-level nets. Consequently, life-cycle states and life-cycle transitions that are semantically matched for the respective transitions, are linked with token messaging and transition messaging arcs.
Forces: Prior semantic life-cycle mapping is required in the heterogeneous system environments of the service consumer and provider. As discussed in the beginning of Subsection 4.5.2, linked transitions might have different life cycles with steps not present in the opposing eSourcing domain or with differently positioned life-cycle steps. Some life-cycle steps may be mutually present but differently termed. Thus, a decision needs to be taken about the semantic equivalence of particular life-cycle steps.
Examples:

- In the example depicted in Figure 4.16, the life-cycle states in the consumer's and provider's transition carry the equal *executing* label. Thus, the respective life-cycle states are linked with a token-messaging arc (see Pattern 5). During the enactment of an eSourcing configuration the life-cycle transitions in the provider sphere propel the life cycle of transitions. Tokens that appear in the source life-cycle state are messaged to the equally labelled target life-cycle state in the consumer domain.

- Equally labelled life-cycle transitions are linked with a transition-messaging arc (see Pattern 6) from the direction of a service provider to a consumer. An example is given in the previous pattern.

The following patterns are focusing on different ways of *polling* changes during the enactment of an eSourcing configuration. Those polled changes help one eSourcing party to monitor and mirror the enactment progress of the opposing eSourcing party. In Figure 4.13 the monitorability patterns that use polling mechanisms are depicted at the top. These patterns are described below in further detail:

Pattern 8 (Token Takeover)
Problem: A provider sphere needs to be cleared of tokens residing in places that do not serve as input nodes to any other transitions. However, the token clearing is triggered from the domain of the service consumer.

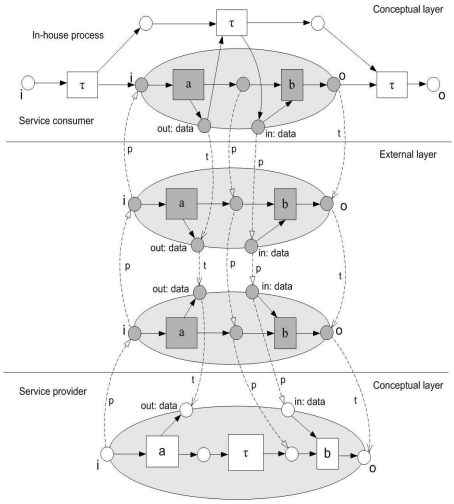

Figure 4.17: Example of token takeover and token polling.

Description: The *out*-labelled interface places are linked in the direction from a consumer sphere to the corresponding provider sphere of a provider. A request is sent periodically from the domain of a service consumer to the equally labelled place of the provider sphere to check whether a token resides there. If the response is positive, the token is removed from the provider domain and placed in the target interface place of the consumer domain.
Examples:

- The *out*-labelled interface place of the provider sphere in Figure 4.17 is connected to a token-takeover arc. When a token is placed in that interface place, token polling from the domain of the service consumer results in a positive response. As a result the token is taken over and placed in the equally labelled interface place of the consumer's domain. Consequently, the provider's *out*-labelled interface place is empty.

Pattern 9 (Token Polling)
Problem: During the enactment of an eSourcing configuration the opposing eSourcing parties want to monitor the progress of state changes. However, for economic reasons only some state changes are of relevance to an eSourcing party and are consequently mirrored according to the opposing domain.
Description: Places that belong to the domains of a service provider and consumer and that are not *out* interface places are linked with each other. Periodically the number of tokens contained in the respective places are checked. When the numbers deviate, the amount of tokens contained in the place of the polling domain is synchronized if it is less than in the target domain. Such synchronization does not affect the token amount contained in the place of the target domain.

Forces: The proper setting of the polling interval is relevant for keeping the polling domain up to date with respect to state changes in the target domain. If the polling intervals are too long, state changes of the target domain can be miss. On the other hand, if the polling intervals are very short, the performance of the applications involved in the enactment of an eSourcing configuration are stressed unnecessarily. Furthermore, it must be stated that early tokens may occur in a linked place of a consumer sphere. Thus, such an early token only enables the output transitions of the place it resides in when the procedure of token polling results in a synchronization of the amount of tokens compared to the linked place in the provider sphere.
Examples:

- The *in*-labelled interface places of Figure 4.17 are linked with a token-polling arc from the domain of the service consumer to the provider. The token is produced by an active input node belonging to the consumer's in-house process and consumed by an active receive node that is located in the provider sphere.

- Places that are not interface places are linked with token-polling arcs in the direction from service consumer to the provider domain. In Figure 4.17 the place between the *a* and *b*-labelled transitions of the provider sphere is polled for tokens. The linked place in the consumer sphere synchronizes its token number.

Pattern 10 (Transition Polling)
Problem: A service consumer wants to monitor when an enabled transition without a life cycle may fire in its domain.
Description: Two transition nodes with equal labels in opposing eSourcing domains are linked in the direction from service consumer to provider. When the consumer's transition is enabled, periodic polling takes place to check if the provider's linked transition has fired. If the response is positive, the consumer transition fires and the polling is stopped.
Forces: The forces exserted on this pattern are comparable to the forces of Pattern 9 during the enactment of an eSourcing configuration with respect to the alignment promptness of a source transition compared to the polled target transition.
Examples:

- An example of transition polling is depicted in Figure 4.18 in connection with transitions that have life-cycles. Both life-cycle transitions labelled *accept* and *complete* occur in transitions of the service consumer and provider domain. The equally labelled life-cycle transition of the service provider is periodically polled. If the response contains the message that firing has been carried out, the consumer transition fires, which results in a change of the life-cycle state.

- Transitions without life cycles that are equally labelled in the domains of the service consumer and provider may be linked with transition-polling arcs. When the consumer's transition is enabled, polling of the target transition in the provider's domain starts to monitor its firing. This is periodically repeated until the response states firing occurred. As a result the consumer transition also fires to follow the provider domain and polling is stopped.

Pattern 11 (Life-Cycle Polling)
Problem: During enactment of an eSourcing configuration the life cycles of transitions in the consumer sphere need to be synchronized with the life-cycles of equally labelled

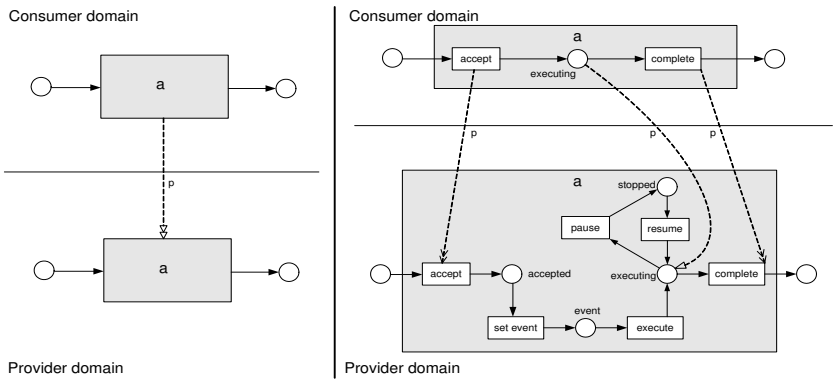

Figure 4.18: Life-cycle polling as a black box and white box.

transitions in the provider's domain. However, the initiative for triggering alignments during service enactment should occur in the consumer domain.

Description: This pattern assumes the life-cycles of transitions consists of a lower-level net (see Subsection 4.5.2). It is a prerequisite that life-cycle states and life-cycle transitions need to be semantically matched across eSourcing domains. Consequently they are linked in the direction from the service consumer to the provider domain with token-polling and transition-polling arcs.

Forces: For this pattern the forces are comparable to those mentioned in Pattern 7 about life-cycle messaging. Additionally, the polling intervals must be set appropriately. The consumer sphere runs the danger of falling behind during enactment when polling is not performed promptly. However, too frequent polling is a burden for the efficiency of applications that are used during the enactment of an eSourcing configuration.

Examples:

- In the example depicted in Figure 4.18 the life-cycle states labelled *executing* in the consumer's and provider's transition are linked with a token-polling arc (see Pattern 9). During enactment of an eSourcing configuration the provider's transition is polled for having reached the life-cycle stage *executing*. If the response results in having a lower number of tokens contained in the respective life-cycle state, the number of tokens is aligned in the domain of the service consumer. Consequently, the tokens in the provider's life-cycle state remain unchanged.

- Equally labelled life-cycle transitions are linked with a transition-polling arc from the direction of a service consumer to a provider domain. An example is given in Pattern 10.

Next, the eSourcing-patterns space in Figure 3.8 contains another eSourcing dimension that is covered in the following section, namely conjoinment.

4.5.3 Conjoinment

Conjoinment is focusing on the way a service provider and consumer use exchange channels. In the example of Figure 3.3.1, the provider sphere, respective contractual spheres, and the consumer sphere contain *in*- and *out*-labelled interface places through

which such conjoinment between the two eSourcing domains takes place. These interface places are connected to dedicated transitions that trigger either the sending or the receiving of business information across organizational domains. These conjoinment transitions carry visible labels in the consumer sphere, the provider sphere, and the respective contractual spheres. Thus, conjoinment transitions are part of the the negotiations between service consumers and providers during the setup phase of an eSourcing configuration.

In Figure 4.19 an extra notation for transitions is presented that is used for further discussing the conjoinment dimension. By using this notation, particular features of conjoinment patterns become apparent. The transitions of Figure 4.19 either receive a message, send a message, or receive and send a message consecutively. Such nodes can either be transitions that fire immediately when enabled or they contain more elaborate life cycles (see Subsection 4.5.2).

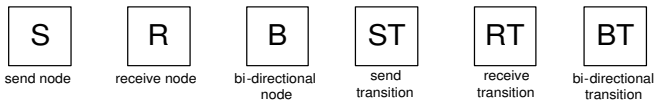

Figure 4.19: Active conjoinment node notation.

In order to make those distinctions visible, additional labels are depicted in Figure 4.19 that are carried by the shown nodes. The labels *S*, *R*, *B* are for transitions with lower level life cycles where the first is sending or initiating an exchange, the second receives or accepts an exchange, and the latter is bidirectional, i.e., the node needs to accept an exchange before it is enabled to initiate a counter exchange after firing. The labels *ST*, *RT*, *BT* are for transitions that are transitions without any contained life cycle where the first is for an exchange-initiating transition, the second an exchange-accepting transition, and the latter node is a bi-directional transition that needs to accept an exchange before an exchange is initiated in response.

In accordance with the values depicted in Figure 3.8 on the conjoinment dimension, the first two patterns described below are part of the conjoinment value *one-directional* and the latter two patterns are part of the value *bi-directional*. The reason for having two patterns deducted from each conjoinment value is that an information exchange can either be initiated from the domain of the service consumer or from the domain of the service provider.

Every pattern specification includes a visualization that consists of two parts. The top shows the conceptual-level in the consumer domain consisting of an in-house process with a contained consumer sphere. For sake of brevity a depiction of the external level and the provider's conceptual level are omitted. If the labels of conjoinment nodes in the top depictions are in brackets, then the service provider is not aware of them as they are part of the in-house process. The same is true for transitions that carry a τ-label. The bottom of the pattern figures show pattern variations that employing different types of conjoinment nodes from Figure 4.19. The specified conjoinment patterns do not contain a description of related forces as they are similar. Instead the forces are given towards the end of this section.

Pattern 12 (Provider-Initiated One-Directional)
Problem: During build time of an eSourcing configuration a construct is required that allows the service consumer and provider to reach consensus on an exchange of information directed from the provider domain to the consumer domain. The created exchange should permit the checking of correct termination.

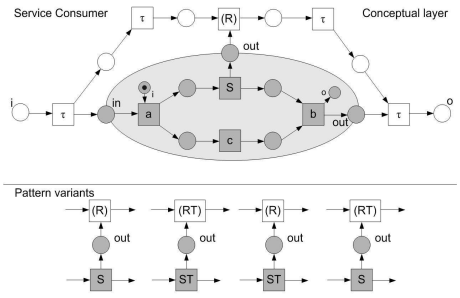

Figure 4.20: Provider-initiated one-directional conjoinment.

Description: All spheres of an eSourcing configuration have a port through which information leaves towards the domain of the service consumer. This port has an input node for triggering the sending of information. In the domain of the service consumer an output node of the port receives the delivered information. This receiving conjoinment node is part of an in-house process outside of the consumer sphere.
Examples:

- A service provider in the automobile industry agrees to supply an engine to an original manufacturer (OEM) of trucks. The OEM demands process mirroring for achieving a tight integration with the provider. Whenever the production of an engine is completed, the OEM demands to have the resulting data from the quality checks delivered for evaluation and controlling purposes.

- An abstract example is depicted at the top of Figure 4.20 where a conceptual-level process in the consumer domain is depicted. The consumer sphere contains a sending conjoinment node connected to an *out*-labelled interface place. These two nodes are replicated in all other spheres that are part of the same eSourcing configuration if they are projected to the consumer's contractual sphere. The in-house process of Figure 4.20 has an *R*-labelled conjoinment node that uses the interface place as an input node. Since the enactment of the sending conjoinment node takes place in the domain of the service provider, an exchange from provider to consumer takes place via the external level.

Pattern 13 (Consumer-Initiated One-Directional)
Problem: The eSourcing parties need to reach consensus on an information exchange directed from the consumer domain to the provider domain. Such an exchange should not endanger the correct termination of the overall eSourcing configuration.
Description: In the domain of the service consumer a sending conjoinment node that initiates information exchange to the domain of the service provider, is part of an in-house process outside of the consumer sphere. Such a transition is connected to a port through which information is injected into the spheres of the eSourcing configuration.

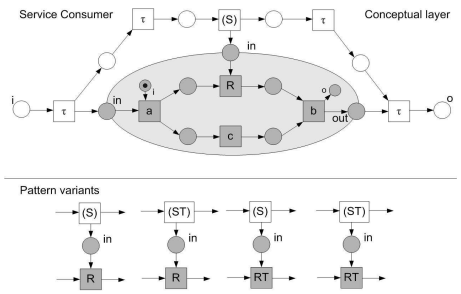

Figure 4.21: Consumer-initiated one-directional conjoinment.

The input port is followed by a node that receives this information and which is located in the consumer sphere, the respective contractual spheres, and the provider sphere.
Examples:

- In alignment with the first business example given in Pattern 12, the exchange is now directed from the OEM to the service provider who delivers engines for truck production. The produced trucks may individually vary depending on customer demand. Thus, for every engine that is produced by the provider, the adjusted engine specifications must be communicated as input to the provider. That way the overall eSourcing configuration is stabile from a process point-of view. However, it is possible to adjust the engine production in a flexibel way to requested engine variations.

- The consumer's in-house process is depicted in Figure 4.21 where it has a sending conjoinment node equipped with an *S* label in brackets and a passive output node that is an *in*-labelled interface place. This interface place in the domain of the service consumer serves as a passive input node for a receiving conjoinment node contained in the consumer sphere. In Figure 4.21 that latter node carries an *R* label. Due to well-directedness, the *in*-labelled interface place is replicated and contained in the provider sphere of the service provider. During enactment of the overall eSourcing configuration an exchange is carried out that is initiated by the consumer and accepted by the provider. If data flows along such control flow that is verified for soundness, it is likely that the exchange from the domain of the service consumer to the provider does not create problems.

Pattern 14 (Provider-Initiated Bi-Directional)
Problem: The service provider has to forward information to the consumer domain that should immediately be answered by a response. Such an exchange should not violate the correct termination of an eSourcing configuration.
Description: A sending conjoinment node that initiates information exchange to the domain of the service consumer is part of the consumer sphere, the contractual spheres, and the consumer. Such a sending conjoinment node is connected to a port at the border of the sphere through which information is exchanged to the domain of the service consumer. The port is connected to a bi-directional conjoinment node that is part of an in-house process outside of the consumer sphere. This bi-directional node receives the delivered information and responds with sending information back to the domain of the service provider through another sphere port. Finally, a receiving conjoinment

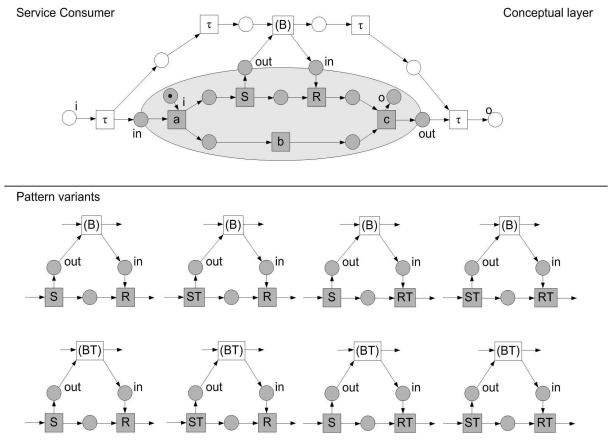

Figure 4.22: Provider-initiated bi-directional conjoinment.

node that is replicated in all spheres of an eSourcing configuration serves as a recipient for the information sent through the response port.
Examples:

- A local service provider is booking a hotel room according to special requirements of a customer. Those requirements are delivered by a travel agency that sources in the service of booking a hotel room. Since the payment format should be kept flexible, the provider must be able to ask the travel agency during enactment how the customer would like to pay for a particular hotel room. The response from the travel agency should be received immediately by the service provider in order to finalize the hotel booking.

- In the top of Figure 4.22 the depicted consumer sphere contains a sending and receiving conjoinment node in the sphere. The first node carries an *S* label and the latter and *R* label. The sending conjoinment node has a passive output node that is an *out*-labelled interface place, which serves an an input node for the bi-directional node in the in-house process. This *B*-labelled conjoinment node produces an immediate response during the exchange. For this reason one passive output node of the bi-directional conjoinment node is an *in*-labelled interface place. Consequently, that interface place is a passive input node for the *R*-labelled receive node contained in the consumer sphere.

Pattern 15 (Consumer-Initiated Bi-Directional)
Problem: The consumer needs to initiate an exchange with the service provider. However, the provider is immediately responding to the opposing eSourcing domain. Such bi-directional information exchange must not endanger the correct termination of an eSourcing configuration.
Description: A sending conjoinment node that is part of the in-house process outside of the consumer sphere, is connected to a port through which information is injected into eSourcing spheres. This port is connected to a bi-directional conjoinment node that is replicated in all spheres of to the same eSourcing configuration. The latter conjoinment node responds with information that is sent through another port back to the domain

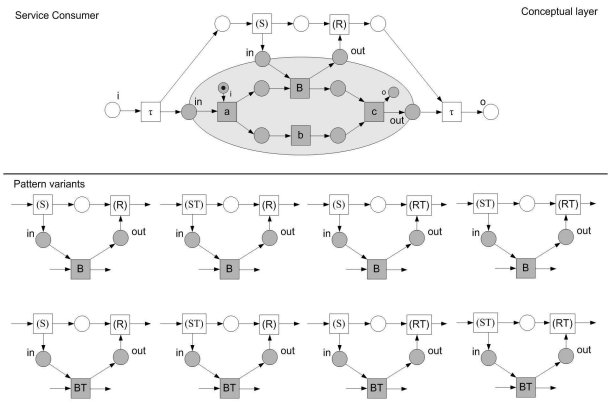

Figure 4.23: Consumer-initiated bi-directional conjoinment.

of the service consumer. Finally, a receiving conjoinment node located in the in-house process outside of the consumer sphere serves as a recipient for the information that is delivered through the second port.

Examples:

- A broker deals with a customer who wants to buy a house. While the process of finding a suitable house is on the way, in parallel a service provider deals with the financial matters related to providing a mortgage. For example, the service provider first needs to evaluate whether the customer of the broker is free of debts. At some point a house is found and the broker exchanges details about the price to the service provider. Based on the evaluated credit-worthiness, the provider needs to respond immediately whether a mortgage can be granted for purchasing the house. Based on the response, the broker goes ahead to offer the house or recommends that the customer needs to look for a less expensive house.

- The top of Figure 4.23 depicts an in-house process that contains a sending and receiving conjoinment node. The first node carries a *S* label and the latter node a *R* label. The sending node has a passive output node that is an *in*-labelled interface place, which denotes an information exchange into a consumer sphere takes place. This interface place serves as a passive input node for the bi-directional conjoinment node in the depicted consumer sphere. A contained *B*-labelled node produces an immediate response by placing a token in an *out*-labelled interface place. The latter interface place is a passive input node for the *R*-labelled receive node contained in the in-house process. Since the *R* and *S*-labelled conjoinment nodes are positioned outside of the sphere, the provider's provider sphere does not contain equally labelled conjoinment nodes. Instead the *in* and *out*-labelled interface places are part of all spheres involved in the eSourcing configuration. Consequently, a *B* or *BT*-labelled conjoinment node is part of the provider sphere.

Finally, it needs to be stated that all four conjoinment patterns are exposed to the same forces. When a conjoinment construct is chosen, the performance characteristics of an exchange during enactment are influenced by incorporated monitorability patterns.

Depending on the conjoinment direction, the *out* and *in*-labelled interface places need to be linked with monitorability arcs. If messaging is applied, token propagation (see Pattern 4) is suitable and if polling is applied, token takeover (see Pattern 8) is appropriate. However, such a monitorability choice influences the speed of conjoinment enactment. If token takeover is used, the defined polling period potentially postpones the conjoinment until the next polling interval starts. If token propagation is chosen, the conjoinment across eSourcing domains is started immediately. However, it is possible that the *R*, *RT*, or *B*-labelled conjoinment node is not yet enabled for responding to the conjoinment exchange.

Let us assume equally labelled conjoinment nodes are part of the consumer sphere, the respective contractual spheres, and the provider sphere. If these transitions are linked for life-cycle monitorability (see Pattern 7 and Pattern 11) then conjoinment construction fails during build time if the lower level life-cycles can't be semantically matched. Furthermore, just as in the case of linking *out* and *in*-labelled interface places, the speed of enacting conjoinment nodes varies depending on the type of employed monitorability links.

4.6 Conclusion

This chapter proposes that intra and inter-organizational knowledge workers should employ patterns for dynamic inter-organizational business process management. Using patterns promises the speedy evaluation and integration of intra-organizational business processes across the domains of collaborating parties. A meta model is described for uniformly storing pattern specifications, ordering the patterns in a taxonomy, and capturing information about technology support of specific patterns. Subsequently a knowledge base can be constructed based on the pattern meta-model for supporting inter-organizational knowledge workers.

For a pattern-based exploration of eSourcing, the suitability analysis of eSourcing from Section 3.4 is used. Thus, the interaction dimensions called assignment and direction and the construction structural dimensions called contractual visibility, monitorability, and conjoinment are used for discovering patterns that are specified in this chapter.

The specified eSourcing patterns are instrumental in two ways. Firstly, the interaction patterns are input for developing a reference architecture that supports the setup- and the enactment phase of eSourcing configurations. Secondly, the construction-elements patterns are used for developing an XML-based specification language for eSourcing configurations.

Chapter 5

Verifying eSourcing Configurations

Contents

An eSourcing configuration that is constructed with the structural patterns of Chapter 4 needs to be verified before enactment for service-agreement adherence and control-flow anomalies, e.g., deadlocks and livelocks. Hence, this chapter formally defines properties of eSourcing configurations and shows that the collapsing method of Chapter 3.3.3 is applicable for ensuring the correct termination of an eSourcing configuration. The formal definitions build on the Petri-net theory of Chapter 2, in particular the pre-existing theory about inter-organizational workflow nets (IOWF-nets). The formal definitions of this chapter point out an extension scope for the verification tool Woflan that is integrated into a proposed reference architecture. The extension permits collaborating parties to independently have their conceptual-level processes verified without forcing collaborating parties to disclose their business internals to each other.

5.1 Introduction

In Chapter 3 the concept of *eSourcing* is proposed, which employs a three-level framework [51] for tackling the complexity of dynamically matching a service consuming and a service providing process. An eSourcing configuration constructed with the structural patterns of Chapter 4 needs to be verified before enactment for service-agreement adherence and control-flow anomalies, e.g., deadlocks and livelocks. These structural patterns include contractual visibility that needs to conclude in a consensus between a service consumer and a service provider.

In this chapter, the properties of eSourcing configurations are formally defined, for which the conceptual understanding about eSourcing established by the pattern-based analysis of Chapter 4 serving as input. The formal definitions build on the Petri-net theory of Chapter 2, in particular the pre-existing theory about inter-organizational workflow nets (IOWF-nets). An approach for establishing an eSourcing configuration is presented that supports the three-level framework proposed in Chapter 3 and Chapter 4. It is shown that the collapsing method of Chapter 3.3.3 can be formalized to check the correct termination of an eSourcing configuration. Furthermore, it is demonstrated that using particular types of contractual visibility suffices for establishing a contractual consensus that only requires local checks within the domains of a service consumer and a service provider, rendering a check of an entire eSourcing configuration by a trusted-third-party component superfluous.

The structure of this chapter is as follows. A running example is used that shows how the processes of a service consumer and a service provider are inter-organizationally matched while correct service termination is ensured. Each section presents different properties of an eSourcing configuration. Based on the Petri-net theory of Chapter 2, Section 5.2 starts with formally investigating the conceptual levels of the service consumer and how a partitioned in-house process can be mapped to inter-organizational workflow nets. Next, Section 5.3 formally defines the domain of a service provider and shows how service refinement is formally supported. The respective conceptual levels are part of a three-level framework [51] (see Section 3.2) for managing the business, conceptual, and technological complexity involved in eSourcing configurations. Section 5.4 defines the projection variations from the conceptual to the external levels and different types of consensus constellations between collaborating parties. In Section 5.5 the properties of eSourcing configurations are defined. Section 5.6 focusses on how eSourcing configurations can be checked for correct termination. Furthermore, the section shows that if black-box projection is not used, an established run-time inter-organizational workflow is guaranteed to be sound and to realize the in-house process without requiring any coordination among collaborating parties. Section 5.7 presents a reference architecture for supporting the methods defined in Section 5.6. Section 5.8 discusses related work and finally, Section 5.9 concludes the chapter.

5.2 Conceptual Level of the Service Consumer

Based on a three-level framework [51], Figure 5.1 shows the top conceptual level where an in-house process initially does not contain a consumer sphere. The bottom conceptual level of Figure 5.1 depicts that a consumer sphere is demarcated in the in-house process. Since this results in a discontinuity in the remainder of the in-house process, it is considered invalid. To resolve this, an extra place with the label *im* is introduced in the middle process of Figure 5.1. Details about the *im* place are presented in the sequel

of the section. The external level depicted in Figure 5.1 is explained in Section 5.4 where contractual spheres are investigated.

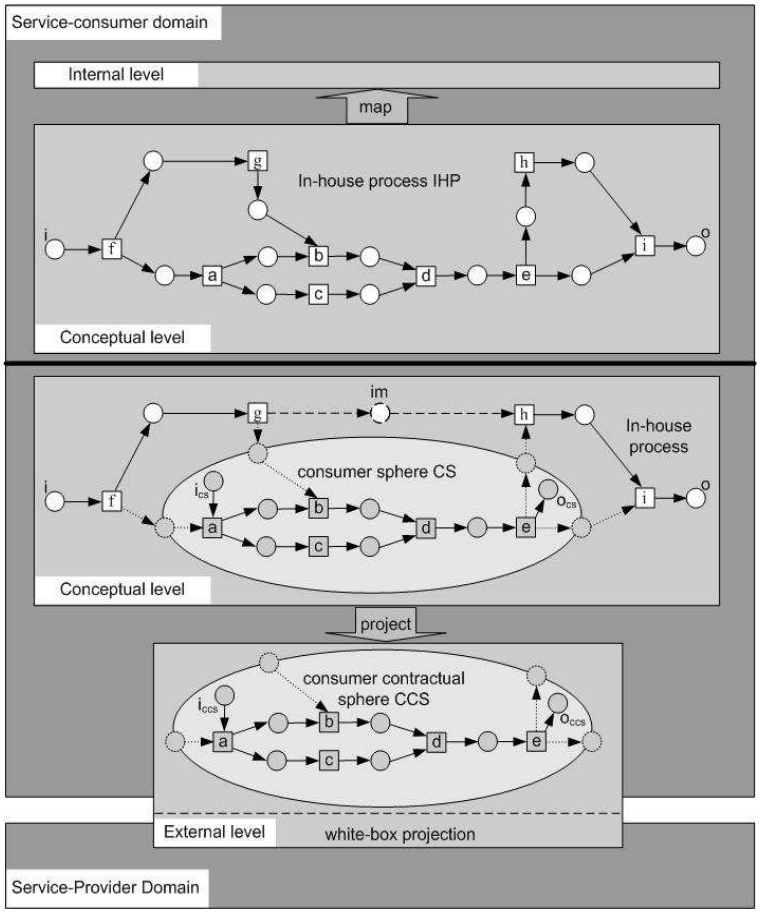

Figure 5.1: The conceptual domain of the service consumer.

Figure 5.1 shows a mapping to the internal level where legacy systems are located, e.g., intra-organizational workflow management systems, ERP systems. The projection of process details to the external level is dealt with in Section 5.4. An in-house process is defined as follows.

Definition 19 (In-house process). *An in-house process is a sound WF-net, i.e.,* $N \in \mathcal{W}$.

See Section 2.3.1 for the definition of soundness. In conformance with the connectedness requirement for WF-nets (see Definition 3 in Section 2.2), the *in-house process* of Figure 5.1 has one unique input and one unique output place. A token that is placed in *i* instantiates one particular in-house process.

The interface places depicted in Figure 5.1 located on the borders of the respective spheres are provisions to enable a systematic exchange of business relevant information between the consumer sphere and the rest of the in-house process. The interface

places are part of what Section 3.4.3 and Section 4.5.3 describe with the specifically introduced eSourcing term *conjoinment*. To denote that the interface places and theirs connected arcs are separated from the consumer sphere in a valid partitioned in-house process, they are depicted with dotted lines. Furthermore, to support the clarity of the formalism in this chapter, the interface places and connected arcs in depicted contractual spheres and provider spheres are also lined in a dotted way.

We formalize eSourcing configurations using the theory of IOWF-nets. In Section 2.4, an IOWF-net has been defined as a set of sound WF-nets connected by channel-flow relations. The following definition, taken from [12], shows that a sound WF-net can be partitioned into an IOWF-net where it is important that such a partitioning is valid.

Definition 20 (Valid partitioning). *Let N be a sound WF-net and Q be an IOWF-net. Q is a valid partitioning of N if and only if Q is sound and N = β(flat(Q)).*

See Section 2.4.1 for the definition of soundness for IOWF-nets. Using Definition 20, we can now define the notion of a partitioned in-house process. An example of such a partition is shown at the bottom of Fig 5.1. A partitioned in-house process is defined as follows.

Definition 21 (Partitioned in-house process, consumer sphere, internal process). *A partitioned in-house process PIHP is a tuple (IHP, I, CS, IP, L, G) where:*

1. *IHP is an in-house process in $\mathcal{W}$;*
2. *I is a set of interface places;*
3. *CS = (P_{CS}, T_{CS}, L_{CS}, F_{CS}, ℓ_{CS}) is a WF-net, called a consumer sphere, that is sound, i.e., CS $\in \mathcal{W}$;*
4. *IP = (P_{IP}, T_{IP}, L_{IP}, F_{IP}, ℓ_{IP}) is a WF-net, called an internal process, that is sound, i.e., IP $\in \mathcal{W}$;*
5. *Instance creation: P_{IP} contains an input (source) place $i \in P_{IP}$ such that $\bullet i = \emptyset$;*
6. *Instance completion: P_{IP} contains an output (sink) place $o \in P_{IP}$ such that $o\bullet = \emptyset$;*
7. *L = ($L_{CS} \cup L_{IP}$) is the set of transition labels;*
8. *G $\subseteq$ (I $\times$ L) $\cup$ (L $\times$ I);*

such that Q = (I, 2, IP, CS, L, G) is an IOWF-net and Q is a valid partitioning of IHP.

Note that channels formalize interface places and the channel flow relation formalizes conjoinment (see Section 4.5.3). From Definition 20, it follows that $I \subseteq P_{IHP}$. The partitioned in-house process focuses on the collaboration of one service consumer with one service provider. However, the collaboration is extensible to one service consumer with several service providers by repeatedly partitioning the internal process.

Next, the mapping of a valid partitioned in-house process to an IOWF-net is explained. Figure 5.2 depicts the IOWF-net *Q* that corresponds to the partitioned in-house process at the bottom of Figure 5.1, using the notation of [12]. The grey-filled boxes are labels in *L* to which the transition labels of the consumer sphere and internal process are assigned. Between the labels the channel places in *I* are depicted that connect

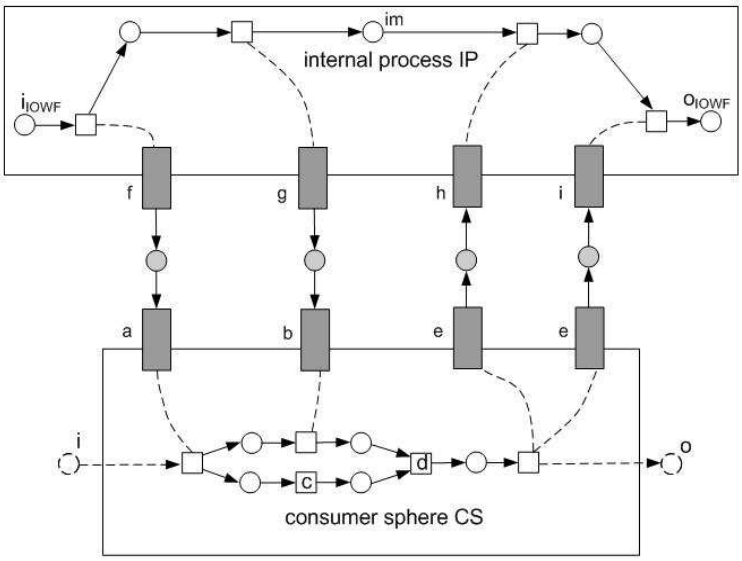

Figure 5.2: The result of mapping from a partitioned in-house process to a IOWF-net.

the two workflow nets. The created channel flows in *G* have the same direction as the conjoinment directions of the running eSourcing example of Figure 5.1.

For a valid partitioning, the internal process, i.e., the remainder of the in-house process without the consumer sphere must again be a sound WF-net. Figure 5.1 shows an in-house process *IHP* and a consumer sphere *CS* for which the resulting internal process *IP* does not fulfill this criterion any more, since *IP* is not a WF net. Thus, the partitioning created by the consumer-sphere demarcation might not be valid.

To overcome this problem, the extension of the to be partitioned workflow net with implicit places has been proposed [12]. A place of a marked P/T-net is said to be implicit or redundant if and only if it does not depend on the number of tokens in the place whether any of its output transitions is enabled by some reachable marking. Note that implicit places do not change the behavior of a process they are used in.

Definition 22 (Implicit place). *Let (N, s) with N = (P, T, L, F, ℓ) be a marked, labeled P/T net. A place $p \in P$ is called implicit in (N, s) if and only if, for all s' reachable from s and transition $t \in p\bullet$, $s' \geq \bullet t \backslash \{p\} \Rightarrow s' \geq \bullet t$.*

In Figure 5.1, adding implicit place *im* to the in-house process results in a valid partitioning of the in-house process (shown in the middle process of Figure 5.1). Thus, the internal process is a sound WF-net. Implicit places and their properties have been studied in [30, 39]. Further details about implicit places and their use in IOWF-nets are contained in [12]. Adding the *im*-labeled place in Figure 5.1 yields a P/T-net which is branching bisimilar to the original net. Next, the conceptual level of the service provider is defined.

5.3 Conceptual Level of the Service Provider

A minimal requirement for a provider sphere is that it sound. A provider sphere is defined as follows.

Definition 23 (Provider sphere). *A provider sphere N is a sound WF-net, i.e., $N \in \mathcal{W}$.*

In an eSourcing configuration, a service provider should not have fixed, standardized routing imposed by the service consumer. Instead, a degree of flexibility is important for internal service refinement in the domain of the service provider to allow the integration of back-office tasks that remain opaque to the service consumer for various reasons, e.g., they constitute a competitive advantage that should not be revealed, the service consumer is not interested in them, and so on.

Figure 5.3 illustrates how a service provider may perform internal refinement. On top of the figure, the service provider contractual sphere is depicted that has the same content compared to the consumer contractual sphere that Figure 5.1 contains. Further details about the external level depicted in Figure 5.3 follow in Section 5.4 where contractual spheres are investigated.

The provider sphere of Figure 5.3 contains additional labels compared to the contractual sphere. However, during enactment the service consumer only perceives process behavior that is part of the external level and not what constitutes the service provider's refinement. To realize such a refinement scenario as depicted in Figure 5.3, *projection inheritance* [14, 27] is employed. In Section 2.5, projection inheritance (see Definition 18) is explained together with the related notions of branching bisimilarity [48] and behavioral equivalence (see Definition 17).

In Figure 5.3 an example is depicted for showing projection inheritance (see Section 2.5). The provider sphere PS_1 is a subclass of *PCS* according to projection inheritance because hiding the inserted transitions does not violate the behavior equivalence the service consumer expects. Firstly, neither hiding the parallel branch with w nor hiding the execution of the inserted x violates the original behavior of *PCS*. Secondly, the same holds for hiding the execution of y that merely postpones the execution of e. In [27] details are contained about the projection-inheritance preserving refinement patterns in PS_1 of using parallel branches, inserted transitions, and loops. Mapping the provider sphere to the internal level of the service provider can be done in a similar way.

The bottom PS_2 sphere of Figure 5.3 shows a violation of projection inheritance in correlation to *PCS* because hiding the inserted newly labeled transitions results in a potential trace where a is followed by d without executing c. The following section demonstrates how collaborating parties of an eSourcing configuration can project their respective spheres to the external level.

5.4 External-Level Properties

The external level of an eSourcing configuration determines how much internal process details are exposed (see Section 4.4). The collaborating parties have the option of projecting different amounts of conceptual-level process content into their respective contractual spheres. To achieve a consensus about the nature of service provision, the respective contractual spheres must match in content. The structure of this section is as follows. First, the properties of a contractual sphere are described in Section 5.4.1,

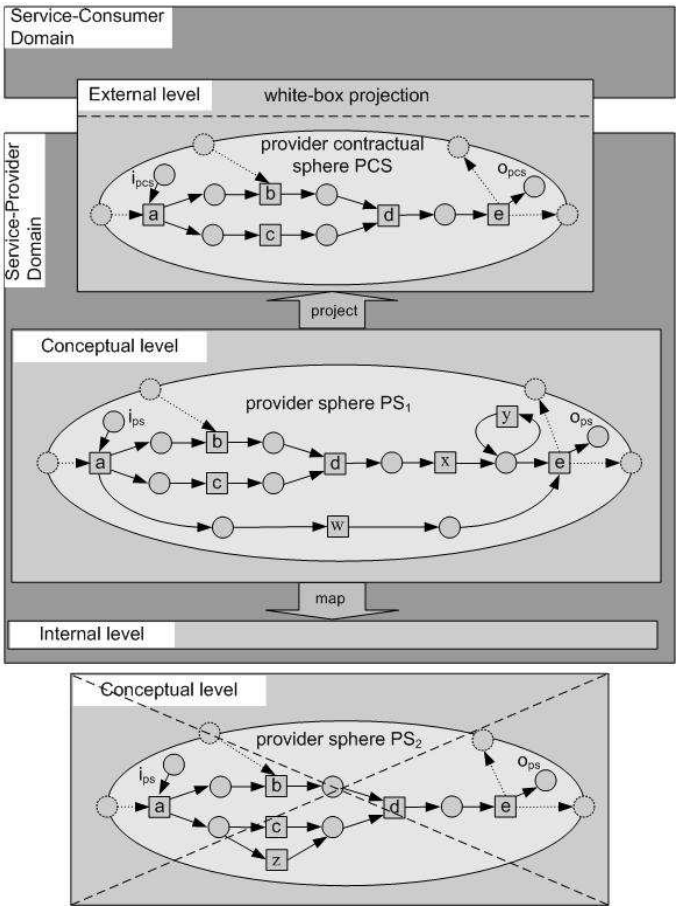

Figure 5.3: The provider contractual sphere and two provider spheres. The bottom one is illegal.

followed by Section 5.4.2 that presents different types of projection options to the external level that collaborating parties can use. Finally, Section 5.4.3 defines when a contractual consensus is achieved in an eSourcing configuration.

5.4.1 Properties of Contractual Sphere

Both the service consumer and the service provider have their own contractual spheres that belong to the external level of an eSourcing configuration.

Definition 24 (Contractual sphere). *A contractual sphere is a labeled P/T-net (P, T, L, F, ℓ) such that $\tau \notin rng(\ell)$. The set of contractual spheres is denoted as $\mathcal{C}$.*

Note that a contractual sphere *CS* does not need to be a WF-net. This is to allow black-box projection. Using τ-labels in the contractual spheres does not make sense from a business point of view as it would result in an eSourcing of invisible actions. Examples of contractual spheres are depicted in Figure 5.1 and Figure 5.3. The figures show there are two contractual spheres, one for the service consumer and for the service

provider, located on the external level of an eSourcing configuration. Having separate contractual spheres for the respective collaborating parties facilitates negotiations until a consensus is reached.

5.4.2 External-Level Projections

The three projection variations called black-box, white-box, grey-box are depicted in several figures of this chapter. The projection options result from the contractual visibilities that Section 4.5.1 describes. As Figure 5.1 shows, in the case of a white-box projection, the consumer sphere is fully projected into the contractual sphere on the external level. The black-box projection example of Figure 5.4 depicts a contractual sphere where only the channels are exposed, which grants the service provider no visibility of process details in the consumer sphere.

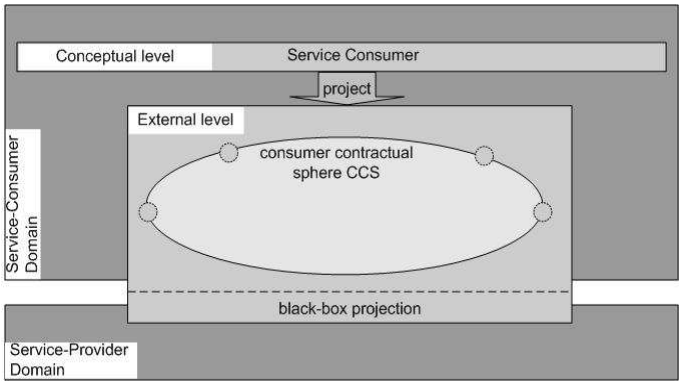

Figure 5.4: A black-box projection to the external level.

For black-box projection the issue arises how a proper conjoinment of the contractual spheres of a service consumer and a service provider can be achieved. The problem occurs because for the formalization of eSourcing the labelling of interface places is omitted. To solve this problem, during the setup phase of an eSourcing configuration, a service consumer can inform the service provider about the conjoinment labels of the channel flows in the consumer contractual sphere.

Black-box projection offers increased flexibility for external-level business process harmonization with the trade off that matching is difficult to achieve as the business-process internals remain opaque. To alleviate this situation, it is necessary that collaborating parties have a mechanism available to support the checking of an eSourcing configuration realized with black-box projection. In Section 5.6.1 such a method will be presented, backed in Section 5.7 by a reference architecture for a trusted-third-party verification that incorporates the Woflan tool [106] for checking the correctness of eSourcing configuration.

Finally, the third option of grey-box projection is depicted in Figure 5.3 where only a subset of the provider sphere is projected to the external level. This subsection formally defines the three projection options.

White-Box Projection

In accordance with the white-box pattern specification given in Section 4.5.1, the entire content of a consumer sphere or a provider sphere is projected to the external level. White-box projection is defined as follows:

Definition 25 (ω-projection). *Let $N_0 = (P_0, T_0, L_0, F_0, \ell_0)$ be a consumer sphere or a provider sphere and $N_1 = (P_1, T_1, L_1, F_1, \ell_1)$ be a contractual sphere in $\mathcal{C}$. There is an ω-projection from N_0 to N_1, written $N_0 \omega N_1$, if and only if $N_0 \equiv N_1$, so N_0 and N_1 are isomorphic.*

Grey-Box Projection

In Figure 5.3 an example of γ-projection is depicted. Grey-box projection is defined as follows.

Definition 26 (γ-projection). *Let $N_{PS} = (P_{PS}, T_{PS}, L_{PS}, F_{PS}, \ell_{PS})$ be a provider sphere and $N_{PCS} = (P_{PCS}, T_{PCS}, L_{PCS}, F_{PCS}, \ell_{PCS})$ a provider contractual sphere in $\mathcal{C}$. There is a γ-projection from N_{PS} to N_{PCS}, written $N_{PS} \gamma N_{PCS}$, if and only if:*

- *N_{PCS} is a sound WF-net, i.e. $N_{PCS} \in \mathcal{W}$;*
- *$N_{PS} \leq_{pj} N_{PCS}$;*
- *$\{ \ell(t) \mid t \in T_{PS} \wedge source(N_{PS}) \in \bullet t \} = \{ \ell(t) \mid t \in T_{PCS} \wedge source(N_{PCS}) \in \bullet t \}$;*
- *$\{ \ell(t) \mid t \in T_{PS} \wedge sink(N_{PS}) \in t\bullet \} = \{ \ell(t) \mid t \in T_{PCS} \wedge sink(N_{PCS}) \in t\bullet \}$.*

For N_{PS} to be a projection-inheritance subclass of N_{PCS}, N_{PCS} must be WF-net and both nets must be weakly sound (see Definition 18 of Section 2.5). Additionally, the labels of the starting transitions must be equal, and the labels of the ending transitions must be equal to support projection inheritance.

Note that γ-projection is limited to provider spheres and must not be performed with consumer spheres. The reason for this limitation is the non-adherence to projection inheritance that may occur during the enactment of an eSourcing configuration. Then a consumer sphere and a provider sphere can have partly deviating control-flow constructs, leading to a violation of projection inheritance and therefore to a lack of contractual adherence. To see why, in Figure 5.5 an example is depicted where both parties use grey-box projection. At the top of Figure 5.5, the bold lined parallel branch is not projected to the external level. The provider sphere at the bottom of Figure 5.5 depicts the bold lined refinement compared to the provider contractual sphere.

Black-Box Projection

In accordance with the black-box pattern specification of Section 4.5.1, the black-box projection does not project any content of the sphere to the external level. In Figure 5.4 an example of β-projection is depicted. The notion of β-projection is defined as follows.

Definition 27 (β-projection). *Let $N_0 = (P_0, T_0, L_0, F_0, \ell_0)$ be a consumer sphere or a provider sphere, $N_1 = (P_1, T_1, L_1, F_1, \ell_1)$ a contractual sphere in $\mathcal{C}$. There is a β-projection from N_0 to N_1, written $N_0 \beta N_1$, if and only if $N_1 = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.*

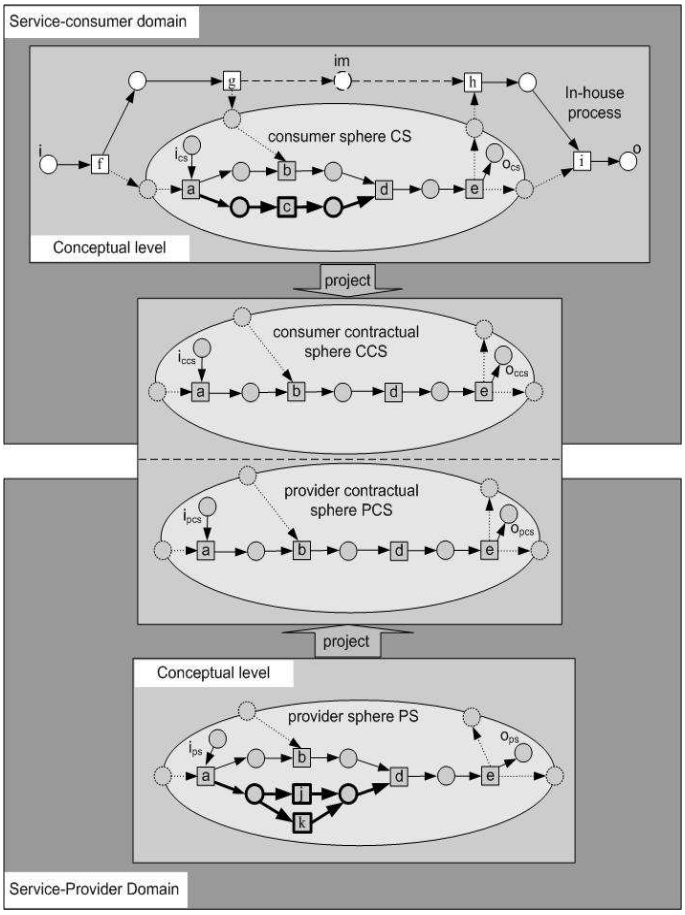

Figure 5.5: An example of both collaborating parties using grey-box projection.

If the contractual spheres result from β-projection, nothing from the conceptual level is exposed. Still, the collaborating parties have to conjoin their spheres through the interface places (channels), which is specified by the channel flow relation (G in Definition 21). This can be solved in an architectural way, by allowing the service consumer to inform the service provider of conjoinment specifics.

With β-projection it is not ensured that an eSourcing configuration is deadlock free. Instead the collaborating parties need to rely on the collapsing method that is informally introduced in Section 3.3.3 and formally presented in Section 5.6. Informally, the collapsing method replaces the consumer sphere of an in-house process with the provider sphere and the resulting net must be sound. For such a soundness check the tool Woflan (see Section 2.6) is instrumental.

5.4.3 Contractual Consensus

For contractual consensus, the contractual spheres of the collaborating parties must be isomorphic. The definition of the contractual consensus is as follows.

Definition 28 (Contractual consensus). *Let CCS be a consumer contractual sphere and PCS be a provider contractual sphere where CCS and PCS are in $\mathcal{C}$. There is a contractual consensus between CCS and PCS if and only if $CCS \equiv PCS$.*

An example of contractual consensus is given by Figure 5.1 and Figure 5.3. The consumer contractual sphere *CCS* of Figure 5.1 and the the provider contractual sphere *PCS* of Figure 5.3 are isomorph although in the first case ω-projection and in the latter case γ-projection results in the respective contractual sphere.

The different processes of a service consumer and service provider are distributed across the three-level framework [51]. The following section consolidates the content presented so far into one definition for an eSourcing configuration.

5.5 eSourcing Configurations

To realize a method for checking eSourcing configurations, it is relevant to first give a definition of an eSourcing configuration. This section presents a definition together with the accompanying properties of an eSourcing configuration. Figure 5.6 depicts a high level overview of the different parts of an eSourcing configuration and how they relate to each other.

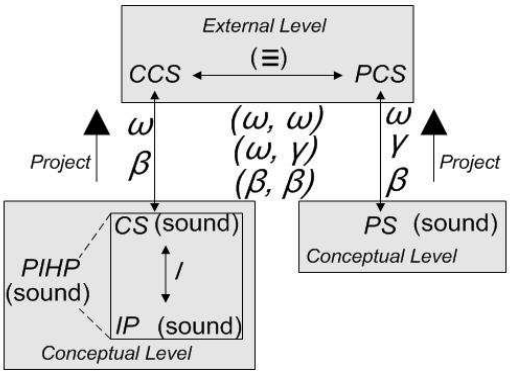

Figure 5.6: A high-level overview of an eSourcing configuration.

Shown at the left bottom of Figure 5.6, a valid partitioned in-house process *PIHP* is a net in $\mathcal{W}$ located on the conceptual level. The interface places (channels) *I* connect the consumer sphere *CS* with the internal process *IP*. On the right side of Figure 5.6 the provider sphere $PS \in \mathcal{W}$ is located on the conceptual level.

The service consumer can choose between an ω-projection and β-projection from the consumer sphere to the consumer contractual sphere *CCS* on the external level. On the other hand, the service provider can choose between an ω-projection, γ-projection, and β-projection. At the top of Figure 5.6, the external level is depicted were the contractual spheres *CCS* and *PCS* are located. To have a contractual consensus between *CCS* and *PCS*, the respective contractual spheres need to be isomorph, so $CCS \equiv PCS$. In the middle, the three projection tuples are depicted that are available for the collaborating parties to establish a contractual consensus between *CCS* and *PCS*. Either the service consumer performs a white-box projection which the service provider comple-

ments either with a white-box projection or a grey-box projection. Or both parties use a black-box projection.

Definition 29 (eSourcing configuration). *An eSourcing configuration is a tuple eSC = (PIHP, PS, CCS, PCS) such that:*

1. *PIHP = (IHP, I, CS, IP, L, G) is a partitioned in-house process;*
2. *PS is a provider sphere;*
3. *CCS is a contractual sphere in $\mathcal{C}$ called a consumer contractual sphere;*
4. *PCS is a contractual sphere in $\mathcal{C}$ called a provider contractual sphere;*
5. *there is a projection relation between CS and CCS on the one hand, and between PS and PCS on the other hand, such that either:*
   - *CSωCCS and PSωPCS, or*
   - *CSωCCS and PSγPCS, or*
   - *CSβCCS and PSβPCS;*
6. *there is contractual consensus: CCS $\equiv$ PCS.*

The definition states in which cases the partitioned in-house process, provider sphere, and the corresponding contractual spheres of the service consumer and service provider are properly related to each other. The next section introduces two methods that are instrumental for checking the correct termination and adherence of the service provider to an agreed upon service provision.

5.6 Checking eSourcing Configurations

For eSourcing configurations a verification method is used that takes advantage of the supporting theorems and their proofs that stem from the IOWF-net domain [12]. Firstly, it is shown how the collapsing method (see Section 3.3.3) for eSourcing configurations can be specified by using the flattening function defined for IOWF-nets (see Section 2.4). Such a flattening is useful for checking soundness, however, it requires that a service consumer and a service provider expose their local processes to a trusted third party. Secondly, it is shown that for certain eSourcing configurations, an established inter-organizational workflow is guaranteed to be sound. Thus, the collapsing method is not needed then to check soundness, and the consumer and provider do not need to expose their processes to some trusted third party. Furthermore, it is ensured that the inter-organizational workflow realizes the in-house process.

5.6.1 Checking Correct Termination Using Collapsing

The practical method with which an eSourcing configuration is checked for control-flow problems and correct service provision is the collapsing method that Figure 5.9 depicts as an example (see Section 3.3.3). The collapsed eSourcing configuration in Figure 5.9 omits the contractual spheres from Section 5.4. By applying the collapsing method, a run-time IOWF-net is yielded, i.e., an IOWF-net that specifies the enactment of inter-organizational collaboration between a service consumer and a service

provider. It is important to check this created process for correct termination, which can be done by flattening the IOWF-net to a WF-net and checking this net for soundness. In addition, the flattened net can be checked for adherence to agreed upon service processing by checking whether the flattened net is a subclass under projection inheritance of the in-house process.

To obtain this run-time IOWF-net, an operator is defined for replacing a consumer sphere that is contained in the valid partitioned in-house process with a provider sphere. The resulting IOWF-net connects the internal process to the provider sphere, so the IOWF-net is linking the domains of a service consumer and a service provider for collaborative enactment.

Definition 30 (CSreplacePS). *Let eSC = (PIHP, PS, CCS, PCS) be an eSourcing configuration with PIHP = (IHP, I, CS, IP, L, G), then CSreplacePS(eSC) is defined as the IOWF-net Q = (I, 2, IP, PS, L, G).*

Hence, $CSreplacePS(eSC)$ is an IOWF-net in which the provider sphere replaces the consumer sphere, i.e., the provider sphere cooperates with the consumer's internal process. Figure 5.7 illustrates the *CSreplacePS* function. The process at the top is the internal process *IP* and the bottom is the replaced provider sphere *PS*. Both processes are connected by channel-flow relations.

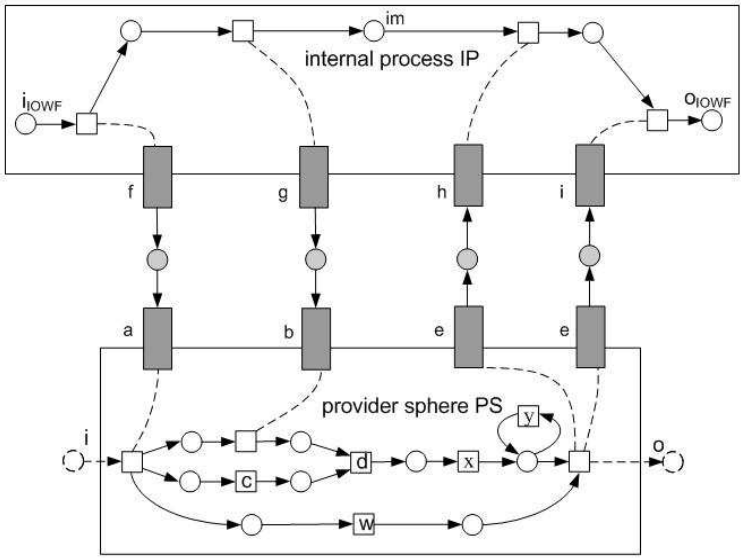

Figure 5.7: The IOWF-net underlying the partitioned in-house process of Figure 5.1, with *CS* replaced by *PS* of Figure 5.3.

Using $CSreplacePS$, we can formally define collapsing method. For the collapsing method the operator $flat$ is used (see Definition 13 of Section 2.4).

Definition 31 (Collapsed net). *Let eSC = (PIHP, CS, PS, CCS, PCS) be an eSourcing configuration. The collapsed net is flat(CSreplacePS(eSC)).*

Figure 5.8 depicts the net that results if the flattening operator *flat* is applied to the IOWF-net in Figure 5.1. At the bottom of Figure 5.9, the collapsed net for the running

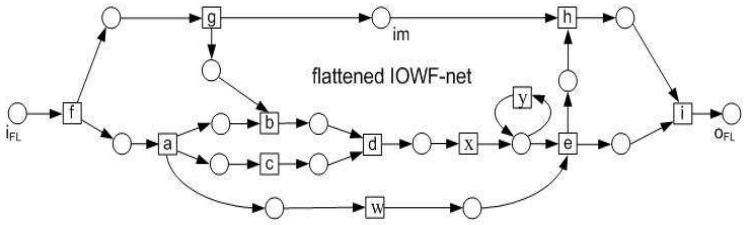

Figure 5.8: The IOWF-net underlying the flattened IOWF-net of Figure 5.7.

example is depicted, which can be verified with the tool Woflan [106] for soundness. Moreover, Woflan can check whether the collapsed net is a subclass according to projection inheritance compared to the in-house process *IHP* that is depicted in Figure 5.1. If β-projection is used, the flattened net need not be a WF-net. In that case Woflan may still be used, although the tool signals the input is not a WF-net. However, for such a soundness and adherence check, the service consumer must send the internal process *IP* and service provider must send the provider sphere *PS* to a trusted third party (see Section 5.7). The following section shows that for eSourcing configurations not using black-box projection, an established inter-organizational workflow is guaranteed to be sound.

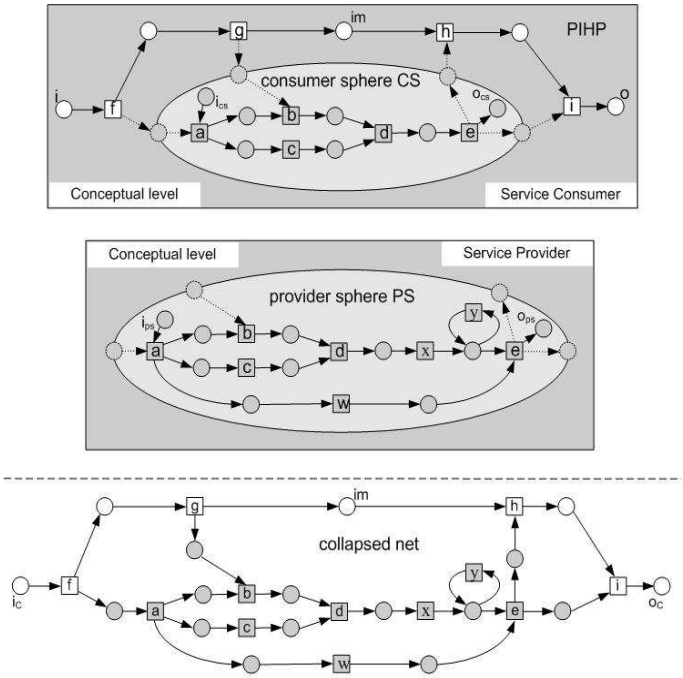

Figure 5.9: A collapsed net.

5.6.2 Checking Correct Termination using Projection Inheritance

In this section, a theorem is introduced, which states that for eSourcing configuration in which black-box projection is not used, the run-time IOWF-net is sound and the collapsed net is a subclass under projection inheritance of the in-house process. The advantage of this approach is that without the need for coordination among the collaborating parties, the resulting IOWF-net is guaranteed to be sound. Additionally, it is guaranteed that the eSourcing configuration realizes the in-house process, i.e., all the tasks specified in the in-house process are executed in the proper order.

Theorem 3 (Compositionality of eSourcing configurations). *Let eSC = (PIHP, PS, CCS, PCS) be an eSourcing configuration with PIHP = (IHP, I, CS, IP, L, G) the partitioned in-house process, CS the consumer sphere, PS the provider sphere, and IP the internal process, with a projection relation between CS and CCS on the one hand, and between PS and PCS on the other hand, such that either CSωCCS and PSωPCS, or CSωCCS and PSγPCS.*

1. *CSreplacePS(eSC) is sound, and*
2. *β(flat(CSreplacePS(eSC))) is a subclass of IHP under projection inheritance, i.e., β(flat(CSreplacePS(eSC))) $\leq_{pj}$ IHP.*

Proof. The proof consists of two parts. Firstly, a theorem is cited from [12] for the compositionality of projection inheritance in the IOWF-net domain, onto which in the second part of this proof the theorem is mapped for the compositionality of projection inheritance for eSourcing configurations.

Part A
IOWF-net Theorem: [12] Let *D = {0,1,...n-1}* be a set of n domains consisting of collaborating parties; N^{publ} is a public workflow in $\mathcal{W}$; Q^{part} = *(C, n,* N_0^{part}, N_1^{part},..., N_{n-1}^{part}, L^{part}, *G)* is the valid partitioning of N^{publ}, where N_k^{part} is a public workflow belonging to domain *k* called the public part of *k*; $Q^{overall}$ = *(C, n,* N_0^{priv}, N_1^{priv},..., N_{n-1}^{priv}, L^{priv}, *G)* is an IOWF-net, where N_k^{priv} = *(*P_k^{priv}, T_k^{priv}, F_k^{priv}, L_k^{priv}, ℓ_k^{priv}*)* is a private workflow belonging to domain *k* such that $N_k^{priv} \leq_{pj} N_k^{part}$; the labels of start and stop transitions are visible and not changed (see Definition 26 for a formalization); and $N^{overall}$ = *β(flat(*$Q^{overall}$*))* then:

1. $Q^{overall}$ is sound, and
2. $N^{overall}$ is a subclass of N^{publ} under projection inheritance, i.e., $N^{overall} \leq_{pj} N^{publ}$.

Part B
Theorem Mapping: Establishing a mapping from the first theorem to the IOWF-net theorem is straightforward:

1. N^{publ} = *IHP*,
2. Q^{part} = *(I, 2, IP, CS,* $L_{IP} \cup L_{CS}$, *G)* where L_{IP} is the label set of *IP* and L_{CS} is the label set of *CS*,
3. $Q^{overall}$ = *CSreplacePS(eSC) = (I, 2, IP, PS, L, G)*,
4. $N^{overall}$ = *β(flat(CSreplacePS(eSC)))*.

For showing the condition $N_k^{priv} \leq_{pj} N_k^{part}$, it must be proven that $PS \leq_{pj} CS$. The projection relation $CS\omega CCS$ results in $CS \equiv CCS$. Since CS is sound (Definition 21) and $CS \equiv CCS$, CCS is sound. Thus $CCS \leq_{pj} CS$. Since $PS\omega PCS$, by similar reasoning we have $PS \leq_{pj} PCS$; and $PS\gamma PCS$ implies $PS \leq_{pj} PCS$ according to Definition 26. A contractual consensus exists if and only if $PCS \equiv CCS$ (see Definition 28). Therefore, $PCS \cong CCS$, so $PCS \leq_{pj} CCS$. Since $\leq_{pj}$ is transitive [12], $PS \leq_{pj} CS$.

The condition that the labels of the start and stop transitions are visible and not changed (see Def 26 for the requirements), follows from the definitions of WF-nets (see Definition 8 in Section 2.3), isomorphism ($\equiv$) (see Definition 6 in Section 2.2) and γ-projection (see Definition 26). □

The essence of the theorem about compositionality of projection inheritance is depicted in Figure 5.10. It shows that if the provider sphere *PS* is a subclass of the consumer sphere *CS* under projection inheritance, then the flattened net without dead transitions *FN* is guaranteed to be a subclass of the in-house process *IHP*. So *FN* and *IHP* do not have to be checked explicitly for deciding whether *FN* is a subclass of *IHP* under projection inheritance.

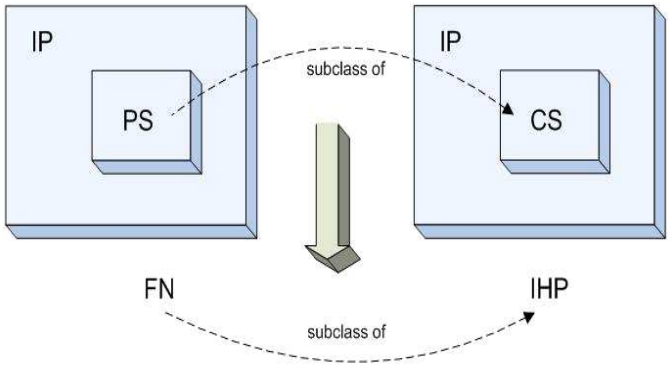

Figure 5.10: The essence of the compositionality of projection inheritance [12].

The consequence of this theorem is that it is possible to check the overall soundness of an eSourcing configuration and the adherence of internal service provision to what is publicly agreed while maintaining independent and mutually opaque process domains of a service consumer and a service provider. The soundness of the eSourcing configuration is guaranteed without the need for any coordination among the service provider and service consumer. Hence, the employment of a trusted third party by the service consumer and the service provider is only required for checking contractual consensus (see Definition 28). It is ensured that the tasks of the consumer sphere are executed in the proper order by the refined service provision.

5.7 A Verifier Component

In the previous section it is demonstrated how an eSourcing configuration can be mapped to a WF-net using an IOWF-net as intermediate notions. The soundness property and projection inheritance of WF-nets can be verified with the analysis tool Woflan

[106, 107, 109] (see Section 2.6). This way it is possible to detect modelling abnormalities, e.g., deadlocks, before workflow enactment, which helps to avoid costly run-time failures.

The approach of Section 5.6.1 can be architecturally supported by the verifier component shown in Figure 5.11. This component offers a trusted third-party service, to which collaborating parties can independently submit their conceptual processes for verification without disclosing internal business details. Note that a verification of correct termination performed by a trusted-third-party component is necessary when a service consumer and a service provider agree on using β-projection to the external level.

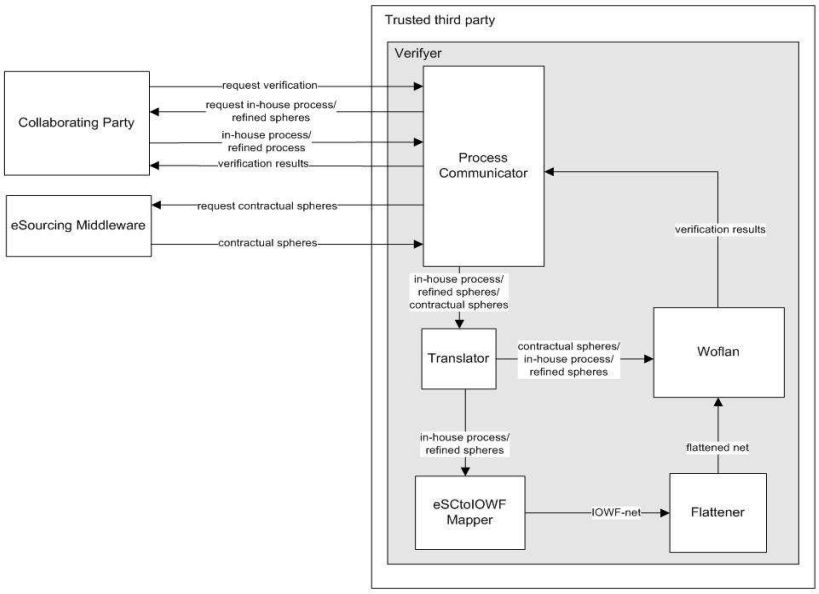

Figure 5.11: The verifier component in detail.

In Figure 5.11, a process-communicator component receives a request from the contracting client belonging to the domain of a collaborating party to perform a verification of a created eSourcing configuration. The process communicator requests the conceptual-level processes of all collaborating parties and the contractual spheres from the eSourcing middleware. Next, the collected in-house process, the provider spheres, and the contractual spheres are delivered to a translator that converts the processes into a format the eSCtoIOWF-mapper component and Woflan can process. The first component delivers the resulting IOWF-net to a flattener component that creates a net, which Woflan verifies for soundness and projection inheritance. For the latter verification type the in-house process is compared with the flattened P/T-net.

Finally, it needs to be stressed that Theorem 3 implies that the trusted-third-party component in Figure 5.11 is not necessary when a service provider and a service consumer agree on performing ω-projection and/or γ-projection. In that case, the third party only needs to check contractual consensus. All other checks can be performed locally by the collaborating parties themselves.

5.8 Related Work

In [12], IOWF-nets are explored in a P2P (Public-To-Private) setup approach that consists of three stages. In the first step, a publicly agreed WF-net is created. Secondly, the public WF-net is partitioned into domains for the collaborating parties. Finally, a private workflow is created for each domain such that the private workflow is a subclass of the corresponding part of the public workflow. Thus, by starting with a publicly agreed WF-net, the P2P approach implies an equal power constellation between collaborating business parties.

The mentioned approach of IOWF-nets suit the current way of technically composing web services. However, observing OEMs and their suppliers in CrossWork industry case studies (see Chapter 7) shows that the business needs of B2B collaboration must be matched better. Typically OEMs play a dominant role in B2B settings and try to exert tight control over their suppliers. Thus, OEMs impose a dominating client-server relationship on their suppliers that are usually tightly integrated into the OEM's in-house process. That way the OEM achieves fast production cycles, which is a competitive advantage. To support B2B collaboration in an electronic way, the client-server nature of inter-organizational business process management must be more strongly emphasized.

Workflow modules have been proposed as an extension of WF nets to model and analyse business processes that are distributed across several web services. In the analysis, a weak notion of soundness is used [66, 73]. However, checks are necessary to determine whether individual modules are compatible [7] and usable [72]. Informally, compatibility tackles the question whether two modules fit together in a way such that the composed system is deadlock-free. Usability investigated the soundness of one given workflow module without considering the actual environment it will by used in. Related is the issue of equivalence between workflow modules that addresses the question whether a module is replaceable by another one while the remaining components stay untouched.

These workflow modules [66, 73] have been succeeded by so-called open workflow nets (oWFN) [92], which have been specified and investigated for the purpose of distributing business processes across web-services. An oWFN is a P/T-net that is enriched with communication places for sending and receiving messages to another oWFN, i.e., a sphere is a special case of an oWFN. The formal definition of oWF nets differs from the definition of WF-nets, but according to [75] open WF-nets can be seen as a liberal version of WF-nets. When the communication places are removed, the inner P/T-net is considered *liberal* as it allows for many output places. Consequently, *weak termination* is realized when the final marking leaves one token in one output place and all other places are empty. For one oWFN a second oWFNS is called a *strategy* when their component-wise union via the channel places is weakly terminating. The resulting composition is again an oWFN net.

If the role of service provider or service consumer is filled by an oWFN [75, 77], a publication and discovery is feasible via a service broker that is followed by a binding. To support the discovery process, a description of the behaviors of all strategies for an oWFN P is provided in the form of reachability trees. The *most permissive strategy* is an oWFN with a *most permissive behavior* that comprises all behaviors of strategies for *P*.

The problem of controllability for oWFN [100] is whether a controller can actually use the functionality that a web service provides. To verify controllability, interaction graphs are constructed for an oWFN and its controller for the objective of investigating adherence to weak termination. The controllability investigation is extended to three

cases: an oWFN with only one port, an oWFN with two ports, and the special case where it is investigated if the controllers of ports can be constructed independently from each other. As the interaction graph for verifying controllability is huge in size, reduction rules [114] have been developed for increasing the computational feasibility of such verification.

oWFN research results might be useful to support the eSourcing setup phase if the collaborating parties use β- projection. Although this projection type allows for collaboration flexibility, the collaborating parties need to rely at the end of a setup on the collapsing method for verifying the correct termination of an eSourcing configuration. Hence, it could be desirable to equip the setup phase that involves β- projection with an additional verification technique. By constructing interaction graphs in a similar manner as for the domain of oWFN, it can be checked during the setup phase of an eSourcing configuration, whether an internal process can use the functionality that is offered by a provider sphere.

For oWFN, tool support is available for a mapping from the industry standard BPEL [40]. The tool BPEL2oWFN [70] is a compiler that translates a BPEL specified business processes into an oWFN. BPEL2oWFN can be used for several checks. With the integration of the tool Fiona [78], it is possible to check for controllability. Fiona automatically analyzes the interactional behavior of an oWFN. It provides two techniques: it checks for the controllability of the given net by computing the interactiong graph, and it calculates within BPEL2oWFN the operating guideline [76] for the net. For computing the states of the graph nodes, the model checking algorithms of the low-level Petri-net analyzer LoLa [98, 99]are used. Within BPEL2oWFN, the tool LoLA is used for validating reduction techniques for place/transition net reachability graphs. LoLA can analyze reachability of a given state or state predicate, boundedness of the net or a place, deadlocks, dead transitions, reversibility, and existence of home states.

5.9 Conclusion

This chapter formally investigates the concept of eSourcing for aligning the business processes of a service consuming and a service providing organization inter-organizationally. The preliminary Petri-net theory of Chapter 2 is adopted for defining the processes located on the conceptual and external levels of an eSourcing configuration. It is defined which conditions must hold in order to achieve contractual consensus between collaborating parties.

A practical collapsing method is given that is instrumental for verifying whether a collapsed eSourcing configuration terminates correctly. The collapsing method can also be used to verify if the provider adheres to an agreed upon service request. Alternatively, it is shown that an eSourcing configuration that does not use black-box projection is guaranteed to be sound and moreover, the collaboration of the internal process with the provider sphere then is guaranteed to realize the in-house process. The latter approach relies on a theorem from IOWF-nets about the compositionality of projection inheritance.

Finally, a reference architecture of a trusted-third party service is proposed for supporting the checking of an eSourcing configuration without forcing collaborating parties to reveal internal business secrets to each other. In this reference architecture the tool Woflan is employed for checking the soundness of an eSourcing configuration before enactment. This reference architecture is needed for supporting the collapsing method when the contractual parties are using β-projection. In this case, a check of the

eSourcing configuration by a trusted-third-party service is necessary to avoid that collaborating parties reveal business internals to each other. For eSourcing configurations that use ω-projection and γ-projection, a trusted third-party service is only needed for checking contractual consensus. Instead, checking for correct termination can be done locally by the collaborating parties themselves.

Chapter 6

Proof-of-Concepts

Contents

This chapter comprises of several proof-of-concept approaches for eSourcing, which demonstrate in short realizations that the core ideas of eSourcing are workable and feasible. Note that this design-science research oriented proof-of-concept does not explicitly pursue the objective of demonstrating the formalizations of Chapter 5, as is customary in purely mathematical research. Firstly, an eSourcing reference architecture (eSRA) is proposed that is guiding for the development of applications to support the establishment of an eSourcing configurations and the subsequent enactment. This reference architecture consists of interacting components distributed across the three-level framework (see Section 3.2) that uses the earlier specified interaction patterns

(see Section 4.4) as input. Next, the XML-based language eSML (electronic Sourcing Markup Language) is presented which allows the specification of eSourcing configurations. eSML incorporates the eSourcing construction patterns of Section 4.5 and elements for representing the resource- and data-flow perspective that are based on respective pattern catalogues. eSRA comprises components for the specification and enactment of eSML instances. Hence, the communication between eSRA components is realized by using eSML code fragments. Finally, two components of eSRA are further explored, namely a pattern knowledge base that supports actors involved in creating an eSourcing configuration, and an evaluation tool that runs the business processes contained in eSML. The pattern knowledge base is relevant as a large number of pattern specifications exist that are relevant for eSourcing. An evaluation tool is needed to ensure that the different perspectives that are specified in one business process can be enacted together without conflicts.

6.1 Introduction

In the introduction of this thesis the design-science research guideline termed *design as an artifact* (see Section 1.4.1) is adopted. Part of the realization of this guideline is a reference architecture for the development of application that support the the setup and enactment of eSourcing configurations. The reference architecture uses the earlier specified interaction patterns (see Section 4.4) as input and comprises components that are distributed across the three-levels framework adopted for eSourcing. On the external level, components are located for the collaborative setup of eSourcing configurations between service providers and consumers. To deliver business-process constructs to the external level, the conceptual level offers components for modelling and storing, verifying, and evaluating business processes constructs that are negotiated on the external level. Furthermore, the conceptual level comprises components that translate data and business-process constructs to be suitable for the components on the respective levels. Finally, the components on the internal level store data, and business-process constructs that are received after a translation from the conceptual level. The business-process constructs are enacted by components that orchestrate legacy systems. Collaborating parties only communicate via the external level with each other and within one domain the components belonging to the external and internal level can only communicate via the conceptual level with each other.

The XML-based modelling language eSML is instrumental for inter-organizationally harmonizing business processes. eSML instances that specify an eSourcing configuration are the result of a setup phase for which application systems are used that are developed in line with eSRA. An eSML instance is interpretable by components of such an application system that orchestrates legacy systems of collaborating parties. eSML specifies inter-organizationally harmonized business processes that also include data-flow and resource specifications.

To support intra- and inter-organizational knowledge workers during the setup phase of an eSourcing configuration, a pattern-knowledge base is employed that has been implemented as a prototype. Such a pattern-knowledge base is necessary as there are many pattern specifications that are relevant for eSourcing [17, 25, 95, 96]. Thus, supporting users with a tool for quickly finding the right pattern for solving a specific problem is relevant. The pattern-knowledge base supports collaborating parties with setting up eSourcing configurations by providing stored pattern specifications and data about which technologies support respective patterns. Registered users may submit

patterns for a reviewing process that results in an extension of the catalogue of stored patterns if the reviewing results in an acceptance. Such accepted patterns enter the publicly available knowledge base.

The inter-organizationally harmonized business processes that are specified in an eSML instance need to be evaluated before enactment. This is necessary to complement the verification of separate perspectives that are specified for a business process. In eSRA a component is part of the conceptual level that has been implemented as a prototype that performs such evaluations of business processes, which are separated for the domain of each collaborating party. After an accompanying verification of different perspectives such as control-flow, the enactment phase of an eSourcing configuration commences.

The structure of this chapter is as follows. In Section 6.2 the eSourcing reference architecture eSRA is presented. The architecture comprises of communicating components on all three levels of an eSourcing configuration and are further decomposed on three refinement levels. Next, Section 6.3 introduces eSML by describing the contained business-process specification constructs with class diagrams [46]. Section 6.4 describes the pattern-knowledge base to support collaborating parties in setting up an eSourcing configuration. Finally, Section 6.5 describes the evaluation tool XRL/flower.

6.2 A Reference Architecture for eSourcing

A reference architecture for supporting electronic interaction between business parties provides the major design principles and specifies the functionalities that must be delivered by such an e-collaboration system. Thus, a reference architecture serves as a starting point for software developers who are occupied with designing and implementing an information system for supporting the automated setup and enactment of business collaboration.

Software development consists of three main phases, the analysis, design, and implementation phases [71]. This section presents a conceptual (also known as a logical architecture) reference architecture for collaboration setup and enactment that facilitate the understanding of the interactions between components and the functionalities provided by the system, and are consequently a good technique for the definition of reference architectures.

The eSourcing reference architecture (eSRA) is designed in accordance with the principle of functional decomposition of a system. This decomposition is also known as ”separation operation” and based on the part-whole principle. Thus, at each refinement level of the eSRA, the identified components provide functionalities that do not overlap with the remaining components that are located at the same level. To achieve completeness, eSRA is designed in a top-down way, i.e., the components on the first level are decomposed into detailing components. The following subsections present three refinement levels of eSRA.

6.2.1 First Detail-Level of eSRA

In Figure 6.1 the highest abstraction level of the eSRA is depicted. In the figure two collaborating parties show the same set of components distributed across an external, conceptual, and internal level. The grey shaded boxes represent components and arcs between the components depict exchanges between the components.

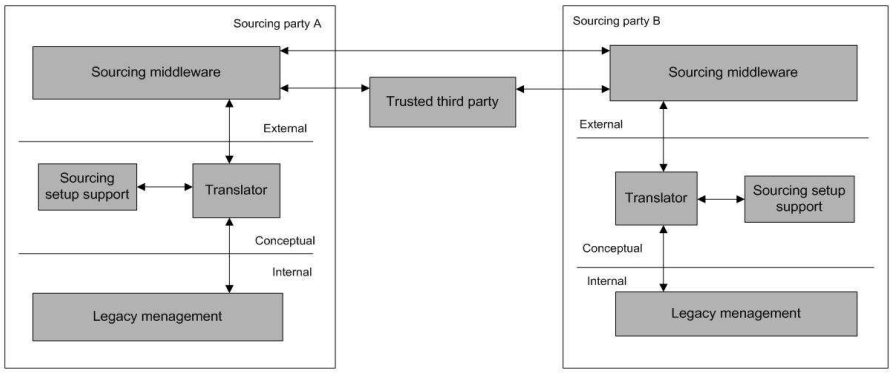

Figure 6.1: Overall Sourcing enactment architecture.

On the external level the *Sourcing middleware* is replicated on the respective external levels of collaborating parties. This component is the main enabler of interoperability and direct information exchange exists between the eSourcing middleware of each collaborating party to synchronize the respective components. Between the collaborating parties a component is located termed *trusted third party* that exchanges information with the eSourcing middleware. A trusted third party is necessary for several reasons. Firstly, collaborating parties expose service demands or service offerings to the trusted third party for public evaluation. Secondly, the trusted third party performs verification of services and checks quality features of eSourcing configurations before enactment. If collaborating parties perform verifications and checks of eSourcing configurations themselves, they would need to reveal competitive secrets to each other, which is undesirable.

The conceptual level of Figure 6.1, depicts two components, namely the translators and the eSourcing setup support. The *translator* component exchanges information between the components located on the external and internal level. The *Sourcing setup support* contains among other functionality tools for modelling business rules and processes. Finally, the internal level depicts a *legacy management* component that interfaces on the one hand with the translator component of the conceptual level and on the other hand with the legacy system of a collaborating party.

6.2.2 Second Detail-Level of eSRA

In this subsection each component of the reference architecture depicted in Figure 6.1, is further refined. The first refinement in Figure 6.2 covers all components that are located on the external level, namely the Sourcing middleware, and the trusted third party. This focus is visualized by grey shading. In contrast, the translator component is not grey shaded as it is refined in a later figure. In all figures of this subsection the refined components of focus are depicted with their exchanges to bordering components.

In Figure 6.2 the eSourcing middleware of one collaborating party is depicted. The Sourcing counterpart contains the same second level components, however, for sake of brevity only the relationship between the coordination-interface components is depicted. Furthermore, Figure 6.2 shows the trusted-third-party component as a white box and its relationship with the Sourcing-middleware component. A dashed arc be-

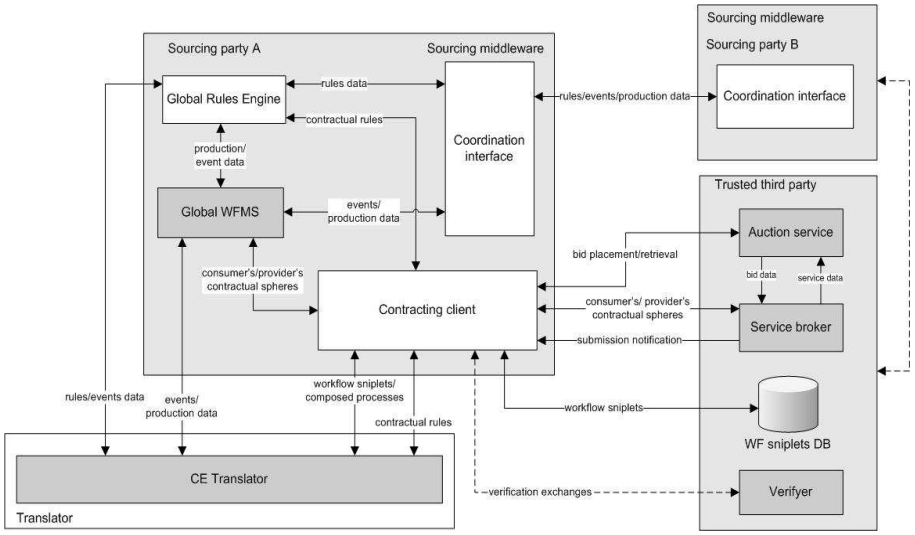

Figure 6.2: External-level collaboration.

tween the trusted-third-party component and the Sourcing-middleware is an abstraction of more detailed information exchange that is described in Section 6.2.3.

In the latter component several refining components are contained. The contracting client component provides support for the management of an e-contracting process. Concretely, the contracting client semi-automatically assembles services by using workflow sniplets that are stored in a corresponding database of the trusted third party. That way the workflow sniplets may be communicated between collaborating organizations. Depending on whether a collaborating party slips into the role of a service consumer or service provider, the contracting client submits or retrieves contractual spheres from a service broker. If a submitted service contains the definition of a concerned party, a submission notification is sent out from the service broker. If several parties are interested in the same service, a bid can be placed with the auction service. The latter component relates the bid data with services stored in the service broker. Finally, when an eSourcing configuration is established, the collaborating parties send their in-house processes and refined spheres to a verifier component for testing the correct termination, i.e., the soundness [10, 62], of the overall eSourcing configuration. The verification results are returned to the collaborating parties. By having a trusted third party perform the verification, the collaborating parties do not have to disclose their internal business details to each other.

When an eSourcing configuration is established, the contracting client distributes the business rules and the processes contained in the contract to the global rules engine and the workflow management system (WFMS) respectively. In order to synchronize the global WFMSs and global rules engines in the Sourcing-middleware components of other collaborating parties, events-, production-, and rules data is communicated via a coordination interface. Production data is for example the specification of a product or data needed for condition statements. Furthermore, the global WFMS and rules engine also exchange production-, and event data with each other.

Finally, the contracting client sends workflow sniplets and composed processes

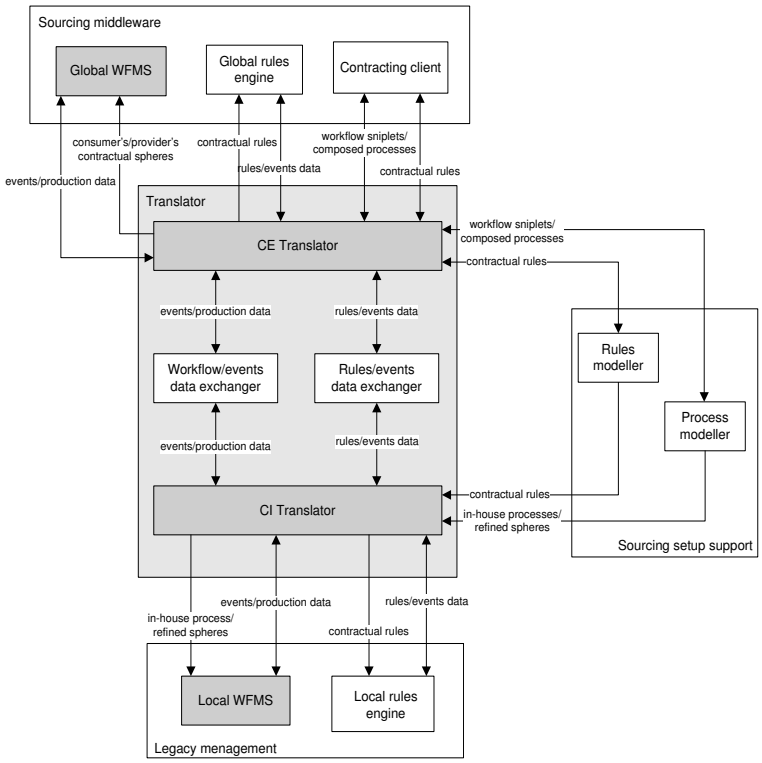

Figure 6.3: Translating between external and internal level.

and contractual rules to the translator. That way the workflow sniplets and composed processes and contractual rules are prepared for the heterogenous system environment that exists on lover internal levels of a collaborating party. Also the global WFMS and rules engine send data to the translator component that is depicted as a white box in Figure 6.3. The translator contains two main translator components for transferring data between the external, conceptual and internal level.

The CE translator component translates data from the conceptual to the external level and vice versa. The component is connected with the rules and process modelers of the Sourcing-setup-support component. The relationships between the CE translator and components contained in the Sourcing middleware is explained above. Two components exchange data between the CE translator and and CI translator, namely the worklfow/events data exchanger and the rules/events data exchanger. Those data exchangers contain information about where data needs to be routed to. For example, several instances of WFMSs and rules engines on the external and internal level may enact several instances of different Sourcing configurations. Furthermore, on the internal level several web services wrap legacy systems to which exchanged data needs to be routed to.

The CI translator component translates data between components of the conceptual and internal levels. From the data-exchanger components, events-, rules-, and production data are translated bi-directionally to the local WFMS and rules engine on the internal level. Furthermore, the CI translator receives contractual rules from the rules

modeler and in-house processes and refined spheres from the process modeler. They are translated to the local WFMS and rules engine on the internal level.

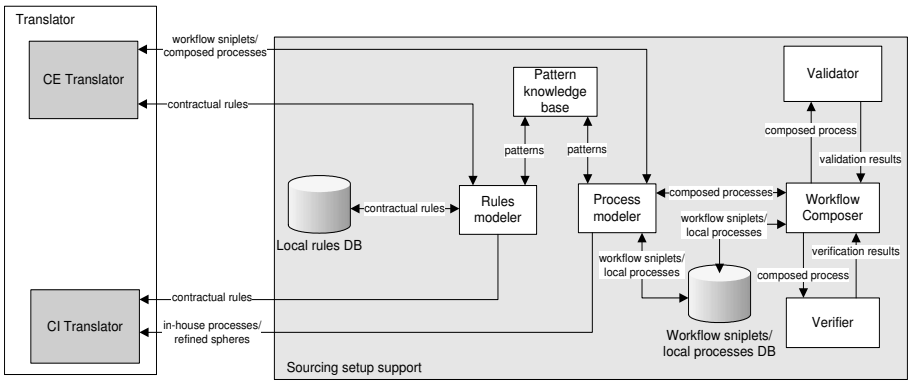

Figure 6.4: Setup functionality.

In Figure 6.4 the Sourcing-setup-support component is located on the conceptual level. The component has two core functions, namely modelling business rules and processes, and composing workflows that are on the one hand evaluated and on the other hand verified for correct termination. Thus, the rules modeler and the process modeler are responsible for the first function for which they are supported by a pattern knowledge base. In Section 6.4 the pattern knowledge base is presented in further detail. The second function is related to the workflow composition component. For composition [44], workflow sniplets or local processes are taken from a dedicated database, which are supplied by the process modeler.

A composed workflow is either an in-house process or a refined sphere and is checked internally in two ways. First, with respect to control flow, correct termination is verified by the tool Woflan [109] for which the process needs to be mapped to a place/transition net. If the net is a WF-net, Woflan checks for structural conflicts, i.e., deadlocks or lack of synchronization. Thus, if the WF-net is verified to terminate correctly, it conforms to the notion of soundness [10]. Secondly, the in-house process or refined sphere needs to be verified for other conflicts, e.g., data-flow or resource.

Although it is desirable to have verification tools for several workflow related perspectives, e.g., data-flow and resource, it is essential to validate the in-house process and refined spheres of an eSourcing configuration with an additional tool. Among other aspects, such a validation is meaningful for testing how the different perspectives fit together for workflow enactment, e.g., the correct functioning of web services that are orchestrated by the processes. In the Sourcing-setup-support component of Figure 6.4 XRL/flower [83] is depicted as a validation tool. In Section 6.5 this tool is presented in further detail. XRL/flower uses XML technology and is implemented in Java on top of the Petri-net Kernel PNK [64]. Standard XML tools can be deployed to parse, check, and handle XRL documents. The XRL enactment application is complemented with a web server allowing actors to interact with the system through the internet.

Finally, Figure 6.5 visualizes a second-level refinement of the legacy-management component. In it, a local WMFS and rules engine constitute the core components. These components are exchanging data between each other and are instrumental for coordinating legacy systems. The business rules and processes that are enacted by

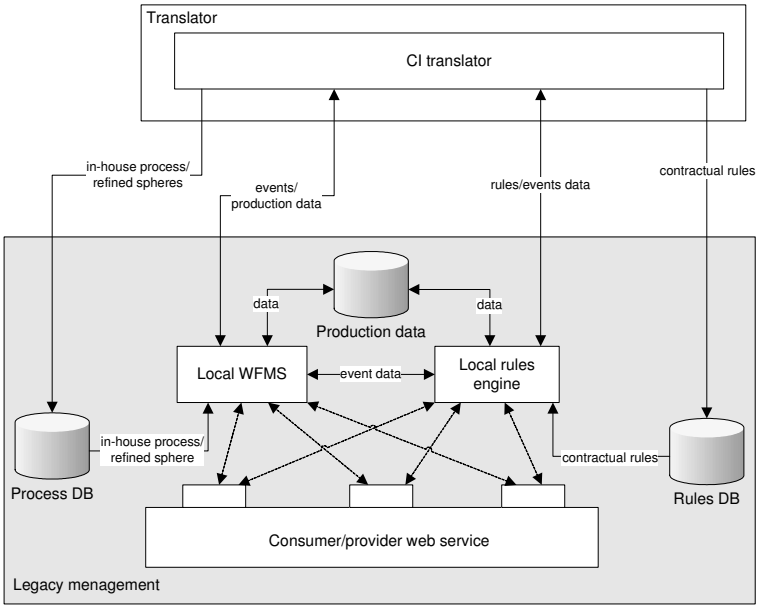

Figure 6.5: Connecting to internal legacy systems.

the WFMS and rules engine are translated down to the internal level by the CI translator. For enactment, the local WFMS and rules engine use a production database. Furthermore, to coordinate the enactment on an internal level and external level, the local WFMS and rules engine communicate events, rules, and production data bidirectionally.

Next, the grey-shaded eSRA components of this section are presented on a third refinement level.

6.2.3 Third Detail-Level of eSRA

In this subsection the dark-grey shaded components of Subsection 6.2.2 are further detailed according to the principles of functional decomposition. First the CE translator and CI translator are refined in Figure 6.6 and Figure 6.8. The refinement of the CE translator depicts a CE projector component that is performing projections between the conceptual and external levels. To perform that function, the CE projector uses a rules database.

Figure 6.6 shows several bidirectional arcs to the CE-projector of which some are related to each other. The rules- and process-modeler components exchange contractual rules and workflow sniplets and composed processes via the CE projector with the contracting client on the external level. The global rules engine receives contractual rules from the CE projector through which rules- and events data is exchanged via the rules- and events data exchanger down to the local rules engine on the internal level. Figure 6.6 depicts a detailed exchange between the CE projector and components of the global WFMS. The enactment engine receives contractual spheres from the service consumer or provider respectively. During enactment, an exchange occurs with the enactment monitor and the conjoinment monitor, which is explained below and depicted

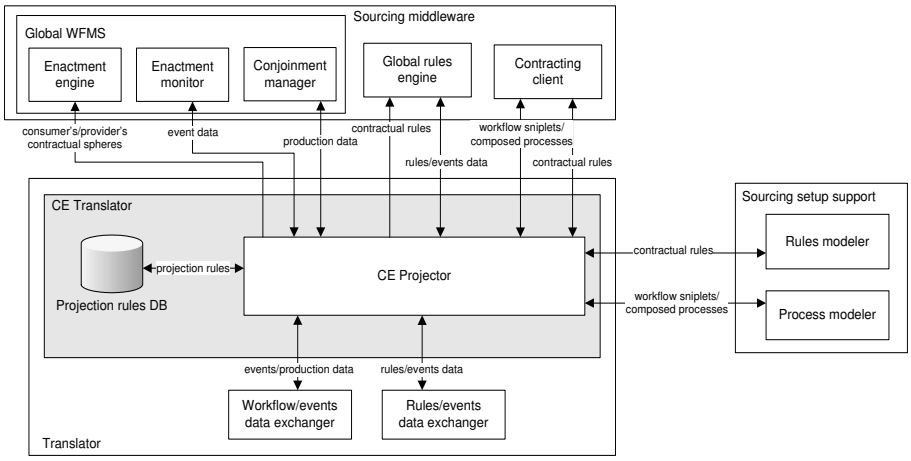

Figure 6.6: The CE translator in detail.

in Figure 6.7. The latter two components exchange events- and production data via the CE projector and the workflow/events data exchanger down to the local enactment monitor and conjoinment monitor that are located on the internal level of the reference architecture.

Figure 6.7 the global WFMS component of the Sourcing middleware that is situated at the external layer, is depicted as a refinement. It shows an enactment engine for the consumer's or provider's consumer spheres that are delivered from the CE translator. Event and production data is created during enactment and also needed for enactment and therefore stored and retrieved from dedicated databases. In order to support the concept of Sourcing, Figure 6.7 shows an enactment-monitor component and conjoinment-monitor component. These components are important to support the Sourcing patterns that are formulated in Chapter 4. Concretely, the enactment monitor is responsible for aligning the enactment progress of internal level processes of the collaborating parties. In Subsection 4.5.2 the patterns are specified that are supported by the enactment-monitor component. Likewise, the conjoinment-manager component supports the conjoinment patterns specified in Subsection 4.5.3. Both the enactment monitor and the conjoinment manager exchange production and event data with components in the domain of the collaborating party via the coordination interface. Furthermore, production and event data is communicated to the internal level via the CE translator to coordinate local components.

The refinement of the CI translator in Figure 6.8 depicts a similar setup as for the CE translator. However, the information exchange to neighboring components differs. The CI translator contains a CI-projector component that projects information between the conceptual and internal level. To do so, a projection-rules database is exchanging rules with the CI projector. With respect to information exchange between the CI translator and its environment, different subsets of arcs depicted in Figure 6.8 are related to each other. The CI projector receives contractual rules from the rules modeler, and in-house processes and refined spheres from the process modeler. The contractual rules are projected to the local rules engine of the internal level. Furthermore, the in-house processes and refined spheres are also projected to the internal level where a process database stores them until the local WFMS loads the processes for enactment.

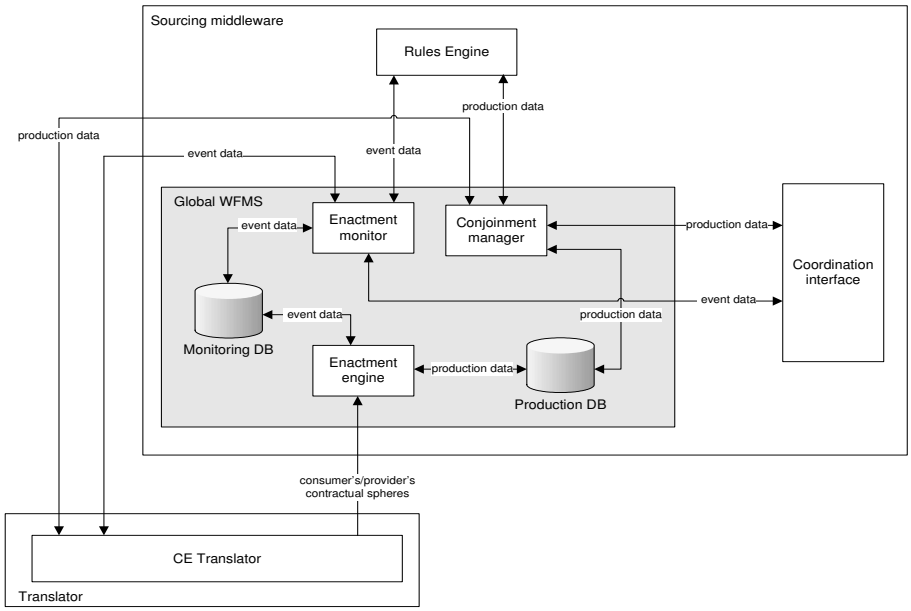

Figure 6.7: The global WFMS in detail.

To coordinate the local WFMS and rules engine with corresponding components on the external level, the CI-projector transfers production , rules, and events data between the internal and external levels of the reference architecture.

The internal level refinement of Figure 6.9 shows a setup that is comparable to the global WFMS of Figure 6.7. The local WFMS contains an enactment engine that receives in-house processes and refined spheres from the process database. Production data that is produced and consumed during process enactment, is exchanged with the production-data database. Event data is exchanged with the local rules engine that carries out contractual rules. Furthermore, the enactment engine exchanges data with ports for the coordination of legacy systems. To coordinate the local enactment progress with the external level, production data and event data are exchanged with the conjoinment manager and the enactment monitor respectively. The latter two components exchange events- and production data via the CI translator with the equally named components located on the external level.

The service-broker refinement within the trusted third party of Figure 6.10 reveals a service-library database that stores contractual spheres of service consumers and providers via the template search engine. The latter component exchanges contractual spheres with the contracting client of the Sourcing middleware that is located on the external level of the collaborating parties. Furthermore, the template search engine exchanges data with the bid-manager component of the auction service. The notifier component checks contractual spheres that are stored in the service library for data about a collaborating party that needs to be informed. If such facts are defined, the notifier informs the specified contracting client of the respective parties about the submission of the contractual sphere. Consequently, informed parties check the contractual sphere and either engage in a bidding procedure or commit to the contractual sphere by instantaneously responding with committing a contractual sphere of equal content

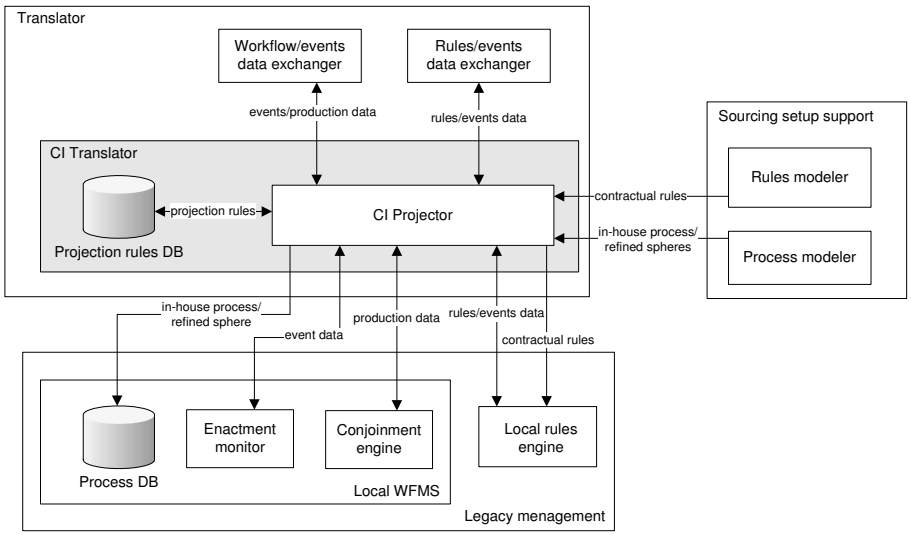

Figure 6.8: The CI translator in detail.

to the trusted third party.

The refined auction service of the trusted third party is depicted in 6.11. In the auction service component the contained bidding library stores bids that are committed and retrieved by a bid manager. This component is communicating with the contracting-client component that places and retrieves bids from the bidding library. As described earlier, the bid manager is exchanging bid- and service data with the template-search-engine component of the service broker. Finally, the last component of the trusted third party is the verifier. In Section 5.7 a verifier architecture is presented that is suitable for the trusted third party. In this architecture the in-house process of a service consumer and the provider sphere are flattened to a P/T-net and consequently verified for correct termination and inheritance relations.

In the remaining sections of this chapter two components of the eSRA reference architecture are further detailed. The component for the pattern knowledge base contained in Figure 6.4 is further elaborated upon with a proposed reference architecture that results from the implementation of a learning prototype. Finally, a proof-of-concept prototype for the evaluation tool contained in Figure 6.4 is presented, namely the web based business-process enactment application called XRL/flower.

Next, the XML-based language eSML is presented that is instrumental for formulating eSourcing configurations.

6.3 eSML - *e*lectronic *S*ourcing *M*arkup *L*anguage

An XML-based modelling language, eSML, is developed as a further proof of the eSourcing concept. It is employed for inter-organizationally harmonizing business processes between collaborating parties by containing the pattern catalogue for eSourcing that are specified in Chapter 4. eSML is applied at the external level where business-process details are publicly disclosed. These details are the result of projecting a larger internal business process that is located on the conceptual layer in the domains of col-

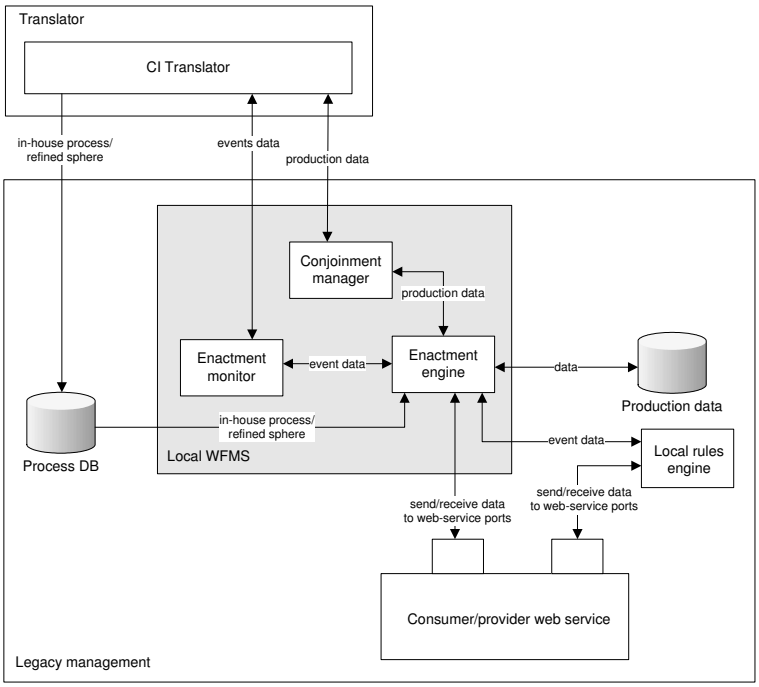

Figure 6.9: The local WFMS in detail.

laborating parties. With this combination of using eSML on the external level and on the conceptual level a separate, bigger business-process definition, the contractual visibility (see Section 4.5.1) for the collaborating counterpart is determined.

In an eSML instance the contractual spheres of a service consumer and a service provider are contained. To achieve a consensus, both contractual spheres must match (see Section 5.4.3). The exposed contractual spheres of the collaborating parties may optionally contain conjoinment nodes (see Section 4.5.3) for the exchange of business information between the opposing business domains. For linking the contractual spheres, a set of monitorability patterns (see Section 4.5.2) is available. That way it is possible to determine how much the enactment progress of an eSourcing configuration may be monitored by a service consumer.

6.3.1 Foundation of eSML

The foundation of eSML is the XML-based language ECML (*E*lectronic *C*ontracting *M*arkup *L*anguage) [23], which is designed for the formulation of electronic contracts. A contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights [56]. In B2B relationships all economic production and exchange processes are organized through contracts. Electronic contracting aims at using information technologies to significantly improve the efficiency and effectiveness of paper contracting, allowing companies to support the newly emerging business paradigms while still being legally protected.

At the highest abstraction a contract in eSML is reduced to answering three questions i.e., the Who, Where, and What question [23]. The *Who* answer concerns the

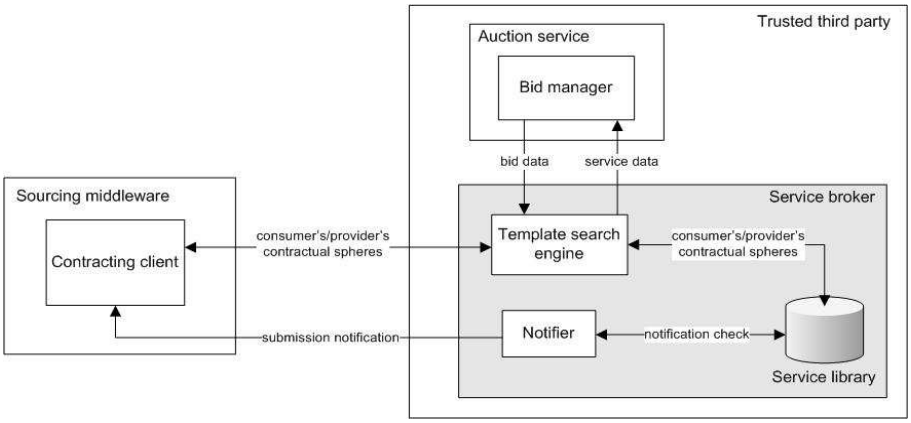

Figure 6.10: The service broker in detail.

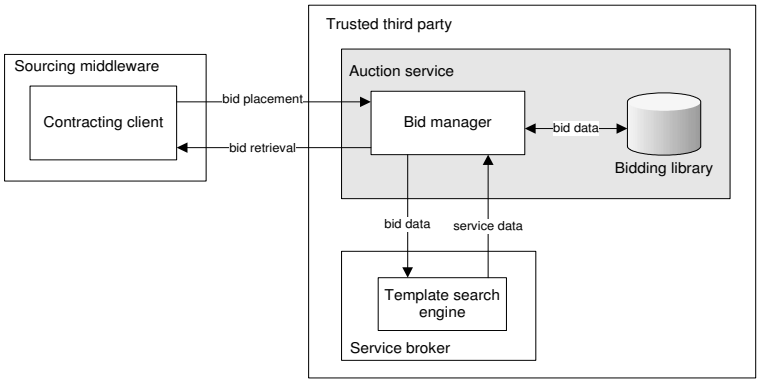

Figure 6.11: The auction service in detail.

actors that participate in the contract establishment and enactment. The *Where* answer specifies the context of a contract, e.g., the legal context, business context. The *What* answer models the exchanged values, rules, and the exchange processes for which XRL [82] is an integral part (see Section 6.5). In its full specification ECML is used for specifying the answers to another questions, namely the *How* answer that specifies the contract content structuring, representation, establishment, and enactment. However, the *How* answer is also not part of eSML as it is covered by the reference architecture for eSourcing.

The following sections present models of constructs and their relationships that are part of the eSML definition. These models are based on the three contracting questions that are described above.

6.3.2 Notation Explanation

Since different perspectives are contained in eSML, it is important to visually distinguish them in the following static UML diagrams [46]. The referred to perspectives are

described in Figure 3.6 of Section 3.4 and paid attention to with a special notation as depicted in the figure below.

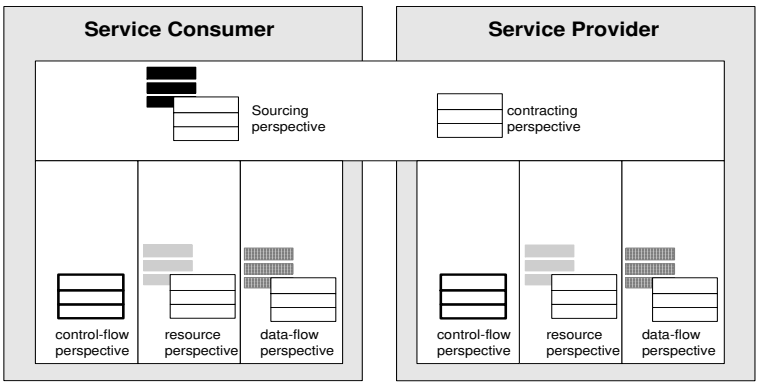

Figure 6.12: Perspective notation.

In Figure 6.12 some UML-class icons are equipped with background shading of different darkness variations and different boldness of lines. Since the inter-organizational concept of eSourcing rests on the three inter-organizational perspectives of control flow, resource, and data flow, eSML also incorporates classes belonging to the latter three.

Visualizing all elements of eSML in a static class diagram results in fairly large models. In order to fit the visualization into this chapter, a way of splitting the overall model into smaller sub-models must be found. Thus, classes that are coupling points between different sub-models are grey shaded and replicated in those sub-models that are adjacent to each other.

6.3.3 eSML Models

As mentioned before, eSML contains three important concepts, namely Who, Where, and What. These three concepts are relevant for structuring eSML in a top-down way for creating a framework in which the perspectives depicted in Figure 6.12 are embedded.

The Who Concept

In Figure 6.13 the classes and relationships belonging to the Who concept are depicted. As mentioned earlier, this concept clearly identifies the contracting parties by including the class `party`. Parties are actors that have rights and obligations, which are listed in the Sourcing configuration. Concerning the relationship cardinalities, it is defined that at least two parties must be stated in a contract. However, it is also possible to have more than two parties defined. For example, an original manufacturer can agree with several suppliers to be included in one contract.

In a contract several third parties may optionally be involved that are termed mediators in the case of eSML. Mediators represented by class `mediator`, participate in the enactment of a Sourcing configuration but their rights/obligations are not stated. Consequently, mediators do not have to sign the e-contract. If their relations with the parties must be defined as legally binding, they can become a party in the same or in

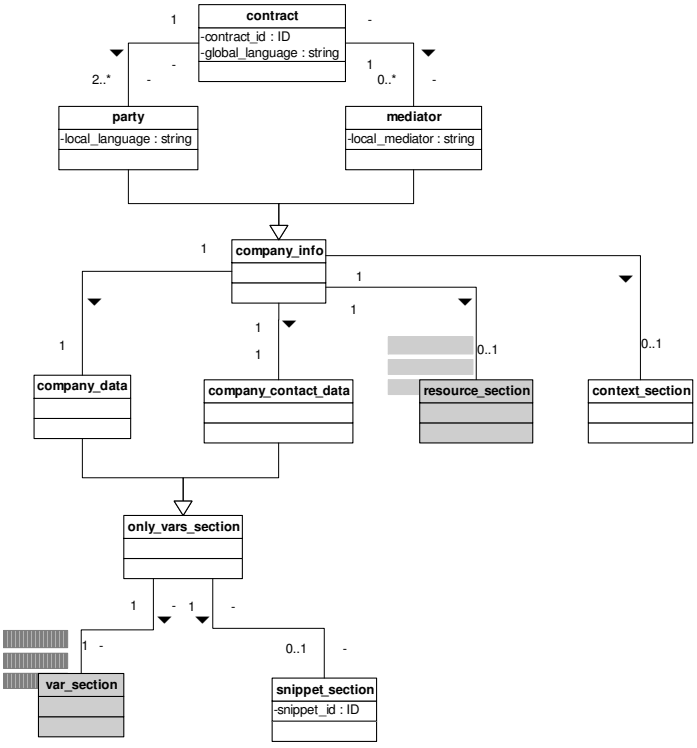

Figure 6.13: Detailed Who model.

a separate contract where their rights and obligations are stated. For example, a mediator could be concerned with verifying whether a Sourcing configuration that is part of a contract can terminate successfully from a control-flow point of view. In order to safeguard business details from each other, the contracting parties can not check such correct termination themselves as they would disclose their business secrets to each other.

Contracting parties and optional numbers of mediators are in a relationship with several other classes. The `company_data` comprises information like the name of a contracting party or mediator, the type of legal organization, and so on, while the `company_contact_data` refers to information related to the geographic location of the Sourcing party. That way a contracting party or a mediator is uniquely identified according to legal requirements.

The third related class termed `resource_section` is the root class of the resource perspective. A contracting `party` or a `mediator` may have attached resource definitions. However, in most cases it might, for example, be superfluous to define resources of mediators, unless a mediator is also primarily involved in commercial exchanges. Resources may comprise of actors and non-actors where the latter can be of a consumable or non-consumable nature. Further details are contained in Appendix A, where figures about the resource perspective are contained.

The classes `company_data` and `company_contact_data` are subclasses of class `only_vars_section`, which contains variables and so called snippets. The

class `var_section` is a docking class belonging to the data-flow perspective that includes company description, trade registration number, VAT registration number, address of registration, etc. The class `snippet_section` references so-called contract snippets that can be attached to particular contract definitions, for example, to attach general terms and conditions.

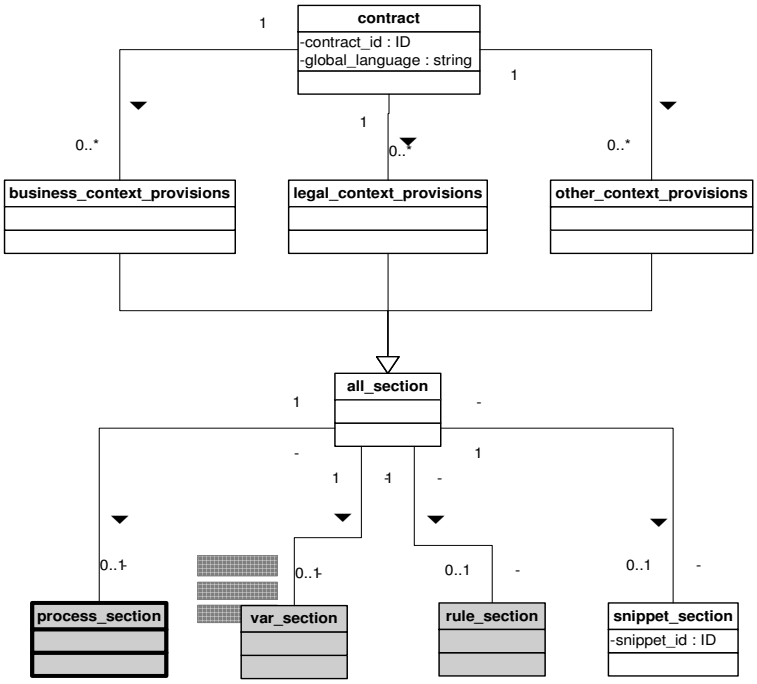

Figure 6.14: Detailed Where model.

The Where Concept

The Where group of concepts contains provisions related to the context of the e-contract. We distinguish two basic aspects of the e-contract context, i.e., the business context and the legal context. Thus two subsections are proposed in the Where section. In addition, a third subsection can be defined to include other e-contract provisions that are not related to the legal and business context.

All three classes named `business_context_provisions`, `legal_context_provisions`, and `other_context_provisions` are subclasses of the grouping class named `all_section`. It references the classes `process_section`, `var_ section`, `rule_section`, and `snippet_section`.

The What Concept

As depicted in Figure 4, the What group contains concepts related to the exchanged values and their related conditions. Two main subsections are distinguished in the What concept, namely the `exchanged_value` and the corresponding `exchange_`

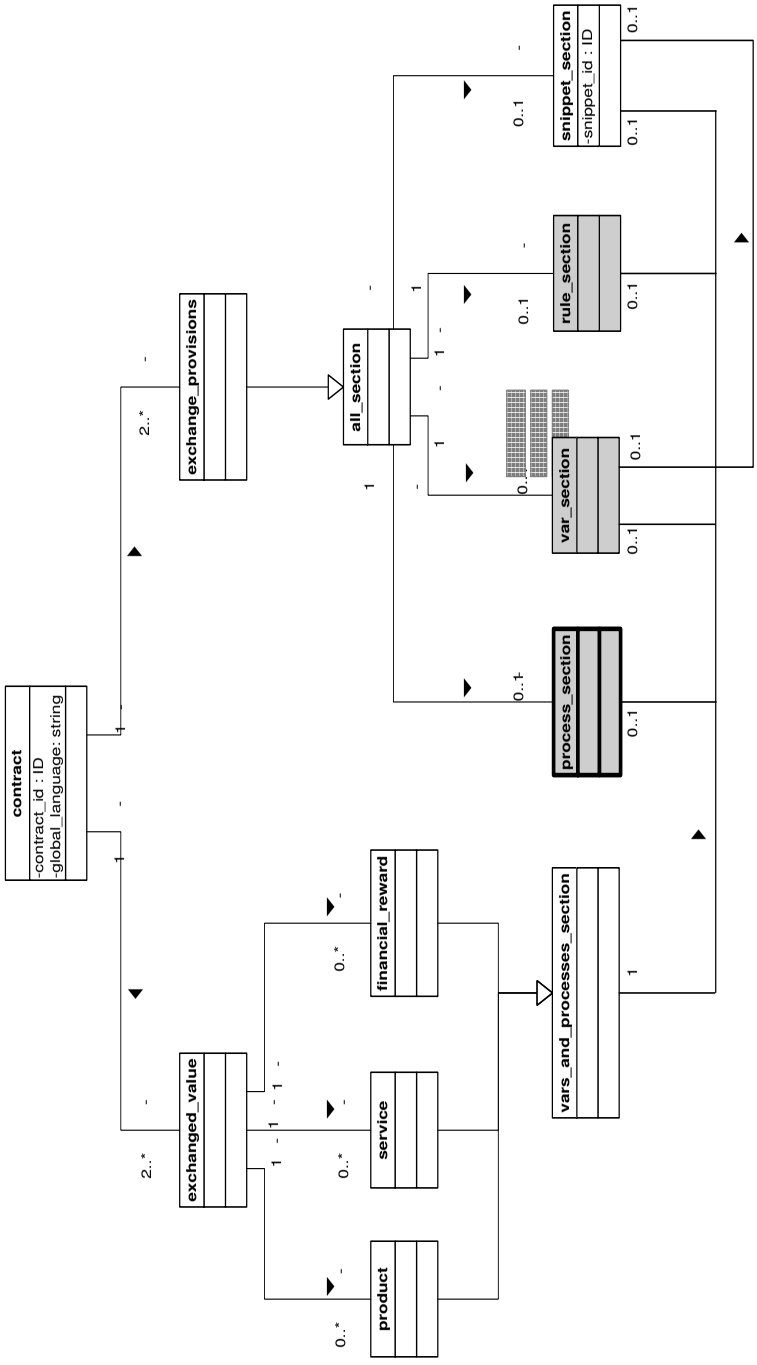

Figure 6.15: Detailed What model.

`provisions` for the value exchange. These classes are defined separately for every respective contractual party involved in contracting.

In a case of product exchange, the product is described by means of data constructs. In a case of service exchange, the service is described through a combination of data that flows through and process constructs. The corresponding financial reward for the received value (in non-barter exchanges) uses the same constructs as a service description subsection. The value exchange provisions subsection requires the use of rule and process specification constructs. Examples for exchange provisions are rules for determining how late payment needs to be handled, how cancellations are dealt with, or definitions for calculating interest adjustments in payments.

The grey shaded classes of Figures 6.13 - 6.15 are docking classes to more elaborate eSML models that are contained in Appendix A. These models cover the contractual perspective and also the control-flow, resource, and data-flow perspective.

For eSML an application environment is required for supporting the different phases of eSourcing collaboration between business parties. The next section offers a reference architecture for setting up and enacting eSourcing configurations that is replicated in the domains of a service consumer and service provider.

6.3.4 Related Work

With respect to industry standards, related work exists. Firstly, abstract BPEL [58] process definitions describe the observable behavior of executable BPEL proces specifications. Abstract BPEL and executable BPEL use the same formal expressive system while the first one permits details to be left out. The intent of abstract BPEL is to provide enough detail for describing the public behavior and to insure conformance at implementation time. Hence, three potential uses of abstract BPEL can be mentioned, namely, the specification of public behavior contracts, the provision of implementation templates, and guards to ensure the executable BPEL processes adhere to what has been agreed upon.

The second notable industry effort is the Web Services Choreography Description Language (WS-CDL) [59] that describes peer-to-peer collaborations of parties by defining their common and complementary observable behavior. Hence, WS-CDL offers a communication bridge between the heterogeneous computational environments used to develop and host applications. Differently to abstract BPEL, WS-CDL explicitly describes how participants interact in a choreography.

When abstract BPEL and WS-CDL are compared with eSML, it can be noted that different collaboration models are realized. Both industry languages address the matching of collaborating processes in a different way than in eSML where a contractual consensus must be established. Moreover, the ability to define the monitorability in a flexible way does not exist in abstract BPEL and WS-CDL. Looking at how the different perspectives are represented in the industry language definitions, it can be noted that eSML is more comprehensive as pattern catalogues for the perspectives control flow, data flow, and resource use are integrated. Additionally, the availability of several types of business rules in eSML allows an extensive specification of contractual clauses that accompany the inter-organizational business process collaboration of a service consumer and service provider.

6.4 A Pattern Knowledge Base Reference Architecture

Inter- and intra-organizational knowledge workers (IKWs) that setup eSourcing collaborations must not have to "reinvent the wheel" every time. Instead, they should be able to use existent pattern catalogues such as those specified in Chapter 4. In Figure 6.4 a pattern knowledge base is depicted to support the modelling of processes. In this section a reference architecture for such a pattern knowledge base is presented that uses the pattern meta-model of Section 4.2 as a foundation.

A pattern for an eSourcing perspective has a lifecycle, which is the starting point for establishing a reference architecture for a pattern knowledge base. A pattern must first be reviewed by a committee before it turns into a quality pattern that is available for IKWs. For the reference architecture an extension of the pattern meta-model is required for capturing additional information about the knowledge-base users and review procedures of patterns.

6.4.1 A Pattern Lifecycle

In Figure 6.16, an activity diagram shows the lifecycle of a pattern, which starts with an initial pattern proposal that is submitted to the knowledge base by a user who has the appropriate authorization level. This initial version of a pattern needs to go through a process of quality assurance before it may be accessible to IKWs who want to search for quality pattern information. Without a review procedure for patterns, the pattern content of the knowledge base may lack quality.

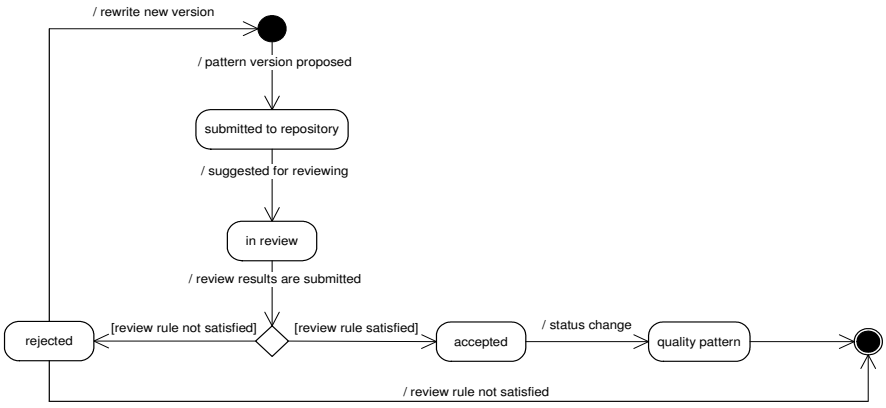

Figure 6.16: The lifecycle of a pattern.

Thus, a knowledge-base user with the authorization of leading a review proposes the pattern for a review process. Repository users with the right skills may volunteer for a review or be explicitly invited by the review leader. Based on a defined review rule, a certain number of review results needs to be submitted for determining whether the pattern is accepted or not. If the review rule is not satisfied, the pattern is rejected and needs to be rewritten as a new proposal. If the review rule is satisfied, the pattern proposal is officially accepted and experiences a status change. Thus, it turns into a quality pattern that is exposed to IKWs for searching. Based on the pattern lifecycle

of Figure 6.16, it is possible to extend the pattern meta-model so that relevant data is captured for running an online application on top.

6.4.2 An Extension for User and Review Management

In Figure 6.17 the `User` package contains relevant classes for handling user and review-related information. On top of Figure 6.17 the `Pattern` package contains class `Pattern`, which is the only connection between the two packages.

The central class of the `User` package is called `RepositoryUser` and contains the attribute `volunteer` for indicating whether a user wants to be a reviewer or not. The user of a knowledge base may slip into several `Roles` which have different authorization levels included that influence how a user can interact with the pattern knowledge base. For example, an analyst is only allowed to browse for information but not allowed to edit a pattern.

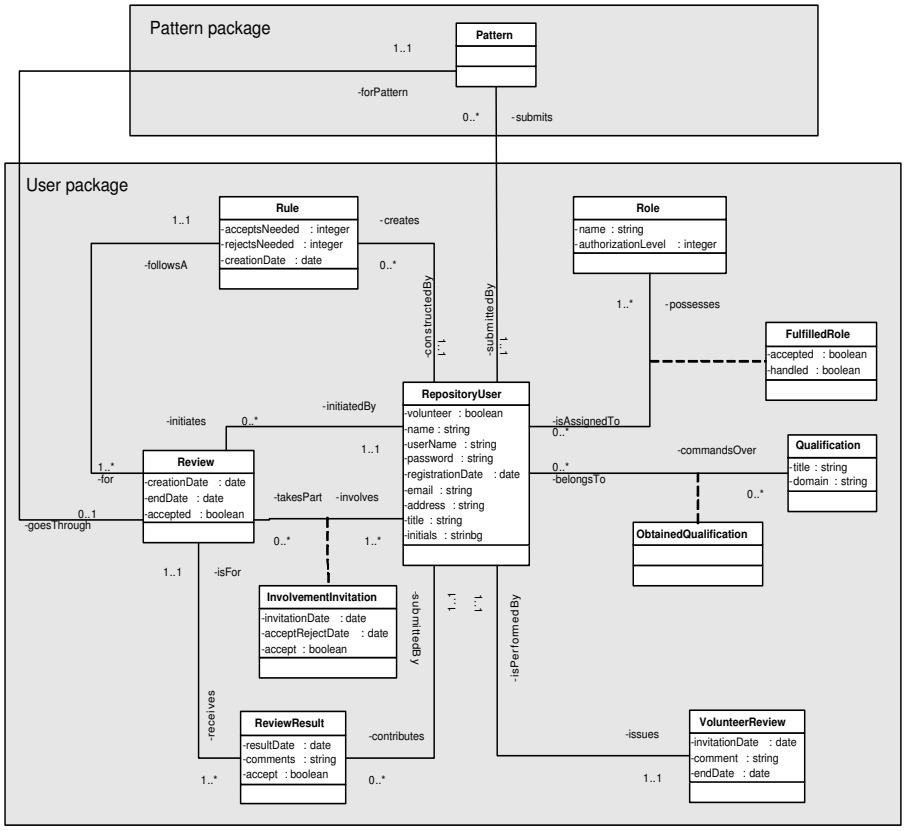

Figure 6.17: Detailed class model of the `UserManagement` package.

The model in Figure 6.17 also depicts the class `Qualification` that is referenced by `RepositoryUser`. The qualifications of a user are relevant for taking part in reviews. Users who do not have required qualifications can not be considered for reviewing patterns. Furthermore, if a user has the appropriate role assigned for leading a review, she can issue invitations for several reviews. Such a knowledge-base user also

needs to define a `Rule` for a review if a predefined rule doesn't exist already. For example, a rule could state that four reviewers out of five must accept a pattern proposal in order to become a quality pattern while two rejections are enough to totally reject the pattern proposal without waiting for the results of the remaining reviewers.

A `Review` is initiated by a knowledge-base user who has the role of a review leader. After assigning a rule to the review, the review leader must find review participants. First, a review leader can look in the pool of users who issued an entry in `Volunteer Review`. Provided a volunteers possess the appropriate qualifications, a review leader creates an entry in class `InvolvementInvatiation` and therefor establishes a relationship between a knowledge-base user and a pattern review. Secondly, if not enough volunteers are available, the review leader searches for rightly qualified knowledge-base users and issues a review-involvement invitation that is either accepted or rejected. If the knowledge base rejects the invitation, the entry created by the review leader is removed from class `InvolvementInvatiation`. Finally, once the required amount of review participants is established, entries in class `ReviewResult` are performed that indicate whether a knowledge-base user accepts or rejects a pattern proposal. Next, a reference architecture is presented for a pattern knowledge base.

6.4.3 An Application Architecture

The pattern lifecycle of Figure 6.16 shows which actions of application users an application architecture must support. An *author* is a user who submits a pattern to the knowledge base. A *review leader* forms a review committee for the evaluation of newly submitted patterns. Registered users of the knowledge base who indicate to be volunteers as *reviewers* are invited by the review leader to form a committee. An IKW with the role termed *analyst* is interested in browsing a knowledge base that contains patterns of different perspectives and corresponding information about their artifact support. As indicated in Figure 3.6, that pattern information helps an analyst to estimate which patterns collaborating business parties support despite their heterogeneous system environments. That way the setup time of inter-organizational business processes is accelerated. Finally, an *administrator* of the pattern knowledge base is required to grant roles to registered users, troubleshoot during pattern reviews, and so on.

The mentioned roles for knowledge-base users are input for an architecture that uses the pattern meta-model as a foundation. The described knowledge-base user types are depicted in Figure 6.18 where bi-directional arrows indicate an exchange with certain modules of the application's web interface. In the interface layer, modules are contained, for user management related interfaces, pattern related interfaces, and review related interfaces.

- The *user management interfaces* offer a knowledge-base user to register, claim various qualifications, and request particular roles. As different roles give a knowledge-base user different rights, the administrator may need to authorize the roles. Once a knowledge-base user is approved, the user management interfaces allow user to login and logout, and modify roles and qualifications.

- The *pattern interfaces* allow users to browse the knowledge base for patterns with a search engine that uses facts from the classes belonging to the taxonomy package (see Figure 4.2) and the support package (see Figure 4.4). The generated lists of patterns can be individually selected for exploring their details.

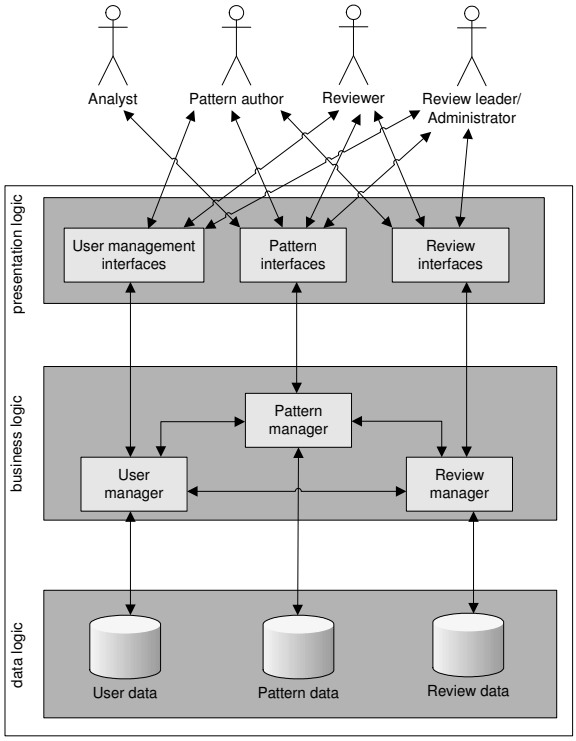

Figure 6.18: The application architecture of the pattern knowledge base.

- *Review interfaces* allow a review leader to set up a review committee consisting of reviewers who either volunteer or are appointed based on their qualifications. In the latter case an appointed reviewer may decline through an interface. After the reviewers explore the properties of a pattern, they submit an accept or reject and their feedback for the pattern author. The latter knowledge-base user checks the feedback from the reviewers through another interface.

The functionality layer of Figure 6.18 shows modules that support the presentation-logic layer, namely the *user manager*, *pattern manager*, and the *review manager*. They process input of the knowledge-base users and control the interfaces that are presented for the signing up and signing in of users, submitting and browsing patterns, performing reviews, and various administration activities. Figure 6.18 depicts the modules of the functionality layer that are referencing each other. For example, to perform a review, the review-manager module uses functionality contained in the user-manager module. As a result, competent review teams are organized with the right qualifications. During a review, functionality from the pattern manager allows a reviewer to explore the context a pattern proposal is embedded in, i.e., the taxonomy location, technology support, relationship with other patterns, and so on. Furthermore, a reference between the pattern-manager module and the user-manager module exists for the same reason of employing functionality from each other. For example, if a knowledge-base user wants to browse for pattern information, she needs to have the role of an analyst, which must be checked by using functionality from the user manager.

The bottom of Figure 6.18 depicts the data layer showing databases for *user data*, *pattern data*, and *review data*. These databases are referenced by the corresponding modules of the functionality layer. The figures of Section 4.2 contain data elements that are in the pattern data. The review and user data is depicted in Figure 6.17 without claiming completeness.

6.5 XRL and XRL/flower

This section starts with an introduction of XRL (eXchangable Routing Language) [82], which is contained in the schema definition of eSML for the purpose of modelling the contractual spheres of a service provide and service consumer. XRL is also an option for modelling the in-house process of a service consumer and the provider sphere of a service provider that are located on the conceptual level of an eSourcing configuration. As a means of evaluation of these processes, the tool XRL/flower exists to be an evaluation tool as proposed by eSRA.

This section is structured as follows. First, a brief introduction to XRL is given, followed by an architecture overview of XRL/flower. Next, the enactment lifecycle of an XRL file is described, followed by a brief presentation of the components comprising XRL/flower.

6.5.1 XRL: an XML-Based Routing Language

XRL an instance-based workflow language that uses XML for the representation of process definitions and Petri nets for its semantics. A catalogue of control-flow patterns [17, 18, 60, 61] is contained in the definition of XRL [20, 82] as routing elements that results in strong control-flow expressive power of XRL. These routing elements are equipped with Petri-net semantics [21], namely, every routing element stands for an equivalent WF-net sniplet that can be connected with other routing elements into a bigger WF-net. The syntax of XRL is completely specified in a DTD and schema definition [82]. An XRL route is a consistent XML document, that is, a well-formed and valid XML file with top element route (see the Appendix A.6). The structure of any XML document forms a tree. In case of XRL, the root element of that tree is the route. This route contains exactly one so-called routing element. A routing element (RE) is an important building block of XRL. It can either be simple (no child routing elements) or complex (one or more child routing elements). A complex routing element specifies whether, when and in which order the child routing elements are done.

XRL provides the following routing elements:

- *Task*. Offer the given step to some resource, wait until the step has been performed, and afterwards set all events for which a child event element exists.
- *Sequence*. Start the child routing elements in the given order and wait until all have been performed.
- *Any_sequence*. Start the child routing elements in any order and wait until all have been performed.
- *Choice*. Start one of the child routing elements and wait until it has been performed.

- *Condition*. If the given condition holds, start the child routing elements of all true child elements in parallel and wait until all have been performed. Otherwise, start the child routing elements of all false child elements in parallel and wait until all have been performed. A condition may have any number (even none) of true and false child elements.
- *Parallel_sync*. Start the child routing elements in parallel and wait until all have been performed.
- *Parallel_no_sync*. Start the child routing elements in parallel but do not wait for any of them.
- *Parallel_part_sync*. Start the child routing elements in parallel and wait until the given number of child routing elements has been performed.
- *Parallel_part_sync_cancel*. Start the child routing elements in parallel, wait until the given number of child routing elements has been performed and cancel the remaining child routing elements if possible.

The routing elements of XRL are based on a thorough analysis of the workflow patterns supported by leading workflow management systems [17]. XRL is an instance-based workflow language that uses XML for the representation of process definitions and Petri-net formalism for its semantics resulting in an unambiguous understanding of XRL. This makes XRL an interesting proposition for eSML instantiations where XRL is used for modelling the contractual spheres of a service consumer and a service provider (see Appendix C).

As shown in [21], the semantics of XRL is expressed in terms of WF-nets (see Section 2.3), which permits the use of theoretical results and standard tools such as Woflan [10, 109] (see Section 2.6) for checking the notion of soundness. The Petri-net semantics of XRL is realized by mapping to PNML [63, 65, 113], an XML-based interchange format that permits the definition of Petri-net types. For that purpose a stylesheet translator is employed that contains mapping rules to PNML for every XRL control-flow construct that are described in [21].

6.5.2 XRL/flower: An Evaluation Tool

XRL/flower [110] is intended to be part of eSRA (see Section 6.2) where it is used as an evaluation tool for the in-house process of a service consumer and for the provider sphere of the service provider. The development of XRL and subsequently XRL/flower can be seen as a reaction to several XML-based standards for business process modelling that have emerged in recent years. Some examples of relevant present and past acronyms are BPEL4WS [40], BPML [33], WSFL [55], WSCI [28], XPDL [38], XLANG [103], and so forth. However, while their semantics and expressive power is suitable for technically composing web services, their satisfactory application in B2B collaboration poses a challenge. In contrast, XRL as a part of eSML is equipped with very clear Petri-net semantics. Differently to the mentioned XML-standards one can determine *before* enactment whether an XRL modelled workflow is sound or not. Such analysis power is crucial for avoiding the occurrence of abnormalities such as deadlocks during carrying out business transactions.

Since XRL is based on both XML for syntax and Petri nets for semantics, standard XML tools can be deployed to parse, check, and handle XRL documents. The Petri-net representation allows for a straightforward implementation of the workflow enactment

engine. XRL constructs are automatically transformed to Petri-net constructs. This allows for an efficient implementation and the system is easy to extend by employing an XSL translator for mapping routing elements to PNML. Thus, for supporting a new control flow primitive, only a transformation to the Petri-net format needs to be added and the engine itself does not need to change.

6.5.3 Architecture of XRL/flower

Figure 6.19 shows the toolset architecture of XRL/flower where grey shaded elements are implemented. Using both the control-flow data for the workflow case and case specific data, the Petri-net engine computes the set of enabled tasks, that is, the set of work items that are ready. The engine sends this set to the work distribution module. Based on information of organizational roles and actors, the work distribution module fills the work item pool. Resources that may carry out those ready worklist items can log into XRL/flower through the web server. If the actor has registered beforehand, an online worklist manager displays the ready items in the web client that are assigned by the work distribution module. By accepting a chosen worklist item, its content is displayed.

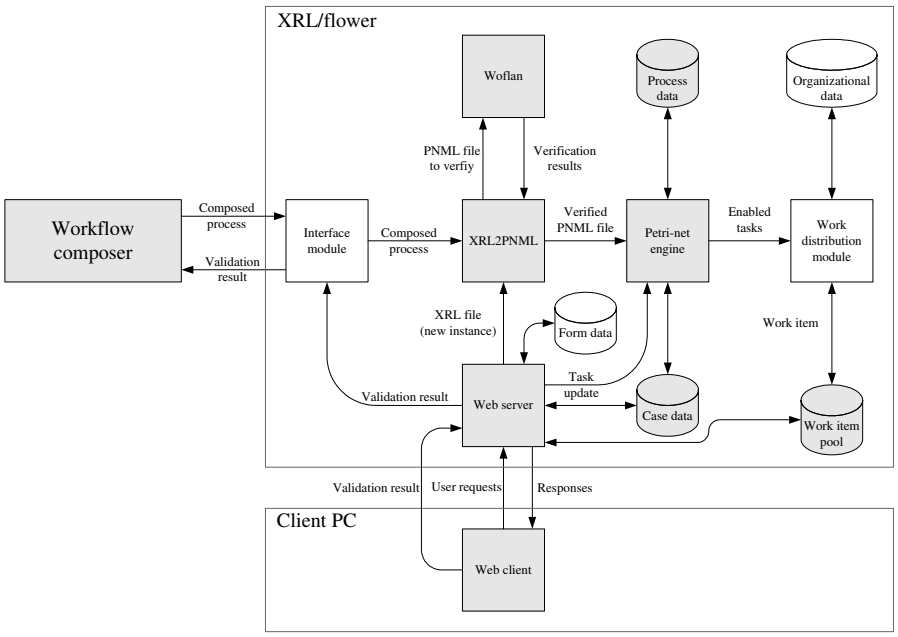

Figure 6.19: XRL/flower architecture [21].

In order to enable an actor to perform an activity, the web server fills the appropriate form template with case specific data for the activity. The web server stores updated case data and signals the Petri-net engine the activity has been completed. The Petri-net engine then recomputes a new set of work items that are ready. The actor can also start an XRL instance by sending the corresponding XRL file to the web server. The web server forwards the XRL file to the XRL2PNML module that transforms XRL to PNML (Petri-Net Markup Language), which is a standard representation language for

a Petri net in XML format [63]. Finally, Figure 6.19 also depicts an interface module of XRL/flower. The interface is used to communicate the composed process and the validation results to the workflow composer, which conforms to Figure 6.4 where a higher level exchange between the two components is explained.

6.5.4 Evaluation with XRL/flower

The activity diagram depicted in Figure 6.20 shows the enactment lifecycle of an XRL modelled workflow. First, the XRL2PNML module performs a transformation from XRL to two PNML files: one for verification and one for enactment sharing similar soundness characteristics. The first PNML file is verified using the Woflan tool. Based on the result either the second PNML file is sent to the Petri-net engine for enactment, or the actor is informed about the XRL instance containing flaws. In the latter case, the actor may either abandon the new instance, or modify it to fix the errors. Of course, the fixed instance is also verified before it is enacted.

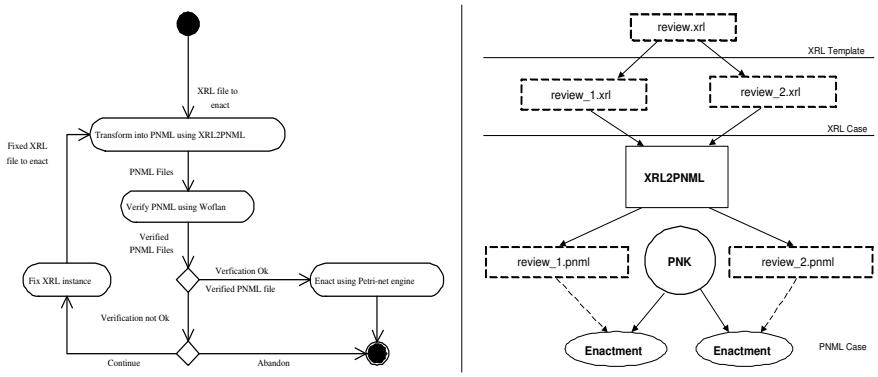

Figure 6.20: XRL case life-cycle.

The right hand visualization of Figure 6.20 serves to complete the understanding of XRL/ flower instance handling. Dashed rectangles represent workflow files, rectangles stand for XRL/flower modules, and circles depict component processes. An XRL workflow may serve as a template of which instance files are created. Every instance of the workflow is uniquely identifyable. After the XRL2PNML module translates every instance to PNML and assuming soundness evaluation succeeds, instances are loaded into the Petri-net engine module that consists of the Petri-net kernel PNK and an enactment application built on top of PNK. For every workflow instance that needs to be carried out, a new enactment application process is created. Thus, multiple instance enactments can be handled concurrently by XRL/flower's Petri-net enactment module.

6.5.5 Component Description

Several parts of the XRL/flower toolset in Figure 6.19 are grey shaded, which means they are largely implemented in the currently available prototype [83]. This section describes the existing functionality of tool modules depicted in Figure 6.19 and how they interact with each other.

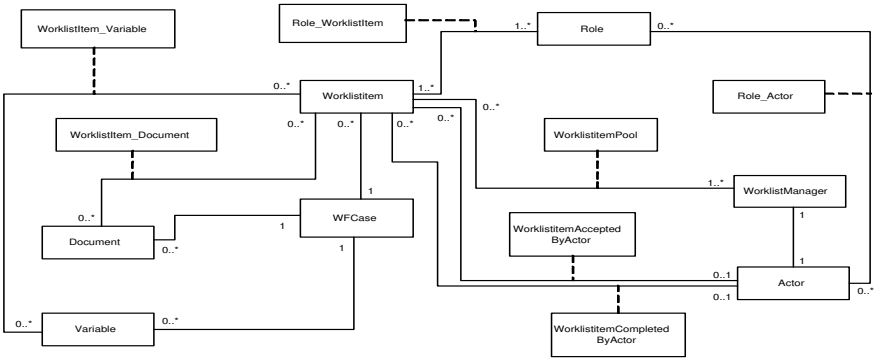

Figure 6.21: Database model of XRL/flower.

A relevant preliminary for describing the component's way of interacting is the database of which a model is depicted in Figure 6.21. Note that this model is independent from the pattern meta-model of Section 4.2 and it's extension for the pattern-knowledge base of Section 6.4.2. The central entity of the model is `WFCase` that contains attributes about every enacted case instance's unique identifier, starting and ending timestamp. A case file can contain variables with optional values present at case start time. Likewise documents can be contained in the case file in combination with a `uri`.

For attaining increased flexibility with respect to organizational fluctuation, a split between organizational roles and actors is sensible. Attributes can be attached to roles into which actors can slip. The database model in Figure 6.21 shows that an actor may slip into multiple roles and a role may be held by many actors. When an actor registers for the first time with XRL/flower, a worklistitem manager entry is inserted in the entity `WorklistManager`.

Ready worklist items are inserted with a start timestamp in the `Worklistitem` entity and a worklist-item-manager identifier is instantaneously added to the worklistitem entry. Variable and document links can be contained in a XRL task definition. Thus, further tuples need to be inserted for assigning the appropriate variables and documents to a particular worklist item entry. The database model of Figure 6.21 also shows entities for the acceptance of a worklist item and completion of an activity that are committed by an actor together with time stamps through the web client.

XRL2PNML

The XRL2PNML module consist of a stylesheet translator containing mapping semantics from XRL [20, 82] to PNML described in [21]. As a result carefully extracted control-flow patterns [21] contained in the XRL modelled workflow (see Appendix A.6) are converted into a WF-net represented in PNML format.

Petri-Net Enactment Engine

The core of this module is the Petri-net kernel PNK on top of which an enactment application is implemented. PNML files representing case instances can be loaded into the PNK that translates workflow node tags into an object instance net. Next, the case name is detected in the PNML file and inserted to the `WFCase` entity. Parsers check

for variable and document tags that are automatically inserted to the database together with their optionally present values.

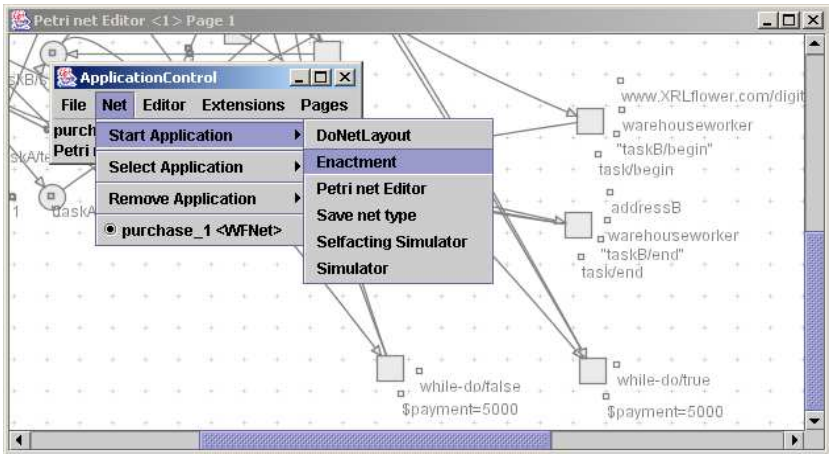

Figure 6.22: Enactment application of the Petri-net enactment module.

Figure 6.22 shows the enactment application selection in front of an editor displaying the PNK loaded workflow instance. After choosing the enactment application, a simple Petri-net firing rule can be started that allocates all enabled transitions and fires them randomly in a loop until no enabled transitions can be detected [110].

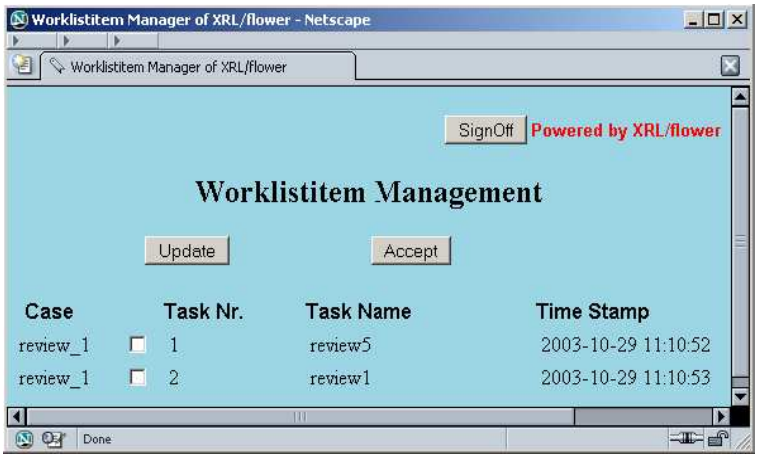

Figure 6.23: Worklistitem manager created by the web server.

Web Server/Web Client

An actor who wants to interact with the XRL/flower system must log in through the web client. If the actor can not be detected in the database, the web server sends a form

to the client for entering assorted data that is inserted through the web server to the database.

Figure 6.24: Display of an activity.

If the actor logging into XRL/flower can be detected in the database, the web server checks the `WorklistitemPool` for ready worklist items. In Figure 6.23the detected set is displayed in the web client. Clicking the `Accept` button triggers the web server to perform a corresponding insertion in the database (see Figure 6.21).

In a next step the web server reads all variable and document entries associated with the activity and displays them in the web client for the actor. A chosen worklist item can be accepted through the web interface, which results in a display of all activity properties as Figure 6.24 shows. An address attribute is displayed delivering a web Service `uri` the actor can use for carrying out an activity operation. The web client allows the actor to choose particular variables and documents for changing the delivered values and commits them to the database. Such changes are relevant for the evaluation result of XPath conditions used in the workflow case.

Figure 6.24 shows a `Complete` button the actor may click after having carried out the activity instruction. Such action commits a corresponding entry in the database containing a timestamp. As a result the web server collects the new set of ready worklist items the Petri-net engine module has generated and inserted to the database and displays them again in the web client. Once the actor has completed carrying out activities with XRL/flower support, clicking the `SignOff` button results in a log off.

6.6 Conclusion

In this chapter several proof-of-concepts are presented, which demonstrate in short realizations that the core ideas of eSourcing are workable and feasible. Hence,a reference architecture is presented that is guiding for the development of applications to support the eSourcing setup phase and the enactment. eSRA is embedded in the three-level framework for supporting the setup and enactment of eSourcing configurations. It is

demonstrated that collaborating parties must not directly connect their domain-specific legacy systems with each other. Instead the three-level framework ensures that a common external denominator is established for achieving inter-organizational business process harmonization.

Next, in this chapter an XML-based proof-of-concept language called eSML is described. eSML is intended for the external level of an eSourcing configuration and based on the electronic contracting language ECML. eSML provides the constructs for defining eSourcing configurations. By adopting the process modelling language XRL for eSML, the control-flow perspective is supported with clearly defined semantics. The data-flow and resource perspectives for business processes are realized by introducing into eSML additional XML constructs that embody specified patterns in the mentioned perspectives. Thus, by combining these perspectives into eSML, a language is created that covers the business needs of inter-organizational business process harmonization.

For two components of eSRA, a further refined architecture and instantiation are presented. Firstly, a detailed model and reference architecture for a pattern knowledge base is presented that supports intra- and inter-organizational knowledge workers in efficiently and effectively setting up eSourcing configurations. Since many patterns are specified in different perspectives, the need arises for such a knowledge base. The architecture shows which components are required to develop a pattern knowledge base and how these components communicate with each other. By using a pattern meta-model as a foundation for the knowledge base, the specified pattern catalogues for eSourcing can be stored and searched as the the knowledge base allows to arrange the patterns in a taxonomy.

Finally, the web-based workflow management system XRL/flower is a "proof of construction" that is instrumental as a validation tool for the eSML formulated business-process specifications. The architecture of XRL/flower is explained, depicting the existent toolset. By employing a translation module XRL2PNML, Woflan can be applied for verifying the soundness property of a workflow *before* enactment. The Petri-net enactment module uses the Petri-net kernel PNK on top of which an enactment application is implemented. No adaptations have to be carried out at the Petri-net enactment module if XRL is extended with a new control-flow element. The XRL/flower database model is presented and explained followed by a detailed discussion of the interaction sequence between the Petri-net enactment module, web-server module, and the database server.

Chapter 7

Cases and Evaluations

Contents

Following the design-science research methodology, results in the creation of artifacts, namely the eSourcing framework together with corresponding pattern catalogues, the reference architecture eSRA, and the eSourcing Markup Language called eSML. The focus of this chapter is an evaluation of the mentioned artifacts with the help of case studies, which are inspired by larger case studies that have been conducted for CrossWork. The concise case studies in this chapter perform an evaluation of the created artifacts, which demonstrate that the eSourcing concept these artifacts represent are workable and feasible. The evaluation requirements are defined based on the business needs of CrossWork industry partners, literature, and expert interviews. After evaluating the artifacts in two case studies, the conclusion of this chapter discusses the evaluation results, i.e., assesses to which extent the list of requirements defined earlier are fulfilled.

7.1 Introduction

For evaluating the concept of eSourcing, business cases from the automobile industry are taken and translated into eSML specifications. Since the case studies for evaluating the eSourcing framework with related patterns, eSRA, and eSML are inspired by CrossWork [3, 4] case studies, it is important to stress the differences. Firstly, in the latter case the emphasis of case studies is on solving practical functional problems that CrossWork industry partners are facing and secondly these case studies are large and extensively documented. In contrast, the two case studies in this chapter need to be brief and the case studies focus more on non-functional requirements. However, the cases in this chapter are inspired by the large CrossWork case studies. Finally, it must be stressed that eSRA is not implemented as a proof-of-concept prototype in CrossWork, instead the latter prototype represents a subset of eSRA.

The structure of this section is as follows. First, Section 7.2 characterizes supply-chain features of the automobile industry. After that, Section 7.3 defines evaluation criteria for the eSourcing framework, the related pattern catalogues, eSRA, and eSML. In Section 7.4 the first case study is presented that focuses on the eSourcing framework, the pattern-catalogue application, and eSRA. The second case study of Section 7.5 focuses on showing an application of eSML constructs. Finally, Section 7.6 concludes this chapter by presenting the evaluation results based on an assessment of criteria.

7.2 Characteristics of Automobile Supply Chains

For evaluating the concept of eSourcing, business cases from the automobile industry are taken and translated into eSML specifications. In the automobile industry, original equipment manufacturers (OEM) have several tiers of suppliers that agree to deliver systems collaboratively. For example, the OEM assembles cars with systems like a cockpit, or an engine, etc. These systems are manufactured by Tier 1 that gets the components for those systems from a Tier 2 supplier.

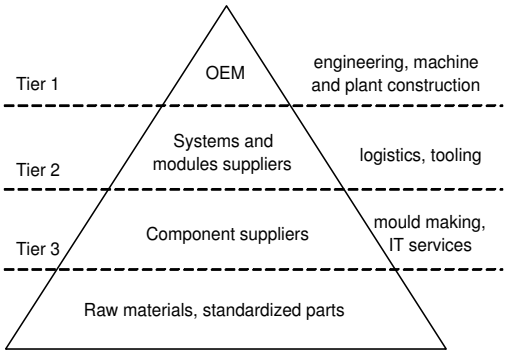

Figure 7.1: Supply-chain hierarchy in the automobile industry.

As depicted in Figure 7.2, the supply chain relationship between an OEM and suppliers resembles a pyramid where the OEM at the top spends considerable time and effort on aligning first and second tier suppliers for achieving the desired service provision. Additionally, the overall number of produced cars and also the number of variants is

going up while the lifetime of car-types is shortening, which means the number of cars per type is decreasing. To deal with the resulting complexity in manufacturing as well as in design and development, OEMs are shifting parts of their activities down the organizational hierarchy. By applying eSourcing with specifying the inter-organizational collaboration with eSML, this coordination effort between collaborating parties is facilitated.

Both CrossWork-inspired case studies are for the assembly and supply of water tanks for trucks. First, the order of the OEM needs to be prepared by service provider *A* who has to find the appropriate automobile-cluster members for carrying out the required nested spheres. The water tank itself consists of a body, a grommet, a motor pump, a dispenser, and a sealing ring. Provider *A* receives the order for the entire automobile cluster from the OEM and organizes the distribution of the production of water-tank parts to partners of the automobile cluster. Thus, it is decided the body and the grommet are produced by service provider *B*, the motor pump and the sealing ring are produced by service provider *C*, and the dispenser is produced by service provider *D*. Finally provider *A* takes over the services of preparing the order and assembling the produced parts to one water tank that can be shipped to the OEM.

7.3 Evaluation Requirements

For the eSourcing framework, the pattern-catalogue application, eSRA, and eSML, a set of functional and non-functional requirements is chosen with specific definitions. Functional requirements specify specific behaviors of a system while non-functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviors. Other terms for non-functional requirements are "quality attributes" and "quality of service requirements". In Table 7.1 presents a requirements classification, stating which are functional and non-functional. The next column assigns the requirements to artifacts. While some requirements in Table 7.1 are evaluated for the several artifacts, their respective definitions differ. The right-hand column lists the case studies, in which the requirement is investigated.

	Type			
Requirements	functional	non-functional	**Artifact**	**Case Study**
Feasibility		X	eSourcing	1
Scalability		X	eSourcing	1
Interoperability		X	eSourcing	1
		X	eSRA	
Coherence	X		eSourcing	1
Applicability		X	Patterns	1
		X	eSRA	
		X	eSML	2
Structuring support	X		Patterns	1
	X		eSML	2
Completeness		X	eSRA	1
Data-flow support	X		eSML	2
Resource-specification support	X		eSML	2

Table 7.1: Requirements for eSourcing, related patterns, eSRA, and eSML.

First, requirements for the *eSourcing framework* are presented. The requirement *feasibility* means that it is possible to model a business case as an eSourcing configuration.

Scalability refers to the ability of the eSourcing framework to combine many collaborating parties into one eSourcing configuration. *Interoperability* focuses on the ability of the eSourcing framework to connect intra-organizational business processes of collaborating parties inter-organizationally so that the parties can reach their respective business goals. *Coherence* must exist between external tasks and their internal counterparts.

For the specified *pattern catalogue* for eSourcing, the requirements are as follows. *Applicability* focuses on the pattern catalogues for eSourcing and how the patterns are relevant for solving problems in dynamic inter-organizational business process collaboration. *Structuring support* means that the patterns are instrumental for the detailed design and analysis phase of an eSourcing configuration.

The reference architecture *eSRA* is assessed with the following requirements. *Completeness* is the quality of comprising the components required for setting up and enacting an eSourcing configuration satisfactorily. *Applicability* states that eSRA is useful for operationalizing inter-organizational collaboration without a direct connection of internal legacy systems. *Interoperability* is the capability of eSRA to realize a technical and conceptual harmonization of intra-organizational business processes for a linked enactment phase.

Several requirements are used for assessing *eSML* in a separate case study. *Structuring support* means that eSML is instrumental for specifying a collaboration where the business processes are inter-organizationally harmonized. *Data-flow support* demands that eSML is suitable for specifying the data-flow within and between the domains of collaborating parties. *Resource-specification support* states that eSML contains the constructs required for specifying organizational facts, production infrastructure and material facts, and the relationship between them. Finally, *applicability* as a requirement for eSML states that the language comprises of relevant modelling constructs for specifying a collaboration in the perspectives eSourcing, control flow, data flow, and resource. In the first case the requirements for the eSourcing framework, the pattern catalogue, and eSRA are evaluated.

7.4 Case 1: eSourcing, Patterns, and eSRA Evaluation

A truck-producing OEM has several suppliers of components that are united in a regional automobile cluster of service providers. Within that cluster the service provider *A* functions as a unique communication party for the entire cluster to the truck producer.

Both, the OEM and the suppliers have historically grown business process that can not be rearranged easily, which is why an external-level process harmonization is required. Thus, the case study applies the *internal-to-external* interaction pattern from Section 4.4.2 of the direction dimension that is explained in Section 3.4.1. The OEM always receives the same system for the truck from the same set of suppliers that are united in a cluster. Within that cluster one supplier serves as the unique communication and coordination party between the OEM and all other suppliers of the cluster. Thus, the *internal-to-external* pattern is combined with the *static* assignment pattern of Section 4.4.1, since the OEM has one predetermined supplier to collaborate with. In this case study it was pointed out in which way components of eSRA (see Section 6.2) are used to support this case study.

Referring to the pattern example of Figure 4.10, the interaction assumes the existence of internal processes in the domains of the service consumer and service provider. The service consumer proposes the creation of an eSourcing configuration. The nego-

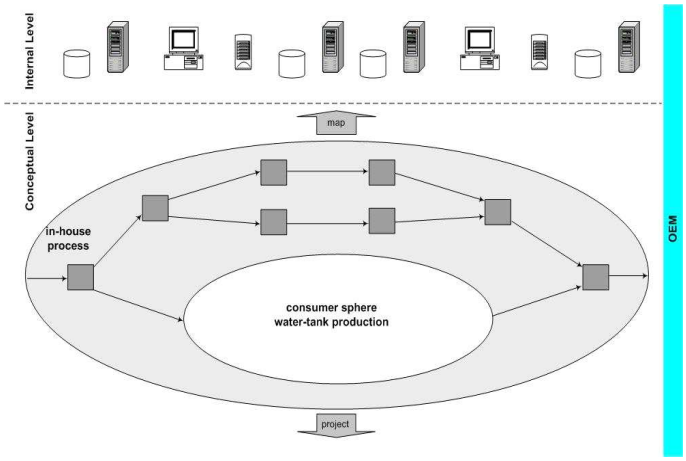

Figure 7.2: OEM in-house process.

tiations about creating a common external-level process are followed by an integration of the contractual spheres in the respective domains. Next, the eSourcing configuration is verified and the extent of monitoring is negotiated. Finally, the enactment of the completed eSourcing configuration commences.

7.4.1 Conceptual-Level Setup

Figure 7.2 depicts the in-house process setup for the OEM. On top, the internal level shows legacy systems that support the OEM's business process. On the conceptual level of Figure 7.2, the in-house process for producing a truck is depicted. Contained in the in-house process is a consumer sphere that delimits a subnet for eSourcing a water tank from service providers. Since for this eSourcing-concept evaluation the original processes are too large to fit into this chapter, the in-house process of Figure 7.3 doesn't use Petri-net formalisms, but a high-level conceptual visualization.

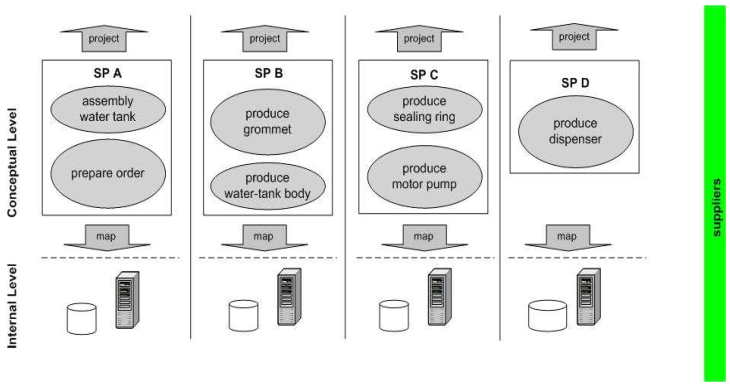

Figure 7.3: Provider domains with their respective provider spheres.

The service provider domains are depicted in Figure 7.3 with their internal setup. Four

provider domains are contained in the figure that are part of an automobile cluster with each containing processes that create parts of a water tank for the OEM. According to Figure 7.3, a service provider may be capable of contributing several processes for producing parts of a water tank. Similarly to the OEM, all service providers have legacy systems on their respective internal levels that need to be integrated in the eSourcing configuration. To do so, the OEM and service providers map their conceptual-level processes on the internal level.

For modelling the conceptual level processes depicted in Figure 7.2 and Figure 7.3, eSRA comprises process-modeler components. Furthermore, the workflow-composer component is available to support the suppliers of the automobile cluster in establishing a composed workflow. The process-modeler and workflow-composer component are part of the Sourcing-setup-support component depicted in Figure 6.4. Although the case study focuses on the business processes, it is assumed that business rules are created with the rules-modeler component that is also depicted in Figure 6.4.

Both processes and rules are stored in their respective databases. To ensure good quality of the conceptual-level processes, the collaborating parties need to utilize the validation and the verification components. Furthermore, each party needs to ensure that conceptual-level processes are mapped to their internal-level legacy systems. Figure 6.3 depicts that processes and rules are delivered to the internal level via the CI translator component of Figure 6.3. Furthermore, in Figure 6.5 the local WFMS and rules engine are the recipients that consequently control web-service wrapped legacy systems.

7.4.2 External-Level Setup

When all the conceptual-level processes are in place in the respective domains, the collaborating parties need to harmonize their respective processes on the external level. In this case study, the OEM proposes to service provider *A* from the automobile cluster the initiation of negotiations for external-level harmonization. The latter party accepts and the OEM and the automobile cluster engage in external-level negotiations.

The OEM starts by filling the gap of the consumer sphere depicted in Figure 7.2 by the rules-modeler and the process-modeler components of the Sourcing-setup-support component of Figure 6.4. Next, the consumer sphere is transferred via the CE translator of Figure 6.3 to the contracting client of the external level. The latter component exposes the OEM's contractual sphere to the service broker of Figure 6.2 that is part of the trusted third party. Note that a trusted-third-party component is not part of the CrossWork prototype. As a result, service brokering is out of focus in CrossWork. Since service provider *A* is the unique contact point for the OEM, the first party has the task to find and organize other service providers for the fulfillment of the externally agreed upon service provision.

Service-provider *A* is notified by the service broker of Figure 6.2 and requests the OEM's contractual sphere for a check. In Figure 7.4 the OEM's consumer sphere is depicted that is entirely projected to the external level. The consumer sphere shows several nested consumer spheres for the production of the water tank. The higher-level consumer sphere is designed to be a fitting subnet within the overall in-house process while each contained consumer spheres is eSourced from a supplier of the automobile cluster.

On the service provider's side of the external level in Figure 7.3, a consensus is created as equal nested spheres are projected. Thus, an overall consensus is given at the end of negotiation phase of the interaction between the OEM and the automobile clus-

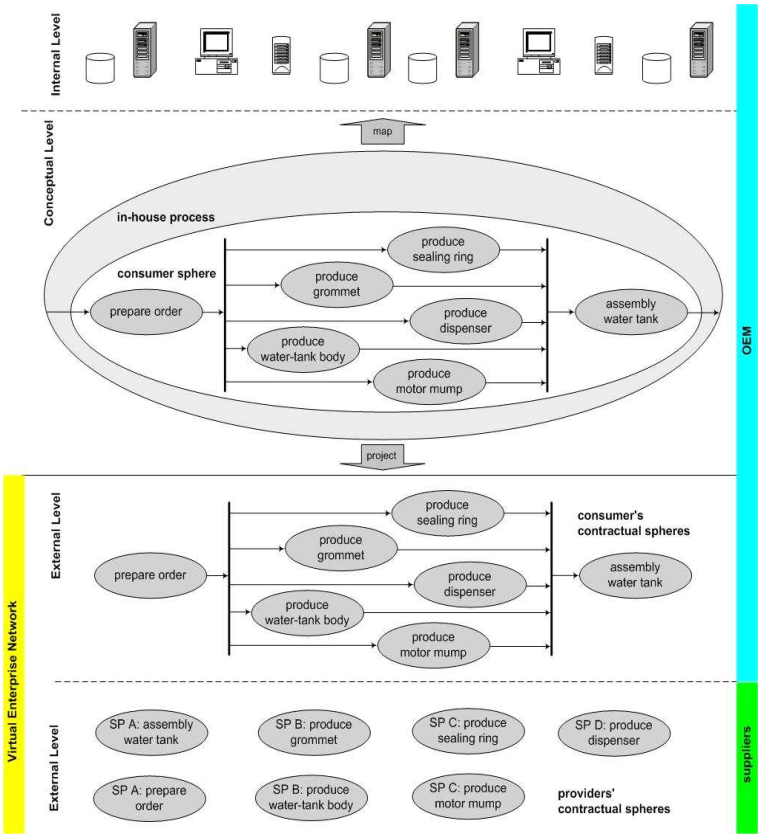

Figure 7.4: External-level contractual spheres.

ter. Optionally, an integration of the contractual spheres of the external level is required on the conceptual level. Such a step is necessary if the negotiation phase has resulted in external-level deviations compared to what is originally projected from the respective conceptual levels. Finally, it needs to be stressed that conceptual-level processes in the domains of the service providers represent refinements of the corresponding external-level contractual spheres. The refinements are additional tasks that are integrated in the provider sphere that remain opaque for the service consumer. In the case study this is illustrated by showing in Figure 7.3 bigger sized ellipses that represent provider spheres compared to the external level of Figure 7.4. For the external-level negotiations the collaborating parties employ from Figure 6.2 the contracting-client component of the Sourcing middleware and service-broker component of the trusted third party.

7.4.3 Selected Spheres in Detail

After the negotiation achieved a preliminary external harmonization of inter-organizational business processes, the following phase in Figure 4.10 of the internal-to-external interaction pattern focuses on the verification of the collaborating processes. Earlier, internal verification ensures that the business processes of collaborating parties enact correctly on their own. However, when intra-organizational processes are

linked inter-organizationally, structural problems may occur. For example, deadlocks that prevent the correct termination of the overall business process.

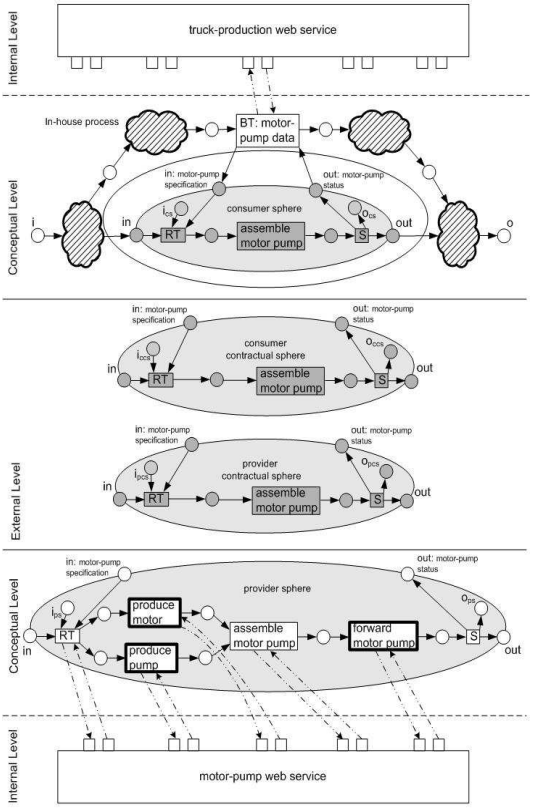

Figure 7.5: Related eSourcing spheres in detail.

In Figure 7.5, a subset of related eSourcing spheres are depicted in detail. The conceptual level of the service consumer shows a consumer sphere that is a subnet of an in-house process. Clouds denote abstracted details from the in-house process. However, it is assumed that an and-split is contained that results in the enactment of parallel branches that is complemented by an and-join at the end. Only one depicted node interacts with the ports of the consumer sphere. This interacting node carries a *BT* label that indicates there is a bi-directional exchange with the consumer sphere modelled, which is part of a conjoinment pattern (see Section 4.5.3). The *BT*-labelled node delivers a motor-pump specification to the *in*-labelled interface place of the consumer sphere after withdrawing such information from the web service of the internal level.

In the consumer sphere a receive node accepts the motor-pump specification. Next, a node is contained for assembling a motor pump. When this node contains further assembly information, the truck producer makes a particular way of assembly mandatory for the service provider if it is assumed the consumer sphere exists before the provider sphere and the contractual spheres. Finally, a send transition returns the status of the motor-pump to the *BT*-labelled node via the *out*-labelled interface place.

The consumer sphere in Figure 7.5 is fully projected to the external. Thus, the

content of the external level consumer contractual sphere and the consumer sphere on the conceptual level are isomorph, which corresponds to a white-box contractual visibility pattern (see Section 4.5.1). According to Section 5.4.3, the service provider has the options of responding either with a grey-box or a white-box projection in order to achieve a contractual consensus.

On the external level of Figure 7.5, the provider contractual sphere is isomorph compared to the consumer contractual sphere, which means the collaborating parties have reached a contractual consensus (see Definition 28). However, the service provider needs to fit the contractual sphere into the internal organizational setup. Additional tasks need to be carried out for manufacturing water-tank components before an assembly takes place and the final water tank needs to be forwarded to the site of the service consumer.

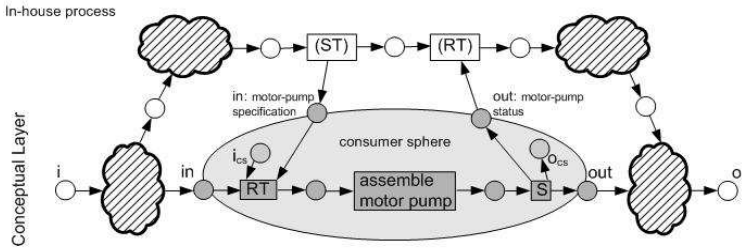

Figure 7.6: Corrected in-house process.

On the conceptual level of the service provider, the contractual sphere is refined by additional nodes, which means a grey-box contractual visibility pattern (see Section 4.5.1) is used by the service provider. Figure 7.5 shows bold lined tasks that represent inserted nodes in the provider sphere. Thus, after receiving the motor-pump specification, the motor and the pump are produced in parallel branches before they are assembled in a joining task. Next, the finished motor pump is forwarded as defined by the truck producer and subsequently quality data is transferred in a document about the motor-pump status to the domain of the service consumer. The tasks focusing on producing and forwarding the motor-pump interact with a web service of the provider.

The created eSourcing configuration of Figure 7.5 must be verified for correct termination before enactment. Thus, the parties independently submit their respective processes to a trusted third party that is depicted in Figure 6.2. In accordance with Theorem 3 in Section 5.6.2, it is not necessary to collapse the eSourcing configuration for checking the correct termination. The reason is that the collaborating parties use a white-box and grey-box projection, which means that local checks of the in-house process and the provider sphere suffice to guarantee the overall eSourcing configuration has sound control-flow. For the eSourcing configuration of Figure 7.5 these local verifications fail because the in-house process can not terminate correctly. The reason is a deadlock contained in the processes that are caused by the arcs of the *BT*-labelled node. As this node needs to wait for the *S*-labelled node to fire, it can never be enabled.

After the local checking of correct termination, the service consumer has to remodel the in-house process. The changed in-house process of Figure 7.6 has the *BT*-labelled node replaced with different conjoinment nodes that establish an exchange between the in-house process and the consumer sphere. Consequently, a repeated local check of the in-house process' control flow by the service consumer succeeds.

7.4.4 Monitorability negotiations

After the inter-organizational processes are verified, the processes need to be linked for a synchronized enactment. In Section 4.5.2 such linking across the domains of collaborating parties is termed monitorability. Several patterns for monitorability and conjoinment were specified for linking processes into an eSourcing configuration [84].

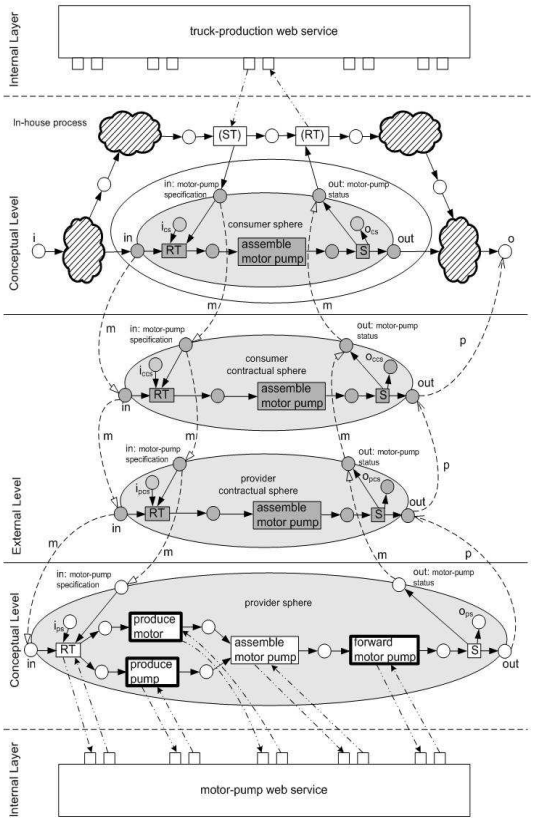

Figure 7.7: Related eSourcing spheres in detail.

In Figure 7.7 two monitorability constructs are used, namely token messaging and token propagation. Token messaging is used for connecting the *in*-labelled interface places. For an enactment application of the eSourcing configuration, token messaging means once the enactment of the in-house process has reached the consumer sphere, such a state is messaged across the organizational domains and the enactment of the provider's provider sphere commences. Token messaging is also used for connecting *in* and *out*-labelled interface places for exchanging the motor-pump specification and the status report across organizational domains. Finally, token propagation is employed for connecting the *out*-labelled interface places of the provider sphere and the consumer sphere via the external level. In the eSourcing configuration of Figure 7.7 this means the enactment of the provider sphere is terminated and this event is communicated to the domain of the OEM where the enactment of the next consumer sphere is starting.

For realizing monitorability, the collaborating parties negotiate directly with each

other without the help of the trusted-third-party component (see Section 6.2.2). Thus, the eSourcing middlewares of the external levels are involved where the global WFMSs and the rules engines need to be linked via the respective coordination interfaces. The agreed upon montorability constructs need to be further realized by appropriately linking the local WFMS and rules engine on the internal level to the conceptual level. When this setup is completed, the enactment of the eSourcing configuration commences.

The next case study is again taken from CrossWork and presented in a condensed version for an eSML evaluation.

7.5 Case 2: eSML Evaluation

The second case study results from CrossWork collaborations with industry partners and focuses on an eSML instance that is used for the external level collaboration definition of two service providers that form a virtual enterprize. This virtual enterprize is publicly advertised to OEM who can integrate the resulting virtual-enterprize service into their in-house process.

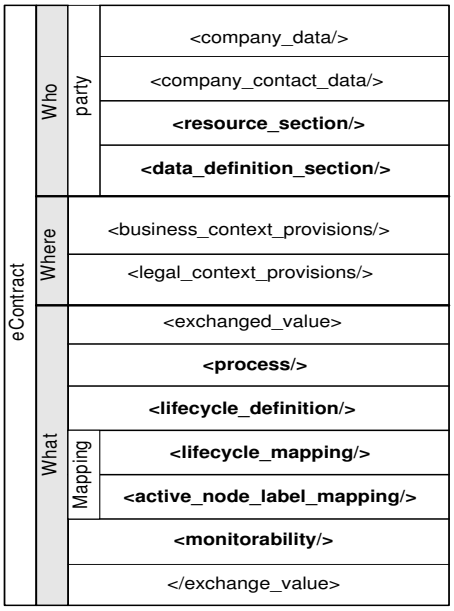

Figure 7.8: An overview of an eSML instantiation.

While the entire eSML instance of this case study is contained in Appendix C, this section explains code extracts that show how eSML is instrumental for specifying an eSourcing configuration on the external level. First, Figure 7.8 gives an overview of the structure in an eSML instantiation. As explained in Section 6.3.1, eSML uses parts of the ECML [23] schema as a foundation. In Figure 7.8 this fact is reflected by considering an entire eSML instance as a contract between collaborating parties and by structuring the eSML content into the blocks Who, Where, and What that are explained in Section 6.3.1. The parts of eSML in Figure 7.8 that are ECML-based are not explained in this case study, namely the definition of company data and company-contact data,

the Where block, and the XRL-based process definition within the exchanged-value definition. In Appendix C code examples are available for the mentioned eSML parts and detailed models explain them in Appendix A.

The bold typed eSML-definition parts in Figure 7.8 are either extensions or modifications that are not part of the ECML foundation. In the Who block extensions for eSML are the resource definition (see Appendix A.5) and the data definition (see Appendix A.10). In the What block, the process definitions are based on XRL, however, extensions have taken place for adopting the conjoinment nodes described in Section 4.5.3 and for linking to the resource- and data-definition sections of eSML. The lifecycle definitions (see Section 4.5.2 and Appendix A.7) focus on the business process and the lifecycle of tasks. In the mapping block of eSML, firstly the lifecycles of the inter-organizationally harmonized business processes are linked, and secondly the lifecycles of tasks from the opposing domains are linked.

Labels of tasks belonging to processes of opposing domains may be considered semantically equal while they actually deviate from each other. To establish a semantic connection, the second part of the mapping block is focussing on the mapping of task labels in the `active_node_label_mapping` tag. Such mapping is relevant for establishing a contractual consensus between collaborating parties. Finally, in the monitorability section of Figure 7.8, it is defined how much visibility is granted to the service consumer (see Section 4.5.2). Next, extracts from eSML code are explained that are contained in Appendix C and it is pointed out how these code extracts related to concepts of previous chapters.

7.5.1 Resource-Perspective Definition

The code extract below is part of the resource definition where an organizational unit is defined as a permanently existing organization. Thus, it is not a unit that dissolves at a certain point in time, e.g., an organization set up for the purpose of managing a project that has a deadline. In line 10 the name of the organizational unit is defined, followed by the definition of the start date. Organizational units may have a business objective assigned. In the code example above this definition is omitted.

```
<permanent_organizational_unit>
    <name>Procurement_Department</name>
    <start_date>2005-01-01</start_date>
    <description/>
    <business_objectives/>
    <resource_nref>
        <resource_type_ref>Department_Head</resource_type_ref>
        <number>1</number>
    </resource_nref>
    <resource_nref>
        <resource_type_ref>Department_Clerk</resource_type_ref>
        <number>33</number>
    </resource_nref>
    <individual_resource>Actor2</individual_resource>
</permanent_organizational_unit>
```

The organizational unit has department members that are defined as roles. These roles are separately specified in the resource section of the eSML file. According to Line 17 the procurement department has one head of department. In Line 16 the `Department_Head` is a unique identifier for an separate extensive role definition.

Similarly, Lines 19-22 specify that the procurement department has a resource assigned with the role name `Department_Clerk`. In the line below the number of individuals is given that slip into the mentioned role of a clerk. Finally, in Line 23 an individual is directly specified as a procurement-department member with the identifier `Actor2`.

7.5.2 Data-Flow Definition

The central part of the data-flow definition in an eSML file is a data package. Such data packages flow through a business process and may even be exchanged to the opposing organizational domain. Following the data-flow pattern specifications [95], in an eSML instantiation, data has a particular visibility ranging from only a task to all instances of a business process and even its environment; a certain interaction type that focuses on the way data is communicated with, e.g, a data package is communicated from a task to another task; or from a block to a different process instance, and so on. Data has different transfer type specifications, e.g, by a copy, a reference, by value, etc. Finally, a data-flow element may be specified to interact with the control-flow perspective, e.g., a pre- or postcondition of data existence for a task, data-value based condition evaluation, etc.

```
<data_definition_section>
    <data_package>
        <package_id>cd</package_id>
        <var_section>
            <string_var tag_name="Bill␣of␣Material" var_id="BOM"
            changeable="false" enabled="enabled">Surrounding Box;
            Gearing</string_var>
        </var_section>
        <document_section>
            <document>
                <document_id>cadDrawing</document_id>
                <name>Cad Drawing of complete GearBox</name>
                <uri>http://www.ve.com/drawings/gearBox.3ds</uri>
            </document>
        </document_section>
    </data_package>
</data_definition_section>
```

The code extract above specifies a data package specifies with a contained variable and a document section. In Line 14 a variable is specified with its attributes. The bill of material is changeable, i.e., the value may be modified, and it is enabled for use. Additionally, Lines 19-23 define a document that is a CAD drawing and is available at a particular `uri`.

```
<receive_transition active_node_id="CO" name="Receive_Order">
    <data>
        <data_flow_direction>input</data_flow_direction>
        <data_package_ref>cd</data_package_ref>
    </data>
    <data>
        <data_flow_direction>input</data_flow_direction>
        <data_package_ref>do</data_package_ref>
    </data>
</receive_transition>
```

Specified above is code where data packages are used with elements in the control-flow of an eSML instantiation. A receive node is specified that receives two data packages from the domain of a collaborating counterpart. In Line 13 the data package with the identifier *cd* is specified as an input. Additionally, in Line 17 the data package with the identifier *id* is equally input to the receiving node. In Appendix C further specification details about the data packages are contained.

7.5.3 Structure of Process-Harmonization Definition

In an eSML instantiation the harmonization of business processes is specified. The code extract below shows how such harmonization is achieved. For every collaborating party an `exchanged_value` section is specified with a service that is either provided or consumed, which depends on the role a collaborating party slips into.

```
<exchanged_value>
    <service>
        <process_section>
            <process tag_name="Gearbox_Production"
            process_id="GB_production">
                <parallel_sync>
                    <sourcing_sphere>
                        XRL-based routing elements
                    </sourcing_sphere>
                    <sourcing_sphere/>
                    <sourcing_sphere/>
                </parallel_sync/>
            </process>
            <lifecycle_definitions/>
            <lifecycle_mappings/>
            <active_node_label_mapping/>
            <monitorability/>
        </process_section>
    </service>
</exchanged_value>
```

The service is represented by an XRL-based (see Section 6.5.2) process definition. Several `sourcing_spheres` may be contained in a process specification. In Line 15 it is specified that three sourcing spheres are embedded in a `parallel_sync` construct, which means the sourcing spheres are contained in parallel branches of control. The sourcing spheres are matched by spheres in `exchanged_value` sections of the same eSML instantiation that belong to opposing collaborating parties. For the matching a grey-box contractual visibility pattern (see Section 4.5.1) is used. In its final state where a consensus is specified in an eSML instantiation, the content of opposing sourcing spheres must match in content. In Line 17 the lengthy XRL-based control flow code is omitted in this code extract. Instead Appendix C contains the entire control-flow code.

In Lines 23 to 26 further eSML constructs are contained that are used for achieving inter-organizational business process harmonization. The constructs for mapping lifecycles and for monitorability specifications are contained in the `exchanged_value` section of the service consumer. The eSML constructs are explained below in detailing code extract.

7.5.4 Lifecycle Definition

In an eSourcing configuration, the heterogeneous system environment of the internal level needs to be inter-organizationally harmonized. The business processes of collaborating parties may have deviating lifecycles on a process and task level. For the enactment phase, it may be relevant to specify a synchronization of the lifecycles (see Section 4.5.2).

```
<lifecycle_definitions>
    <process_lifecycle>
        <lifecycle_sequence>
            <atomic_state name="VE_process_ready"
            tag_name="ready"/>
            <transition name="VE_process_start_enactment"
            tag_name="start_enactment"/>
            ....more lifecycle specifications.......
            tag_name="ended"/>
        </lifecycle_sequence>
    </process_lifecycle>
    <active_node_lifecycle/>
</lifecycle_definitions>
```

Above, it is shown that lifecycles for a process are specified with control-flow constructs. In Line 13 a not further decomposable atomic state is defined. However, in a lifecycle states are possible that contain further nested states. Accordingly, eSML contains a `nesting_state` construct that comprises lower-level states. The lifecycle of a process or a task is propelled by transitions of which Line 15 shows an example. In Line 21 the `active_node_lifecycle` the lifecycle of a task is defined with the same control-flow constructs that are used for the specification of process lifecycles.

7.5.5 Mapping Definitions

If a heterogenous system environment with different lifecycles is harmonized in one eSourcing configuration, it may be important for the enactment infrastructure to specify in an eSML instantiation how the respective lifecycles fit together. As the previous case study shows, in an eSourcing configuration several service providers are included with one service consumer. Thus, for lifecycle harmonization it is relevant to include all service providers. The respective lifecycle steps that are specified as equal may have diverting names but are still semantically equivalent. The same holds for the mapping of task labels from the domains of opposing parties.

The code extract below shows how lifecycles are mapped. In Line 11 the mapping of process lifecycles starts with first specifying the lifecycle label of the service consumer. From Line 16 onwards the semantically equivalent labels of two service providers are specified. For every lifecycle step this specification needs to be repeated. From Line 25 onwards the same specification approach is used for mapping lifecycle steps of tasks that belong to the domains of opposing parties.

```
<lifecycle_mappings>
    <process_lifecycle_mapping mapping_name="process_ready"
    node_type="lifecycle_state">
        <consumer_sphere>OEM_Sphere1</consumer_sphere>
        <consumer_active_node>OEM_process_ready
        </consumer_active_node>
```

```
        <provider>
            <provider_sphere>Provider_SP1_1</provider_sphere>
            <provider_active_node>SP1_process_idle
            </provider_active_node>
            <provider_sphere>Provider_SP2</provider_sphere>
            <provider_active_node>SP1_process_idle
            </provider_active_node>
        </provider>
    </process_lifecycle_mapping>
    <active_node_lifecycle_mapping mapping_name="node_complete"
    node_type="lifecycle_transition">
        <consumer_sphere>OEM_Sphere1</consumer_sphere>
        <consumer_active_node>OEM_active_node_complete
        </consumer_active_node>
        <provider>
            <provider_sphere>Provider_SP1_1</provider_sphere>
            <provider_active_node>SP1_active_node_complete
            </provider_active_node>
            <provider_sphere>Provider_SP2</provider_sphere>
            <provider_active_node>SP2_active_node_complete
            </provider_active_node>
        </provider>
    </active_node_lifecycle_mapping>
</lifecycle_mappings>
```

For the mapping of task labels a code extract is given below. In Line 11 the specification of a service-consumer task label starts by first naming the process a task is contained in followed by the label of a task. In Line 13 similar specifications are given for the domain of the service provider. Differently to lifecycle mappings, there is always one label of the service consumer that is mapped to a label of one service provider because the concept of eSourcing assumes a task is always serviced by one provider.

```
<active_node_label_mapping>
    <consumer_process>GB_production</consumer_process>
    <consumer_active_node>CO</consumer_active_node>
    <provider_process>PP_SP1_1</provider_process>
    <provider_active_node>Local_CO</provider_active_node>
</active_node_label_mapping>
```

7.5.6 Monitorability Definition

The last construct of the process harmonization code in Section 7.5.3 is for specifying with which monitorability patterns the business processes of collaborating parties are linked. The more monitorability patterns are specified, the more enactment progress the service consumer is able to follow. Many monitorability patterns are specified in Section 4.5.2 to cater for differing linking functionalities in a heterogenous system environment of eSourcing configurations.

```
<monitorability>
    <polling/>
    <messaging>
        <transition_messaging>
            <consumer_sphere>SP1_Sphere1</consumer_sphere>
            <consumer_active_node>CO</consumer_active_node>
```

```
            <provider>
                <provider_sphere>PP_SP1_1</provider_sphere>
                <provider_active_node>Local_CO</provider_active_node>
            </provider>
        </transition_messaging>
    <messaging>
</monitorability>
```

An example for a monitorability specification is given above with two parts, namely one for the specification of polling constructs and one for specifying messaging constructs. In Lines 13-20 a transition-messaging monitorability construct is defined. First, the transition identifier in the domain of the service consumer is specified that represents the target node. In Lines 16-19 the source node for the messaging construct is specified that is located in the domain of the service provider.

7.6 Conclusion

In the first case study of Section 7.4 all requirements for eSourcing are evaluated. The case study shows it is feasible to use the framework of eSourcing for inter-organizational business process collaboration. The eSourcing framework fulfills the requirement of scalability as it is demonstrated how an OEM can collaborate with several service providers in one eSourcing configuration. The inter-operability requirement is fulfilled for eSourcing as the case study in Section 7.4 demonstrates how the OEM and the service providers reach a collaboration consensus. With reaching a consensus for inter-organizational collaboration, the setup phase of an eSourcing configuration ends and the enactment phase commences. Finally, the requirement of coherence is fulfilled if it is assumed that the external-level tasks are complemented by conceptual-level business processes where labels match with the external-level contractual spheres.

The eSourcing related pattern catalogue of Chapter 4 is applicable in the case study of Section 7.4. Only a small number of patterns are used, namely the interaction pattern (see Section 4.4.2) called internal-to-external, for contractual visibility (see Section 4.5.1) the service provider uses a white box pattern and the service providers use grey-box patterns. In Figure 7.5 a consumer-initiated bi-directional conjoinment pattern is (wrongly) applied and in the initial setup phase of the eSourcing configuration of Figure 7.7, two one-directional conjoinment patterns are used. Finally, several monitorability patterns (see Section 4.5.2) are used for linking the consumer sphere, the contractual spheres, and the provider sphere with each other. Finally, by using the eSourcing patterns, the structuring of collaborations for eSourcing configurations is possible. That is demonstrated by Figure 4.13 where the deadlock problem caused by a wrong application of a bi-directional conjoinment pattern in Figure 7.5 is solved by using two one-directional conjoinment patterns instead.

For the case study in Section 7.4, the application of eSRA components for realizing an internal-to-external interaction pattern demonstrate the completeness of the reference architecture. For sake of brevity it can not be demonstrated in this chapter how all interaction patterns are supported by eSRA components. Furthermore, it is not investigated in detail how eSRA supports the enactment phase of an eSourcing configuration. The interoperability requirement of eSRA is also fulfilled in Section 7.4, as it is shown how the processes of collaborating parties are harmonized on the external level of an eSourcing configuration. By using a conceptual level, an abstraction is achieved from the technical features of legacy systems on the internal levels of the

respective collaborating parties. The conceptual-level processes are harmonized on the external level of the eSourcing configuration. By having many monitorability patterns available for linking the processes of collaborating parties, the heterogenous nature of legacy systems is paid attention to.

In the second case study of Section 7.5, it is demonstrated with code extracts how eSML is instrumental for structuring the inter-organizational collaboration of business-process harmonization. The code extracts are taken from a full eSML instantiation in Appendix C. The data-flow support requirement is realized in eSML by adopting data-flow patterns [95] and combining them with conjoinment and monitorabilty elements for inter-organization data-flow specification. The resource-specification support requirement is fulfilled by integrating an extensive resource model into eSML (see Appendix A.5). Both data-flow and resource support are integrated with the control-flow elements of eSML. Although only code extracts of eSML are presented in Section 7.5 and only one full eSML instantiation is given in Appendix A.5, these examples show that eSML fulfills the applicability requirement.

Chapter 8

An Outlook for eSourcing

Contents

The concept of eSourcing increases the effectiveness and efficiency for enterprizes in electronically integrating and enacting inter-organizational business processes. In this chapter, an outlook for eSourcing is presented that addresses the issue of additionally required mechanisms for dynamic and transactionally safeguarded e-business collaboration. In the context of eSourcing, dynamic stands for automatically integrated business process relations that are forged between business parties by matching structures of respective processes. Additionally, it is important to ensure the safety of dynamic e-business collaboration with using an electronic business transactions concept. In this PhD thesis the focus for eSourcing lies on matching business processes inter-organizationally. For this outlook chapter it is explored what the context of eSourcing is, what dynamic mechanisms are lacking for fitting into this context, and what the features of a suitable transaction concept are for automatically integrated business process relations.

8.1 Introduction

The concept of eSourcing is embedded in a broader framework with the objective of automatically integrated business process relations that can be characterized by the following key features. Firstly, this broader framework that eSourcing is embedded

in, is an open collaboration system with several participating parties, resources, rules, processes, etc., that is in exchange with its environment. For example, the participating parties are service providers, service consumers, various mediating parties, and so forth. The resources may consists of production materials, machines, organizations, and so on.

Since automatically integrated business process relations are open, the structure is not stable as the relationship of the parties, resources, rules, processes changes over time. For example, a mediator must be added, the service provision must be modified, instead of paying one amount, the payment is performed in parts after certain periods. Furthermore, automatically integrated business process relations are complex as they comprise many highly interrelated elements. When the elements of a business relationship change or the relationships between involved elements, a new state is entered that may change again after some time. For example, in an existing e-business collaboration an additional service provider must be included, or a service provider must be eliminated from an e-business collaboration because of a decreased level of trust. Dynamic mechanisms provide a feedback loop that propels an e-business collaboration from one state to the next.

When the broader context of eSourcing is considered, scope exists for dynamic e-business collaboration extensions in the eSourcing concept. This chapter explores how eSourcing relates to and overlaps with other concepts that are relevant for automatically integrated business process relations as their integration promises a total automation of the setup, enactment and post-enactment By exploring and relating dynamic mechanisms and drafting an e-business transaction concept for safeguarding these business relations, the extension potential for eSourcing is detected.

The structure of this chapter is as follows. In Section 8.2 the context of eSourcing is explained based on a figure. Next, Section 8.3 proposes dynamic mechanisms that eSourcing should adopt for better supporting automated business relations. To safeguard the automatically integrated business process relations, an e-business transaction concept is outlines in Section 8.4. Finally, Section 8.5 concludes this chapter.

8.2 The Context of eSourcing

Based on the description of features in the introduction, automatically integrated business process relations are complex, open, dynamic systems of interrelated elements that pursue the objective of providing inter-organizational business services for financial compensation. Such systems experience state changes of elements and their relationships, which are governed by dynamic mechanisms that prevent the business relations from turning into a chaotic system of unpredictable behavior.

In Figure 8.1, the context of eSourcing is modelled by showing several contained concepts. To tackle the involved complexity three dimensions are depicted. One dimension is a lifecycle of an automatically integrated business relation that has a setup, enactment, and post-enactment phase [23]. During the setup phase an offer is made to form an eCommunity [67] that is eventually defined in a business-network model (BNM). Changes of community members and their relationships result in updates of the BNM, e.g., the service provision is modified, a party of the e-business collaboration changes, and so on. Such changes result in a new state of a community, which is visualized by a separate dimension in Figure 8.1. The collaborating parties engage in forming one or many eContracts [23] until signatures exist for them and the eContract(s) are stored. An eContract is defined in accordance with the earlier definition

given in Section 6.3.1.

Following Section 6.3, a real-world contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights. During an eContract enactment, service provisions are consumed for some compensation. In this phase exceptional eContract-enactment situations may occur that are either resolved or lead to the termination of individual enactment phases. Finally, the post-enactment phases characterized by compensations and rollbacks if any exceptions are not resolved during the enactment phases. Eventually, when no eContracts need to be established and enacted any more, a community dissolves.

The third dimension focusses on establishing a separation of concerns with the values pragmatic, semantic, and syntactic. Pragmatic collaboration captures the willingness of parties to perform the necessary activities. The willingness to participate involves the capability to perform requested actions and policies that dictate whether the action is preferable for a party to be involved in an eCommunity. Semantic collaboration means that a message content is understood in the same way by a sender and receiver. Finally, syntactic collaboration means that messages can be transported from one application to another and correctly processed.

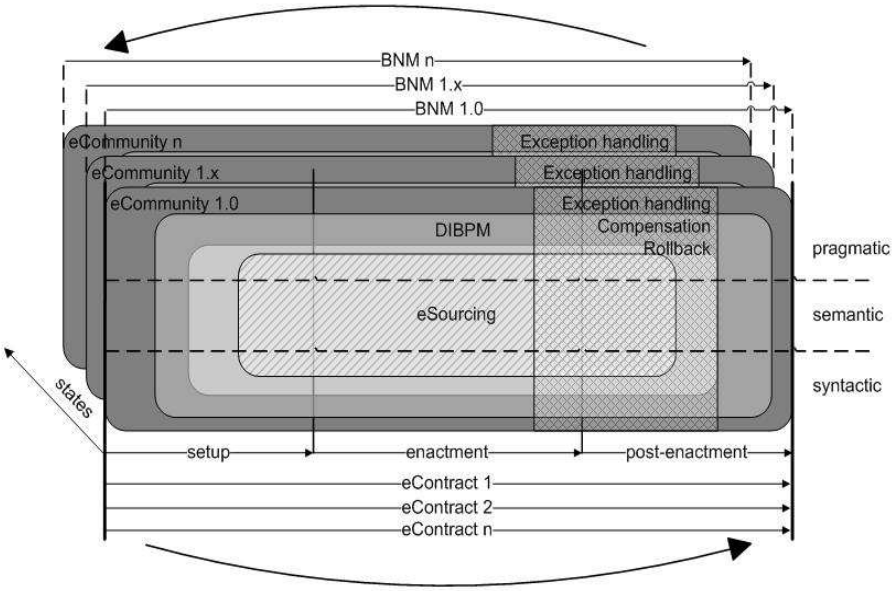

Figure 8.1: The context of eSourcing.

The context of eSourcing is summed up in Figure 8.1 that shows nested concepts of which eSourcing is the core concept. Following Section 3.3, eSourcing is embedded in DIBPM [50], which is defined in Section 3.2. On the highest level, the concept of eCommunities [67] is depicted in Figure 8.1 that contains the first two mentioned concepts. An eCommunity is a specific collaboration with special operations, agreements and states. An eCommunity carries identities and is managed according to the BNM information. Collaborating parties join or leave an eCommunity either voluntarily or by community decision.

The differences between the concepts depicted in Figure 8.1 are as follows. While

eSourcing is a concept for externally harmonizing business process views, the eCommunity approach is supporting a federated model of interoperability of systems that does not deal with inter-organizational business-process management, but monitors the interactions between the domains of collaborating parties that are part of an eContract. eSourcing provides an infrastructure for integrating business processes with a different degree of mutual content visibility and enactment monitorablity. In contrast, eCommunity management facilities provide support for loose coupling and emphasizes the need of predefined, dynamic (breeding environment) type discipline for the benefit of automated negotiation and monitoring. In this process, the harmonization of eContracts takes place while defining models and types.

To realize this broader framework of Figure 8.1 for the objective of automatically integrated dynamic business process collaboration, eSourcing needs to be extended to allow an integration with the broader context. In the following section this extension scope for eSourcing is explored by investigating important dynamic mechanisms that are part of the concepts of eCommunities [67] and DIBPM [50] and discussing how they relate to the current state of the art of eSourcing.

8.3 Dynamic Mechanisms for Extending eSourcing

Based on Figure 8.1, dynamic and automatically integrated business process relations require supporting mechanisms that go further than inter-organizationally harmonizing business processes. By evaluating the concepts of Figure 8.1, a gap exists between eSourcing and the depicted broader context. As business relations between collaborating parties are arranged in communities that experience state changes, mechanisms for their life-cycle management are of importance. The collaborating members need to negotiate on the one hand the conditions for their community membership and also the content of eContracts. Thus, mechanisms for negotiation support are relevant to fill the gap. Next, mechanisms for the collaborative establishment and management of such eContracts must be adopted. As services are important at all stages of a business collaboration, mechanisms for service management should be adopted.

In the following sections, the mentioned dynamic mechanisms for automatically integrated business process relations are further elaborated upon. These dynamic mechanisms are inspired by the eCommunity concept [67]. First, the category of mechanisms is described and consequently specific types are explained. It is stated to which extent dynamic mechanisms are already present in the eSourcing concept.

8.3.1 eCommunity-Lifecycle Management

An eCommunity lifecycle is a series of stages through which a business collaboration moves. The specific functional blocks supporting a lifecycle are discussed below, of which all are currently part of the eCommunity [67] concept. The following mechanisms are listed in accordance with an eCommunity lifecycle.

1. The lifecycle starts with a *BNM former* that is instrumental for the formation of a BNM where defined roles and interactions between roles are governed by policies and consolidated so that the structure and behavior of a collaborating community is set up. In this case a first negotiation between potential partners involves comparing and matching strategic and pragmatic goals in the network.

2. The *populator* represents a breeding process where services are selected for eCommunity roles. Here a match needs to be established between the candidate attribute values and constraints for roles in a BNM.

3. The *BNM joiner* goes a step further and establishes a grouping of similar models with suitable transformers and adapters for the configuration of communication channels so that information exchange becomes understood correctly. Furthermore, a deadlock free message exchange needs to be ensured.

4. Next, the *membership manager* retrieves potential participants for an eCommunity for the defined roles of a BNM based on a service-type repository where providers publish their service offers. The resulting eCommunity contract is a subject for re-negotiation that results in a trajectory of states.

5. The *state support* is a mechanism that manages the change of major reorganizations of the collaboration structure in a BNM. A state is a period where the roles and services of the BNM participants are stable while subsequent states have different roles and services involved.

6. Finally, the *state-transition synchronizer* governs state changes based on transition rules. Some participants that fill BNM roles reappear in the next state, while others leave the community. The transitions between states lead to synchronizations between partners where the constraints in a BNM that govern their behavior need to be adjusted.

Mapped to Figure 8.1 the first four dynamic mechanisms are part of the setup phase and the latter two mechanisms are part of the enactment phase. A post-enactment phase of an eCommunity may be considered a dynamic mechanism that results in an orderly termination where the members agree to part, the behavior governing rules and linked business processes are uncoupled, and the technological infrastructures in the respective community-member domains are disconnected (see Section 7.4).

In the eSourcing concept this dynamic eCommunity-lifecycle management mechanism is not contained. However, since the scalability of the eSourcing concept may extend to more than two collaborating parties, the integration with mechanisms for managing the lifecycles of eCommunities is feasible. For an extension the results of the described function blocks need to be reflected in eSML instantiations. For example, it can not be possible that a collaborating party is defined in an eSML instantiation that isn't member of a community. Furthermore, the roles defined during the BNM formation must be reflected in eSML. Also the policies governing the interaction between collaborating parties need to be adopted in individual eSML instantiations. The information exchanges between communication channels must be reflected in the eSML data-flow specifications. Finally, the deadlock-freeness of message exchanges can be ensured by correctly terminating inter-organizational business processes and the services integrated in a BNM are also part of an eSML instantiation.

8.3.2 Negotiation Support

Several functional blocks are contained in this dynamic mechanisms that is fully represented in the eCommunity concept and partly in DIBPM. Negotiation is a process where collaborating parties address disputes, agree upon courses of action, bargain for individual or collective advantage, or craft outcomes that serve their mutual interests.

The *eCommunity membership negotiator* is used during the setup phase and the enactment phase. In the first case members need to match their strategic and pragmatic goals to form an eCommunity. During the enactment phase the membership may be altered. For example, a member may be replaced with a different one on an own decision or because of being voted out by the other eCommunity members. The membership negotiator is a pragmatic mechanism because a willingness must exist to be part of an eCommunity or not.

The *BNM negotiator* supports the establishment of BNMs that define the collaboration of eCommunity members. A BNM defines the structure and behavior of the collaborating community through roles and interactions between them. Additionally, assignment rules are defined for the service offer, and conformance rules are instrumental to limit acceptable behavior during an eCommunity operation. The BNM negotiator is used during the setup and enactment phase of a business collaboration and applied as a pragmatic mechanisms.

The *coordination elector* generates an eCommunity member who functions as a coordinator who is elected during the first negotiation round among the participant candidates. The coordinator gathers and considers the agreement proposals, disagreements, and possible counter offers and merges them into one contract. A coordination elector is used during the setup phase and the enactment of a business collaboration. During the enactment phase a contract may be subject of modification requests for which a coordinator is instrumental. Furthermore, this dynamic mechanism is pragmatic as the wills of collaborating parties are collected for generating a coordinator.

A *policy negotiator* is responsible for the collaborative establishment of a set of guiding policies. These policies govern the alternative behavior patterns in a BNM. Thus, the policies dictate whether a potential action is preferable for a collaborating party to be involved in. A policy negotiator is used in the setup and enactment phase of a business collaboration and is of a pragmatic and semantic nature. A variant of a policy negotiator is also part of DIBPM when it is taken into account that contractual policies are also part of an eSourcing configuration.

Finally, the *eContract negotiator* supports negotiations on the exchanged values, the content of the eContract, and so on. On a subordinate level the *business-process negotiator* is a central mechanism for the collaborative establishment of an inter-organizational business process. Thus, the result of such negotiation in the case of automatically integrated business process relations may be an eSourcing configuration as presented in previous chapters. The business-process negotiator is used during the setup and enactment phase of a business collaboration and is of a pragmatic and semantic nature. The mechanism needs to respect clear semantics because there must be mutual clarity about the meaning of used constructs in a business process. In DIBPM a variant of a business-process negotiator is part of the concept.

In the eSourcing concept there is no support for the first four function blocks. For the eSourcing concept an adoption to these function blocks means that eSML instantiations must be updated every time the negotiations result in a change of community members, rules, or policies. However, for the function block termed eContract negotiator, the eSourcing concept provides a foundation. As eSML instantiations contain inter-organizationally harmonized business processes and supporting policies in the form of contractual business rules, the policy negotiator and eContract negotiator mechanisms is supported. The existence of a service-broker and auction-service components (see Figure 6.2) of eSRA is a supporting foundation for the latter two mechanisms.

8.3.3 eContract Management

The main issue for the eContract management is the detection and management of minor and major breaches of contractual agreements. Minor breaches can be resolved and major breaches result in the termination of an eContract. Such breaches are perceived by a *breach detector* that notifies a *recovery component* that decides whether a recovery action within the domain of a collaborating party is sufficient or if the occurrence may be ignored all together.

For a serious breach that can not be resolved locally, a *breach manager* is activated that initiates a negotiation between the eCommunity members in which the corrective actions are decided. The breach manager is used during the enactment and post-enactment phase. If no corrective action can be agreed upon, the eCommunity is terminated.

In the eSourcing concept no eContract management mechanisms are included yet. However, an adoption is necessary as the eSourcing concept also uses eSML instantiations for formulating the contracts between collaborating parties that may be breached during enactment. For integrating eContract management, an extension of eSRA on the external level is required where the contracting-client component of Figure 6.2 is equipped with a breach detector, breach manager, and recovery component. For further details about eContract management, in [23] an elaborate reference architecture for electronic contracting is described.

8.3.4 Service Management

In automatically integrated business process relations, services that are implemented by business processes are heavily used at every stage of an eContract. Service management is relevant for all concepts that are contained in Figure 8.1.

The *service discoverer* uses a repository with service offers that need to be matched with service types contained in a BNM or in an eContract. A service type is an abstract definition of business service functionality. The service publication functionality for registering services in a shared repository may be similar to UDDI [29]. The service discoverer scans through the repository and generates a set of service offers that are candidates for service-type matching based on published functional properties such as the service-interface signature, service behavior, requirements for technical bindings, and so on.

Next, the *service selector* chooses a particular service offer for a service type that is part of the set generated by the service discoverer. A service offer is chosen based on trust and service quality related evaluations that are non-functional properties. After a performed conformance validation the match with the service type needs to be established.

Finally, the *local service manager* the the domain of each collaborating party controls the local operating environment and is involved in message exchanges with the eCommunity contract that is an agent itself. The local service manager use knowledge about local services and their various management methods.

For the automatically integrated business process relations framework depicted in Figure 8.1, an exploration of service-management mechanisms on all contextual levels of eSourcing is necessary. The application of services is instrumental for all lifecycle stages of an eCommunity and the eContracts it comprises. In eSRA a service broker allows a collaborating party to publicly register services. However, neither is a trust evaluation component integrated in eSRA nor are quality of service issues considered.

Local service management in eSRA is situated on the internal level (see Figure 6.5) where a local workflow management system and a local rules engine orchestrate legacy systems.

The post-enactment phase of automatically integrated business process relations is not extensively covered by the discussed dynamic mechanisms of this section. The next section outlines the ingredients of an e-business transaction concept for automatically integrated business process relations that is meant to safeguard dynamic inter-organizational business collaboration.

8.4 Outlining an eBusiness-Transaction Concept

For eSourcing, a transaction concept is important to ensure reliability in business collaboration. However, with the apparent complexity involved in eSourcing and its broader context of automatically integrated business process relations, no single transaction model is able to meet all requirements. Traditionally, a transaction is a unit of interaction with a database management system that must be treated in a coherent and reliable way independent of other transactions. From a technical standpoint, current web service composition approaches are confronted with the transactional challenges of relaxed atomicity, where intermediate results may be kept without rollback despite the failure to complete the overall execution of a composite service. Secondly, composed services are dynamic as they can be automatically selected at run-time based on specific requests [80]. A solution approach is to compose web services also from a transaction perspective where first the transactional requirements are defined together with acceptable termination states. During the web-service composition phase, the transactional properties of every service can be matched with these prior defined transactional requirements. However, for eSourcing such a technology centered transaction concept does not suffice as business relevance must be taken into account.

As previous chapters show, in eSourcing configurations spheres are used to demarcate process parts that are provided by a collaborating party. The theory of spheres of control [31] originates from the domain of traditional database transactions. So called workflow spheres [69] expand the transaction theory into the dynamic world of complex business processes. Those concepts are applied in [105] for analyzing atomicity criteria dependencies and atomicity spheres. This work, does not relate the workflow concepts of highly dynamic inter-organization processes. In the work of [104] a substantial emphasis is put on the characteristic atomicity properties of e-business. These unconventional atomicities for spheres in eBT are explored and related [87] to each other along the categories system-level atomicity, business-interaction atomicity, and operational-level atomicity. These atomicities need to be part of a transaction model that pays attention to the business realities that form the context of an eSourcing configuration.

8.4.1 Business-Aware Transaction Models

An e-business transaction (eBT) concept must go further than transactions for the database domain if it should support eSourcing and if it needs to ensure consistent state changes of a business collaboration. An eBT needs to reflect the operational business semantics that accompanies inter-organizationally harmonized business processes. Thus, the business needs and their objectives between collaborating parties drive business transactions, e.g., business commitments, mutual obligations, the exchange of

monetary resources, and so on.

eBTs are long lived, involve collaboration at multiple levels, are characterized by unconventional behavioral features, and include multiple parties that exchange services for compensation. The successful completion of a business transaction results in consistent state changes that reflect the objectives of multi-party business collaboration. Furthermore, business transactions are a means of ensuring that collaborating parties have a common semantic understanding of their collaboration.

In [88] the key components of business transactions are described, namely commitment exchange, the parties involved in the commitment, business constraints and invariants that apply to the messages exchanged between collaborating parties, and business objects that are operated upon by business activities. In eSML these key components are contained. An eSML instantiation represents a definition of a commitment exchange between collaborating parties that is only complete, when the collaborating parties reach a consensus about the contained commitment exchange definition. In an eSML instantiation the collaborating parties can be specified. The definition of business constraint and invariants is supported in an eSML instantiation in the form of business rules and definitions for the Where concept (see Section 6.3.3). Finally, in eSML business objects in the form of documents can be defined and assigned to tasks.

The foundation for supporting the definition of business transactions is created in eSML. However, for integrating eBTs, the eSourcing concept needs to be extended with an extra XML-based specification language for the setup time that describes additional elements for business transactions, e.g, quality of service or specifying common business functions like payment, certified delivery, non repudiation, goods, and so on. The Business Transaction Model Language BTML [88] comprises of elements that allow to specify such special business transaction elements.

On the internal level of an eSourcing configuration, many legacy systems safeguarded by diverse transaction concepts are located that must be inter-organizationally integrated. The following section discusses ongoing research about a business transaction framework (BTF) that may enable eSourcing configurations to integrate inter-organizationally the differing transaction concepts of internal-level legacy systems.

8.4.2 Abstract Transaction Constructs

The need for a comprehensive and flexible transactional support is addressed in the XTC project (eXecution of Transactional Contracted electronic services) [112] that is funded by the Dutch Organization for Scientific Research (NWO). By means of a BTF, the XTC project lays a transactional foundation for processes in contract-driven and service-oriented environment. In XTC, existing transaction models are integrated into abstract transaction constructs (ATC) so that their implementation details are hidden. The heterogeneous infrastructures of eSourcing configurations are catered for by selecting ATCs to form an overall transaction scheme. Thus, ATCs are building blocks of a BTF that are composed into specific transaction models

8.5 Conclusion

This chapter presents an outlook for eSourcing that explores what dynamic mechanisms are required for supporting the setup, enactment, and post-enactment of business networks and associated eContracts in dynamic electronic business collaborations. Automatically integrated business relations comprises several contained concepts with

eSourcing as a core embedded in DIBPM and the eCommunity concept. The members and their behavior-governing rules of an eCommunity are defined in business-network models that have one or several eContracts associated. For supporting such automatically integrated business process relations, dynamic mechanisms are investigated that eSourcing must either adopt or connect to.

Partly hierarchical, unconventional e-business transaction atomicities are considered for safeguarding eSourcing configurations. These unconventional types of atomicities need to be employed for securing on the one hand the business-interaction protocols of collaborating parties during the setup phases of electronic communities and electronic contracts. On the other hand these atomicities also safeguard during enactment the consumption of service provision for corresponding compensation. In an eSourcing configuration the spheres must be additionally demarcated with e-business atomicities.

The ingredients of an e-business transaction for eSourcing are explored. Research results from the ongoing XTC project are considered where a business transaction framework is established by combining differing abstract transaction constructs that are ordered in a taxonomy. For eSourcing configurations, these ATCs need to be adopted for establishing a composed transaction that spans across the domains of collaborating organizations and incorporates the conceptual and internal business processes that orchestrate heterogenous legacy systems. The advantage of ATCs is that different transaction concepts can be composed to accommodate this heterogenous system environment that is integrated in eSourcing configurations.

Chapter 9

Conclusion

Contents

In this thesis several results can be distinguished that are summed up below and combined with an assessment of the research results. Next, final remarks for this thesis discuss remaining issues for the eSourcing concept that could not be sufficiently addressed.

9.1 Summary of Research Findings

The answers given below sum up the research findings of this thesis. The sequence of subsections corresponds to the sequence of research questions listed in Section 1.4.2. First, every subsection answers the main research question and their additional subquestions. The answers are followed by concluding remarks.

9.1.1 Formal Preliminaries

In providing a formal grounding for the concept of eSourcing, Petri-net formalism is used for which a considerable body of theory exists for the domain of business processes. By employing specific Petri-net theory, the semantics of the control-flow perspective of business processes is unambiguously defined. A subclass of Petri-net theory is adopted, namely WF-nets. For inter-organizationally linking business processes, IOWF-nets are adopted that can be flattened to a place/transition-net.

By adopting WF-nets, the soundness property is available for verifying the correct termination of processes. After applying the flattening method, the soundness property

can also be used for verifying the correct termination of IOWF-nets. In using projection inheritance for comparing consumer spheres and provider spheres, collaborating parties retain a business relevant degree of internal adjustment flexibility that remains opaque to the opposing collaborating party. That way, organizations are enabled to keep their business secrets and must not have fixed, standardized routing imposed on themselves. With the adoption of clear control-flow semantics, it is possible to check a configuration at the end of the setup phase of an eSourcing configuration for problems that render a successful enactment impossible. That way the need for expensive exception handling and compensation steps is reduced.

9.1.2 The eSourcing Concept

As a framework for conducting dynamic inter-organizational electronic business, this thesis introduces the eSourcing perspective that supports the establishment and enactment of inter-organizational business process collaboration. eSourcing integrates three crucial issues, namely conceptual, business, and technological complexity management; control-flow rigor; and relevance for the domain of investigation, i.e., dynamic inter-organizational business process collaboration.

A three-level business process framework is used in eSourcing to manage the conceptual, business, and technological complexity involved in inter-organizational business process collaboration. The informally introduced collapsing method for eSourcing allows the eSourcing parties to check during build time if they adhere to process-behavior requirements and whether the enactment of an eSourcing configuration can successfully terminate. These checks are possible without revealing the internals of the collaborating opponent.

With eSourcing a framework is established for the structural matching of business processes. It allows companies to identify the benefits of carrying out electronically inter-organizational business process collaboration where a conceptual harmonization between collaborating domains takes place. It is not required to directly link the respective information infrastructures, and collaborating parties have the freedom to protect their competitive advantages.

9.1.3 A Pattern-Based Exploration

For discovering patterns for eSourcing, a suitability analysis forms the foundation, which is a top-down exploration approach resulting from exploring relevant characteristics of dynamic inter-organizational business process collaboration. To explore the eSourcing framework for its setup phase and for its structural elements, the following method is chosen. Dynamic inter-organizational business process collaboration contains several feature dimensions in the form of axes that create a logical space. On every axis, dimension values are located that detail the feature an axis represents. By taking a subset of axes, a logical space is created that represents a particular perspective.

Many pattern catalogues exist for perspectives that are relevant to dynamic inter-organizational business process collaboration. While their specifications share many commonalities, the terminology used for specifying patterns for different perspectives is not uniform. Still, the pattern catalogues form a foundation for exploring the nature of patterns and for extracting a common specification terminology. The results of this exploration result in a pattern meta-model for uniformly specifying patterns,

ordering the patterns in a taxonomy, and capturing information about technology support of specific patterns that establishes relevance for eSourcing. Deducted from the suitability analysis, two multi-dimensional spaces are created for exploring the setup phase and structural elements of eSourcing configurations. Firstly, the interaction dimensions called assignment and direction and secondly the construction dimensions called contractual visibility, monitorability, and conjoinment are used for discovering and specifying patterns.

Using the specified patterns allows for an evaluation and integration of intra-organizational business processes across the domains of collaborating parties. By using conceptually formulated and technology independent patterns for the establishment of eSourcing configurations, it becomes unnecessary for intra- and inter-organizational knowledge workers to continuously "reinvent the wheel". The insight gained by specifying interaction patterns about how collaborating parties are in exchange with each other during the setup phase of an eSourcing configuration, is input for developing a reference architecture that supports the setup- and the enactment phase of eSourcing configurations. The construction-element patterns are used for developing a specification language for eSourcing configurations.

9.1.4 Formal Properties of eSourcing

Shifting to a formal exploration of eSourcing, different options for reaching a consensus about service provision and service consumption are defined. To reach a consensus, the projected contractual spheres of collaborating parties must be isomorph on the external level of an eSourcing configuration. The projection options result in three relationship variants between the consumer sphere and the consumer contractual sphere, and the provider sphere and the provider contractual sphere, namely a white-box, grey-box, or a black-box projection. To avoid enactment problems, the service consumer may only use white-box and black-box projection, while the service provider may use all three projection options. For reaching a contractual consensus, if the service consumer uses white-box projection, the service provider may either respond with white-box or grey-box projection, while black-box projection must always be complemented with black-box projection by the collaborating counterpart.

On the conceptual level of the consumer domain, an eSourcing configuration consists of an in-house process with a consumer sphere as a contained subnet. On the external level, a contractual sphere for every collaborating party is located and on the service provider's conceptual level the provider sphere is situated. The in-house process and the provider sphere form an IOWF-net. To perform service refinement without violating the externally agreed upon service provision, process refinement patterns are available, namely a parallel branch, a loop, and an inserted task. These patterns ensure that the projection-inheritance relationship between the consumer sphere and the provider sphere is not violated if white-box or grey-box projection are performed.

eSourcing is backed by a proven theorem from IOWF-nets about the compositionality of projection inheritance that states under certain conditions, a subflow can be replaced by a subclass subflow without endangering soundness. However, further research is required for supporting a setup phase of an eSourcing configuration where black-box projection is used. This projection type offers collaborating parties design freedom for their conceptual-level business processes with the drawback that it is challenging to achieve a sound eSourcing configuration. Instead, it should be formally explored how the compatibility and usability of the consumer sphere and the provider sphere can be checked during the setup phase of eSourcing configurations.

9.1.5 Prototypes for eSourcing

Based on the electronic contracting language ECML, the eSourcing-formulation language eSML is constructed for the external level of an eSourcing configuration. The eSourcing pattern catalogues of this thesis are used for constructing the syntax of eSML. By employing additional pre-existing pattern catalogues for the data-flow and resource perspectives, further XML constructs are included in eSML. Respecting design principles that pay attention to a separation of business, conceptional, and technological concerns, the eSourcing reference architecture called eSRA is constructed for the development of applications that support the eSourcing setup phase and the enactment, which is embedded in the three-level framework.

With the adoption of the process modelling language XRL in eSML, contained business process definitions have clearly defined control-flow semantics. That way the Woflan application becomes instrumental for verifying the soundness property of a collapsed eSourcing configuration *before* enactment. The XRL formulated business-process specifications that are contained in eSML instantiations can be validated with the web-based toolset XRL/flower. XRL/flower is built on existing technology that allows for an efficient implementation and the system is easy to extend by employing an XSL translator for mapping routing elements to PNML. Thus, for supporting a new control flow primitive, only a transformation to the Petri-net format needs to be added and the engine itself does not need to change. The XRL enactment application is complemented with a web server, allowing actors to interact with the system through the internet. The pattern-knowledge base application uses a pattern meta-model as a foundation and stores pattern catalogues of various business-process collaboration perspectives. A pattern lifecycle is the starting point for establishing a reference architecture for a pattern knowledge base. The pattern meta-model is extended for capturing additional information about the knowledge-base users and review procedures of patterns.

9.1.6 Case Studies

The thesis illustrates two case studies from the automobile industry that were conducted for the CrossWork project. These two case studies are instrumental for evaluating a set of functional and non-functional requirements for eSourcing and the related pattern catalogues, eSRA, and eSML. The first case study describes the application of eSourcing with the related pattern catalogues and eSRA in a scenario where an OEM is eSourcing from an automobile cluster that collaboratively produces, assembles and delivers a water-tank. The second case study shows how eSML is applied for specifying on the external level of an eSourcing configuration the collaboration for producing a gear box. The requirements are for eSourcing feasibility, scalablity, interoperability, and coherence; for the pattern catalogues applicability and structuring support; for eSRA interoperability, applicability, and completeness; for eSML applicability, structuring support, data-flow support, and resource specification support.

The case studies in this thesis provide an illustration of feasibility for the eSourcing approach. However, the considerable complexity of eSourcing and the related pattern catalogues, eSRA, and eSML requires the conduction of additional, more elaborate case studies with additional requirements for evaluating a more complete list of functional and non-functional requirements.

9.1.7 An Outlook

The broader context for eSourcing is explored to fill the gap for realizing dynamic inter-organizational business process collaboration. Consequently, eSourcing is embedded in additional tiers of broader concepts with DIBPM being a second encapsulating tier. As a third tier, the eCommunity concept contains the first two mentioned concepts. An eCommunity is a specific collaboration with special operations, agreements and states.

The detected need for additional dynamic mechanisms in eSourcing is inspired by considering the eCommunity concept as the third context tier that encapsulates eSourcing. The dynamic mechanisms are called eCommunity-lifecycle management, negotiation support, eContract management, and service management. To safeguard the enactment of eSourcing configurations, partly hierarchical, unconventional e-business transaction atomicities are considered for eSourcing spheres that need to help in securing on the one hand the business-interaction protocols of collaborating parties, and on the other hand the consumption of service provision for monetary compensation. Furthermore, the ingredients of an e-business transaction for eSourcing are outlined, inspired by research results from the ongoing XTC project. For eSourcing, a electronic business transaction framework is required for combining differing abstract transaction constructs that are ordered in a taxonomy. These abstract transaction constructs must be combined in an eSourcing configuration to support automatically integrated business relations.

For forging automatically integrated business process relations, the eSourcing concept needs to be extended for adopting dynamic mechanisms that result from the broader conceptual context that eSourcing is embedded in. A more elaborate investigation is required for operationalizing the dynamic mechanisms in eSourcing, which results in extensions of eSML and eSRA. For creating an e-business transaction concept for eSourcing, a case study must be conducted where the research results from the XTC project are applied. That way an extension scope is detected for eSML and eSRA, namely, the introduction of an e-business transaction concept to safeguard dynamic inter-organizational business process collaboration.

9.2 Final Remarks

With the emergence of service-oriented technologies and their application for supporting business activities, it becomes possible for companies to inter-organizationally align their business processes and transact via electronic marketplaces. The concept of eSourcing is a contribution into this direction, which is enabled by a move towards web service ecosystems. According to [24], a web service ecosystems is a logical collection of web services whose exposure and access are subject to constraints, which are characteristic of business service delivery.

For web service ecosystems several critical issues exist, namely, the extent to which different service supply and distribution roles are supported in a service supply and distribution network, capturing semantics for web services to support their discovery in a flexible way, supporting long-running multi-party service interactions in inter-organizational business processes, capturing non-functional properties in service descriptions to manage the quality of service level agreements, and enabling automated respecification of services that support business processes in order to compose and bind them inter-organizationally in ways unforeseeable at design time.

In eSourcing several of these issues are paid attention to in varying degrees. For the

first issue of role support, eSourcing is integrating an elaborate resource model in its language representation termed eSML. In Section 6.3.3 the *Who*-concept is presented for eSML that is refined with further detailing models in Appendix A.5. Condition statements that are associated to tasks and their related services, use resource specifications for roles binding. Furthermore, the eSourcing reference architecture eSRA lays a foundation for developing system applications that support the brokerage and mediation of process-oriented services in an electronic marketplace.

The issue of including semantics in eSourcing is not explicitly dealt with in this thesis. However, it is addressed in the CrossWork project [2] where domain-specific ontologies for goal decomposition and automated team formation in the automobile industry have been developed. Such domain-specific ontologies are linked into eSML instantiations to support business collaboration in an eSourcing configuration. In eSML instantiations, the collaborating parties and their resource details can be specified together with legal context specifications for covering the *What*-concept (see Section 6.3.3). Hence, it is feasible to use these facts and to extend eSourcing into the direction of semantics-supported service discovery.

Long-running multi-party business-process interactions in eSourcing should be secured with an appropriate e-business transaction concept. While Section 8.4 gives a direction for such an e-business transaction concept, it also is shown that this is a matter of ongoing research. Hence, the current state of eSourcing does not include an e-business transaction concept for safeguarding inter-organizational business processes. Currently, for an eSML instantiation, business rules can be specified in a way that exceptional situations are managed, followed by a business-process rollback. The purpose of such rules is the specification of contractual clauses that cover, e.g., a penalty situation.

The topic of service quality for eSourcing is not dealt with in this thesis. It is an open research issue to investigate what non-functional properties are required in differing industry domains for ensuring good quality of service. While eSML allows the specification of business relevant facts like price or a delivery timetable, further non-functional business properties like trust, reputation, promises, penalties, escalation, and dispute resolution mechanisms are open issues.

An automated respecification of services that support business processes is a necessary extension for eSourcing where business collaboration is set up in a dynamic way. Respecification means that the nodes and links between them, and the interfaces of a services are changed. By supporting service interface adaptation in eSourcing, interfaces could be kept as generic as possible while adapting to functions peculiar to implementation or prone to change. In [41] such interface adaptation is investigated in a declarative way where a visual language allows pairs of provided and required interfaces to be linked through algebraic expressions. Hence, the tools that are developed for supporting such interface adaptation should be integrated in eSRA for enhancing the inter-organizational linking support in eSourcing collaboration.

A respecification approaches of a process is achievable by looking at the data-flow dependencies [45]. With a given set of services and their interdependencies, an algorithm constructs fully automatically a structured composition that satisfies the given dependencies. However, the user must still give input to the algorithm by annotating the dependency graphs, which is less work is than annotating services with formal pre- and post-conditions, as usually required by most other comparable service composition approaches. The constructed compositions use only basic workflow patterns [19].

Finally a respecification may focus on the differing views of a business process that is offered to a collaborating counterpart [43]. Here a process view hides details of

an internal process that are secret to or irrelevant for the consumer. In a formal two-step approach, customized process views are constructed on structured process models. Firstly, from an internal structured process model, internal activities that need to be hidden are aggregated into a non-customized process view. Secondly, by aggregating and omitting activities from the non-customized view that are not requested by the consumer, a customized process view is constructed .

Considering the schema definition in Appendix B, the complexity of specifying eSourcing configurations becomes apparent. According to [24], web service ecosystems need even more extensive specifications when non-functional requirements and ontologies are taken into account. To support intra- and inter-organizational knowledge workers who have the task assigned of setting up eSourcing configurations, visual modelling support is relevant. However, so far there is no visual language existent for eSML. In [117], a service behavior modelling language is presented that is created according to a clearly defined set of requirements, namely abstraction, comprehensibility, and suitability. This modelling language is supported by a tool that performs a static analysis. For eSourcing a similar development is necessary where first the requirements of a visual modelling language needs to be defined. Consequently, when it is assumed in a semi-automated environment that human involvement for the setup phase of an eSourcing configuration is at least required for final checks and modifications, there is a need for tool-supported eSML instantiation specifications with incorporated automatic analysis algorithms that check the correctness of eSourcing configurations.

Appendix A

Further Refining eSML Models

Contents

The models of Section 6.3 contain docking classes to sub-models for process definitions, capturing resource facts of contracting parties, variable and rule definitions. Thus, the following sub-sections and figures are presenting further detailing sub-models.

A.1 Process Definition Model

Process definitions are used in the shamrock concepts of Where and What. Looking at Figure A.1, the process model contains classes that belong to two different perspectives, namely the control-flow and the eSourcing perspective. The classes in Figure A.1 are appropriately background shaded.

A `process_section` may contain no or multiple process definitions. A process is a type of route, which is the root class for a process definition in XRL, which is a

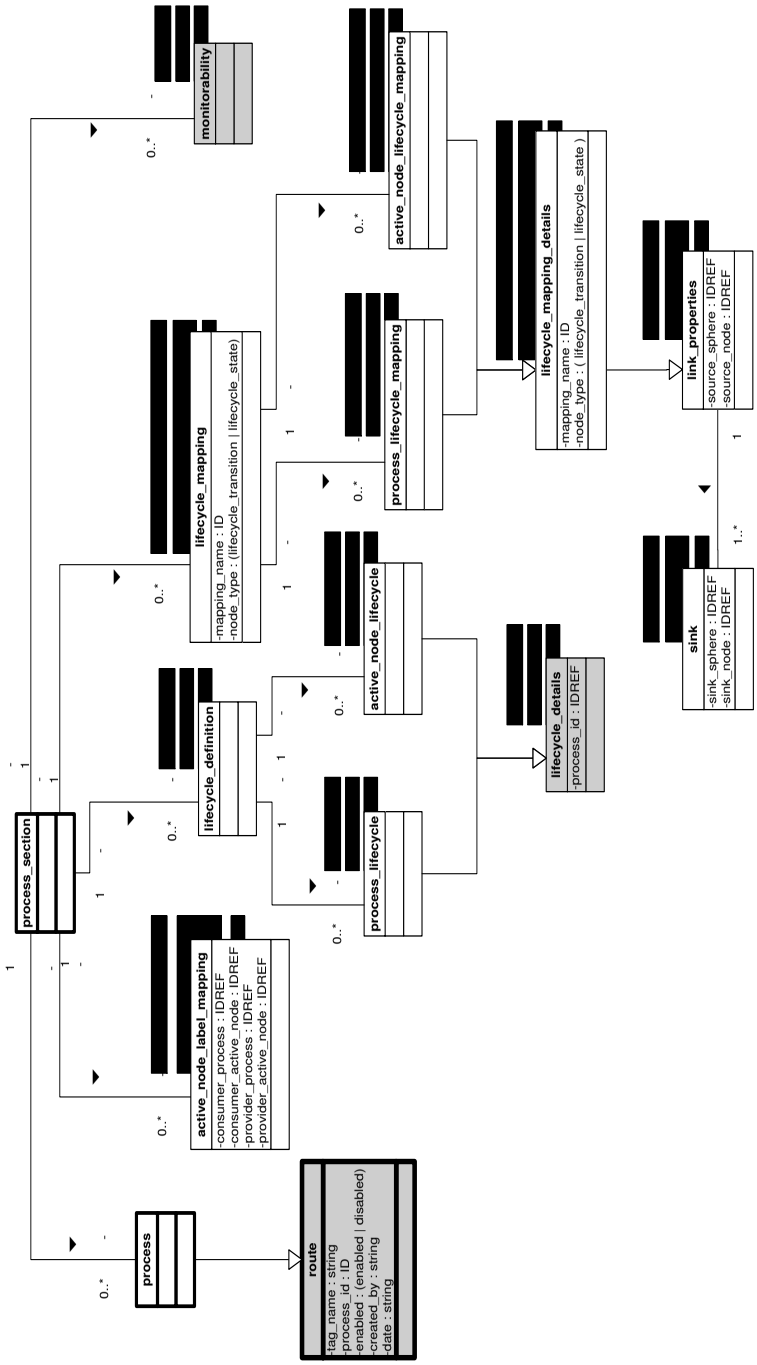

Figure A.1: Model of `process_section`.

subset of eSML. The route is a docking class that is further refined in Figure A.8 and Figure A.9 and explained at a later stage.

All remaining classes of the process_section in Figure A.9 are part of the eSourcing perspective. The majority of those classes are dealing with defining and mapping the life-cycles of service consumer's and provider's processes that are involved in an eSourcing configuration.

The class life_cycle_definition can be contained in a process_section multiple times. It is instrumental for defining life-cycle stages of entire processes or merely active nodes that are part of respective processes. Such definitions are assigned to the processes of a consumer and provider. The classes process_lifecycle and active_node_lifecycle are subclasses of lifecycle_details. The latter class is a docking class belonging to the eSourcing perspective that is explained in further detail below.

Once the lifecycles of respective processes and their contained active nodes are defined, the case may occur that labels are named differently but express equal tasks. Thus, the class active_node_label_mapping serves to define such semantic equivalence that is important for verifying projection inheritance of a consumer sphere and the refinement sphere of a service provider.

As the class lifecycle_mapping indicates, it is also important to map life-cycle stages of different processes and active nodes belonging to the domain of a service consumer and provider. Such labels may be differently named but semantically considered equal. Mapping is important for application support of the enactment of an eSourcing configuration in eSML.

Finally, a docking class for monitorability is contained in the process_section model of Figure A.9 that is further refined in Figure A.9. Montitorability is dealing with different ways of linking nodes belonging to the service consumer and service provider process. Section A.8 gives a more elaborate description.

A.2 Data Package Integration

In an eSourcing configuration a forged contract usually defines variables and documents that are relevant for enactment. The figure below shows how such data is integrated in an eSourcing configuration.

A contract in Figure A.2 may reference one data_definition_section, which in return references one or several data packages. These data packages optionally contain a set of variables and documents.

A.3 Variable Definition

The data items contained in Figure A.3 are data definitions in an eSourcing configuration that are to be used by the contract enactment system. Furthermore, data items can have the function of variables in programming languages, allowing an item to be referenced by other e-contract elements. As in many programming languages, it is essential that data items belong to a specific data type. This allows easier contract processing and sets basic constraints on the allowed values for a data element. Two classes of required data types are identified, i.e., standard data types and special data types. Next, the identified required data types are listed. As rge variable definitions stem from ECML, in [23] further details and examples can be found.

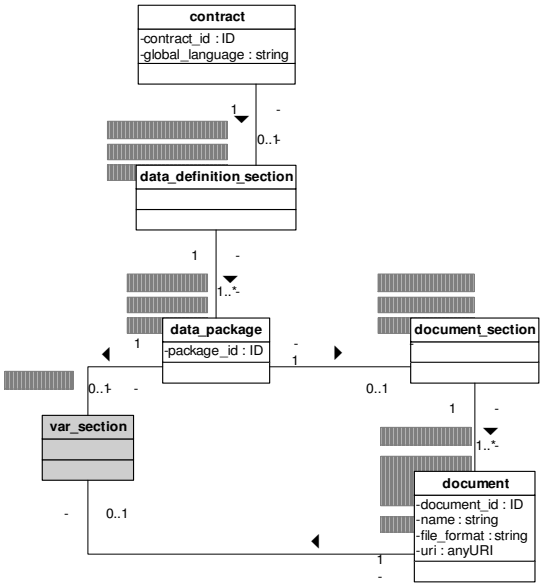

Figure A.2: Data package integration in a contract.

A.3.1 Standard Data Types

Standard data types required in an eSourcing language are:

- *String*. The string type is required to support the definition of string data items.
- *Number*. The number data type is required to support the definition of number-data items.
- *Boolean*. The boolean data type is required to support the definition of boolean data items.
- *Set*. The set data type is required to support the definition of sets of data items.
- *List*. The list data type is required for the definition of ordered sets.
- *Record*. The record data type is required for the definition of bundles of data items.

A.3.2 Special Data Types

In addition to the standard data types, the need for data types is identified that are of importance for eSourcing but are not widely accepted in programming languages. Such special data types can be seen as a specialization of the standard data types. Next, the required special data types are described.

- *Date/Time*. The date and time data types are required for the support of definition of date/time data items.

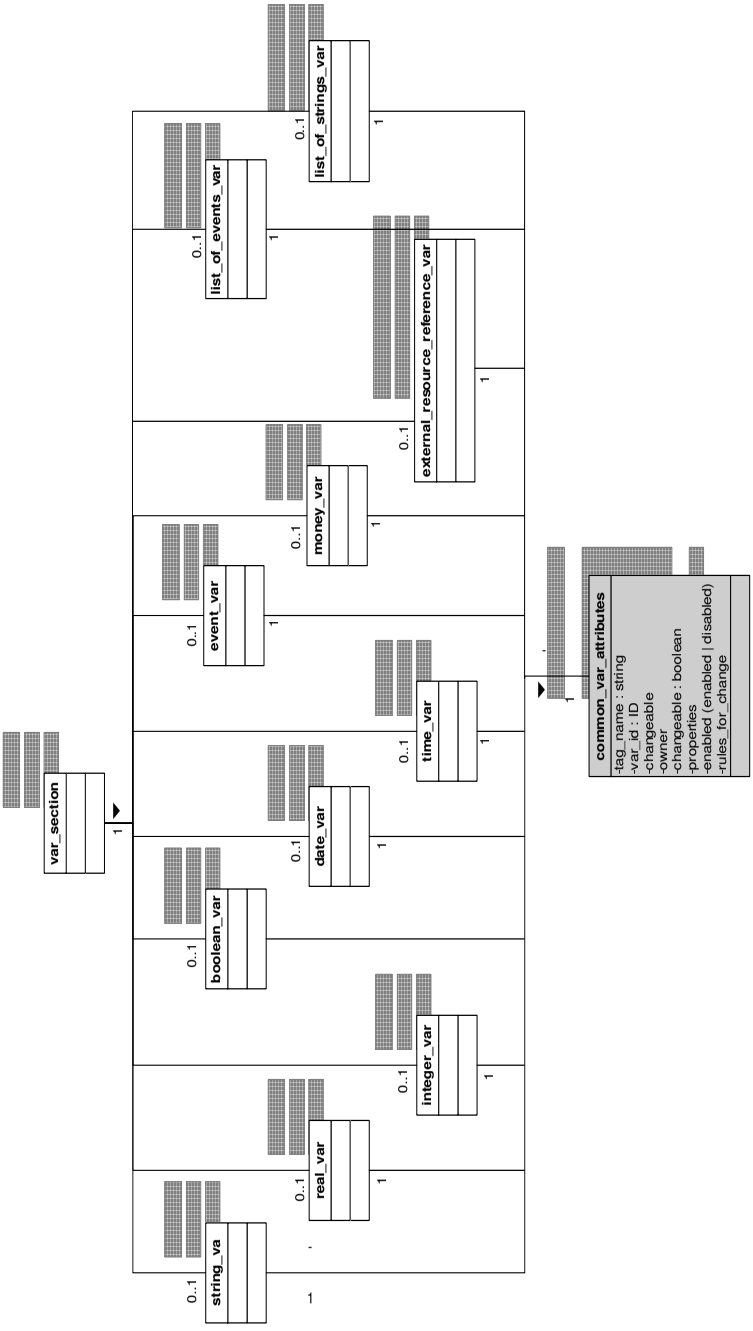

Figure A.3: Model of `var_section`.

- *Money*. The money data type is required for the definition of amounts of money from a specific currency.

- *Event*. The event data type is required to specify events that can occur during contract enactment.

- *External Resource Reference (ERR)*. The ERR data type is required for the specification of references to external contract elements.

A.4 Defining Rules

A rule construct is required for the specification of contract provisions. In Figure A.4 the eSML model of contained rules is depicted. Contract rules are business rules relevant to the contracting relations between the contracting parties. In the existing literature four basic types of business rules are identified, i.e., integrity rules, derivation rules, reaction rules, and deontic assignments. Examples of each of these four rule classes can be identified in existing business contracts. This shows that constructs for each of them has to be provided in an e-contract specification language. In addition, a free-text rule construct can be used for the expression of any provisions targeted only for reading by humans. Every business rule can either be in an enabled or disabled state. Two rules can conflict with each other when both apply at the same time. Thus, the prioritization of execution of rules is required, i.e., when two or more rules have conditions that are satisfied, prioritization can be used to suppress the execution of some of them. In the following subsections, the rules are briefly described and in [23] further details are contained.

A.4.1 Integrity Rules

Integrity rules represent assertions that must be satisfied in the evolving states and state transitions. Two sub-classes of the integrity rules exist, i.e., state constraints and process constraints (also referred to as dynamic constraints).

- *State constraints* set constraints on states of objects at any point in time.

- *Dynamic constraints* set restrictions on transitions from one state to another.

A.4.2 Derivation Rules

A derivation rule prescribes the way for obtaining the value of a data element. Two types of derivation rules can be distinguished, i.e., computational and linguistic rules.

- In *Computational rules* the value of a data element is calculated through a mathematical expression.

- In *Linguistic rules*, information for a data element is derived from existing information (or from information that itself can be derived).

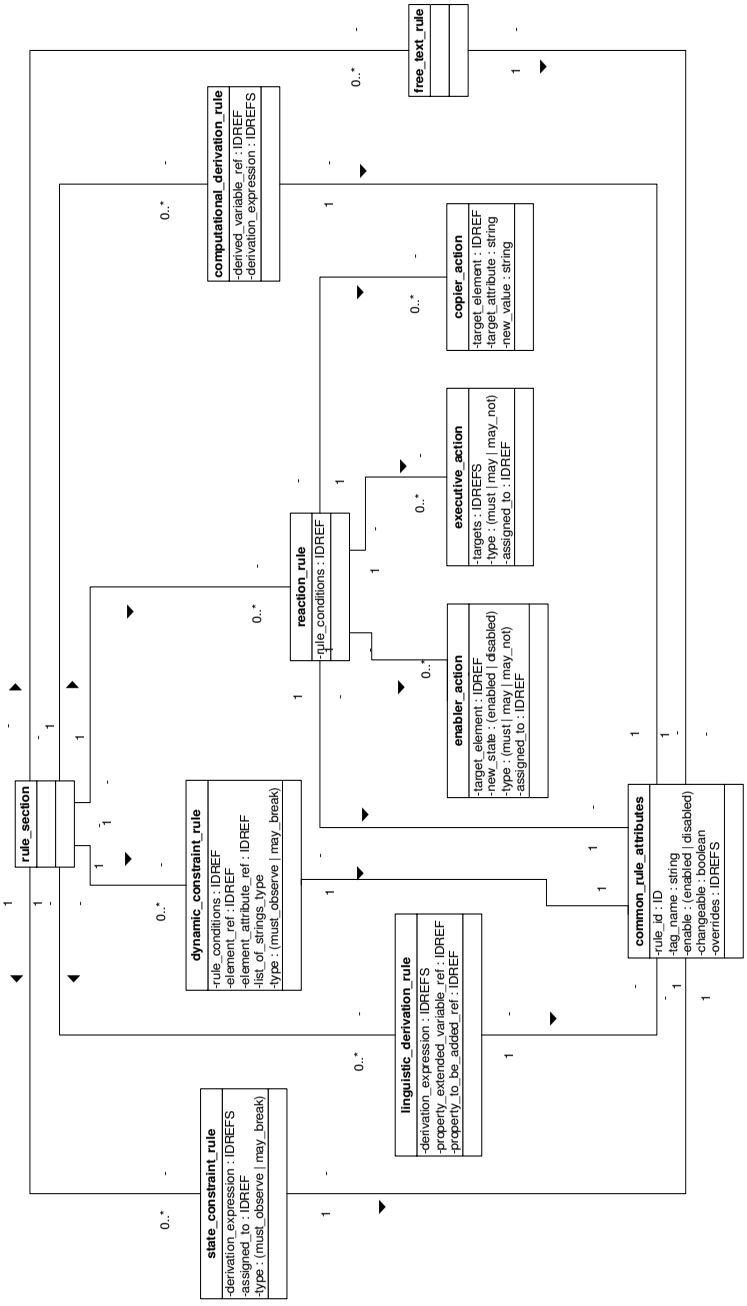

Figure A.4: Model of `rule_section`.

A.4.3 Reaction Rules

Reaction rules are rules that lead to invocation of actions (execution of processes, changing of data values, etc.) in response to certain events, provided that certain state conditions are true. These rules are often referred to as Event Condition Action (ECA) rules. Condition Action (CA) rules can be seen as special cases of the ECA type of rule. The structure of an ECA rule is encoded in its name. An ECA rule construct must provide a definition of the events and conditions for the firing of a rule, and the actions to be performed. Three classes of reaction rules are distinguished according to the type of action defined in the action sub-construct, i.e., enabler, executive, and copier reaction rules.

- *Enabler reaction rules* create/delete or enable/disable e-contract elements.

  In an eSourcing configuration, the free and rule governed updates are agreed upon in advance. For this reason, e-contracts contain the text that has to be added or deleted in its content. Often complete data item, rule, or process specification (or a set of the) is disabled initially or during the contract enactment. These clauses of the contract must be indicated as non-active parts of the e-contract. When an enabler reaction rule fires, a data item, a rule, or a process can change its status (to enabled or disabled). Thus, enabler ECA rules do not directly delete or create new content but only disable or enable previously defined contract elements. When a rule of this type evaluates to true, the contract management system has to cater for the change of the state of the relevant e-contract elements. Enabler rules require the action sub-construct to contain the new state and the identification string of the element whose state is to be changed. This limited functionality of enabler rules required in e-contracts can be delivered by copier reaction rules as well (copier reaction rules are discussed below).

- *Executive reaction rules* cause a process to be triggered or a rule to fire.

- *Copier reaction rules* set the value of a data element.

A.4.4 Deontic Rules

Deontic assignments regulate the assignments of powers, duties, etc. Deontic assignments indicate the rights, obligations, and prohibitions each party (or a person/role from a party) is given/imposed during enactment.

A.4.5 Free-Text Rules

Formalization of certain provisions is not necessary as it does not deliver any value. A free-text rule construct contains the rule in a free text form and indicates that it contains a rule that is to be monitored and evaluated by humans. This allows its automatic extraction from the e-contract and representation to the e-contracting parties for manual evaluation.

A.5 The Resource Section

Resources can either be actors or non actors that are involved in an organization. In the Figures A.5, A.6, A.7 the full resource model is depicted split into three parts. Resource

facts defined that way can be used for three types of task assignment. Either tasks are assigned directly to actors, or a role is defined with a task that can be filled by several actors. Finally, a more complex formal language can be employed for defining which resources are potentially carrying out a task. Such a formal language may take into account capabilities, command over particular resources, authorities, and so on. eSML does not cater for such a formal language. However, a task definition offers a tag where one of the three options may be used for resource definition. The three resource models are connected by three docking classes, namely `organizational_unit`, `resource_type`, and `individual_resource`.

In Figure A.5 the central class is `organizational_unit` which has two subclasses, namely `temporary_organizational_unit` and `permanent_organizational_unit`. Both subclasses have a `start_date` and the first subclass has an `end_date`. An organizational unit has a correlation class related that can be used to describe how two units are correlated to each other. Furthermore, it is possible to relate organizational units in a hierarchy to each other. An `organizational_unit` comprises of a collection and a `resource_type`, where the latter class is described in connection with Figure A.6.

A collection has three subclasses, namely a `concrete_collection`, a `mixed_collection`, and a `typed_collection`. The first subclass comprises a number of actors that are either real humans or agents. A `typed_collection` references a set of optionally contained roles of actors, or resource-type references such as space, machines, or production material. Consequently, a `mixed_collection` combines both collection types and references actors and typed collections. Finally, an individual resource may be available for a task to particular degree.

As mentioned before, an `individual_resource` has an actor as a subclass. Such an actor may be directly assigned to a task. As Figure A.6 shows, an actor reference various assigned appointments that have a start and an end. An actor references one or many roles that can be delegated to other actors. Furthermore, an actor has also one or many organizational positions that are related to organizational units. Such an organizational position may mean several privileges are attached that are also related to roles.

A role is a subclass of a `resource_type` and may be filled by several actors. Besides already mentioned, several capabilities may be required to fill a role. Furthermore, a role can give certain power that can also be delegated to other roles for a limited time. Power that is attached to a role is also related to capabilities and privileges.

In the third part of the resource model depicted in Figure A.7, the focus lies on non actors. A `non_actor` is a subclass of an `individual_resource` and has two further subclasses, namely space and `consumable_resource`. Space is located in a building, has a `room_number` and a capacity. A `consumable_resource` has equally two subclasses, which are `production_material` and machine.

All those subclasses of `non_actor` inherit a contact and phone attribute. The purpose for providing such human contact details is the availability of somebody when "things go wrong". For example, in a tightly integrated just-in-time supply chain, major losses can be expected when a machine fails or certain `production_material` isn't available as scheduled. As a result one crucial contact must be available to avoid a collapse of the overall supply chain.

The classes `production_material` and `machine` both reference the classes `rate_of_usage` and capacity. The first class is instrumental to define how much `production_material` a particular machine consumes in a specific time period. The class capacity also links `production_material` to a machine. The class can

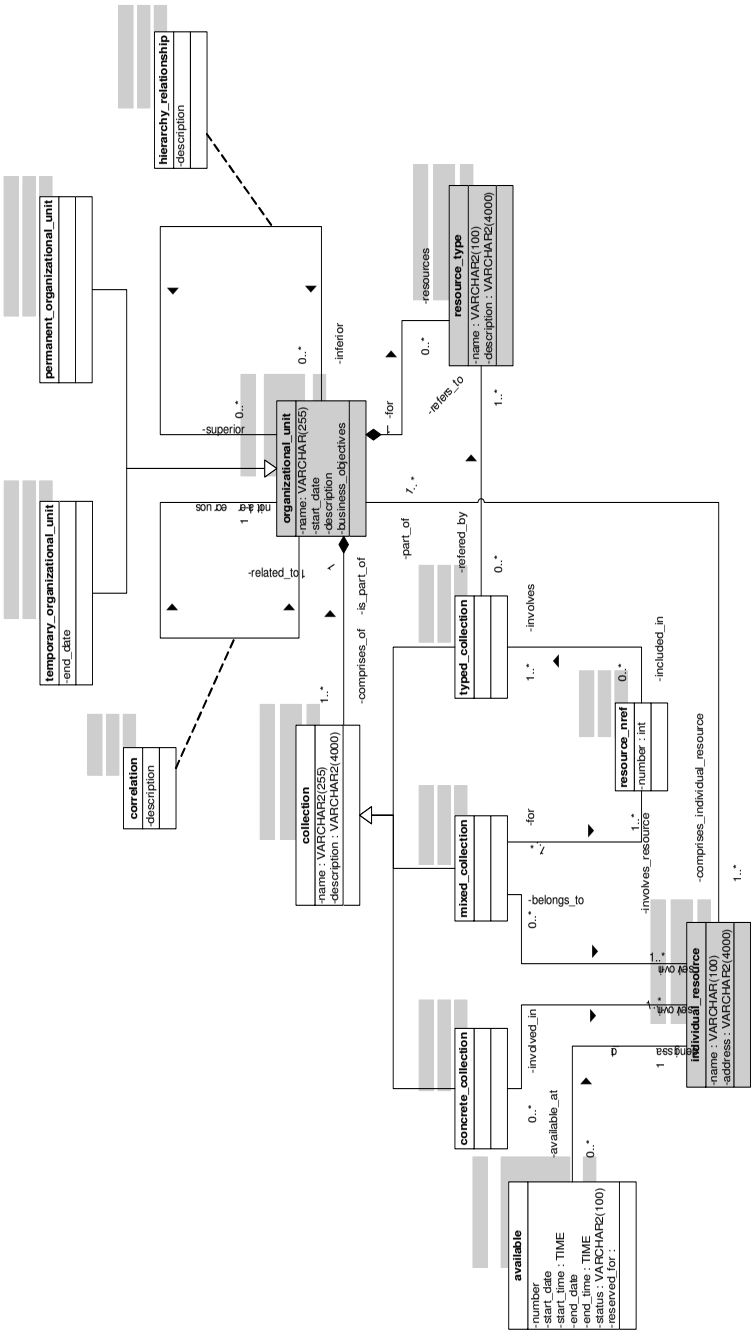

Figure A.5: First part of the `resource_section` Model.

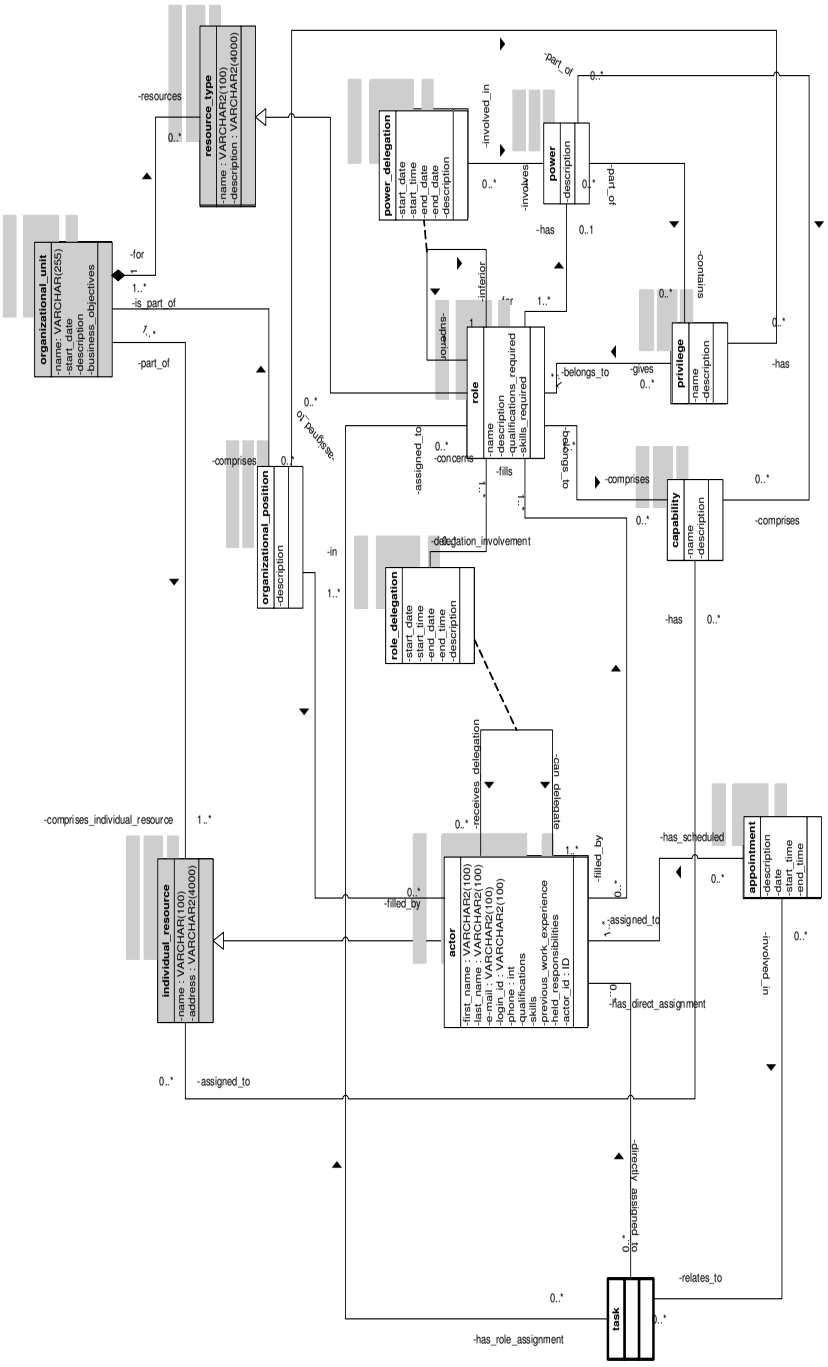

Figure A.6: Second part of the `resource_section` Model.

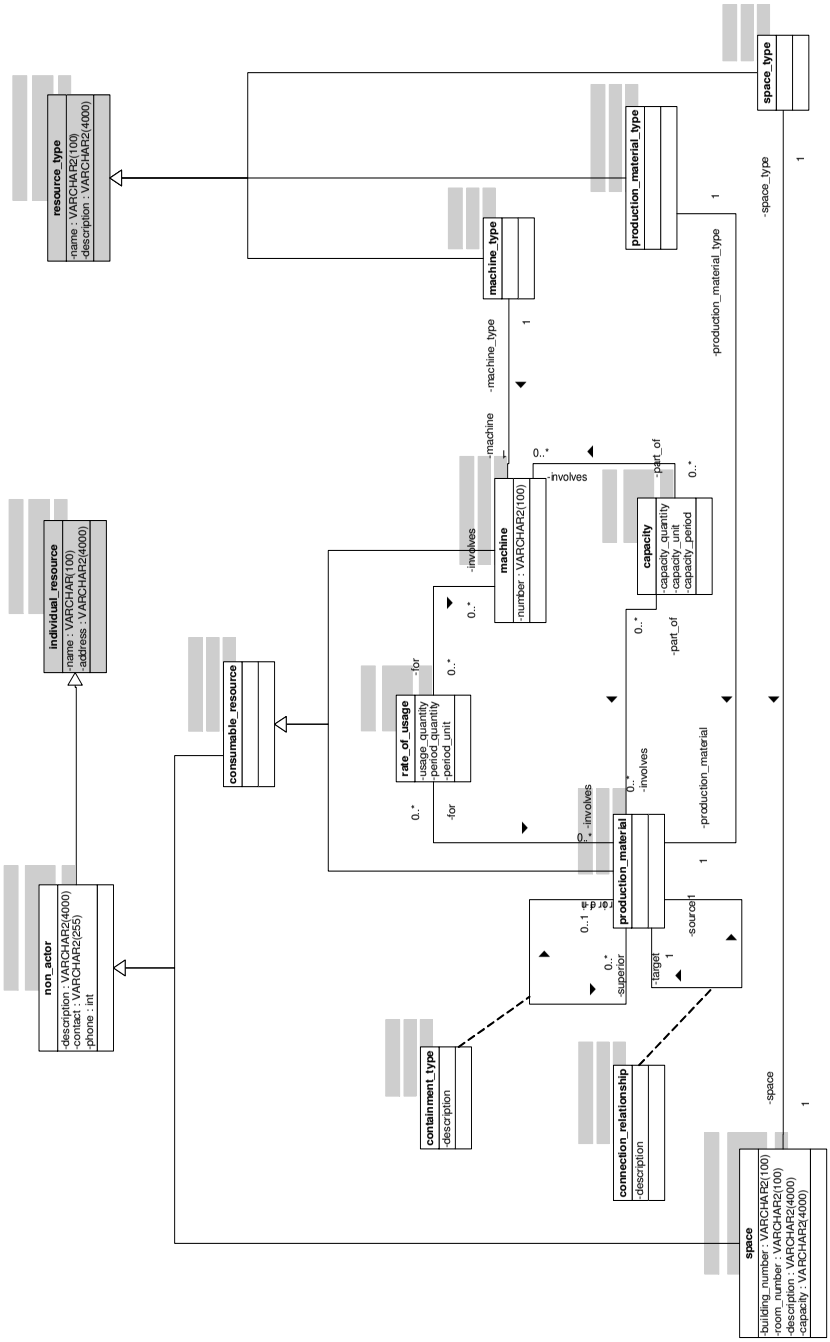

Figure A.7: Third part of the `resource_section` Model.

be used to define, for example, be 50 kg of a particular product that a machine can produce per hour.

Different types of `production_material` can be related with each other. Thus, two relating classes are provided, namely `containment_type` and `connection_relationship`. The first class helps to define that a particular material is part of another material in a sense of a hierarchy. The latter class defines a peer-to-peer relationship between differing `production_material`.

Finally, the classes space, machine, and `production_material` are related to the docking class `resource_type`. To achieve this relationship, `resource_type` has three subclasses called `machine_type`, `production_material_type`, and `space_type`.

A.6 The XRL-Based Route Model

The route model is depicted in the Figure A.8 and Figure A.9 and comprises of classes that are part of the control-flow, eSourcing, and workflow-data perspective while the majority of classes belong to the first perspective. The grey shaded classes are replicated in both Figure A.8 and Figure A.9 and therefore relate the figures.

The top class of the elements is route that references a group definition called `common_elements`. In Figure A.8 that group all control-flow classes are contained of which several classes recursively reference class `common_elements` again. A detailed description of the semantics of the control-flow elements can be found at [17]. By placing a data definition into the `common_elements` group, the workflow-data pattern 2 is supported. Thus, data visibility can be defined on a group level, e.g., a sequence, `parallel_sync`, etc. If the data definition takes place on a rout level, the range stretches over the entire process. Furthermore, adding data definition in the common elements allows visibility on a process and block level.

In Figure A.8 several member classes of group `common_elements` belong to the eSourcing perspective. Concretely, the classes named `receive_task`, `send_task`, `receive_transition`, and `send_transition` are part of the conjoinment dimension of the eSourcing perspective. Additionally, the classes `bi_directional_transition` and `bi_directional_task` equally belong to the conjoinment dimension. The semantics of these classes is contained in the deliverable Section 4.5.3 where the eSourcing perspective patterns are described.

Finally, the class `data_scope` is part of the data perspective that is referenced by class route. By using data scopes, data elements can be defined which are accessible by a subset of the tasks in a case. For example, the initial tax estimate variable is only used within the Gather Return Data, Examine Similar Claims and Prepare Initial Return tasks in the Prepare Tax Return process.

The motivation for data scopes is where several tasks within a workflow coordinate their actions around common data elements or a set of data elements, it is useful to be able to define data elements that are bound to that subset of tasks in the overall workflow process. One of the major justifications for scopes in workflows is that they provide a means of binding data elements, error and compensation handlers to sets of related tasks within a case. This allows for more localized forms of recovery action to be undertaken in the event that errors or concurrency issues are detected.

The routing elements are extended by contextual and logistic attributes. The contextual attributes are necessary for human-centric modelling and contains aspects such as goal, risk assessment, project accomplishment, and handbook reference. The other

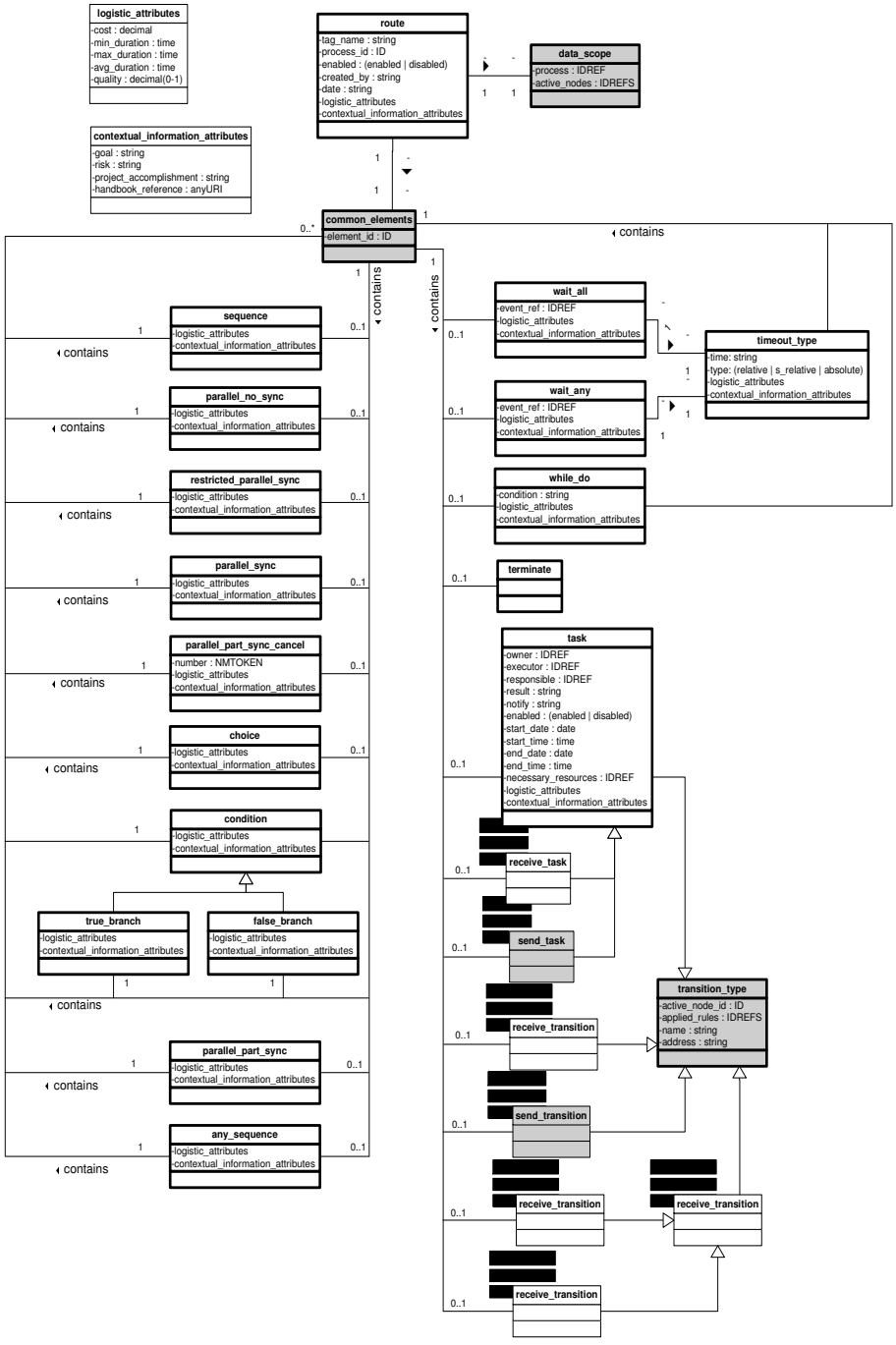

Figure A.8: Thirst part of the `route`.

attributes capturing costs, durations and quality (= probability of valid output) of the execution of an element are needed for the evaluation of the workflow under logistic aspects such as minimum duration.

The second part of the route model depicted in Figure A.9 contains an equal number of classes that belong to the control-flow and the eSourcing perspective. Two classes are members of the workflow-data perspective. The grey shaded docking classes belonging to the control-flow perspective and class `data_scope` are explained in the part one of the route model. The latter class references class data, which is central for the data-flow perspective. Since this latter class is a grey shaded docking class, is explained more elaborately in Figure A.13.

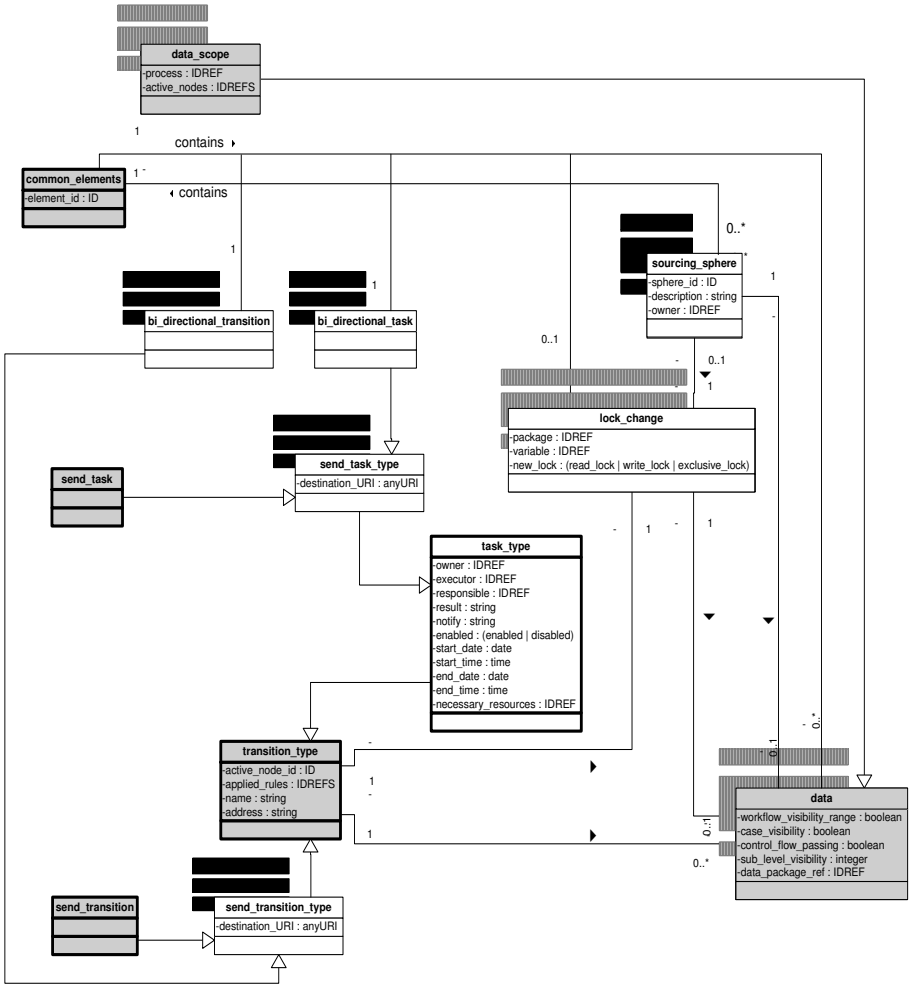

Figure A.9: Second part of `route`.

The classes `bi_directional_task`, `bi_directional_transition`, `sending_task_type`, and `sending_task_transition` all belong to the conjoinment dimension of the eSourcing perspective. For further details Section 4.5.3 may be consulted.

The class sourcing_sphere is mutually related to class common_elements. A sourcing_sphere is instrumental to support multi-lateral contracting of one service consumer and several providers who are all involved in one eSourcing configuration. In such a case the service consumer places one entire process on the external layer of an eSourcing configuration. However, for each provider involved in the multi-lateral contract, the consumer process contains an eSourcing sphere. Thus, one provider's contractual sphere on the external layer is assigned to one sourcing_sphere of the consumer's contractual sphere.

Finally, class lock_change is used to impose locks on data packages that are used in an eSourcing configuration. The lock types available for a data_package_ref are read lock, write lock, and exclusive lock. Since there are several visibility levels of data packages, class lock_change is referenced by several other classes in Figure A.9. The referred to visibility levels that are directly supported by eSML are on an active-node level, block level, and sphere level. Correspondingly, class lock_change referenced by other classes in the second part of the route model.

A.7 Lifecycle Details

The model about different types of life-cycle elements in Figure A.10 is an adjacent sub-model to route in Figure A.8 where the class lifecycle_details is contained as a replica. Consequently, the classes of Figure A.10 are used to define the lifecycles of processes and active nodes that are part of an eSourcing configuration. Accordingly, Figure A.10 depicts that lifecycle_details is a subclass of lifecycle_elements. Compared to Figure A.8, a similar pattern emerges where routing elements belonging to the control-flow-perspective are extracted for lifecycle definitions in the eSourcing perspective.

Similar to Figure A.8 there exists a mutual reference between class lifecycle_elements and the extracted lifecycle control-flow classes. The semantics of those control-flow classes is similar to Figure A.8 and described in more elaborate detail in [17].
However, some classes are contained for lifecycle definitions that are not part of a process definition. Concretely, states are introduced that are either of the class nesting_state or atomic_state. The purpose of the first class is to cater for the observation that nested states are common in practice when workflow management systems are examined. On a lower level of a nesting state further lifecycle elements may be used, including another instance of nesting_state. On the other hand, the class atomic_state can not be further refined on a lower level. Finally, transitions propel the lifecycle of a process or active node from one state to the next.

A.8 The Monitorability Model

The classes belonging to Figure A.11 all belong to the monitorability dimension of the eSourcing perspective and are instrumental for linking active and passive nodes that belong to the respective contractual spheres of a service consumer and provider. On a process level nodes are only active, i.e., task, transition, send task, receive task, send transition, receive transition, bi-directional task, and bi-directional transition. The only two cases of passive nodes exist on a life-cycle level of tasks where nested states and atomic states exist.

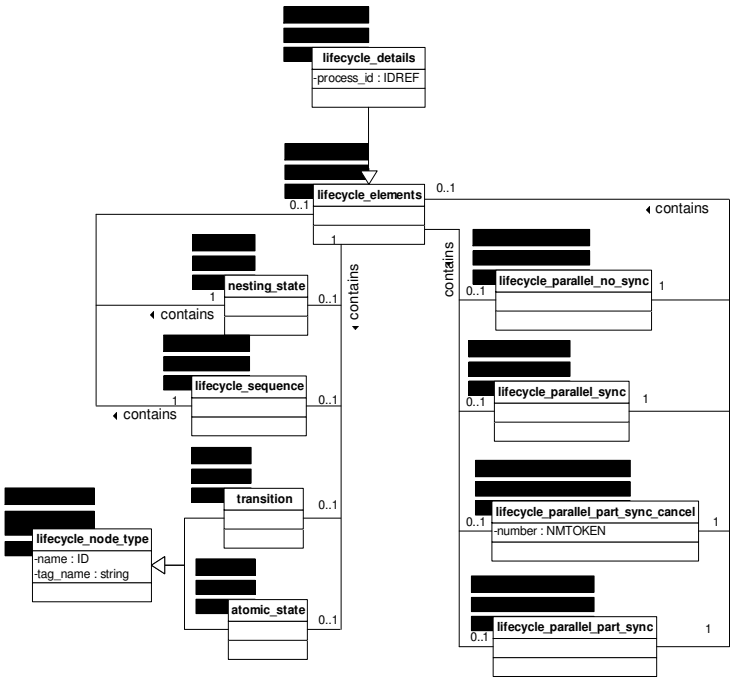

Figure A.10: Model of `lifecycle_details`.

By defining monitoring links between nodes of respective eSourcing domains, it is possible for one contracting party to observe the progress of process enactment of the eSourcing counterpart. Usually the monitoring direction is from service consumer to provider. However, it is also imaginable that it can be necessary for the service provider to observe enactment progress of the service consumer.

The monitorability classes of Figure A.11 either perform the polling or the messaging of enactment progress. Assuming that polling is used from service consumer to provider, the consumer frequently requests the status of the linked node, which is enacted in the domain of the service provider. When an enactment change is perceived, the linked node in the domain of the service consumer follows the change. When an active node with a life-cycle is polled, state changes or transition firings are mirrored. When a transition on a process level is polled, information is returned whether the linked node has fired. Subsequently the firing is equally mirrored in the domain of the service consumer. Finally, the class `enactment_termination` is instrumental for polling whether the enactment of service provision is completed. In that case the eSourcing sphere of the consumer also terminates and the rest of the consumer's in-house process commences with enactment.

Messaging classes are used to link nodes from the process enacting domain to the eSourcing counterpart that wants to observe. Thus, when two transitions in opposing eSourcing domains are linked, the firing is messaged to the other domain where that behavior is mirrored by the linked node. When a task is linked, lifecycle messaging must be employed. Thus, when a lifecycle state is entered and left again, these events are messaged to the task in the other domain where the lifecycle-state changes are followed. Equally lifecycle-transition changes are mirrored in the domain of the

Figure A.11: Model of monitorabilty.

message recipient. The class `enactment_propagation` is used when enactment of the eSourcing sphere in the domain of the service consumer is started. This enactment commencement is propagated to the domain of the eSourcing counterpart where service provision commences.

A.9 The Transition-Type Model

In Figure A.12 the class `transition_type` is depicted in further detail. This class is contained in Figure A.8 and Figure A.9 that are the first and second part of the route model. In those latter figures class `transition_type` is central for all active nodes belonging to the control-flow perspective and the conjoinment dimension of the eSourcing perspective. As Figure A.12 shows, class `transition_type` is also a central connection to the workflow-data perspective. The `lock_type` is initially set or not set when a data package is defined may be altered with the `lock_change` tag.

The `common_var_attributes` contains properties that are used for all simple and complex variables used in eSML. The class `common_var_attributes` is replicated as a docking class in Figure A.3 where many variable classes are depicted. Also class event employs the properties of `common_var_attributes`.

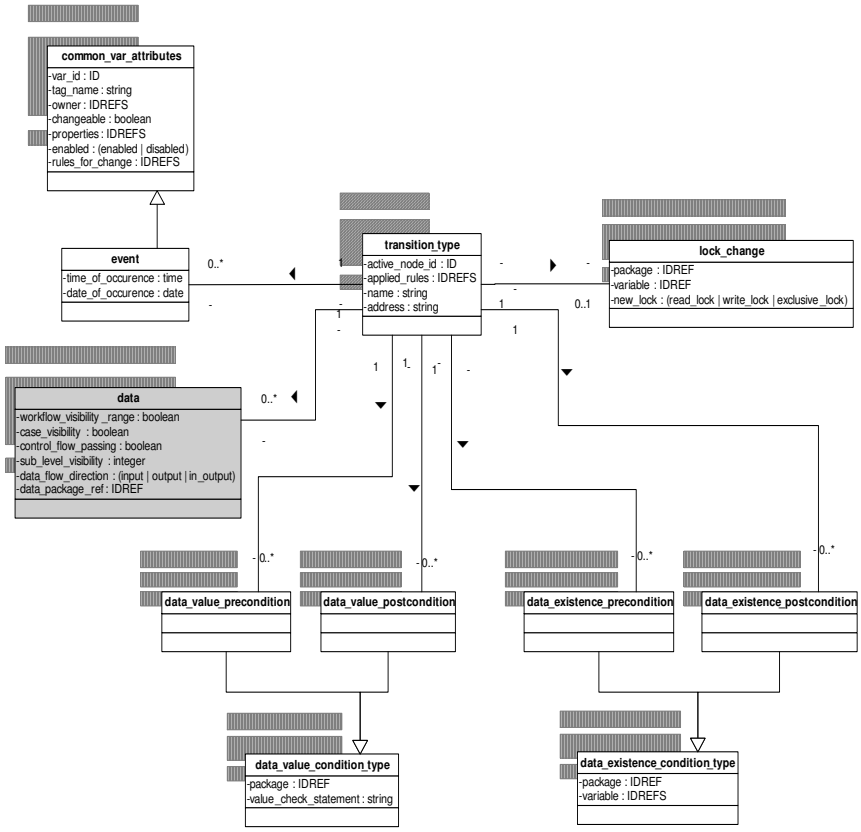

Figure A.12: Model of `transition_type`.

The only grey shaded class in Figure A.12 is named data, which is central for the workflow-data perspective. An instance of data may reference one or many data-package references, which contain different variables and/or document definitions. Next, the properties of class data are explained. If workflow_visibility_range is true, a package is visible in all cases of a process template for all active nodes contained. If case_visibility is true, a data package is visible for all cases.

The property data_flow_direction supports the passing of data elements from a block task instance to the corresponding sub-workflow that defines its implementation. When no further explicit assignment definition is given, no data passing is performed since data is stored in a global data store. That means all lower-level elements are automatically aware of the data package. If additional assignment tags are used, then either data passing is performed via a dedicated data channel or via an integrated control and data channel, i.e., in the latter case data flows along control flow.

The property control_flow_passing is instrumental for supporting data_flow_direction. When data_types are defined on a block level, setting control_flow_passing to true means that an integrated control and data channel is used. As a result data flows from one node to the next along control flow.

By using sub_level_visibility in combination with a data_package_ref definition for a control-flow block element, it can be determined to which lower level the data_package_ref is visible. For example, if a block has 5 lower levels of routing elements and level 4 is defined in a sub_level_visibility tag, then elements located on the lowest level don't have visibility of the data package, i.e., the 5th level below the definition level of the data package in question.

In Figure A.12 the class transition_type references several other classes that are part of the workflow-data perspective. With the referenced classes data_existence_precondition the presence of a variable can be checked as a prerequisite for enactment of an active node and with data_existence_postcondition the presence of a variable can be defined as a postcondition that must be fulfilled after the completed enactment of an active node. The superclass data_existence_condition_type contains properties for defining which variable in which package is targeted.

Finally, the classes data_value_precondition and data_value_postcondition can be used to define pre- and postconditions for the enactment of active nodes. However, differently to the prior case where variable existence was the criteria, this time the variables must have a particular value. Therefore, the superclass data_value_condition_type contains a property for checking the value of a variable.

A.10 The Data Model

In the last submodel depicted in Figure A.13, many classes are contained to support further data-flow models. Subsequently all classes belong exclusively to the data-flow perspective. The central class is data, which can be found replicated in Figure A.9 and Figure A.12 as a grey shaded docking class.

With case_visibility_range the set of cases can be specified where a data package is equally visible. Class passing_destination supports passing data packages from task to task. Class passing_destination is instrumental for two different cases. When data_type definition is on block level, then explicit data passing from a block level to a contained lower-level element can be performed with

passing_destination. Furthermore, with passing_destination task-level data packages can be assigned explicitly to the higher level, e.g., a block.

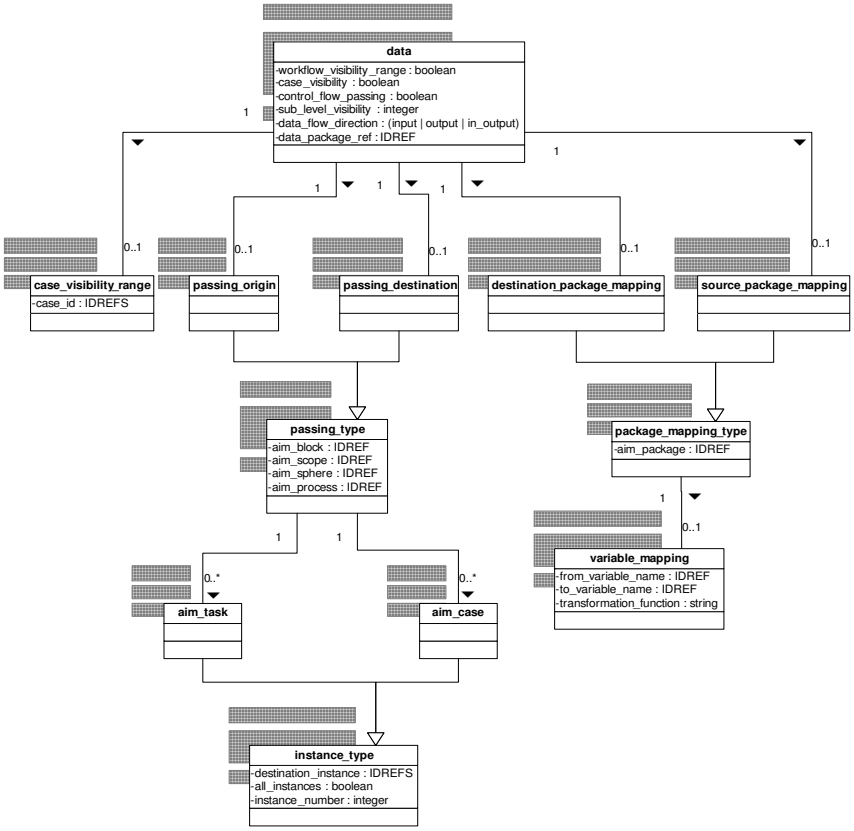

Figure A.13: Model of data.

Instead of passing a data package per se, destination_package_mapping and source_package_mapping map parts of a data package in one process node on properties of another data package at some other location.

In Figure A.13 passing_type is a superclass of passing_origin and passing_destination. Thus, depending on the visibility level for destination passing, the properties contained in passing_type are used for detailed definitions in the subclasses passing_origin and passing_destination.

The referenced classes aim_task and aim_case may be identified differently. It could be that several instances of a task exist within a case. An eSourcing configuration can be equally instantiated several times. Thus, the superclass instance_type is employable for identifying if either all instances or a subset of either tasks or cases is addressed for data visibility.

Finally, in Figure A.13 package_mapping_type serves as a superclass of destination_package_mapping and source_package_mapping. The superclass contains a property for targeting a data package of which the variables can be mapped on the variable of another package. During that mapping a transformation function may be applied. For that purpose package_mapping_type references

class `variable_mapping`. The following section is the eSML schema definition that the models of Section 6.3 and of this chapter describe.

Appendix B

eSML Schema

```
<?xml version="1.0" encoding="uTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by
Alex Norta (Technische Universiteit Eindhoven)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!-- PART 1 -->
    <!-- Next, the definition of the data types required in an
    e-contract language are provided. We use the built in the XML
    schema data types and extend them with the required additional
    attributes for a data item. The common attributes for all data
    items are listed in the group of attributes
    "common_var_attributes". We start with the definition of the
    standard data types and continue with the definition
    of the special data types . At the end of this part of the
    eSML schema, we list three predefined set types (i.e.,
    "state_type", "constraint_type", "currency_type"). These constant
    sets are used as the data type of attributes of contract elements
    that are defined later on in the language schema.-->
    <!-- definition of standard Variable TYPES-->
    <xs:attributeGroup name="common_var_attributes">
        <xs:attribute name="tag_name" type="xs:string"
        use="required"/>
        <xs:attribute name="var_id" type="xs:ID" use="required"/>
        <xs:attribute name="owner" type="xs:IDREFS"
        use="optional"/>
        <xs:attribute name="changeable" type="xs:boolean"
        use="required"/>
        <xs:attribute name="properties" type="xs:IDREFS"
        use="optional"/>
        <xs:attribute name="enabled" type="state_type"
        use="required"/>
        <xs:attribute name="rules_for_change" type="xs:IDREFS"
        use="optional"/>
    </xs:attributeGroup>
    <!-- Tag "name" attribute is a name from an agreed
    ontology. "var_id" attribute is the unique identifier
    of the data item. "owner" attribute contains a
    reference to an owner of the data item who is allowed
```

```
1038    to update the value of the data item."changeable"
1039    indicates if the value of the data element can be changed.
1040    "properties" attribute indicates the properties a data
1041    element has. "enabled" attribute indicates if the data
1042    item is currently enabled. The value of this attribute
1043    is from the defined state_type."rules_for_change" attribute
1044    is used to indicate the rules in the contract that must
1045    evaluate to true in order a change to take place. It is
1046    optional and is used solely for improving the efficiency
1047    of the monitoring systems.-->
1048    <xs:complexType name="string_type">
1049        <xs:simpleContent>
1050            <xs:extension base="xs:string">
1051                <xs:attributeGroup ref="common_var_attributes"/>
1052            </xs:extension>
1053        </xs:simpleContent>
1054    </xs:complexType>
1055    <xs:complexType name="real_type">
1056        <xs:simpleContent>
1057            <xs:extension base="xs:decimal">
1058                <xs:attributeGroup ref="common_var_attributes"/>
1059            </xs:extension>
1060        </xs:simpleContent>
1061    </xs:complexType>
1062    <xs:complexType name="integer_type">
1063        <xs:simpleContent>
1064            <xs:extension base="xs:integer">
1065                <xs:attributeGroup ref="common_var_attributes"/>
1066            </xs:extension>
1067        </xs:simpleContent>
1068    </xs:complexType>
1069    <xs:complexType name="boolean_type">
1070        <xs:simpleContent>
1071            <xs:extension base="xs:boolean">
1072                <xs:attributeGroup ref="common_var_attributes"/>
1073            </xs:extension>
1074        </xs:simpleContent>
1075    </xs:complexType>
1076    <!-- Sample definition of data items from the List data type.
1077    Additional list types can be defined by designers in the schema
1078    or in the e-contract
1079    itself.-->
1080    <xs:simpleType name="list_of_strings_type">
1081        <xs:list itemType="xs:string"/>
1082    </xs:simpleType>
1083    <xs:complexType name="list_of_events_type">
1084        <xs:sequence>
1085            <xs:element name="event" type="event_type"
1086            maxOccurs="unbounded"/>
1087        </xs:sequence>
1088    </xs:complexType>
1089    <!-- For the definition of data items from the Record data
1090    type, contract designers can use the build in XML Schema
1091    Complex Type element.-->
```

```
1092    <!-- Definition of special data types -->
1093    <xs:complexType name="date_type">
1094        <xs:simpleContent>
1095            <xs:extension base="xs:date">
1096                <xs:attributeGroup ref="common_var_attributes"/>
1097            </xs:extension>
1098        </xs:simpleContent>
1099    </xs:complexType>
1100    <xs:complexType name="time_type">
1101        <xs:simpleContent>
1102            <xs:extension base="xs:time">
1103                <xs:attributeGroup ref="common_var_attributes"/>
1104            </xs:extension>
1105        </xs:simpleContent>
1106    </xs:complexType>
1107    <xs:complexType name="money_type">
1108        <xs:simpleContent>
1109            <xs:extension base="xs:decimal">
1110                <xs:attributeGroup ref="common_var_attributes"/>
1111                <xs:attribute name="currency" type="currency_type"
1112                use="required"/>
1113            </xs:extension>
1114        </xs:simpleContent>
1115    </xs:complexType>
1116    <xs:complexType name="event_type">
1117        <xs:simpleContent>
1118            <xs:extension base="xs:boolean">
1119                <xs:attributeGroup ref="common_var_attributes"/>
1120                <xs:attribute name="time_of_occurence" type="xs:time"
1121                use="optional"/>
1122                <xs:attribute name="date_of_occurence" type="xs:date"
1123                use="optional"/>
1124            </xs:extension>
1125        </xs:simpleContent>
1126    </xs:complexType>
1127    <!-- For either pushing out or pulling in external data -->
1128    <xs:simpleType name="access_type">
1129        <xs:restriction base="xs:string">
1130            <xs:enumeration value="push"/>
1131            <xs:enumeration value="pull"/>
1132        </xs:restriction>
1133    </xs:simpleType>
1134    <!-- For referencing variables that are defined in a data_package
1135    -->
1136    <xs:complexType name="external_resource_reference_type">
1137        <xs:simpleContent>
1138            <xs:extension base="xs:anyURI">
1139                <xs:attributeGroup ref="common_var_attributes"/>
1140                <xs:attribute name="access" type="access_type"
1141                use="optional"/>
1142                <xs:attribute name="resource_state"
1143                type="list_of_resource_types" use="optional"/>
1144                <xs:attribute name="is_legally_binding"
1145                type="xs:boolean" use="required"/>
```

```
1146            </xs:extension>
1147        </xs:simpleContent>
1148    </xs:complexType>
1149    <!--Documents may lso be a part of data packages for data flow.
1150    -->
1151    <xs:complexType name="document_type">
1152        <xs:sequence>
1153            <xs:element name="document_id" type="xs:ID"/>
1154            <xs:element name="name" type="xs:string"
1155            maxOccurs="unbounded"/>
1156            <xs:element name="file_format" type="xs:string"
1157            minOccurs="0" maxOccurs="unbounded"/>
1158            <xs:element name="uri" type="xs:anyURI"
1159            maxOccurs="unbounded"/>
1160            <xs:element name="var_section"
1161            type="variables_def_section" minOccurs="0"/>
1162            <!-- specify the structure of the document-->
1163        </xs:sequence>
1164    </xs:complexType>
1165    <xs:complexType name="document_def_section">
1166        <xs:sequence maxOccurs="unbounded">
1167            <xs:element name="document_read" type="document_type"
1168            minOccurs="0" maxOccurs="unbounded"/>
1169            <xs:element name="document_create" type="document_type"
1170            minOccurs="0" maxOccurs="unbounded"/>
1171            <xs:element name="document_update" type="document_type"
1172            minOccurs="0" maxOccurs="unbounded"/>
1173        </xs:sequence>
1174    </xs:complexType>
1175    <xs:complexType name="list_of_documents">
1176        <xs:sequence maxOccurs="unbounded">
1177            <xs:element name="document" type="document_type"/>
1178        </xs:sequence>
1179    </xs:complexType>
1180    <xs:complexType name="internal_resource_reference_type">
1181        <xs:simpleContent>
1182            <xs:extension base="xs:anyURI">
1183                <xs:attributeGroup ref="common_var_attributes"/>
1184                <xs:attribute name="referenced_package"
1185                type="xs:IDREFS" use="required"/>
1186                <xs:attribute name="referenced_variable"
1187                type="xs:IDREFS" use="required"/>
1188            </xs:extension>
1189        </xs:simpleContent>
1190    </xs:complexType>
1191    <!-- Snippets allow inclusion of predefined contract parts.-->
1192    <xs:complexType name="snippet_type">
1193        <xs:sequence>
1194            <xs:any maxOccurs="unbounded"/>
1195        </xs:sequence>
1196        <xs:attribute name="snippet_id" type="xs:ID"
1197        use="required"/>
1198    </xs:complexType>
1199    <!-- Definition of rule/process state type-->
```

```
1200    <xs:simpleType name="state_type">
1201        <xs:restriction base="xs:string">
1202            <xs:pattern value="enabled|disabled"/>
1203        </xs:restriction>
1204    </xs:simpleType>
1205    <!-- Definition of deontic operators -->
1206    <xs:simpleType name="constraint_type">
1207        <xs:restriction base="xs:string">
1208            <xs:enumeration value="must"/>
1209            <xs:enumeration value="may"/>
1210            <xs:enumeration value="may_not"/>
1211        </xs:restriction>
1212    </xs:simpleType>
1213    <!-- Definition of static constraints deontic operators -->
1214    <xs:simpleType name="static_constraint_type">
1215        <xs:restriction base="xs:string">
1216            <xs:enumeration value="must_observe"/>
1217            <xs:enumeration value="may_break"/>
1218        </xs:restriction>
1219    </xs:simpleType>
1220    <!-- Definition of currency types -->
1221    <xs:simpleType name="currency_type">
1222        <xs:restriction base="xs:string">
1223            <xs:enumeration value="USD"/>
1224            <xs:enumeration value="EUR"/>
1225            <xs:enumeration value="GBP"/>
1226            <xs:enumeration value="cAD"/>
1227            <xs:enumeration value="BGN"/>
1228        </xs:restriction>
1229    </xs:simpleType>
1230    <!-- Definition of resource state types -->
1231    <xs:simpleType name="list_of_resource_types">
1232        <xs:list itemType="resource_state_type"/>
1233    </xs:simpleType>
1234    <xs:simpleType name="resource_state_type">
1235        <xs:restriction base="xs:string">
1236            <xs:enumeration value="available"/>
1237            <xs:enumeration value="unavailable"/>
1238            <xs:enumeration value="changed"/>
1239        </xs:restriction>
1240    </xs:simpleType>
1241    <!-- PART 2 -->
1242    <!-- In the second part of the eSML schema, we provide
1243    the constructs required for the definition of the integrity,
1244    derivation, and reaction contract rules. The common for
1245    all three rule types attributes are extracted in the
1246    attribute group "common_rule_attributes". A sample set
1247    of operators that can be used in expressions is provided
1248    with an illustrative purpose (e.g., the "unary_operator"
1249    type). The complete set of operators must additionally be
1250    defined. -->
1251    <!-- Definition of common rule attributes. In addition to
1252    the attributes "tag_name", "rule_id", "changeable",
1253    "enabled" explained in the data item definition section,
```

```
1254    the following attributes are defined:
1255    The "overrides" attribute indicates which other rule are
1256    overridden by this rule if they fire together.-->
1257    <xs:attributeGroup name="common_rule_attributes">
1258        <xs:attribute name="tag_name" type="xs:string"
1259        use="required"/>
1260        <xs:attribute name="rule_id" type="xs:ID"
1261        use="required"/>
1262        <xs:attribute name="enabled" type="state_type"
1263        use="required"/>
1264        <xs:attribute name="changeable" type="xs:boolean"
1265        use="required"/>
1266        <xs:attribute name="overrides" type="xs:IDREFS"
1267        use="optional"/>
1268    </xs:attributeGroup>
1269    <!-- Definition of some operators used in expressions
1270    in rules.     -->
1271    <xs:simpleType name="unary_operator">
1272        <xs:restriction base="xs:string">
1273            <xs:enumeration value="plus"/>
1274            <xs:enumeration value="minus"/>
1275            <xs:enumeration value="sqrt"/>
1276            <xs:enumeration value="percent"/>
1277        </xs:restriction>
1278    </xs:simpleType>
1279    <xs:simpleType name="unary_boolean_operator">
1280        <xs:restriction base="xs:string">
1281            <xs:enumeration value="not"/>
1282        </xs:restriction>
1283    </xs:simpleType>
1284    <!-- Next, we start with the definition of Integrity rules
1285    (distinguishing the two classes of integrity rules, namely
1286    the "state_constraint_rule_type" and the
1287    "dynamic_constraint_rule_type").
1288    The state_constraint_rule_type construct is a simple
1289    Boolean expression. The "assigned_to"
1290    attribute indicates the party (if any) to which the
1291    constraint is assigned. The "type" attribute indicates if
1292    the rule obligates the assigned party to observe the rule
1293    or allows it to break the rule.
1294    The dynamic_constraint_rule_type construct consists of
1295    four elements. If the boolean expression evaluates to
1296    true the rule is in force and the referenced element
1297    "element_ref" (or attribute of the element
1298    "element_attribute_ref") may/may not (depending
1299    on the value of the value of the type attribute) receive
1300    a new value indicating its new state. The
1301    "restricted-nonrestricted_values" ellement contains a set
1302    of allowed/disallowed values.  -->
1303    <!-- Definition of expressions in rule types.
1304    Expressions in rules are presented as strings that
1305    are to be parsed by the contract interpretation software
1306    in order to obtain an expression tree. The expression
1307    strings can be mixed with references to earlier defined
```

```
1308    variables.-->
1309    <xs:complexType name="expression" mixed="true">
1310        <xs:sequence>
1311            <xs:element name="expression_variable_ref"
1312            type="xs:IDREF" minOccurs="0"
1313            maxOccurs="unbounded"/>
1314        </xs:sequence>
1315    </xs:complexType>
1316    <!-- Definition of the possible rule types and their
1317    sub-constructs -->
1318    <xs:complexType name="state_constraint_rule_type">
1319        <xs:sequence>
1320            <xs:element name="rule_conditions"
1321            type="expression"/>
1322        </xs:sequence>
1323        <xs:attributeGroup ref="common_rule_attributes"/>
1324        <xs:attribute name="assigned_to" type="xs:IDREF"
1325        use="optional"/>
1326        <xs:attribute name="type"
1327        type="static_constraint_type" use="optional"/>
1328    </xs:complexType>
1329    <xs:complexType name="dynamic_constraint_rule_type">
1330        <xs:sequence>
1331            <xs:element name="rule_conditions"
1332            type="expression"/>
1333            <xs:element name="element_ref" type="xs:IDREF"/>
1334            <xs:element name="element_attribute_ref"
1335            type="xs:IDREF" minOccurs="0"/>
1336            <xs:element name="restricted-nonrestricted_states"
1337            type="list_of_strings_type"/>
1338        </xs:sequence>
1339        <xs:attribute name="type" type="constraint_type"/>
1340        <xs:attributeGroup ref="common_rule_attributes"/>
1341    </xs:complexType>
1342    <!-- Next, we define the Derivation rule constructs
1343    (distinguishing the two types of derivation rules, namely
1344    "computational_derivation_rule_type" and
1345    "linguistic_derivation_rule_type". The "derived_variable_ref"
1346    attribute contains a reference to the data items the value
1347    of which is derived with this rule. "derivation_expression"
1348    contains a Boolean expression that when evaluating to true
1349    fires the rule.
1350    In "linguistic_derivation_rule_type", the
1351    "property_extended_variable_ref" contains a reference to the
1352    data item that is given a property when the rule fires.
1353    "property_to_be_added_ref" contains a reference to a data
1354    item that contains the property to be added. In this
1355    approach, we require the properties to be defined as data
1356    items in the e-contract, as in this way they can be
1357    referenced by many components and will be from a clearly
1358    defined data type. However, it is possible linguistic
1359    derivation rules to be completely replaced by copier reaction
1360    rules that simply state the new value to be given to the
1361    data item (in a string format).-->
```

```
1362    <xs:complexType name="computational_derivation_rule_type">
1363        <xs:sequence>
1364            <xs:element name="derived_variable_ref"
1365            type="xs:IDREFS"/>
1366            <xs:element name="derivation_expression"
1367            type="expression"/>
1368        </xs:sequence>
1369        <xs:attributeGroup ref="common_rule_attributes"/>
1370    </xs:complexType>
1371    <xs:complexType name="linguistic_derivation_rule_type">
1372        <xs:sequence>
1373            <xs:element name="derivation_expression"
1374            type="expression"/>
1375            <xs:element name="property_extended_variable_ref"
1376            type="xs:IDREF"/>
1377            <xs:element name="property_to_be_added_ref"
1378            type="xs:IDREF"/>
1379        </xs:sequence>
1380        <xs:attributeGroup ref="common_rule_attributes"/>
1381    </xs:complexType>
1382    <!-- Next, the eSML constructs for the support of Reaction
1383    rules follow. A reaction rule can be from one of the three
1384    subtypes, i.e., enabler, executive, and copiers. To support
1385    each of the three types, we provide three sub-constructs
1386    for each of them. Each reaction rule fires when its
1387    "rule_conditions" evaluates to true.
1388    The "enabler_action_type" has two ellements. The
1389    "target_element" contains a reference to the element the
1390    state of which will be changed. "new_state" indicates the
1391    new state that must be assigned to the element (enabled or
1392    disabled). The "type" attribute indicates the deontic
1393    assignment to a party (defined in the "assigned_to"
1394    attribute), i.e., if a party that is owner of the rule
1395    is obliged/allowed/forbidden to perform the update of the
1396    status of the element. The "executive_action_type"
1397    sub-construct indicates through the "Targets" element the
1398    execution of which process specifications must
1399    be/may be/cannot be started when the rule fires. The "type"
1400    attribute indicates the deontic assignment to a party,
1401    i.e., if a party (defined in the "assigned_to" attribute)
1402    is obliged/allowed/forbidden to start the execution of the
1403    process. The "copier_action_type" sub-construct indicates
1404    which element (the "target_element" element) or its
1405    attribute (the "target_attribute" attribute) is given
1406    a new value (the "new_value" attribute). Again the "type"
1407    attribute indicates the deontic assignment on the owner
1408    of the rule.-->
1409    <xs:complexType name="enabler_action_type">
1410        <xs:sequence>
1411            <xs:element name="target_element" type="xs:IDREF"/>
1412            <xs:element name="new_state" type="state_type"/>
1413        </xs:sequence>
1414        <xs:attribute name="type" type="constraint_type"/>
1415        <xs:attribute name="assigned_to" type="xs:IDREF"
```

```
1416        use="optional"/>
1417    </xs:complexType>
1418    <xs:complexType name="executive_action_type">
1419        <xs:sequence>
1420            <xs:element name="targets" type="xs:IDREFS"/>
1421        </xs:sequence>
1422        <xs:attribute name="type" type="constraint_type"/>
1423        <xs:attribute name="assigned_to" type="xs:IDREF"
1424        use="optional"/>
1425        <xs:attribute name="repeatable" type="xs:boolean"/>
1426    </xs:complexType>
1427    <xs:complexType name="copier_action_type">
1428        <xs:sequence>
1429            <xs:element name="target_element" type="xs:IDREF"/>
1430            <xs:element name="target_attribute"
1431            type="xs:string" minOccurs="0"/>
1432            <xs:element name="new_value"
1433            type="list_of_strings_type" minOccurs="0"/>
1434            <!-- besides the assignment of one value, a party
1435            can be allowed to set one of a set of values (for
1436            example in the control of a proces status). -->
1437        </xs:sequence>
1438        <xs:attribute name="type" type="constraint_type"/>
1439        <xs:attribute name="assigned_to" type="xs:IDREF"
1440        use="optional"/>
1441    </xs:complexType>
1442    <xs:complexType name="reaction_rule_type">
1443        <xs:sequence>
1444            <xs:element name="rule_conditions" type="expression"/>
1445            <xs:choice>
1446                <xs:element name="enabler_action"
1447                type="enabler_action_type"/>
1448                <xs:element name="executive_action"
1449                type="executive_action_type"/>
1450                <xs:element name="copier_action"
1451                type="copier_action_type"/>
1452            </xs:choice>
1453        </xs:sequence>
1454        <xs:attributeGroup ref="common_rule_attributes"/>
1455    </xs:complexType>
1456    <!-- Next, we provide the construct for free-text
1457    rules. -->
1458    <xs:complexType name="free_text_rule_type">
1459        <xs:simpleContent>
1460            <xs:extension base="xs:string">
1461                <xs:attributeGroup
1462                ref="common_rule_attributes"/>
1463            </xs:extension>
1464        </xs:simpleContent>
1465    </xs:complexType>
1466    <!-- PART 3 -->
1467    <!-- For the definition of process specification constructs
1468    in eSML we use as a foundation XRL (eXchangeable Routing
1469    Language). XRL is an instance-based workflow language that
```

```
1470    uses XML for the representation of process definitions and
1471    Petri nets for its semantics. A few modifications have been
1472    applied on XRL. These extensions are marked with
1473    commentaries. XRL is a prescriptive language which defines
1474    explicitly the control flow between activities.
1475    XRL was chosen as process specification language for several
1476    reasons. First, its XML representation and short schema
1477    definition allow easy integration and extension to suit the
1478    eSML goals. Second, the underlying Petri Net semantics in
1479    XRL allows the Petri net representation of an XRL process
1480    specification to be analyzed using state-of-the-art analysis
1481    techniques and tools. Finally, the experience and background
1482    of using XRL in the IS group at the Technical University
1483    of Eindhoven allowed easier implementation and maintenance
1484    in eSML. More information about XRL and its original
1485    DTD/schema can be found at:
1486    http://tmitwww.tm.tue.nl/staff/anorta/XRL/xrlHome.html  -->
1487    <xs:simpleType name="probability">
1488        <xs:restriction base="xs:decimal">
1489            <xs:minInclusive value="0"/>
1490            <xs:maxInclusive value="1"/>
1491        </xs:restriction>
1492    </xs:simpleType>
1493    <xs:attributeGroup name="logistic_attributes">
1494        <xs:attribute name="cost" type="xs:decimal" u
1495        se="optional"/>
1496        <xs:attribute name="min_duration" type="xs:time"
1497        use="optional"/>
1498        <xs:attribute name="max_duration" type="xs:time"
1499        use="optional"/>
1500        <xs:attribute name="avg_duration" type="xs:time"
1501        use="optional"/>
1502        <xs:attribute name="quality" type="probability"
1503        use="optional"/>
1504        <!-- quality is defined as how high is the probability
1505        that the active node (task, block, process) outputs
1506        the required data-->
1507    </xs:attributeGroup>
1508    <xs:attributeGroup name="contextual_information_attributes">
1509        <xs:attribute name="goal" type="xs:string"
1510        use="optional"/>
1511        <xs:attribute name="risk" type="xs:string"
1512        use="optional"/>
1513        <xs:attribute name="project_accomplishment"
1514        type="xs:string" use="optional"/>
1515        <xs:attribute name="handbook_reference"
1516        type="xs:anyURI" use="optional"/>
1517    </xs:attributeGroup>
1518    <xs:group name="common_elements">
1519        <xs:choice>
1520            <xs:element name="interface_in"
1521            type="interface_type"/>
1522            <xs:element name="interface_out"
1523            type="interface_type"/>
```

```
1524            <xs:element name="element_id"
1525            type="xs:ID" minOccurs="0"/>
1526            <xs:element name="task" type="task_type"/>
1527            <xs:element name="sequence"
1528            type="sequence_type"/>
1529            <xs:element name="any_sequence"
1530            type="any_sequence_type"/>
1531            <xs:element name="choice" type="choice_type"/>
1532            <xs:element name="condition"
1533            type="condition_type"/>
1534            <xs:element name="parallel_sync"
1535            type="parallel_sync_type"/>
1536            <xs:element name="parallel_no_sync"
1537            type="parallel_no_sync_type"/>
1538            <xs:element name="parallel_part_sync"
1539            type="parallel_part_sync_type"/>
1540            <xs:element name="parallel_part_sync_cancel"
1541            type="parallel_part_sync_cancel_type"/>
1542            <xs:element name="restricted_parallel_sync"
1543            type="restricted_parallel_sync_type"/>
1544            <xs:element name="wait_all" type="wait_all_type"/>
1545            <xs:element name="wait_any" type="wait_any_type"/>
1546            <xs:element name="while_do" type="while_do_type"/>
1547            <xs:element name="terminate" type="terminate_type"/>
1548            <xs:element name="send_task" type="send_task_type"/>
1549            <xs:element name="receive_task" type="task_type"/>
1550            <xs:element name="bi_directional_task"
1551            type="send_task_type"/>
1552            <xs:element name="send_transition"
1553            type="send_transition_type"/>
1554            <xs:element name="receive_transition"
1555            type="transition_type"/>
1556            <xs:element name="bi_directional_transition"
1557            type="send_transition_type"/>
1558            <xs:element name="sourcing_sphere"
1559            type="sourcing_sphere_type" minOccurs="0"
1560            maxOccurs="unbounded"/>
1561            <xs:element name="data" type="data_type"
1562            minOccurs="0" maxOccurs="unbounded"/>
1563            <xs:element name="lock_change" type="lock_change_type"
1564            minOccurs="0"/>
1565            <!-- By placing a data definition into the
1566            common_elements group, data visibility can be defined on
1567            a group level, e.g., a sequence, parallel_sync, etc. If the
1568            data definition takes place on a rout level, the range
1569            stretches over the entire process. -->
1570            <!-- Adding data definition in the common elements allows
1571            visibility on a process and block level. -->
1572        </xs:choice>
1573    </xs:group>
1574    <!-- The lock_type supports workflow data patterns 30 and 31.
1575    Locks that are initially set or not set when a data package is
1576    defined may be altered with the  lock_change tag. -->
1577    <xs:complexType name="lock_change_type">
```

```
1578        <xs:sequence>
1579            <xs:element name="package" type="xs:IDREF"
1580            maxOccurs="unbounded"/>
1581            <xs:element name="variable" type="xs:IDREF"
1582            minOccurs="0" maxOccurs="unbounded"/>
1583            <xs:element name="new_lock" type="lock_type"/>
1584        </xs:sequence>
1585    </xs:complexType>
1586    <!-- The lock_type supports workflow data patterns 29
1587    and 30 -->
1588    <xs:simpleType name="lock_type">
1589        <xs:restriction base="xs:string">
1590            <xs:enumeration value="read_lock"/>
1591            <xs:enumeration value="write_lock"/>
1592            <xs:enumeration value="exclusive_lock"/>
1593            <xs:enumeration value="none"/>
1594        </xs:restriction>
1595    </xs:simpleType>
1596    <xs:complexType name="lock_definition_type">
1597        <xs:sequence>
1598            <xs:element name="lock_owner" type="xs:IDREFS"
1599            maxOccurs="unbounded"/>
1600            <xs:element name="lock" type="lock_type"/>
1601            <!-- A lock owner may be a resource or a modelling
1602            construct, e.g., a scope, a process, etc. -->
1603        </xs:sequence>
1604    </xs:complexType>
1605    <!-- Data-definition tags. -->
1606    <!-- A package of data may either be input or output. -->
1607    <!--May be used to support worflow data pattern 9 and 10.
1608    States whether a package is input, output, or in_out -->
1609    <xs:simpleType name="data_flow_direction_type">
1610        <xs:restriction base="xs:string">
1611            <xs:enumeration value="input"/>
1612            <xs:enumeration value="output"/>
1613            <xs:enumeration value="in_output"/>
1614        </xs:restriction>
1615    </xs:simpleType>
1616    <xs:complexType name="instance_type">
1617        <xs:sequence>
1618            <xs:element name="destination_instance"
1619            type="xs:IDREFS"/>
1620            <xs:choice>
1621                <xs:element name="all_instances"
1622                type="xs:boolean" minOccurs="0"/>
1623                <xs:element name="instance_number"
1624                type="xs:integer" minOccurs="0"/>
1625                <!-- Target all instances of a task
1626                -->
1627                <!-- Target a particular task-instance
1628                number -->
1629            </xs:choice>
1630        </xs:sequence>
1631    </xs:complexType>
```

```
1632    <!-- Packages can be either passed from a source or to
1633    a target. -->
1634    <!--A passing destination defines where a package is
1635    passed on to. -->
1636    <!-- Workflow data pattern 13 is supported by defining
1637    a destination case for a package that is valid in
1638    another case. -->
1639    <!-- Also workflow data patterns 11 and 12 are supported
1640    if the a data package passing from task to task is
1641    including a multiple-instance task. Such a task is
1642    located in a parallel_non_sync blok. -->
1643    <xs:complexType name="passing_type">
1644        <xs:choice>
1645            <xs:element name="aim_task" type="instance_type"
1646            minOccurs="0" maxOccurs="unbounded"/>
1647            <xs:element name="aim_block" type="xs:IDREF"
1648            minOccurs="0" maxOccurs="unbounded"/>
1649            <xs:element name="aim_scope" type="xs:IDREF"
1650            minOccurs="0" maxOccurs="unbounded"/>
1651            <xs:element name="aim_sphere" type="xs:IDREF"
1652            minOccurs="0" maxOccurs="unbounded"/>
1653            <xs:element name="aim_case" type="instance_type"
1654            minOccurs="0" maxOccurs="unbounded"/>
1655            <xs:element name="aim_process" type="xs:IDREF"
1656            minOccurs="0" maxOccurs="unbounded"/>
1657        </xs:choice>
1658    </xs:complexType>
1659    <!-- Variables and their values of one package can be
1660    mapped to variables of a destination package. -->
1661    <xs:complexType name="variable_mapping_type">
1662        <xs:sequence>
1663            <xs:element name="from_variable_name"
1664            type="xs:IDREF"/>
1665            <xs:element name="to_variable_name"
1666            type="xs:IDREF"/>
1667            <xs:element name="transformation_function"
1668            type="xs:string" minOccurs="0"/>
1669            <!-- A variable that is mapped may first
1670            be transformed by a function. -->
1671        </xs:sequence>
1672    </xs:complexType>
1673    <xs:complexType name="package_mapping_type">
1674        <xs:sequence>
1675            <xs:element name="aim_package"
1676            type="xs:IDREF"/>
1677            <xs:element name="variable_mapping"
1678            type="variable_mapping_type" minOccurs="0"
1679            maxOccurs="unbounded"/>
1680        </xs:sequence>
1681    </xs:complexType>
1682    <!-- Data may be visible in one or many cases. -->
1683    <xs:complexType name="case_visibility_type">
1684        <xs:sequence maxOccurs="unbounded">
1685            <xs:element name="case_id" type="xs:IDREFS"/>
```

```
1686            </xs:sequence>
1687        </xs:complexType>
1688        <!-- Coverage of workflow data patterns 33 til including 36.
1689        -->
1690        <!--This tag is useful for active nodes to define as a
1691        precondition for enactment the existance of a particular
1692        variable.-->
1693        <xs:complexType name="data_existence_condition_type">
1694            <xs:sequence>
1695                <xs:element name="package" type="xs:IDREF"/>
1696                <xs:element name="variable" type="xs:IDREF"
1697                maxOccurs="unbounded"/>
1698            </xs:sequence>
1699        </xs:complexType>
1700        <!-- Applicable  in active nodes to check the value of a
1701        particular variable.-->
1702        <xs:complexType name="data_value_condition_type">
1703            <xs:sequence>
1704                <xs:element name="package" type="xs:IDREF"/>
1705                <xs:element name="value_check_statement"
1706                type="xs:string" maxOccurs="unbounded"/>
1707            </xs:sequence>
1708        </xs:complexType>
1709        <!-- Below data-package definitions are given for supporting
1710        various data-flow patterns.-->
1711        <!--Workflow data pattern 2 is supported by defining data
1712        package visibility on a block level.-->
1713        <!-- Supports workflow data pattern 9. When no further
1714        explicit assignment definitions given, the third option
1715        of no data passing in workflow data pattern 9 is performed.
1716        That means all lower-level elements are automatically aware
1717        of the data package. If additional assignment tags are used,
1718        then either data passing is performed via a dedicated
1719        data channel or via an integrated control and data channel,
1720        i.e., in the latter case data flows along control flow.-->
1721        <!-- case_visibility supports workflow data pattern 5.
1722        Here it can be defined if a data package is visible in
1723        all current cases of a workflow process.-->
1724        <!-- With case_visibility_range a limitation in accordance
1725        with workflow data pattern 5 is achieved. The set of cases
1726        can be specified where a data package is equally visible.-->
1727        <!-- passing_destination supports workflow data pattern
1728        8 for passing data patterns from task to task.-->
1729        <!--passing_destination is also useful for supporting
1730        workflow data pattern 9 and 10.  When data_type definition
1731        is on block level. Then explicit data passing from a block
1732        level to a contained lower-level element can be performed.
1733        In case of supporting pattern 10, task-level data packages
1734        can be assigned explicitly to the higher level, e.g.,
1735        a block.-->
1736        <!--control_flow_passing is useful for workflow data
1737        pattern 9. When data_types are defined on a block level,
1738        setting control_flow_passing to true means that an integrated
1739        control and data channel is used. That means data flows
```

```
1740      from one node to the next along control flow. -->
1741      <!--If workflow_visibility_range is true, a package is
1742      visible in all cases of a process for all active nodes
1743      contained. Supports workflow data pattern 6 -->
1744      <!--If case_visibility is true, a data package is
1745      visibility for all cases. Covers workflow data pattern 5.
1746      Comparable to class variables in OO.-->
1747      <!-- Workflow data pattern 4 is supported  if a data
1748      package is defined for a task that is instantiated several
1749      times, e.g., when a task is part of a parallel_no_sync
1750      block. -->
1751      <!--Workflow data patterns concerned with internal and
1752      external data passing (patterns 8 - 25) can be supported
1753      with the tags passing_destination and passing_origin in
1754      combination with various tags concerned with visibility.
1755      -->
1756      <!--Workflow data patterns 26 and 27 are concerned with
1757      incoming and outgoing data transfer by value. Such support
1758      can be achieved when data packages are defined for respective
1759      tasks and cases. When data needs to be passed from one
1760      task to another by value, a mapping of a variable and value
1761      can first take place to a case package. Subsequently the
1762      variable mapping to a destination task can be repeated
1763      from a case level.  -->
1764      <!--Workflow data patterns 27  is supported by using a
1765      data package on a case level from which variables are
1766      retrieved into a data package on a task level. The data
1767      with their values are placed back into the case data
1768      package when the task is near completion.-->
1769      <!--Workflow data patterns 25 and 26 are supported with
1770      using tags passing_destination and passing_origin.-->
1771      <!--Workflow data patterns 31 and 32 are supported with
1772      using destination_package_mapping and source_package_mapping.
1773      Instead of passing a data package per se, parts of a data
1774      package in one process node are mapped on properties of
1775      another data package at some other location. -->
1776      <!--Workflow data patterns 33 till 39 are covered by
1777      business rules data are not part of the data_type tag. -->
1778      <!--By using sub_level_visibility in combination with
1779      a data_package definition for a control-flow block element,
1780      it can be determined to which lower level the data_package
1781      is visible. For example, if a block has 5 lower levels of
1782      routing elements and level 4 is defined in a
1783      sub_level_visibility tag, then elements located on the
1784      lowest level don't␣have␣visibility␣of␣the␣data␣package,
1785 ␣␣␣␣i.e.,␣the␣5th␣level␣below␣the␣definition␣level␣of␣the
1786 ␣␣␣␣data␣package␣in␣question.␣-->
1787 ␣␣␣␣<xs:complexType␣name="data_type">
1788 ␣␣␣␣␣␣␣␣<xs:sequence>
1789 ␣␣␣␣␣␣␣␣␣␣␣␣<xs:element␣name="data_flow_direction"
1790 ␣␣␣␣␣␣␣␣␣␣␣␣type="data_flow_direction_type"/>
1791 ␣␣␣␣␣␣␣␣␣␣␣␣<xs:element␣name="workflow_visibility_range"
1792 ␣␣␣␣␣␣␣␣␣␣␣␣type="xs:boolean"␣minOccurs="0"/>
1793 ␣␣␣␣␣␣␣␣␣␣␣␣<xs:element␣name="case_visibility"␣type="xs:boolean"
```

```
1794 ␣␣␣␣␣␣␣␣␣␣␣␣minOccurs="0"/>
1795             <xs:element name="case_visibility_range"
1796              type="case_visibility_type" minOccurs="0"/>
1797              <xs:element name="passing_destination"
1798              type="passing_type" minOccurs="0"/>
1799              <xs:element name="passing_origin"
1800              type="passing_type" minOccurs="0"/>
1801              <xs:element name="control_flow_passing"
1802              type="xs:boolean" minOccurs="0"/>
1803              <xs:element name="destination_package_mapping"
1804              type="package_mapping_type" minOccurs="0"
1805              maxOccurs="unbounded"/>
1806              <xs:element name="source_package_mapping"
1807              type="package_mapping_type" minOccurs="0"
1808              maxOccurs="unbounded"/>
1809              <xs:element name="sub_level_visibility"
1810              type="xs:integer" minOccurs="0"/>
1811              <xs:element name="data_package_ref"
1812              type="xs:IDREF"/>
1813          </xs:sequence>
1814      </xs:complexType>
1815      <!--A scope comprises of several active nodes on a process
1816      that share a set of data. It covers workflow data pattern 3
1817      -->
1818      <xs:complexType name="data_scope_type">
1819          <xs:complexContent>
1820              <xs:extension base="data_type">
1821                  <xs:sequence>
1822                      <xs:element name="process" type="xs:IDREF"/>
1823                      <xs:element name="active_nodes"
1824                      type="xs:IDREFS" maxOccurs="unbounded"/>
1825                  </xs:sequence>
1826              </xs:extension>
1827          </xs:complexContent>
1828      </xs:complexType>
1829      <!--The following definition creates packages of data that
1830      may comprise of variables together with their optional values
1831      and documents.-->
1832      <xs:complexType name="data_package_type">
1833          <xs:sequence>
1834              <xs:element name="package_id" type="xs:ID"/>
1835              <xs:element name="var_section"
1836              type="variables_def_section" minOccurs="0"/>
1837              <xs:element name="document_section"
1838              type="list_of_documents" minOccurs="0"/>
1839          </xs:sequence>
1840      </xs:complexType>
1841      <!--Conjoinment perspective supporting types. -->
1842      <!--<xs:complexType name="conjoinment_base_type">
1843          <xs:sequence>
1844              <xs:element name="destination_URI"
1845              type="xs:anyURI"/>
1846          </xs:sequence>
1847      </xs:complexType>-->
```

```
<xs:complexType name="send_task_type">
    <xs:complexContent>
        <xs:extension base="task_type">
            <xs:sequence>
                <xs:element name="destination_URI"
                type="xs:anyURI" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="send_transition_type">
    <xs:complexContent>
        <xs:extension base="transition_type">
            <xs:sequence>
                <xs:element name="destination_URI"
                type="xs:anyURI"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="sourcing_sphere_type">
    <xs:sequence>
        <xs:element name="sphere_id" type="xs:ID"/>
        <xs:element name="owner" type="xs:IDREF"
        minOccurs="0"/>
        <xs:element name="description" type="xs:string"
        maxOccurs="unbounded"/>
        <xs:element name="data" type="data_type"
        minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="lock_change"
        type="lock_change_type" minOccurs="0"/>
        <xs:group ref="common_elements"
        maxOccurs="unbounded"/>
        <xs:element name="interface_in"
        type="interface_type"/>
        <xs:element name="interface_out"
        type="interface_type"/>
        <!-- should refer to a party -->
    </xs:sequence>
    <!-- When multi-lateral contracting takes place,
    the consumer process contains multiple Sourcing
    spheres. Each Sourcing sphere in the consumer
    process is complemented by a service providing
    process. In the latter case sphere comprises the
    entire service-provision process. -->
</xs:complexType>
<xs:complexType name="any_sequence_type">
    <xs:sequence>
        <xs:group ref="common_elements"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup
    ref="logistic_attributes"/>
    <xs:attributeGroup
```

```
1902            ref="contextual_information_attributes"/>
1903        </xs:complexType>
1904        <xs:complexType name="choice_type">
1905            <xs:sequence>
1906                <xs:group ref="common_elements"
1907                maxOccurs="unbounded"/>
1908            </xs:sequence>
1909            <xs:attributeGroup ref="logistic_attributes"/>
1910            <xs:attributeGroup
1911            ref="contextual_information_attributes"/>
1912        </xs:complexType>
1913        <xs:complexType name="condition_type">
1914            <xs:choice minOccurs="0" maxOccurs="unbounded">
1915                <xs:element name="true_branch"
1916                type="true_branch_type"/>
1917                <xs:element name="false_branch"
1918                type="false_branch_type"/>
1919            </xs:choice>
1920            <xs:attribute name="condition"
1921            type="xs:string" use="required"/>
1922            <xs:attribute name="description"
1923            type="xs:string"/>
1924            <xs:attributeGroup ref="logistic_attributes"/>
1925            <xs:attributeGroup
1926            ref="contextual_information_attributes"/>
1927        </xs:complexType>
1928        <xs:complexType name="false_branch_type">
1929            <xs:sequence>
1930                <xs:group ref="common_elements"/>
1931            </xs:sequence>
1932            <xs:attributeGroup ref="logistic_attributes"/>
1933            <xs:attributeGroup
1934            ref="contextual_information_attributes"/>
1935        </xs:complexType>
1936        <xs:complexType name="parallel_no_sync_type">
1937            <xs:sequence>
1938                <xs:group ref="common_elements"
1939                maxOccurs="unbounded"/>
1940            </xs:sequence>
1941            <xs:attributeGroup ref="logistic_attributes"/>
1942            <xs:attributeGroup
1943            ref="contextual_information_attributes"/>
1944        </xs:complexType>
1945        <xs:complexType name="parallel_part_sync_type">
1946            <xs:sequence>
1947                <xs:group ref="common_elements"
1948                maxOccurs="unbounded"/>
1949            </xs:sequence>
1950            <xs:attribute name="number" type="xs:NMTOKEN"
1951            use="required"/>
1952            <xs:attributeGroup ref="logistic_attributes"/>
1953            <xs:attributeGroup
1954            ref="contextual_information_attributes"/>
1955        </xs:complexType>
```

```
1956    <xs:complexType name="parallel_part_sync_cancel_type">
1957        <xs:sequence>
1958            <xs:group ref="common_elements"
1959            maxOccurs="unbounded"/>
1960        </xs:sequence>
1961        <xs:attribute name="number" type="xs:NMTOKEN"
1962        use="required"/>
1963        <xs:attributeGroup ref="logistic_attributes"/>
1964        <xs:attributeGroup
1965        ref="contextual_information_attributes"/>
1966    </xs:complexType>
1967    <xs:complexType name="parallel_sync_type">
1968        <xs:sequence>
1969            <xs:group ref="common_elements"
1970            maxOccurs="unbounded"/>
1971        </xs:sequence>
1972        <xs:attributeGroup ref="logistic_attributes"/>
1973        <xs:attributeGroup
1974        ref="contextual_information_attributes"/>
1975    </xs:complexType>
1976    <xs:complexType name="restricted_parallel_sync_type">
1977        <xs:sequence>
1978            <xs:group ref="common_elements"
1979            maxOccurs="unbounded"/>
1980        </xs:sequence>
1981        <xs:attributeGroup ref="logistic_attributes"/>
1982        <xs:attributeGroup
1983        ref="contextual_information_attributes"/>
1984    </xs:complexType>
1985    <xs:complexType name="sequence_type">
1986        <xs:sequence>
1987            <xs:group ref="common_elements"
1988            maxOccurs="unbounded"/>
1989        </xs:sequence>
1990        <xs:attributeGroup ref="logistic_attributes"/>
1991        <xs:attributeGroup
1992        ref="contextual_information_attributes"/>
1993    </xs:complexType>
1994    <!--data_existence_precondition supports workflow data
1995    pattern 34.-->
1996    <!--data_existence_postcondition supports workflow data
1997    pattern 36.-->
1998    <!--data_value_precondition supports workflow data pattern
1999    35.-->
2000    <!--data_value_postcondition supports workflow data pattern
2001    37.-->
2002    <xs:complexType name="transition_type">
2003        <xs:sequence>
2004            <xs:element name="event" type="event_type"
2005            minOccurs="0" maxOccurs="unbounded"/>
2006            <xs:element name="data" type="data_type"
2007            minOccurs="0" maxOccurs="unbounded"/>
2008            <xs:element name="lock_change"
2009            type="lock_change_type" minOccurs="0"/>
```

```
2010            <xs:element name="applied_rules" type="xs:IDREFS"
2011            minOccurs="0" maxOccurs="unbounded"/>
2012            <xs:element name="data_existence_precondition"
2013            type="data_existence_condition_type" minOccurs="0"
2014            maxOccurs="unbounded"/>
2015            <xs:element name="data_value_precondition"
2016            type="data_value_condition_type" minOccurs="0"
2017            maxOccurs="unbounded"/>
2018            <xs:element name="data_existence_postcondition"
2019            type="data_existence_condition_type" minOccurs="0"
2020            maxOccurs="unbounded"/>
2021            <xs:element name="data_value_postcondition"
2022            type="data_value_condition_type" minOccurs="0"
2023            maxOccurs="unbounded"/>
2024            <!-- Support of workflow data pattern 1. Data block
2025            visibility for the  task level.-->
2026            <!-- this tag covers pattern 1 of the workflow data
2027            patterns. That way data visibility is realized on a
2028            task level. -->
2029        </xs:sequence>
2030        <xs:attribute name="active_node_id" type="xs:ID"
2031        use="required"/>
2032        <xs:attribute name="name" type="xs:string" use="required"/>
2033        <xs:attribute name="address" type="xs:string"
2034        use="optional"/>
2035    </xs:complexType>
2036    <!-- interface_type is relevant for defining the interfaces
2037    for supporting a black box pattern -->
2038    <xs:complexType name="interface_type">
2039        <xs:complexContent>
2040            <xs:extension base="transition_type">
2041            </xs:extension>
2042        </xs:complexContent>
2043    </xs:complexType>
2044    <xs:complexType name="task_type">
2045        <xs:complexContent>
2046            <xs:extension base="transition_type">
2047                <xs:attribute name="owner" type="xs:IDREF"
2048                use="optional"/>
2049                <xs:attribute name="executor"
2050                type="xs:IDREF" use="optional"/>
2051                <xs:attribute name="responsible"
2052                type="xs:IDREF" use="optional"/>
2053                <xs:attribute name="necessary_resources"
2054                type="xs:IDREF" use="optional"/>
2055                <xs:attribute name="result" type="xs:string"/>
2056                <xs:attribute name="notify" type="xs:string"/>
2057                <xs:attribute name="enabled" type="state_type"/>
2058                <xs:attribute name="start_date"
2059                type="xs:date" use="optional"/>
2060                <xs:attribute name="start_time"
2061                type="xs:time" use="optional"/>
2062                <xs:attribute name="end_date" type="xs:date"
2063                use="optional"/>
```

```
2064                <xs:attribute name="end_time" type="xs:time"
2065                use="optional"/>
2066                <xs:attributeGroup ref="logistic_attributes"/>
2067                <xs:attributeGroup
2068                ref="contextual_information_attributes"/>
2069            </xs:extension>
2070        </xs:complexContent>
2071    </xs:complexType>
2072    <xs:complexType name="terminate_type"/>
2073    <xs:complexType name="timeout_type">
2074        <xs:sequence>
2075            <xs:group ref="common_elements" minOccurs="0"/>
2076        </xs:sequence>
2077        <xs:attribute name="time" type="xs:string" use="required"/>
2078        <xs:attribute name="type" default="absolute">
2079            <xs:simpleType>
2080                <xs:restriction base="xs:NMTOKEN">
2081                    <xs:enumeration value="relative"/>
2082                    <xs:enumeration value="s_relative"/>
2083                    <xs:enumeration value="absolute"/>
2084                </xs:restriction>
2085            </xs:simpleType>
2086        </xs:attribute>
2087        <xs:attributeGroup ref="logistic_attributes"/>
2088        <xs:attributeGroup ref="contextual_information_attributes"/>
2089    </xs:complexType>
2090    <xs:complexType name="true_branch_type">
2091        <xs:sequence>
2092            <xs:group ref="common_elements"/>
2093        </xs:sequence>
2094        <xs:attributeGroup ref="logistic_attributes"/>
2095        <xs:attributeGroup ref="contextual_information_attributes"/>
2096    </xs:complexType>
2097    <xs:complexType name="wait_all_type">
2098        <xs:choice maxOccurs="unbounded">
2099            <xs:element name="event_ref" type="xs:IDREF"/>
2100            <xs:element name="timeout" type="timeout_type"/>
2101        </xs:choice>
2102        <xs:attributeGroup ref="logistic_attributes"/>
2103        <xs:attributeGroup ref="contextual_information_attributes"/>
2104    </xs:complexType>
2105    <xs:complexType name="wait_any_type">
2106        <xs:choice maxOccurs="unbounded">
2107            <xs:element name="event_ref" type="xs:IDREF"/>
2108            <xs:element name="timeout" type="timeout_type"/>
2109        </xs:choice>
2110        <xs:attributeGroup ref="logistic_attributes"/>
2111        <xs:attributeGroup ref="contextual_information_attributes"/>
2112    </xs:complexType>
2113    <xs:complexType name="while_do_type">
2114        <xs:sequence>
2115            <xs:group ref="common_elements"/>
2116        </xs:sequence>
2117        <xs:attribute name="condition" type="xs:string"
```

```
2118        use="required"/>
2119        <xs:attributeGroup ref="logistic_attributes"/>
2120        <xs:attributeGroup
2121        ref="contextual_information_attributes"/>
2122    </xs:complexType>
2123    <xs:complexType name="route">
2124        <xs:sequence>
2125            <xs:group ref="common_elements"/>
2126            <xs:element name="data_scope"
2127            type="data_scope_type" minOccurs="0"
2128            maxOccurs="unbounded"/>
2129            <!--A Data_scope comprises of several active
2130            nodes, i.e., task, transition, and all active
2131            conjoinment nodes. Data_scopes cover workflow
2132            data pattern 3.-->
2133        </xs:sequence>
2134        <xs:attribute name="tag_name" type="xs:string"
2135        use="required"/>
2136        <xs:attribute name="process_id" type="xs:ID"
2137        use="required"/>
2138        <xs:attribute name="enabled" type="state_type"/>
2139        <xs:attribute name="created_by" type="xs:string"/>
2140        <xs:attribute name="date" type="xs:string"/>
2141        <xs:attributeGroup ref="logistic_attributes"/>
2142        <xs:attributeGroup
2143        ref="contextual_information_attributes"/>
2144        <!-- Control over the process and monitoring rights
2145        and obligations are specified through deontic
2146        assignments and reaction rules (executive for
2147        monitoring and copier reaction rules for control
2148        rights)-->
2149        <!-- extension -->
2150    </xs:complexType>
2151    <!-- PART 4 Syntax -->
2152    <!-- In this part, we provide the syntax definitions
2153    of eSML, i.e., the structure of eSML. First, we
2154    provide three basic structures, i.e., the Variable
2155    definition section, Rule definition section, and Process
2156    definition section. In each of these sections can
2157    respectively be defined data constructs, rule constructs,
2158    and process constructs.-->
2159    <xs:complexType name="variables_def_section">
2160        <xs:sequence maxOccurs="unbounded">
2161            <xs:choice>
2162                <xs:element name="string_var"
2163                type="string_type"/>
2164                <xs:element name="real_var"
2165                type="real_type"/>
2166                <xs:element name="integer_var"
2167                type="integer_type"/>
2168                <xs:element name="boolean_var"
2169                type="boolean_type"/>
2170                <xs:element name="date_var"
2171                type="date_type"/>
```

```
2172                <xs:element name="time_var"
2173                type="time_type"/>
2174                <xs:element name="event_var"
2175                type="event_type"/>
2176                <xs:element name="money_var"
2177                type="money_type"/>
2178                <xs:element
2179                name="external_resource_reference_var"
2180                type="external_resource_reference_type"/>
2181                <xs:element name="list_of_events_var"
2182                type="list_of_events_type"/>
2183                <xs:element name="list_of_strings_var"
2184                type="list_of_strings_type"/>
2185                <xs:any/>
2186                <!-- <xs:element
2187                name="internal_resource_reference_var"
2188                type="internal_resource_reference_type"/>
2189                -->
2190                <!--Supports workflow data pattern 7.
2191                This way environment data can be brought
2192                into a data package.-->
2193                <!-- Required to allow the definition of
2194                user-defined complex types and lists -->
2195            </xs:choice>
2196            <xs:element name="lock_definition"
2197            type="lock_definition_type" minOccurs="0"/>
2198            <!-- For  data definitions of a data package
2199            an initial lock can be placed.-->
2200        </xs:sequence>
2201    </xs:complexType>
2202    <xs:complexType name="rule_def_section">
2203        <xs:sequence maxOccurs="unbounded">
2204            <xs:choice>
2205                <xs:element name="state_constraint_rule"
2206                type="state_constraint_rule_type"
2207                minOccurs="0" maxOccurs="unbounded"/>
2208                <xs:element name="dynamic_constraint_rule"
2209                type="dynamic_constraint_rule_type"
2210                minOccurs="0" maxOccurs="unbounded"/>
2211                <xs:element name="computational_derivation_rule"
2212                type="computational_derivation_rule_type"
2213                minOccurs="0" maxOccurs="unbounded"/>
2214                <xs:element name="linguistic_derivation_rule"
2215                type="linguistic_derivation_rule_type"
2216                minOccurs="0" maxOccurs="unbounded"/>
2217                <xs:element name="reaction_rule"
2218                type="reaction_rule_type" minOccurs="0"
2219                maxOccurs="unbounded"/>
2220                <xs:element name="free_text_rule"
2221                type="free_text_rule_type" minOccurs="0"
2222                maxOccurs="unbounded"/>
2223            </xs:choice>
2224        </xs:sequence>
2225    </xs:complexType>
```

```
2226    <!-- Next, the monitorability patterns are defined.-->
2227    <xs:complexType name="sink_type">
2228        <xs:sequence maxOccurs="unbounded">
2229            <xs:element name="provider_sphere"
2230            type="xs:IDREF"/>
2231            <xs:element name="provider_active_node"
2232            type="xs:IDREF"/>
2233        </xs:sequence>
2234    </xs:complexType>
2235    <xs:complexType name="link_properties">
2236        <xs:sequence>
2237            <xs:element name="consumer_sphere"
2238            type="xs:IDREF"/>
2239            <xs:element name="consumer_active_node"
2240            type="xs:IDREF"/>
2241            <xs:element name="provider"
2242            type="sink_type"/>
2243        </xs:sequence>
2244    </xs:complexType>
2245    <xs:complexType name="enactment_properties">
2246        <xs:sequence>
2247            <xs:element name="consumer_sphere"
2248            type="xs:IDREF"/>
2249            <xs:element name="provider_sphere"
2250            type="xs:IDREF"/>
2251        </xs:sequence>
2252    </xs:complexType>
2253    <xs:complexType name="lifecycle_link_properties">
2254        <xs:sequence>
2255            <xs:element name="consumer_sphere"
2256            type="xs:IDREF"/>
2257            <xs:element name="consumer_active_node"
2258            type="xs:IDREF"/>
2259            <xs:element name="consumer_lifecycle_node"
2260            type="xs:IDREF"/>
2261            <xs:element name="provider_sphere"
2262            type="xs:IDREF"/>
2263            <xs:element name="provider_active_node"
2264            type="xs:IDREF"/>
2265            <xs:element name="provider_lifecycle_node"
2266            type="xs:IDREF"/>
2267        </xs:sequence>
2268    </xs:complexType>
2269    <xs:complexType name="lifecycle_polling_patterns">
2270        <xs:sequence maxOccurs="unbounded">
2271            <xs:choice>
2272                <xs:element
2273                name="lifecycle_transition_polling"
2274                type="lifecycle_link_properties"
2275                minOccurs="0" maxOccurs="unbounded"/>
2276                <xs:element
2277                name="lifecycle_state_polling"
2278                type="lifecycle_link_properties"
2279                minOccurs="0" maxOccurs="unbounded"/>
```

```
2280            </xs:choice>
2281        </xs:sequence>
2282    </xs:complexType>
2283    <xs:complexType name="lifecycle_messaging_patterns">
2284        <xs:sequence maxOccurs="unbounded">
2285            <xs:choice>
2286                <xs:element
2287                name="lifecycle_transition_messaging"
2288                type="lifecycle_link_properties"
2289                minOccurs="0" maxOccurs="unbounded"/>
2290                <xs:element
2291                name="lifecycle_state_messaging"
2292                type="lifecycle_link_properties"
2293                minOccurs="0" maxOccurs="unbounded"/>
2294            </xs:choice>
2295        </xs:sequence>
2296    </xs:complexType>
2297    <xs:complexType name="polling_patterns">
2298        <xs:choice maxOccurs="unbounded">
2299            <xs:element name="enactment_takeover"
2300            type="enactment_properties" minOccurs="0"
2301            maxOccurs="unbounded"/>
2302            <xs:element name="termination_takeover"
2303            type="enactment_properties" minOccurs="0"
2304            maxOccurs="unbounded"/>
2305            <xs:element name="transition_polling"
2306            type="link_properties" minOccurs="0"
2307            maxOccurs="unbounded"/>
2308            <xs:element name="lifecycle_polling"
2309            type="lifecycle_polling_patterns"
2310            minOccurs="0" maxOccurs="unbounded"/>
2311        </xs:choice>
2312    </xs:complexType>
2313    <xs:complexType name="messaging_patterns">
2314        <xs:sequence maxOccurs="unbounded">
2315            <xs:choice>
2316                <xs:element name="enactment_propagation"
2317                type="enactment_properties" minOccurs="0"
2318                maxOccurs="unbounded"/>
2319                <xs:element name="termination_propagation"
2320                type="enactment_properties" minOccurs="0"
2321                maxOccurs="unbounded"/>
2322                <xs:element name="transition_messaging"
2323                type="link_properties" minOccurs="0"
2324                maxOccurs="unbounded"/>
2325                <xs:element name="lifecycle_messaging"
2326                type="lifecycle_messaging_patterns"
2327                minOccurs="0" maxOccurs="unbounded"/>
2328            </xs:choice>
2329        </xs:sequence>
2330    </xs:complexType>
2331    <xs:complexType name="monitorability_patterns">
2332        <xs:sequence>
2333            <xs:element name="polling" type="polling_patterns"
```

```
2334            minOccurs="0"/>
2335            <xs:element name="messaging"
2336            type="messaging_patterns" minOccurs="0"/>
2337        </xs:sequence>
2338    </xs:complexType>
2339    <!-- The definition of life cycles is given. Life cycles may
2340    be present for  a process level, or on a task level. A life
2341    cycle has control-flow too.-->
2342    <xs:complexType name="lifecycle_node_type">
2343        <xs:attribute name="name" type="xs:ID" use="required"/>
2344        <xs:attribute name="tag_name" type="xs:string" u
2345        se="required"/>
2346    </xs:complexType>
2347    <xs:complexType name="lifecycle_sequence_type">
2348        <xs:sequence>
2349            <xs:group ref="lifecycle_elements"
2350            maxOccurs="unbounded"/>
2351        </xs:sequence>
2352    </xs:complexType>
2353    <!-- A nesting state contains further life-cycle nodes. -->
2354    <xs:complexType name="nesting_state_type">
2355        <xs:sequence>
2356            <xs:group ref="lifecycle_elements"
2357            maxOccurs="unbounded"/>
2358        </xs:sequence>
2359    </xs:complexType>
2360    <xs:complexType name="lifecycle_parallel_sync_type">
2361        <xs:sequence>
2362            <xs:group ref="lifecycle_elements"
2363            maxOccurs="unbounded"/>
2364        </xs:sequence>
2365    </xs:complexType>
2366    <xs:complexType name="lifecycle_parallel_no_sync_type">
2367        <xs:sequence>
2368            <xs:group ref="lifecycle_elements"
2369            maxOccurs="unbounded"/>
2370        </xs:sequence>
2371    </xs:complexType>
2372    <xs:complexType name="lifecycle_parallel_part_sync_type">
2373        <xs:sequence>
2374            <xs:group ref="lifecycle_elements"
2375            maxOccurs="unbounded"/>
2376        </xs:sequence>
2377        <xs:attribute name="number" type="xs:NMTOKEN"
2378        use="required"/>
2379    </xs:complexType>
2380    <xs:complexType
2381    name="lifecycle_parallel_part_sync_cancel_type">
2382        <xs:sequence>
2383            <xs:group ref="lifecycle_elements"
2384            maxOccurs="unbounded"/>
2385        </xs:sequence>
2386        <xs:attribute name="number" type="xs:NMTOKEN"
2387        use="required"/>
```

```
2388    </xs:complexType>
2389    <!-- Below, all elements of a life cycle are grouped
2390    together. -->
2391    <xs:group name="lifecycle_elements">
2392        <xs:choice>
2393            <xs:element name="transition"
2394            type="lifecycle_node_type"/>
2395            <xs:element name="nesting_state"
2396            type="nesting_state_type"/>
2397            <xs:element name="atomic_state"
2398            type="lifecycle_node_type"/>
2399            <xs:element name="lifecycle_sequence"
2400            type="lifecycle_sequence_type"/>
2401            <xs:element name="lifecycle_parallel_sync"
2402            type="lifecycle_parallel_sync_type"/>
2403            <xs:element name="lifecycle_parallel_no_sync"
2404            type="lifecycle_parallel_no_sync_type"/>
2405            <xs:element name="lifecycle_parallel_part_sync"
2406            type="lifecycle_parallel_part_sync_type"/>
2407            <xs:element name="lifecycle_parallel_part_sync_cancel"
2408            type="lifecycle_parallel_part_sync_cancel_type"/>
2409        </xs:choice>
2410    </xs:group>
2411    <xs:complexType name="lifecycle_details">
2412        <xs:sequence>
2413            <xs:group ref="lifecycle_elements"/>
2414        </xs:sequence>
2415        <xs:attribute name="process_id" type="xs:IDREF"/>
2416    </xs:complexType>
2417    <xs:complexType name="lifecycles">
2418        <xs:sequence>
2419            <xs:element name="process_lifecycle"
2420            type="lifecycle_details" minOccurs="0"
2421            maxOccurs="unbounded"/>
2422            <xs:element name="active_node_lifecycle"
2423            type="lifecycle_details" minOccurs="0"
2424            maxOccurs="unbounded"/>
2425        </xs:sequence>
2426    </xs:complexType>
2427    <!-- Next, the mapping of life-cycle stages is defined-->
2428    <xs:complexType name="lifecycle_mapping_details">
2429        <xs:complexContent>
2430            <xs:extension base="link_properties">
2431                <xs:attribute name="mapping_name"
2432                type="xs:ID" use="required"/>
2433                <xs:attribute name="node_type">
2434                    <xs:simpleType>
2435                        <xs:restriction base="xs:string">
2436                            <xs:pattern
2437                            value="lifecycle_transition|
2438                            lifecycle_state"/>
2439                        </xs:restriction>
2440                    </xs:simpleType>
2441                </xs:attribute>
```

```
2442            </xs:extension>
2443        </xs:complexContent>
2444    </xs:complexType>
2445    <xs:complexType name="mapping_details">
2446        <xs:sequence>
2447            <xs:element name="process_lifecycle_mapping"
2448            type="lifecycle_mapping_details" minOccurs="0"
2449            maxOccurs="unbounded"/>
2450            <xs:element name="active_node_lifecycle_mapping"
2451            type="lifecycle_mapping_details" minOccurs="0"
2452            maxOccurs="unbounded"/>
2453        </xs:sequence>
2454    </xs:complexType>
2455    <xs:complexType name="provider_type">
2456        <xs:sequence maxOccurs="unbounded">
2457            <xs:element name="provider_process"
2458            type="xs:IDREF"/>
2459            <xs:element name="provider_active_node"
2460            type="xs:IDREF"/>
2461        </xs:sequence>
2462    </xs:complexType>
2463    <xs:complexType name="active_node_label_mapping_type">
2464        <xs:sequence>
2465            <xs:element name="consumer_process"
2466            type="xs:IDREF"/>
2467            <xs:element name="consumer_active_node"
2468            type="xs:IDREF"/>
2469            <xs:element name="provider_process"
2470            type="xs:IDREF"/>
2471            <xs:element name="provider_active_node"
2472            type="xs:IDREF"/>
2473            <xs:element name="provider"
2474            type="provider_type"/>
2475        </xs:sequence>
2476    </xs:complexType>
2477    <!-- Next, the resource-perspective definition for
2478    covering the organizational aspect.-->
2479    <!-- An organizational unit and its subclasses is
2480    defined. These subclasses are permanent_organizational_unit
2481    and temporary_organizational_unit -->
2482    <xs:group name="organizational_unit_elements">
2483        <xs:sequence>
2484            <xs:element name="name" type="xs:ID"/>
2485            <xs:element name="start_date" type="xs:date"
2486            maxOccurs="unbounded"/>
2487            <xs:element name="description" type="xs:string"
2488            maxOccurs="unbounded"/>
2489            <xs:element name="business_objectives"
2490            type="xs:string" maxOccurs="unbounded"/>
2491        </xs:sequence>
2492    </xs:group>
2493    <xs:complexType name="temporary_organizational_unit_type">
2494        <xs:complexContent>
2495            <xs:extension base="organizational_unit_type">
```

```
2496                <xs:sequence>
2497                    <xs:element name="end_date" type="xs:date"/>
2498                </xs:sequence>
2499            </xs:extension>
2500        </xs:complexContent>
2501    </xs:complexType>
2502    <xs:complexType name="permanent_organizational_unit_type">
2503        <xs:complexContent>
2504            <xs:extension base="organizational_unit_type"/>
2505        </xs:complexContent>
2506    </xs:complexType>
2507    <xs:complexType name="organizational_unit_type">
2508        <xs:sequence>
2509            <xs:group ref="organizational_unit_elements"/>
2510            <xs:element name="resource_type" type="xs:IDREF"
2511            minOccurs="0" maxOccurs="unbounded"/>
2512            <xs:element name="resource_nref"
2513            type="resource_nref_type" minOccurs="0"
2514            maxOccurs="unbounded"/>
2515            <xs:element name="collection" type="xs:IDREF"
2516            minOccurs="0" maxOccurs="unbounded"/>
2517            <xs:element name="individual_resource"
2518            type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
2519        </xs:sequence>
2520    </xs:complexType>
2521    <xs:complexType name="organizational_position_type">
2522        <xs:sequence>
2523            <xs:element name="description" type="xs:string"
2524            minOccurs="0" maxOccurs="unbounded"/>
2525            <xs:element name="organizational_unit"
2526            type="xs:IDREF" maxOccurs="unbounded"/>
2527            <xs:element name="actor" type="xs:IDREF"
2528            minOccurs="0" maxOccurs="unbounded"/>
2529            <xs:element name="privilege" type="xs:IDREF"
2530            minOccurs="0" maxOccurs="unbounded"/>
2531        </xs:sequence>
2532    </xs:complexType>
2533    <!-- Several resource_type may be part of organizational_units.
2534    A resource_type has subclasses, namely a role, a machine,
2535    space, and production_material. Furthermore, a resource_type
2536    is referred by a typed_collection.-->
2537    <xs:group name="resource_perspective">
2538        <xs:choice minOccurs="0" maxOccurs="unbounded">
2539            <xs:element name="machine_type" type="machine_type_type"
2540            minOccurs="0" maxOccurs="unbounded"/>
2541            <xs:element name="production_material_type"
2542            type="production_material_type_type" minOccurs="0"
2543            maxOccurs="unbounded"/>
2544            <xs:element name="space_type" type="space_type_type"
2545            minOccurs="0" maxOccurs="unbounded"/>
2546            <xs:element name="capability" type="capability_type"
2547            minOccurs="0" maxOccurs="unbounded"/>
2548            <xs:element name="privilege" type="privilege_type"
2549            minOccurs="0" maxOccurs="unbounded"/>
```

```
2550            <xs:element name="power" type="power_type" minOccurs="0"
2551            maxOccurs="unbounded"/>
2552            <xs:element name="power_delegation"
2553            type="power_delegation_type" minOccurs="0"
2554            maxOccurs="unbounded"/>
2555            <xs:element name="concrete_collection"
2556            type="concrete_collection_type" minOccurs="0"
2557            maxOccurs="unbounded"/>
2558            <xs:element name="mixed_collection"
2559            type="mixed_collection_type" minOccurs="0"
2560            maxOccurs="unbounded"/>
2561            <xs:element name="typed_collection"
2562            type="typed_collection_type" minOccurs="0"
2563            maxOccurs="unbounded"/>
2564            <xs:element name="resource_nref"
2565            type="resource_nref_type" minOccurs="0"
2566            maxOccurs="unbounded"/>
2567            <xs:element name="available" type="available_type"
2568            minOccurs="0" maxOccurs="unbounded"/>
2569            <xs:element name="space" type="space_type"
2570            minOccurs="0" maxOccurs="unbounded"/>
2571            <xs:element name="production_material"
2572            type="production_material_type" minOccurs="0"
2573            maxOccurs="unbounded"/>
2574            <xs:element name="containment_type"
2575            type="containment_type_type" minOccurs="0"
2576            maxOccurs="unbounded"/>
2577            <xs:element name="machine" type="machine_type"
2578            minOccurs="0" maxOccurs="unbounded"/>
2579            <xs:element name="capacity" type="capacity_type"
2580            minOccurs="0" maxOccurs="unbounded"/>
2581            <xs:element name="rate_of_usage"
2582            type="rate_of_usage_type" minOccurs="0"
2583            maxOccurs="unbounded"/>
2584            <xs:element name="actor" type="actor_type"
2585            minOccurs="0" maxOccurs="unbounded"/>
2586            <xs:element name="role_delegation"
2587            type="role_delegation_type" minOccurs="0"
2588            maxOccurs="unbounded"/>
2589            <xs:element name="appointment"
2590            type="appointment_type" minOccurs="0"
2591            maxOccurs="unbounded"/>
2592            <xs:element name="correlation"
2593            type="correlation_type" minOccurs="0"
2594            maxOccurs="unbounded"/>
2595            <xs:element name="hierarchy_relationship"
2596            type="hierarchy_relationship_type"
2597            minOccurs="0" maxOccurs="unbounded"/>
2598            <xs:element name="temporary_organizational_unit"
2599            type="temporary_organizational_unit_type"
2600            minOccurs="0" maxOccurs="unbounded"/>
2601            <xs:element name="permanent_organizational_unit"
2602            type="permanent_organizational_unit_type"
2603            minOccurs="0" maxOccurs="unbounded"/>
```

```
            <xs:element name="organizational_position"
            type="organizational_position_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="role" type="role_type"
            minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="connection_relationship"
            type="connection_relationship_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="resource_type"
            type="resource_type_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="collection"
            type="collection_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="individual_resource"
            type="individual_resource_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="non_actor" type="non_actor_type"
            minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="consumable_resource"
            type="consumable_resource_type" minOccurs="0"
            maxOccurs="unbounded"/>
            <xs:element name="organizational_unit"
            type="organizational_unit_type" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:choice>
    </xs:group>
    <xs:group name="resource_type_elements">
        <xs:sequence>
            <xs:element name="name" type="xs:ID"/>
            <xs:element name="description" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
    <xs:complexType name="resource_type_type">
        <xs:sequence>
            <xs:group ref="resource_type_elements"/>
            <xs:element name="typed_collection" type="xs:IDREF"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="machine_type_type">
        <xs:complexContent>
            <xs:extension base="resource_type_type"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="production_material_type_type">
        <xs:complexContent>
            <xs:extension base="resource_type_type">
                <xs:sequence>
                    <xs:element name="production_material"
                    type="xs:IDREF"/>
                </xs:sequence>
            </xs:extension>
```

```
2658            </xs:complexContent>
2659        </xs:complexType>
2660        <xs:complexType name="space_type_type">
2661            <xs:complexContent>
2662                <xs:extension base="resource_type_type">
2663                    <xs:sequence>
2664                        <xs:element name="space" type="xs:IDREF"/>
2665                    </xs:sequence>
2666                </xs:extension>
2667            </xs:complexContent>
2668        </xs:complexType>
2669        <!--A role definition  and all related entities.-->
2670        <xs:group name="role_elements">
2671            <xs:sequence>
2672                <xs:element name="qualifications_required"
2673                type="xs:string" minOccurs="0"
2674                maxOccurs="unbounded"/>
2675                <xs:element name="skills_required" type="xs:string"
2676                minOccurs="0" maxOccurs="unbounded"/>
2677            </xs:sequence>
2678        </xs:group>
2679        <xs:complexType name="role_type">
2680            <xs:complexContent>
2681                <xs:extension base="resource_type_type">
2682                    <xs:sequence>
2683                        <xs:group ref="role_elements"/>
2684                        <xs:element name="power" type="xs:IDREF"
2685                        minOccurs="0" maxOccurs="unbounded"/>
2686                        <xs:element name="capability"
2687                        type="xs:IDREF" minOccurs="0"
2688                        maxOccurs="unbounded"/>
2689                        <!-- this is refering to required
2690                        capabilities for a certain role -->
2691                    </xs:sequence>
2692                </xs:extension>
2693            </xs:complexContent>
2694        </xs:complexType>
2695        <xs:group name="feature_elements">
2696            <xs:sequence>
2697                <xs:element name="name" type="xs:ID"/>
2698                <xs:element name="description" type="xs:string"
2699                minOccurs="0" maxOccurs="unbounded"/>
2700            </xs:sequence>
2701        </xs:group>
2702        <xs:complexType name="capability_type">
2703            <xs:sequence>
2704                <xs:group ref="feature_elements"/>
2705            </xs:sequence>
2706        </xs:complexType>
2707        <xs:complexType name="privilege_type">
2708            <xs:sequence>
2709                <xs:group ref="feature_elements"/>
2710                <xs:element name="role" type="xs:IDREF"/>
2711            </xs:sequence>
```

```
2712    </xs:complexType>
2713    <xs:complexType name="power_type">
2714        <xs:sequence>
2715            <xs:group ref="feature_elements"/>
2716            <xs:element name="capability" type="xs:IDREF"
2717            minOccurs="0" maxOccurs="unbounded"/>
2718            <xs:element name="privilege" type="xs:IDREF"
2719            minOccurs="0" maxOccurs="unbounded"/>
2720        </xs:sequence>
2721    </xs:complexType>
2722    <xs:complexType name="power_delegation_type">
2723        <xs:sequence>
2724            <xs:group ref="timing_elements"/>
2725            <xs:element name="superior_role" type="xs:IDREF"
2726            minOccurs="0" maxOccurs="unbounded"/>
2727            <xs:element name="inferior_role" type="xs:IDREF"
2728            minOccurs="0" maxOccurs="unbounded"/>
2729            <xs:element name="power" type="xs:IDREF"
2730            minOccurs="0"/>
2731        </xs:sequence>
2732    </xs:complexType>
2733    <!--The definition of a collection and all subclasses.
2734    These subclasses are mixed_collection, concrete_collection,
2735    and typed_collection-->
2736    <xs:group name="collection_elements">
2737        <xs:sequence>
2738            <xs:element name="name" type="xs:ID"/>
2739            <xs:element name="description" type="xs:string"
2740            minOccurs="0" maxOccurs="unbounded"/>
2741        </xs:sequence>
2742    </xs:group>
2743    <xs:complexType name="collection_type">
2744        <xs:sequence>
2745            <xs:group ref="collection_elements"/>
2746            <xs:element name="organizational_unit"
2747            type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
2748        </xs:sequence>
2749    </xs:complexType>
2750    <xs:complexType name="concrete_collection_type">
2751        <xs:complexContent>
2752            <xs:extension base="collection_type">
2753                <xs:sequence>
2754                    <xs:element name="individual_resource"
2755                    type="xs:IDREF" maxOccurs="unbounded"/>
2756                </xs:sequence>
2757            </xs:extension>
2758        </xs:complexContent>
2759    </xs:complexType>
2760    <xs:complexType name="mixed_collection_type">
2761        <xs:complexContent>
2762            <xs:extension base="collection_type">
2763                <xs:sequence>
2764                    <xs:element name="resource_nref"
2765                    type="resource_nref_type" maxOccurs="unbounded"/>
```

```
2766                    <xs:element name="individual_resource"
2767                    type="xs:IDREF" maxOccurs="unbounded"/>
2768                </xs:sequence>
2769            </xs:extension>
2770        </xs:complexContent>
2771    </xs:complexType>
2772    <xs:complexType name="typed_collection_type">
2773        <xs:complexContent>
2774            <xs:extension base="collection_type">
2775                <xs:sequence>
2776                    <xs:element name="resource_nref_type"
2777                    type="resource_nref_type" maxOccurs="unbounded"/>
2778                </xs:sequence>
2779            </xs:extension>
2780        </xs:complexContent>
2781    </xs:complexType>
2782    <xs:complexType name="resource_nref_type">
2783        <xs:sequence>
2784            <xs:element name="resource_type_ref"
2785            type="xs:IDREF"/>
2786            <xs:element name="number" type="xs:integer"/>
2787            <xs:element name="typed_collection_reference"
2788            type="xs:IDREF"/>
2789            <xs:element name="mixed_collection_reference"
2790            type="xs:IDREF"/>
2791        </xs:sequence>
2792    </xs:complexType>
2793    <!-- Resource related definitions. -->
2794    <xs:group name="individual_resource_elements">
2795        <xs:sequence>
2796            <xs:element name="name" type="xs:ID"/>
2797            <xs:element name="address" type="xs:string"
2798            maxOccurs="unbounded"/>
2799        </xs:sequence>
2800    </xs:group>
2801    <xs:complexType name="individual_resource_type">
2802        <xs:sequence>
2803            <xs:group ref="individual_resource_elements"/>
2804            <xs:element name="available" type="xs:IDREF"
2805            minOccurs="0" maxOccurs="unbounded"/>
2806            <xs:element name="concrete_collection"
2807            type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
2808            <xs:element name="mixed_collection"
2809            type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
2810            <xs:element name="organizational_unit"
2811            type="xs:IDREF" maxOccurs="unbounded"/>
2812            <xs:element name="capability" type="xs:IDREF"
2813            minOccurs="0" maxOccurs="unbounded"/>
2814        </xs:sequence>
2815    </xs:complexType>
2816    <xs:group name="available_type_elements">
2817        <xs:sequence>
2818            <xs:element name="number" type="xs:ID"/>
2819            <xs:element name="start_date" type="xs:date"
```

```
2820            minOccurs="0"/>
2821            <xs:element name="start_time" type="xs:time"
2822            minOccurs="0"/>
2823            <xs:element name="end_date" type="xs:date"
2824            minOccurs="0"/>
2825            <xs:element name="end_time" type="xs:time"
2826            minOccurs="0"/>
2827            <xs:element name="status" type="xs:string"
2828            minOccurs="0"/>
2829            <xs:element name="reserved_for" type="xs:string"
2830            minOccurs="0" maxOccurs="unbounded"/>
2831        </xs:sequence>
2832    </xs:group>
2833    <xs:complexType name="available_type">
2834        <xs:sequence>
2835            <xs:group ref="available_type_elements"/>
2836            <xs:element name="individual_resource"
2837            type="xs:IDREF" minOccurs="0"/>
2838        </xs:sequence>
2839    </xs:complexType>
2840    <!-- non_actor definition with their subclasses.-->
2841    <xs:group name="non_actor_elements">
2842        <xs:sequence>
2843            <xs:element name="description" type="xs:string"
2844            minOccurs="0" maxOccurs="unbounded"/>
2845            <xs:element name="contact" type="xs:string"
2846            minOccurs="0" maxOccurs="unbounded"/>
2847            <xs:element name="phone" type="xs:integer"
2848            minOccurs="0" maxOccurs="unbounded"/>
2849        </xs:sequence>
2850    </xs:group>
2851    <xs:complexType name="non_actor_type">
2852        <xs:complexContent>
2853            <xs:extension base="individual_resource_type">
2854                <xs:sequence>
2855                    <xs:group ref="non_actor_elements"/>
2856                </xs:sequence>
2857            </xs:extension>
2858        </xs:complexContent>
2859    </xs:complexType>
2860    <xs:group name="space_elements">
2861        <xs:sequence>
2862            <xs:element name="building_number"
2863            type="xs:string"/>
2864            <xs:element name="room_number"
2865            type="xs:string"/>
2866            <xs:element name="description"
2867            type="xs:string" minOccurs="0"
2868            maxOccurs="unbounded"/>
2869            <xs:element name="capacity" type="xs:string"/>
2870        </xs:sequence>
2871    </xs:group>
2872    <xs:complexType name="space_type">
2873        <xs:complexContent>
```

```
2874            <xs:extension base="non_actor_type">
2875                <xs:sequence>
2876                    <xs:group ref="space_elements"/>
2877                </xs:sequence>
2878            </xs:extension>
2879        </xs:complexContent>
2880    </xs:complexType>
2881    <xs:complexType name="consumable_resource_type">
2882        <xs:complexContent>
2883            <xs:extension base="non_actor_type"/>
2884        </xs:complexContent>
2885    </xs:complexType>
2886    <xs:complexType name="production_material_type">
2887        <xs:complexContent>
2888            <xs:extension
2889            base="consumable_resource_type"/>
2890        </xs:complexContent>
2891    </xs:complexType>
2892    <xs:complexType name="containment_type_type">
2893        <xs:sequence>
2894            <xs:element name="superior_production_material"
2895            type="xs:IDREF"/>
2896            <xs:element name="inferior_production_material"
2897            type="xs:IDREF"/>
2898            <xs:element name="description" type="xs:string"
2899            maxOccurs="unbounded"/>
2900        </xs:sequence>
2901    </xs:complexType>
2902    <xs:complexType name="connection_relationship_type">
2903        <xs:sequence>
2904            <xs:element name="source_production_material"
2905            type="xs:IDREF"/>
2906            <xs:element name="target_production_material"
2907            type="xs:IDREF"/>
2908            <xs:element name="description" type="xs:string"
2909            maxOccurs="unbounded"/>
2910        </xs:sequence>
2911    </xs:complexType>
2912    <xs:complexType name="machine_type">
2913        <xs:complexContent>
2914            <xs:extension base="consumable_resource_type">
2915                <xs:sequence>
2916                    <xs:element name="number" type="xs:string"
2917                    maxOccurs="unbounded"/>
2918                </xs:sequence>
2919            </xs:extension>
2920        </xs:complexContent>
2921    </xs:complexType>
2922    <xs:group name="capacity_elements">
2923        <xs:sequence>
2924            <xs:element name="amount" type="xs:decimal"/>
2925            <xs:element name="unit" type="xs:string"/>
2926        </xs:sequence>
2927    </xs:group>
```

```
2928    <xs:complexType name="capacity_type">
2929        <xs:sequence>
2930            <xs:group ref="capacity_elements"/>
2931            <xs:element name="machine" type="xs:IDREF"/>
2932            <xs:element name="production_material" type="xs:IDREF"
2933            minOccurs="0" maxOccurs="unbounded"/>
2934        </xs:sequence>
2935    </xs:complexType>
2936    <xs:group name="rate_of_usage_elements">
2937        <xs:sequence>
2938            <xs:element name="usage_quantity" type="xs:decimal"/>
2939            <xs:element name="usage_period" type="xs:decimal"/>
2940            <xs:element name="period_unit" type="xs:string"/>
2941        </xs:sequence>
2942    </xs:group>
2943    <xs:complexType name="rate_of_usage_type">
2944        <xs:sequence>
2945            <xs:group ref="rate_of_usage_elements"/>
2946            <xs:element name="machine" type="xs:IDREF"/>
2947            <xs:element name="production_material" type="xs:IDREF"
2948            minOccurs="0" maxOccurs="unbounded"/>
2949        </xs:sequence>
2950    </xs:complexType>
2951    <!-- actor definition with their subclasses.-->
2952    <xs:group name="actor_elements">
2953        <xs:sequence>
2954            <xs:element name="actor_id" type="xs:ID"/>
2955            <xs:element name="first_name" type="xs:string"/>
2956            <xs:element name="last_name" type="xs:string"/>
2957            <xs:element name="email" type="xs:string"
2958            maxOccurs="unbounded"/>
2959            <xs:element name="login_id" type="xs:string"/>
2960            <xs:element name="phone" type="xs:string"
2961            maxOccurs="unbounded"/>
2962            <xs:element name="qualification" type="xs:string"
2963            minOccurs="0" maxOccurs="unbounded"/>
2964            <xs:element name="skill" type="xs:string" minOccurs="0"
2965            maxOccurs="unbounded"/>
2966            <xs:element name="previous_work_experience"
2967            type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
2968            <xs:element name="held_responsibility" type="xs:string"
2969            minOccurs="0" maxOccurs="unbounded"/>
2970        </xs:sequence>
2971    </xs:group>
2972    <xs:complexType name="actor_type">
2973        <xs:sequence>
2974            <xs:group ref="actor_elements"/>
2975            <xs:element name="role" type="xs:IDREF" minOccurs="0"
2976            maxOccurs="unbounded"/>
2977        </xs:sequence>
2978    </xs:complexType>
2979    <xs:group name="timing_elements">
2980        <xs:sequence>
2981            <xs:element name="description" type="xs:string"
```

```
2982            minOccurs="0" maxOccurs="unbounded"/>
2983            <xs:element name="start_date" type="xs:date"
2984            minOccurs="0"/>
2985            <xs:element name="start_time" type="xs:time"
2986            minOccurs="0"/>
2987            <xs:element name="end_date" type="xs:date"
2988            minOccurs="0"/>
2989            <xs:element name="end_time" type="xs:time"
2990            minOccurs="0"/>
2991        </xs:sequence>
2992    </xs:group>
2993    <xs:complexType name="role_delegation_type">
2994        <xs:sequence>
2995            <xs:group ref="timing_elements"/>
2996            <xs:element name="source_actor" type="xs:IDREF"/>
2997            <xs:element name="target_actor" type="xs:IDREF"/>
2998        </xs:sequence>
2999    </xs:complexType>
3000    <xs:complexType name="appointment_type">
3001        <xs:sequence>
3002            <xs:group ref="timing_elements"/>
3003            <xs:element name="actor" type="xs:IDREF"
3004            maxOccurs="unbounded"/>
3005            <xs:element name="task" type="xs:IDREF"
3006            maxOccurs="unbounded"/>
3007        </xs:sequence>
3008    </xs:complexType>
3009    <!-- The relationships of organizational_unit entities
3010    to each other is established. That includes hierarchy
3011    relationships in terms of superior and inferior, and
3012    a correlation of one organizational_unit to another one,
3013    e.g., a relationship exists because bith are involved
3014    in the same project or in the same business process,
3015    etc.-->
3016    <xs:complexType name="correlation_type">
3017        <xs:sequence>
3018            <xs:element name="description" type="xs:string"
3019            maxOccurs="unbounded"/>
3020            <xs:element name="related_to" type="xs:IDREF"/>
3021            <xs:element name="relation_source" type="xs:IDREF"/>
3022        </xs:sequence>
3023    </xs:complexType>
3024    <xs:complexType name="hierarchy_relationship_type">
3025        <xs:sequence>
3026            <xs:element name="description" type="xs:string"
3027            maxOccurs="unbounded"/>
3028            <xs:element name="superior" type="xs:IDREF"/>
3029            <xs:element name="inferior" type="xs:IDREF"/>
3030        </xs:sequence>
3031    </xs:complexType>
3032    <!-- Next, the process definition wraps up all respective,
3033    available elements.-->
3034    <xs:complexType name="process_def_section">
3035        <xs:sequence>
```

```
3036            <xs:element name="process" type="route"
3037            maxOccurs="unbounded"/>
3038            <xs:element name="lifecycle_definitions"
3039            type="lifecycles" minOccurs="0"/>
3040            <xs:element name="lifecycle_mappings"
3041            type="mapping_details" minOccurs="0"/>
3042            <xs:element name="active_node_label_mapping"
3043            type="active_node_label_mapping_type"
3044            minOccurs="0" maxOccurs="unbounded"/>
3045            <xs:element name="monitorability"
3046            type="monitorability_patterns" minOccurs="0"/>
3047        </xs:sequence>
3048    </xs:complexType>
3049    <!-- Next, we define the possible combinations of data
3050    constructs, rule constructs and process constructs that
3051    can be used in an e-contract sub-structure. -->
3052    <xs:complexType name="only_vars_section">
3053        <xs:sequence>
3054            <xs:element name="var_section"
3055            type="variables_def_section"/>
3056            <xs:element name="process_section"
3057            type="process_def_section" minOccurs="0"/>
3058            <xs:element name="snippet_section"
3059            type="snippet_type" minOccurs="0"/>
3060            <!-- required to support the inclusion of
3061            externally defined data items.-->
3062        </xs:sequence>
3063    </xs:complexType>
3064    <xs:complexType name="vars_and_processes_section">
3065        <xs:sequence>
3066            <xs:element name="var_section"
3067            type="variables_def_section" minOccurs="0"/>
3068            <xs:element name="process_section"
3069            type="process_def_section" minOccurs="0"/>
3070            <xs:element name="snippet_section"
3071            type="snippet_type" minOccurs="0"/>
3072        </xs:sequence>
3073    </xs:complexType>
3074    <xs:complexType name="all_section">
3075        <xs:sequence>
3076            <xs:element name="var_section"
3077            type="variables_def_section" minOccurs="0"/>
3078            <xs:element name="rule_section"
3079            type="rule_def_section" minOccurs="0"/>
3080            <xs:element name="process_section"
3081            type="process_def_section" minOccurs="0"/>
3082            <xs:element name="snippet_section"
3083            type="snippet_type" minOccurs="0"/>
3084        </xs:sequence>
3085    </xs:complexType>
3086    <!-- definition of contract sub-structures of
3087    the 4W structure-->
3088    <xs:complexType name="resource_section_type">
3089        <xs:sequence>
```

```
3090            <xs:group ref="resource_perspective"
3091            minOccurs="0" maxOccurs="unbounded"/>
3092        </xs:sequence>
3093    </xs:complexType>
3094    <xs:complexType name="company_info">
3095        <xs:sequence>
3096            <xs:element name="company_data"
3097            type="only_vars_section"/>
3098            <xs:element name="company_contact_data"
3099            type="only_vars_section"/>
3100            <xs:element name="resource_section"
3101            type="resource_section_type" minOccurs="0"/>
3102            <xs:element name="context_section"
3103            type="only_vars_section" minOccurs="0"/>
3104        </xs:sequence>
3105        <xs:attribute name="local_language"
3106        type="xs:string"/>
3107    </xs:complexType>
3108    <xs:complexType name="value_types">
3109        <xs:choice>
3110            <xs:element name="product"
3111            type="vars_and_processes_section"
3112            minOccurs="0" maxOccurs="unbounded"/>
3113            <xs:element name="service"
3114            type="vars_and_processes_section"
3115            minOccurs="0" maxOccurs="unbounded"/>
3116            <xs:element name="financial_reward"
3117            type="vars_and_processes_section"
3118            minOccurs="0" maxOccurs="unbounded"/>
3119        </xs:choice>
3120    </xs:complexType>
3121    <xs:complexType name="list_of_data_packages">
3122        <xs:sequence maxOccurs="unbounded">
3123            <xs:element name="data_package"
3124            type="data_package_type"/>
3125        </xs:sequence>
3126    </xs:complexType>
3127    <!-- Finally, we define the root element of
3128    the schema and the complete contract
3129    structure -->
3130    <xs:element name="contract">
3131        <xs:complexType>
3132            <xs:sequence>
3133                <xs:element name="party"
3134                type="company_info"
3135                maxOccurs="unbounded"/>
3136                <xs:element name="mediator"
3137                type="company_info" minOccurs="0"
3138                maxOccurs="unbounded"/>
3139                <xs:element name="data_definition_section"
3140                type="list_of_data_packages" minOccurs="0"/>
3141                <xs:element name="business_context_provisions"
3142                type="all_section" minOccurs="0"
3143                maxOccurs="unbounded"/>
```

```
3144                    <xs:element name="legal_context_provisions"
3145                    type="all_section" minOccurs="0"
3146                    maxOccurs="unbounded"/>
3147                    <xs:element name="other_context_provisions"
3148                    type="all_section" minOccurs="0"
3149                    maxOccurs="unbounded"/>
3150                    <xs:element name="exchanged_value"
3151                    type="value_types" minOccurs="2"
3152                    maxOccurs="unbounded"/>
3153                    <xs:element name="exchange_provisions"
3154                    type="all_section" minOccurs="2"
3155                    maxOccurs="unbounded"/>
3156                    <xs:element name="agreed_ontology"
3157                    type="external_resource_reference_type"
3158                    maxOccurs="unbounded"/>
3159                    <!--WHO Section contains the description
3160                    of the parties and the possible mediators.
3161                    -->
3162                    <!--WHERE Section contains the description
3163                    of the context of the agreement.-->
3164                    <!--WHAT Section contains the description
3165                    of the exchanged goods/services and the
3166                    conditions for the exchange.-->
3167                </xs:sequence>
3168                <xs:attribute name="contract_id" type="xs:ID"/>
3169                <xs:attribute name="global_language" type="xs:string"/>
3170                <xs:attribute name="web_service_uri" type="xs:string"/>
3171            </xs:complexType>
3172        </xs:element>
3173    </xs:schema>
```

Appendix C

eSML Instantiation

```
<?xml version="1.0" encoding="UTF-8"?>
<contract xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\eSML\eSML_lite.xsd"
contract_id="CrossWorkD2.2" global_language="English">
    <!-- Beginning of What concept-->
    <party>
        <company_data>
            <var_section>
                <string_var tag_name="CompanyName" var_id="SP1"
                changeable="false" enabled="enabled">Super
                 Supplier ABC</string_var>
                <string_var tag_name="RegisterNr" var_id="ID2"
                changeable="false"
                 enabled="enabled">1111111</string_var>
                <integer_var tag_name="BankNr" var_id="ID3"
                 changeable="true"
                 enabled="enabled">111000</integer_var>
                <!-- The changeable attribute indicates that the
                 value of the element can be changed by the
                 Owner (DeTelegraaf) freely (without any rules
                 applied as there is not reference to
                 applicable rules; IDNumber illustrates that
                 it is not an often referenced element and does
                 not require a special string; -->
                <integer_var tag_name="BankNr" var_id="ID4"
                changeable="true"
                enabled="enabled">1111111</integer_var>
                <string_var tag_name="Property"
                var_id="AccountManagerSp1" changeable="false"
                enabled="enabled">Account Manager</string_var>
                <string_var tag_name="ContactPerson"
                var_id="ContactPersonSp1" changeable="true"
                enabled="enabled">Max Spar</string_var>
            </var_section>
        </company_data>
        <company_contact_data>
            <var_section>
                <string_var tag_name="MailAddress" var_id="ID5"
```

```
1038                changeable="true" enabled="enabled">Postbox
1039                111, 1000 Linz</string_var>
1040               <string_var tag_name="VisitingAddress"
1041               var_id="ID6" changeable="true"
1042               enabled="enabled">Bergstreet 11, 1111
1043               Linz</string_var>
1044               <integer_var tag_name="Tel" var_id="Tel1"
1045               changeable="true"
1046               enabled="enabled">010111111</integer_var>
1047               <integer_var tag_name="Tel" var_id="Tel2"
1048               changeable="true"
1049               enabled="enabled">01011112</integer_var>
1050               <integer_var tag_name="Fax" var_id="Fax1"
1051               changeable="true"
1052               enabled="enabled">01011113</integer_var>
1053           </var_section>
1054       </company_contact_data>
1055       <resource_section>
1056           <role>
1057               <name>Department_Head</name>
1058           </role>
1059           <role>
1060               <name>Department_Clerk</name>
1061           </role>
1062           <permanent_organizational_unit>
1063               <name>Procurement_Department</name>
1064               <start_date>2005-01-01</start_date>
1065               <description/>
1066               <business_objectives/>
1067               <resource_nref>
1068                   <resource_type_ref>Department_Head
1069                   </resource_type_ref>
1070                   <number>1</number>
1071               </resource_nref>
1072               <resource_nref>
1073                   <resource_type_ref>Department_Clerk
1074                   </resource_type_ref>
1075                   <number>33</number>
1076               </resource_nref>
1077               <individual_resource>Actor2
1078               </individual_resource>
1079           </permanent_organizational_unit>
1080           <permanent_organizational_unit>
1081               <name>Logistic_Department</name>
1082               <start_date>2005-01-01</start_date>
1083               <description/>
1084               <business_objectives/>
1085               <resource_nref>
1086                   <resource_type_ref>Department_Head
1087                   </resource_type_ref>
1088                   <number>1</number>
1089               </resource_nref>
1090               <resource_nref>
1091                   <resource_type_ref>Department_Clerk
```

```
                    </resource_type_ref>
                    <number>200</number>
                </resource_nref>
            </permanent_organizational_unit>
            <permanent_organizational_unit>
                <name>Quality_Assurance_Department</name>
                <start_date>2005-01-01</start_date>
                <description/>
                <business_objectives/>
                <resource_nref>
                    <resource_type_ref>Department_Head
                    </resource_type_ref>
                    <number>1</number>
                </resource_nref>
                <resource_nref>
                    <resource_type_ref>Department_Clerk
                    </resource_type_ref>
                    <number>15</number>
                </resource_nref>
            </permanent_organizational_unit>
            <permanent_organizational_unit>
                <name>Engineering_Department</name>
                <start_date>2005-01-01</start_date>
                <description/>
                <business_objectives/>
                <resource_nref>
                    <resource_type_ref>Department_Head
                    </resource_type_ref>
                    <number>1</number>
                </resource_nref>
                <resource_nref>
                    <resource_type_ref>Department_Clerk
                    </resource_type_ref>
                    <number>150</number>
                </resource_nref>
                <individual_resource>Actor1
                </individual_resource>
            </permanent_organizational_unit>
            <permanent_organizational_unit>
                <name>Production_Department</name>
                <start_date>2005-01-01</start_date>
                <description/>
                <business_objectives/>
                <resource_nref>
                    <resource_type_ref>Department_Head
                    </resource_type_ref>
                    <number>1</number>
                </resource_nref>
                <resource_nref>
                    <resource_type_ref>Department_Clerk
                    </resource_type_ref>
                    <number>1500</number>
                </resource_nref>
                <resource_nref>
```

```
1146                    <resource_type_ref>Driller
1147                    </resource_type_ref>
1148                    <number>35</number>
1149                </resource_nref>
1150            </permanent_organizational_unit>
1151            <typed_collection>
1152                <name>MAN_Decision_Team</name>
1153                <resource_nref_type>
1154                    <resource_type_ref>Department_Head
1155                    </resource_type_ref>
1156                    <number>5</number>
1157                </resource_nref_type>
1158            </typed_collection>
1159            <machine_type>
1160                <name>Driller </name>
1161            </machine_type>
1162            <actor>
1163                <actor_id>Actor2 </actor_id>
1164                <first_name>S</first_name>
1165                <last_name>Buyer</last_name>
1166                <email>sbuyer@abs.com</email>
1167                <login_id>sb</login_id>
1168                <phone>9999</phone>
1169                <role>Department_Head </role>
1170            </actor>
1171            <actor>
1172                <actor_id>Actor1 </actor_id>
1173                <first_name>Max</first_name>
1174                <last_name>Design </last_name>
1175                <email/>
1176                <login_id>mdesign </login_id>
1177                <phone/>
1178                <role>Department_Clerk </role>
1179            </actor>
1180        </resource_section>
1181        <context_section>
1182            <var_section>
1183                <string_var tag_name="goal_priority"
1184                var_id="goalPrio" changeable="true"
1185                enabled="enabled">
1186                4*time+3*quality+2*cost+1*flexibility
1187                </string_var>
1188            </var_section>
1189        </context_section>
1190    </party>
1191    <party>
1192        <company_data>
1193            <var_section>
1194                <string_var tag_name="CompanyName" var_id="SP2"
1195                 changeable="false" enabled="enabled">Component
1196                 Supplier XYZ</string_var>
1197                <string_var tag_name="RegisterNr" var_id="ID12"
1198                changeable="false" enabled="enabled">222222
1199                </string_var>
```

```
1200                <integer_var tag_name="BankNr" var_id="ID13"
1201                changeable="true" enabled="enabled">2222000
1202                </integer_var>
1203                <!-- The changeable attribute indicates that the
1204                value of the element can be changed by the Owner
1205                (DeTelegraaf) freely (without any rules applied
1206                as there is not reference to applicable rules;
1207                IDNumber illustrates that it is not an often
1208                referenced element and does not require a special
1209                string; -->
1210                <integer_var tag_name="BankNr" var_id="ID14"
1211                changeable="true" enabled="enabled">2222222
1212                </integer_var>
1213                <string_var tag_name="Property"
1214                var_id="AccountManagerSp2" changeable="false"
1215                enabled="enabled">Account Manager</string_var>
1216                <string_var tag_name="ContactPerson"
1217                var_id="ContactPersonSp2" changeable="true"
1218                enabled="enabled" owner="SP2">Felix Pods
1219                </string_var>
1220            </var_section>
1221        </company_data>
1222        <company_contact_data>
1223            <var_section>
1224                <string_var tag_name="MailAddress" var_id="ID15"
1225                changeable="true" enabled="enabled">Postbox 2,
1226                2000 Steyr</string_var>
1227                <string_var tag_name="VisitingAddress"
1228                var_id="ID16" changeable="true"
1229                enabled="enabled">Dommelroad 22, 2222 Steyr
1230                </string_var>
1231                <integer_var tag_name="Tel" var_id="Tel1_Sp2"
1232                changeable="true" enabled="enabled">02022222
1233                </integer_var>
1234                <integer_var tag_name="Tel" var_id="Tel2_Sp2"
1235                changeable="true" enabled="enabled">02022221
1236                </integer_var>
1237                <integer_var tag_name="Fax" var_id="Fax1_Sp2"
1238                changeable="true" enabled="enabled">02022223
1239                </integer_var>
1240            </var_section>
1241        </company_contact_data>
1242        <resource_section/>
1243    </party>
1244    <party>
1245        <company_data>
1246            <var_section>
1247                <string_var tag_name="CompanyName" var_id="VE"
1248                changeable="false" enabled="enabled">
1249                Virtual Enterprise NoAE</string_var>
1250                <string_var tag_name="Property"
1251                var_id="Coordinator" changeable="false"
1252                enabled="enabled">NoAE coordinator
1253                </string_var>
```

```
1254                <string_var tag_name="ContactPerson"
1255                var_id="ContactPerson_NoAE" changeable="true"
1256                enabled="enabled">Felix Pods</string_var>
1257            </var_section>
1258        </company_data>
1259        <company_contact_data>
1260            <var_section>
1261                <string_var tag_name="Address"
1262                var_id="AddressOeM" changeable="false"
1263                enabled="enabled">Steyr/Austria</string_var>
1264            </var_section>
1265        </company_contact_data>
1266        <resource_section/>
1267    </party>
1268    <data_definition_section>
1269        <data_package>
1270            <package_id>cd</package_id>
1271            <var_section>
1272                <string_var tag_name="Bill␣of␣Material"
1273                var_id="BOM" changeable="false"
1274                enabled="enabled">Surrounding Box; Gearing
1275                </string_var>
1276            </var_section>
1277            <document_section>
1278                <document>
1279                    <document_id>cadDrawing</document_id>
1280                    <name>Cad Drawing  of complete GearBox
1281                    </name>
1282                    <uri>
1283                    http://www.ve.com/drawings/gearBox.3ds
1284                    </uri>
1285                </document>
1286            </document_section>
1287        </data_package>
1288        <data_package>
1289            <package_id>do</package_id>
1290            <document_section>
1291                <document>
1292                    <document_id>do_doc</document_id>
1293                    <name>Development Order Document</name>
1294                    <uri/>
1295                    <var_section>
1296                        <string_var tag_name="OrderContent"
1297                        var_id="OC" changeable="true"
1298                        enabled="enabled">Develop and Produce
1299                        a GearBox </string_var>
1300                        <string_var tag_name="SupplierID"
1301                        var_id="SID"
1302                        changeable="true" enabled="enabled">
1303                        Suppllier1
1304                        </string_var>
1305                        <string_var tag_name="BuyerID"
1306                        var_id="BID" changeable="true"
1307                        enabled="enabled">Buyer1</string_var>
```

```
1308                </var_section>
1309            </document>
1310        </document_section>
1311    </data_package>
1312    <data_package>
1313        <package_id>Box</package_id>
1314        <var_section>
1315            <string_var tag_name="Surrounding␣Box"
1316            var_id="SB" changeable="true" enabled="enabled">
1317            Surrounding Box Deliverable Data</string_var>
1318        </var_section>
1319    </data_package>
1320    <data_package>
1321        <package_id>GearBox</package_id>
1322        <var_section>
1323            <string_var tag_name="Complete␣Gear␣Box"
1324            var_id="cGB" changeable="true" enabled="enabled">
1325            Complete GearBox Deliverable Data</string_var>
1326        </var_section>
1327    </data_package>
1328    <data_package>
1329        <package_id>cdBox</package_id>
1330        <var_section>
1331            <string_var tag_name="Bill␣of␣Materia␣for␣box"
1332            var_id="BOM_box" changeable="false"
1333            enabled="enabled">Surrounding Box, screws
1334            </string_var>
1335        </var_section>
1336        <document_section>
1337            <document>
1338                <document_id>cadDrawing_Box</document_id>
1339                <name>Cad Drawing  of surrounding Box</name>
1340                <uri>
1341                http://www.ve.com/drawings/surroundingBox.3ds
1342                </uri>
1343            </document>
1344        </document_section>
1345    </data_package>
1346    <data_package>
1347        <package_id>doBox</package_id>
1348        <document_section>
1349            <document>
1350                <document_id>do_Box_doc</document_id>
1351                <name>Surrounding Box Development Order Document
1352                </name>
1353                <uri/>
1354                <var_section>
1355                    <string_var tag_name="BoxOrderContent"
1356                    var_id="BoxOC" changeable="true"
1357                    enabled="enabled">
1358                    Develop and Produce a Surrounding Box
1359                    </string_var>
1360                    <string_var tag_name="SupplierID"
1361                    var_id="BoxSID" changeable="true"
```

```
1362                        enabled="enabled">Supplier1 </string_var>
1363                        <string_var tag_name="BuyerID"
1364                        var_id="BoxBID" changeable="true"
1365                        enabled="enabled">Buyer1 </string_var>
1366                    </var_section>
1367                </document>
1368            </document_section>
1369        </data_package>
1370        <data_package>
1371            <package_id>BankStatement </package_id>
1372            <var_section>
1373                <string_var tag_name="Account␣Number"
1374                var_id="ANr" changeable="false" enabled="enabled">
1375                1111</string_var>
1376                <string_var tag_name="Account␣Holder"
1377                var_id="AH" changeable="false"
1378                enabled="enabled">VE</string_var>
1379            </var_section>
1380        </data_package>
1381        <data_package>
1382            <package_id>TransferBill </package_id>
1383            <var_section>
1384                <string_var tag_name="Sender␣Account"
1385                var_id="SenderAcount" changeable="false"
1386                enabled="enabled">1111</string_var>
1387                <string_var tag_name="Receiver␣Account1"
1388                var_id="ReceiverAccount1" changeable="true"
1389                enabled="enabled">2222</string_var>
1390                <string_var tag_name="Receiver␣Account2"
1391                var_id="ReceiverAccount2" changeable="true"
1392                enabled="enabled">33333</string_var>
1393            </var_section>
1394        </data_package>
1395    </data_definition_section>
1396    <!--    end of Who concept  -->
1397    <!-- Beginning of Where concept -->
1398    <business_context_provisions>
1399        <rule_section>
1400            <reaction_rule tag_name="ComplainRule"
1401            rule_id="ComplainRule" enabled="enabled"
1402            changeable="false">
1403                <rule_conditions>
1404                CurrentDate&lt;ContractDate+60
1405                </rule_conditions>
1406                <executive_action type="may"
1407                assigned_to="VE"
1408                repeatable="true">
1409                    <targets>Complain</targets>
1410                </executive_action>
1411            </reaction_rule>
1412            <reaction_rule tag_name="ForceMajeureRule"
1413            rule_id="ForceMajeureRule" enabled="enabled"
1414            changeable="false">
1415                <rule_conditions>
```

```
1416                (ForceMajeure=true) AND
1417                (ForceMajeure.DateofOccurance+30&lt;CurrentDate)
1418                </rule_conditions>
1419                <executive_action type="may"
1420                assigned_to="VE" repeatable="false">
1421                    <targets>TerminateContract
1422                    </targets>
1423                </executive_action>
1424            </reaction_rule>
1425            <reaction_rule tag_name="TerminateRule"
1426            rule_id="TerminateRule2" enabled="enabled"
1427            changeable="false">
1428                <rule_conditions>ContractBreach=true
1429                </rule_conditions>
1430                <executive_action type="may" assigned_to="VE">
1431                    <targets>TerminateContract</targets>
1432                </executive_action>
1433            </reaction_rule>
1434            <reaction_rule tag_name="TerminateRule"
1435            rule_id="TerminateRule1" enabled="enabled"
1436            changeable="false">
1437                <rule_conditions>(BankruptcyOfVE=true) OR
1438                (MoratoriumByVE=true)</rule_conditions>
1439                <executive_action type="may" assigned_to="SP1">
1440                    <targets>TerminateContract</targets>
1441                </executive_action>
1442            </reaction_rule>
1443        </rule_section>
1444        <process_section>
1445            <process tag_name="Complain" process_id="Complain"
1446            enabled="enabled">
1447                <task name="SendComplain" active_node_id="scomp"
1448                owner="VE" executor="VE" enabled="enabled"/>
1449            </process>
1450            <process tag_name="TerminateContract"
1451            process_id="TerminateContract" enabled="enabled">
1452                <sequence>
1453                    <task name="NotifyServiceProviders"
1454                    active_node_id="nsproviders"
1455                    address="someWebService" executor="VE"
1456                    enabled="enabled"/>
1457                    <task name="StopContract"
1458                    active_node_id="scontract"
1459                    address="someWebService" executor="VE"
1460                    enabled="enabled"/>
1461                </sequence>
1462            </process>
1463        </process_section>
1464    </business_context_provisions>
1465    <legal_context_provisions>
1466        <var_section>
1467            <external_resource_reference_var
1468            tag_name="GeneralProvisions" var_id="ID31"
1469            changeable="true"
```

```
1470            enabled="enabled" is_legally_binding="true"
1471            owner="VE" resource_state="available">
1472            http://service.VE.nl/tarieven/website/index.php?39
1473            </external_resource_reference_var>
1474        </var_section>
1475    </legal_context_provisions>
1476    <!--    end of Where concept -->
1477    <!--    Beginning of What concept -->
1478    <exchanged_value>
1479        <product>
1480            <var_section>
1481                <string_var tag_name="Name"
1482                var_id="name_end_product" changeable="false"
1483                enabled="enabled">Gear Box</string_var>
1484            </var_section>
1485        </product>
1486    </exchanged_value>
1487    <exchanged_value>
1488        <product>
1489            <var_section>
1490                <string_var tag_name="Name"
1491                var_id="NameComponent1" changeable="false"
1492                enabled="enabled">Gearing System</string_var>
1493            </var_section>
1494        </product>
1495    </exchanged_value>
1496    <exchanged_value>
1497        <product>
1498            <var_section>
1499                <string_var tag_name="Name"
1500                var_id="NameComponent2" changeable="false"
1501                enabled="enabled">Surrounding Box
1502                </string_var>
1503            </var_section>
1504        </product>
1505    </exchanged_value>
1506    <exchanged_value>
1507        <service>
1508            <process_section>
1509                <process tag_name="Gearbox_Production"
1510                process_id="GB_production">
1511                    <sequence>
1512                        <sourcing_sphere>
1513                            <sphere_id>SP1_Sphere1</sphere_id>
1514                            <owner>SP1</owner>
1515                            <description>
1516                            This is the first sourcing sphere
1517                            of the SP1
1518                            </description>
1519                            <sequence>
1520                                <receive_transition
1521                                active_node_id="CO"
1522                                name="Receive_Order">
1523                                    <data>
```

```
                    <data_flow_direction>
                    input
                    </data_flow_direction>
                    <data_package_ref>
                    cd
                    </data_package_ref>
               </data>
               <data>
                    <data_flow_direction>
                    input
                    </data_flow_direction>
                    <data_package_ref>
                    do
                    </data_package_ref>
               </data>
          </receive_transition>
          <task
          active_node_id="Check_Gearbox_order"
          name="Check_Order"
          owner="VE"
          executor="SP1">
               <data>
                    <data_flow_direction>
                    input
                    </data_flow_direction>
                    <data_package_ref>
                    cd
                    </data_package_ref>
               </data>
               <data>
                    <data_flow_direction>
                    input
                    </data_flow_direction>
                    <data_package_ref>
                    do
                    </data_package_ref>
               </data>
          </task>
          <parallel_sync>
               <sequence>
                    <task
                    name="Develop_Gearing"
                    active_node_id="DevG"
                    owner="VE"
                    executor="SP1">
                         <event
                         tag_name="cdBox_available"
                         var_id="cdBox_available"
                         changeable="true"
                         enabled="enabled">
                         false
                         </event>
                         <data>
                              <data_flow_direction>
```

```
                output
                </data_flow_direction>
                <data_package_ref>
                cdBox
                </data_package_ref>
            </data>
        </task>
        <task
        name="Produce_Gearing"
        active_node_id="ProG"
        owner="VE"
        executor="SP1"/>
</sequence>
<sequence>
<wait_any>
    <event_ref>
    cdBox_available
    </event_ref>
</wait_any>
<task
name="Order_Box"
active_node_id="OrdB"
owner="VE"
executor="SP1">
    <event
        tag_name="doBox_available"
        var_id="doBox_available"
        changeable="true"
        enabled="enabled">
        false
    </event>
    <data>
        <data_flow_direction>
        output
        </data_flow_direction>
        <data_package_ref>
        doBox
        </data_package_ref>
    </data>
</task>
<send_task
    active_node_id="send_Order_Box_CD"
    name="SendOrderCD">
        <data>
            <data_flow_direction>output
            </data_flow_direction>
            <data_package_ref>
            doBox
            </data_package_ref>
        </data>
        <data>
            <data_flow_direction>
            output
        </data_flow_direction>
```

```
                                <data_package_ref>
                                cdBox
                                </data_package_ref>
                            </data>
<destination_URI>
http://www.ve.com/boxorder.wsdl?port=12212?operation=receive
</destination_URI>
        </send_task>
            <sourcing_sphere>
                <sphere_id>Sphere_SP2</sphere_id>
                <owner>SP2</owner>
                <description>
                Sorucing Sphere of Supllier SP2
                </description>
                <sequence>
                    <receive_task
                        name="Receive_Box_Order"
                        active_node_id="ReceiveBO"
                        owner="VE"
                        executor="SP2">
                            <data>
                                <data_flow_direction>
                                input
                                </data_flow_direction>
                                <data_package_ref>
                                doBox
                                </data_package_ref>
                            </data>
                            <data>
                                <data_flow_direction>
                                input
                                </data_flow_direction>
                                <data_package_ref>
                                cdBox
                                </data_package_ref>
                            </data>
                        </receive_task>
                        <task
                        name="Check_Box_Order"
                        active_node_id="CBO"
                        owner="VE"
                        executor="SP2">
                            <data>
                                <data_flow_direction>
                                input
                                </data_flow_direction>
                                <data_package_ref>
                                doBox
                                </data_package_ref>
                            </data>
                        </task>
                        <task
                        name="Develop_Box"
                    active_node_id="DevB"
```

```
1686                                    owner="VE"
1687                                    executor="SP2">
1688                                        <data>
1689                                            <data_flow_direction>
1690                                            input
1691                                            </data_flow_direction>
1692                                            <data_package_ref>
1693                                            cdBox
1694                                            </data_package_ref>
1695                                        </data>
1696                                    </task>
1697                                    <task
1698                                    name="Produce_Box"
1699                                    active_node_id="ProB"
1700                                    owner="VE"
1701                                    executor="SP2">
1702                                        <event
1703                                        tag_name="Box_available"
1704                                        var_id="Box_available"
1705                                        changeable="true"
1706                                        enabled="enabled">
1707                                        false
1708                                        </event>
1709                                        <data>
1710                                            <data_flow_direction>
1711                                            output
1712                                            </data_flow_direction>
1713                                            <data_package_ref>
1714                                            Box
1715                                            </data_package_ref>
1716                                        </data>
1717                                    </task>
1718                                    <send_task
1719                                    name="Send_Box"
1720                                    active_node_id="SendB"
1721                                    owner="VE"
1722                                    executor="SP2">
1723                                        <data>
1724                                            <data_flow_direction>
1725                                            output
1726                                            </data_flow_direction>
1727                                            <data_package_ref>
1728                                            Box
1729                                            </data_package_ref>
1730                                        </data>
1731                    <destination_URI>
1732                    http://www.ve.com/receivePort.wsdl?operation=1111
1733                    </destination_URI>
1734                                    </send_task>
1735                                </sequence>
1736                                <!-- end of seq within SP2 -->
1737                            </sourcing_sphere>
1738                            <!-- end of Sphere_Sp2 -->
1739                                    </sequence>
```

```
                                <!-- end seq lower branch
                                within parall_sync -->
                            </parallel_sync>
                        </sequence>
                    </sourcing_sphere>
                    <sourcing_sphere>
                        <sphere_id>Sphere_SP1_2</sphere_id>
                        <owner>SP1</owner>
                        <description>
                        This is the second sphere of
                        provider SP1
                        </description>
                        <sequence>
                            <receive_task
                            active_node_id="Receive_Box"
                            name="Receive␣Surrounding␣Box">
                                <data>
                                    <data_flow_direction>
                                    input
                                    </data_flow_direction>
                                    <data_package_ref>
                                    Box
                                    </data_package_ref>
                                </data>
                            </receive_task>
                            <task
                            name="Assemble_Gear_Box"
                            active_node_id="AssB"
                            owner="VE" executor="SP1">
                                <data>
                                    <data_flow_direction>
                                    output
                                    </data_flow_direction>
                                    <data_package_ref>
                                    GearBox
                                    </data_package_ref>
                                </data>
                            </task>
                        </sequence>
                    </sourcing_sphere>
                </sequence>
            </process>
            <lifecycle_definitions>
                <process_lifecycle>
                    <lifecycle_sequence>
                        <atomic_state
                        name="VE_process_ready"
                        tag_name="ready"/>
                        <transition
                        name="VE_process_start_enactment"
                        tag_name="start_enactment"/>
                        <atomic_state
                        name="VE_process_enacting"
                        tag_name="enacting"/>
```

```
1794                        <transition
1795                        name="VE_process_finish_enactment"
1796                        tag_name="finish_enactment"/>
1797                        <atomic_state
1798                        name="VE_process_ended"
1799                        tag_name="ended"/>
1800                    </lifecycle_sequence>
1801                </process_lifecycle>
1802                <active_node_lifecycle>
1803                    <lifecycle_sequence>
1804                        <atomic_state name="VE_active_node_ready"
1805                        tag_name="ready"/>
1806                        <lifecycle_parallel_sync>
1807                            <lifecycle_sequence>
1808                                <transition
1809                                name="VE_active_node_accept"
1810                                tag_name="accept"/>
1811                                <atomic_state
1812                                name="VE_active_node_executing"
1813                                tag_name="executing"/>
1814                            </lifecycle_sequence>
1815                            <lifecycle_sequence>
1816                                <transition
1817                                name="VE_active_node_bypass"
1818                                tag_name="bypass"/>
1819                                <atomic_state
1820                                name="VE_active_node_bypassed"
1821                                tag_name="bypassed"/>
1822                            </lifecycle_sequence>
1823                        </lifecycle_parallel_sync>
1824                        <transition name="VE_active_node_complete"
1825                        tag_name="complete"/>
1826                        <atomic_state name="VE_active_completed"
1827                        tag_name="completed"/>
1828                    </lifecycle_sequence>
1829                </active_node_lifecycle>
1830            </lifecycle_definitions>
1831            <lifecycle_mappings>
1832                <process_lifecycle_mapping
1833                mapping_name="process_ready"
1834                node_type="lifecycle_state">
1835                    <consumer_sphere>SP1_Sphere1
1836                    </consumer_sphere>
1837                    <consumer_active_node>
1838                    VE_process_ready
1839                    </consumer_active_node>
1840                    <provider>
1841                        <provider_sphere>PP_SP1_1
1842                        </provider_sphere>
1843                        <provider_active_node>
1844                        SP1_process_idle
1845                        </provider_active_node>
1846                        <provider_sphere>PP_SP2
1847                        </provider_sphere>
```

```
                    <provider_active_node>
                    SP1_process_idle
                    </provider_active_node>
               </provider>
          </process_lifecycle_mapping>
          <process_lifecycle_mapping
          mapping_name="process_start"
          node_type="lifecycle_transition">
               <consumer_sphere>SP1_Sphere1
               </consumer_sphere>
               <consumer_active_node>
               VE_process_start_enactment
               </consumer_active_node>
               <provider>
                    <provider_sphere>PP_SP1_1
                    </provider_sphere>
                    <provider_active_node>
                    SP1_process_start
                    </provider_active_node>
                    <provider_sphere>PP_SP2
                    </provider_sphere>
                    <provider_active_node>
                    SP1_process_start
                    </provider_active_node>
               </provider>
          </process_lifecycle_mapping>
          <process_lifecycle_mapping
          mapping_name="process_enacting"
          node_type="lifecycle_state">
               <consumer_sphere>SP1_Sphere1
               </consumer_sphere>
               <consumer_active_node>
               VE_process_enacting
               </consumer_active_node>
               <provider>
                    <provider_sphere>
                    PP_SP1_1
                    </provider_sphere>
                    <provider_active_node>
                    SP1_process_enacting
                    </provider_active_node>
                    <provider_sphere>PP_SP2
                    </provider_sphere>
                    <provider_active_node>
                    SP2_process_enacting
                    </provider_active_node>
               </provider>
          </process_lifecycle_mapping>
          <process_lifecycle_mapping
          mapping_name="process_finish"
          node_type="lifecycle_transition">
               <consumer_sphere>SP1_Sphere1
               </consumer_sphere>
               <consumer_active_node>
```

```
1902                VE_process_finish_enactment
1903                </consumer_active_node>
1904                <provider>
1905                    <provider_sphere>PP_SP1_1
1906                    </provider_sphere>
1907                    <provider_active_node>
1908                    SP1_process_finish
1909                    </provider_active_node>
1910                    <provider_sphere>PP_SP2
1911                    </provider_sphere>
1912                    <provider_active_node>
1913                    SP2_process_finish_enactment
1914                    </provider_active_node>
1915                </provider>
1916            </process_lifecycle_mapping>
1917            <process_lifecycle_mapping
1918            mapping_name="process_ended"
1919            node_type="lifecycle_state">
1920                <consumer_sphere>SP1_Sphere1
1921                </consumer_sphere>
1922                <consumer_active_node>
1923                VE_process_ended
1924                </consumer_active_node>
1925                <provider>
1926                    <provider_sphere>
1927                    PP_SP1_1
1928                    </provider_sphere>
1929                    <provider_active_node>
1930                    SP1_process_ended
1931                    </provider_active_node>
1932                    <provider_sphere>PP_SP2
1933                    </provider_sphere>
1934                    <provider_active_node>
1935                    SP2_process_ended
1936                    </provider_active_node>
1937                </provider>
1938            </process_lifecycle_mapping>
1939            <active_node_lifecycle_mapping
1940            mapping_name="node_ready"
1941            node_type="lifecycle_state">
1942                <consumer_sphere>SP1_Sphere1
1943                </consumer_sphere>
1944                <consumer_active_node>
1945                VE_active_node_ready
1946                </consumer_active_node>
1947                <provider>
1948                    <provider_sphere>
1949                    PP_SP1_1
1950                    </provider_sphere>
1951                    <provider_active_node>
1952                    SP1_active_node_ready
1953                    </provider_active_node>
1954                    <provider_sphere>
1955                    PP_SP2
```

```
                    </provider_sphere>
                    <provider_active_node>
                    SP2_active_node_ready
                    </provider_active_node>
                </provider>
            </active_node_lifecycle_mapping>
            <active_node_lifecycle_mapping
            mapping_name="node_accept"
            node_type="lifecycle_transition">
                <consumer_sphere>SP1_Sphere1
                </consumer_sphere>
                <consumer_active_node>
                VE_active_node_accept
                </consumer_active_node>
                <provider>
                    <provider_sphere>
                    PP_SP1_1
                    </provider_sphere>
                    <provider_active_node>
                    SP1_active_node_accept
                    </provider_active_node>
                    <provider_sphere>PP_SP2
                    </provider_sphere>
                    <provider_active_node>
                    SP2_active_node_accept
                    </provider_active_node>
                </provider>
            </active_node_lifecycle_mapping>
            <active_node_lifecycle_mapping
            mapping_name="node_complete"
            node_type="lifecycle_transition">
                <consumer_sphere>
                SP1_Sphere1
                </consumer_sphere>
                <consumer_active_node>
                VE_active_node_complete
                </consumer_active_node>
                <provider>
                    <provider_sphere>
                    PP_SP1_1
                    </provider_sphere>
                    <provider_active_node>
                    SP1_active_node_complete
                    </provider_active_node>
                    <provider_sphere>PP_SP2
                    </provider_sphere>
                    <provider_active_node>
                    SP2_active_node_complete
                    </provider_active_node>
                </provider>
            </active_node_lifecycle_mapping>
        </lifecycle_mappings>
        <active_node_label_mapping>
            <consumer_process>GB_production
```

```
2010                </consumer_process>
2011                <consumer_active_node>CO
2012                </consumer_active_node>
2013                <provider_process>PP_SP1_1
2014                </provider_process>
2015                <provider_active_node>Local_CO
2016                </provider_active_node>
2017            </active_node_label_mapping>
2018            <active_node_label_mapping>
2019                <consumer_process>GB_production
2020                </consumer_process>
2021                <consumer_active_node>
2022                Check_Gearbox_order
2023                </consumer_active_node>
2024                <provider_process>PP_SP1_1
2025                </provider_process>
2026                <provider_active_node>
2027                Local_Check_Gearbox_order
2028                </provider_active_node>
2029            </active_node_label_mapping>
2030            <active_node_label_mapping>
2031                <consumer_process>GB_production
2032                </consumer_process>
2033                <consumer_active_node>DevG
2034                </consumer_active_node>
2035                <provider_process>PP_SP1_1
2036                </provider_process>
2037                <provider_active_node>Local_DevG
2038                </provider_active_node>
2039            </active_node_label_mapping>
2040            <active_node_label_mapping>
2041                <consumer_process>GB_production
2042                </consumer_process>
2043                <consumer_active_node>ProG
2044                </consumer_active_node>
2045                <provider_process>PP_SP1_1
2046                </provider_process>
2047                <provider_active_node>Local_ProG
2048                </provider_active_node>
2049            </active_node_label_mapping>
2050            <active_node_label_mapping>
2051                <consumer_process>GB_production
2052                </consumer_process>
2053                <consumer_active_node>OrdB
2054                </consumer_active_node>
2055                <provider_process>PP_SP1_1
2056                </provider_process>
2057                <provider_active_node>Local_OrdB
2058                </provider_active_node>
2059            </active_node_label_mapping>
2060            <active_node_label_mapping>
2061                <consumer_process>GB_production
2062                </consumer_process>
2063                <consumer_active_node>
```

```
2064                send_Order_Box_CD
2065                </consumer_active_node>
2066                <provider_process>PP_SP1_1
2067                </provider_process>
2068                <provider_active_node>
2069                Local_send_Order_Box_CD
2070                </provider_active_node>
2071            </active_node_label_mapping>
2072            <active_node_label_mapping>
2073                <consumer_process>GB_production
2074                </consumer_process>
2075                <consumer_active_node>ReceiveBO
2076                </consumer_active_node>
2077                <provider_process>PP_SP2
2078                </provider_process>
2079                <provider_active_node>
2080                Local_ReceiveBO
2081                </provider_active_node>
2082            </active_node_label_mapping>
2083            <active_node_label_mapping>
2084                <consumer_process>GB_production
2085                </consumer_process>
2086                <consumer_active_node>CBO
2087                </consumer_active_node>
2088                <provider_process>PP_SP2
2089                </provider_process>
2090                <provider_active_node>Local_CBO
2091                </provider_active_node>
2092            </active_node_label_mapping>
2093            <active_node_label_mapping>
2094                <consumer_process>GB_production
2095                </consumer_process>
2096                <consumer_active_node>DevB
2097                </consumer_active_node>
2098                <provider_process>PP_SP2
2099                </provider_process>
2100                <provider_active_node>Local_DevB
2101                </provider_active_node>
2102            </active_node_label_mapping>
2103            <active_node_label_mapping>
2104                <consumer_process>GB_production
2105                </consumer_process>
2106                <consumer_active_node>ProB
2107                </consumer_active_node>
2108                <provider_process>PP_SP2
2109                </provider_process>
2110                <provider_active_node>Local_ProB
2111                </provider_active_node>
2112            </active_node_label_mapping>
2113            <active_node_label_mapping>
2114                <consumer_process>GB_production
2115                </consumer_process>
2116                <consumer_active_node>SendB
2117                </consumer_active_node>
```

```
2118                    <provider_process>PP_SP2
2119                    </provider_process>
2120                    <provider_active_node>Local_SendB
2121                    </provider_active_node>
2122                </active_node_label_mapping>
2123                <active_node_label_mapping>
2124                    <consumer_process>GB_production
2125                    </consumer_process>
2126                    <consumer_active_node>Receive_Box
2127                    </consumer_active_node>
2128                    <provider_process>PP_SP1_2
2129                    </provider_process>
2130                    <provider_active_node>
2131                    Local_Receive_Box
2132                    </provider_active_node>
2133                </active_node_label_mapping>
2134                <active_node_label_mapping>
2135                    <consumer_process>GB_production
2136                    </consumer_process>
2137                    <consumer_active_node>AssB
2138                    </consumer_active_node>
2139                    <provider_process>PP_SP1_2
2140                    </provider_process>
2141                    <provider_active_node>Local_AssB
2142                    </provider_active_node>
2143                </active_node_label_mapping>
2144                <monitorability>
2145                    <polling>
2146                        <enactment_takeover>
2147                            <consumer_sphere>
2148                            SP1_Sphere1
2149                            </consumer_sphere>
2150                            <provider_sphere>
2151                            PP_SP1_1
2152                            </provider_sphere>
2153                        </enactment_takeover>
2154                        <termination_takeover>
2155                            <consumer_sphere>
2156                            SP1_Sphere1
2157                            </consumer_sphere>
2158                            <provider_sphere>
2159                            PP_SP1_1
2160                            </provider_sphere>
2161                        </termination_takeover>
2162                        <termination_takeover>
2163                            <consumer_sphere>
2164                            Sphere_SP1_2
2165                            </consumer_sphere>
2166                            <provider_sphere>
2167                            PP_SP1_2
2168                            </provider_sphere>
2169                        </termination_takeover>
2170                    </polling>
2171                    <messaging>
```

```
<enactment_propagation>
    <consumer_sphere>
    Sphere_SP2
    </consumer_sphere>
    <provider_sphere>
    PP_SP2
    </provider_sphere>
</enactment_propagation>
<termination_propagation>
    <consumer_sphere>
    Sphere_SP2
    </consumer_sphere>
    <provider_sphere>PP_SP2
    </provider_sphere>
</termination_propagation>
<enactment_propagation>
    <consumer_sphere>
    Sphere_SP1_2
    </consumer_sphere>
    <provider_sphere>
    PP_SP1_2
    </provider_sphere>
</enactment_propagation>
<transition_messaging>
    <consumer_sphere>SP1_Sphere1
    </consumer_sphere>
    <consumer_active_node>CO
    </consumer_active_node>
    <provider>
        <provider_sphere>PP_SP1_1
        </provider_sphere>
        <provider_active_node>
        Local_CO
        </provider_active_node>
    </provider>
</transition_messaging>
<lifecycle_messaging>
    <lifecycle_state_messaging>
        <consumer_sphere>
        SP1_Sphere1
        </consumer_sphere>
        <consumer_active_node>
        DevG
        </consumer_active_node>
        <consumer_lifecycle_node>
        VE_active_node_accept
        </consumer_lifecycle_node>
        <provider_sphere>PP_SP1_1</provider_sphere>
        <provider_active_node>
        Local_DevG
        </provider_active_node>
        <provider_lifecycle_node>
        SP1_active_node_accept
        </provider_lifecycle_node>
```

```
                        </lifecycle_state_messaging>
                    </lifecycle_messaging>
                </messaging>
                <!--More monitorability can be defined here
                depending on preference.-->
            </monitorability>
        </process_section>
    </service>
    <service>
        <process_section>
            <process tag_name="ProviderProcess_SP1_2"
            process_id="PP_SP1_2">
                <sequence>
                    <receive_task
                    active_node_id="Local_Receive_Box"
                    name="Receive␣Surrounding␣Box">
                        <data>
                            <data_flow_direction>
                            input
                            </data_flow_direction>
                            <data_package_ref>
                            Box
                            </data_package_ref>
                        </data>
                    </receive_task>
                    <task name="Assemble_Gear_Box"
                    active_node_id="Local_AssB"
                    owner="SP1" executor="SP1">
                        <data>
                            <data_flow_direction>
                            output
                            </data_flow_direction>
                            <data_package_ref>
                            GearBox
                            </data_package_ref>
                        </data>
                    </task>
                </sequence>
            </process>
        </process_section>
    </service>
    <service>
        <process_section>
            <process tag_name="ProviderProcess_SP1_1"
            process_id="PP_SP1_1">
                <sequence>
                    <receive_transition
                    active_node_id="Local_CO"
                    name="Receive_Order">
                        <data>
                            <data_flow_direction>
                            input
                            </data_flow_direction>
                            <data_package_ref>
```

```
2280                    cd</data_package_ref>
2281                </data>
2282                <data>
2283                    <data_flow_direction>
2284                    input
2285                    </data_flow_direction>
2286                    <data_package_ref>do
2287                    </data_package_ref>
2288                </data>
2289            </receive_transition>
2290            <task
2291            active_node_id="Local_Check_Gearbox_order"
2292            name="Check_Order"
2293            owner="SP1" executor="SP1">
2294                <data>
2295                    <data_flow_direction>input
2296                    </data_flow_direction>
2297                    <data_package_ref>cd
2298                    </data_package_ref>
2299                </data>
2300                <data>
2301                    <data_flow_direction>
2302                    input</data_flow_direction>
2303                    <data_package_ref>do
2304                    </data_package_ref>
2305                </data>
2306            </task>
2307            <parallel_sync>
2308                <sequence>
2309                    <task name="Develop_Gearing"
2310                    active_node_id="Local_DevG"
2311                    owner="SP1" executor="SP1">
2312                        <event
2313                        tag_name="cdBox_available"
2314                        var_id="Local_cdBox_available"
2315                        changeable="true"
2316                        enabled="enabled">false </event>
2317                        <data>
2318                            <data_flow_direction>
2319                            output
2320                            </data_flow_direction>
2321                            <data_package_ref>cdBox
2322                            </data_package_ref>
2323                        </data>
2324                    </task>
2325                    <task name="Produce_Gearing"
2326                    active_node_id="Local_ProG"
2327                    owner="SP1" executor="SP1"/>
2328                </sequence>
2329                <sequence>
2330                    <wait_any>
2331                        <event_ref>cdBox_available
2332                        </event_ref>
2333                </wait_any>
```

```
                                    <task name="Order_Box"
                                    active_node_id="Local_OrdB"
                                    owner="SP1" executor="SP1">
                                        <event
                                        tag_name="doBox_available"
                                        var_id="Local_doBox_available"
                                        changeable="true"
                                        enabled="enabled">false </event>
                                        <data>
                                            <data_flow_direction>output
                                            </data_flow_direction>
                                            <data_package_ref>doBox
                                            </data_package_ref>
                                        </data>
                                    </task>
                                    <send_task
                                    active_node_id="Local_send_Order_Box_CD"
                                    name="SendOrderCD">
                                        <data>
                                            <data_flow_direction>output
                                            </data_flow_direction>
                                            <data_package_ref>doBox
                                            </data_package_ref>
                                        </data>
                                        <data>
                                            <data_flow_direction>output
                                            </data_flow_direction>
                                            <data_package_ref>cdBox
                                            </data_package_ref>
                                        </data>
                                        <destination_URI>
http://www.ve.com/boxorder.wsdl?port=12212?operation=receive
                                        </destination_URI>
                                    </send_task>
                                </sequence>
                            </parallel_sync>
                        </sequence>
                    </process>
                    <lifecycle_definitions>
                        <process_lifecycle>
                            <lifecycle_sequence>
                                <atomic_state name="SP1_process_idle"
                                tag_name="idle"/>
                                <transition name="SP1_process_start"
                                tag_name="start"/>
                                <atomic_state
                                name="SP1_process_enacting"
                                tag_name="enacting"/>
                                <transition name="SP1_process_finish"
                                tag_name="finish"/>
                                <atomic_state name="SP1_process_ended"
                                tag_name="completed"/>
                            </lifecycle_sequence>
                            </process_lifecycle>
```

```
                    <active_node_lifecycle>
                        <lifecycle_sequence>
                            <atomic_state
                            name="SP1_active_node_ready"
                            tag_name="ready"/>
                            <lifecycle_parallel_sync>
                                <lifecycle_sequence>
                                    <transition
                                    name="SP1_active_node_accept"
                                    tag_name="accept"/>
                                    <atomic_state
                                    name="SP1_active_node_executing"
                                    tag_name="executing"/>
                                </lifecycle_sequence>
                            </lifecycle_parallel_sync>
                            <transition
                            name="SP1_active_node_complete"
                            tag_name="complete"/>
                            <atomic_state
                            name="SP1_active_completed"
                            tag_name="completed"/>
                        </lifecycle_sequence>
                    </active_node_lifecycle>
                </lifecycle_definitions>
            </process_section>
        </service>
        <service>
            <process_section>
                <process tag_name="ProviderProcess_SP2"
                process_id="PP_SP2">
                    <sequence>
                        <receive_task
                        name="Receive_Box_Order"
                        active_node_id="Local_ReceiveBO"
                        owner="SP2" executor="SP2">
                            <data>
                                <data_flow_direction>
                                input
                                </data_flow_direction>
                                <data_package_ref>doBox
                                </data_package_ref>
                            </data>
                            <data>
                                <data_flow_direction>
                                input
                                </data_flow_direction>
                                <data_package_ref>
                                cdBox
                                </data_package_ref>
                            </data>
                        </receive_task>
                        <task
                        name="Check_Box_Order"
                        active_node_id="Local_CBO"
```

```
2442                    owner="SP2" executor="SP2">
2443                        <data>
2444                            <data_flow_direction>
2445                            input
2446                            </data_flow_direction>
2447                            <data_package_ref>
2448                            doBox
2449                            </data_package_ref>
2450                        </data>
2451                    </task>
2452                    <task name="Develop_Box"
2453                    active_node_id="Local_DevB"
2454                    owner="SP2" executor="SP2">
2455                        <data>
2456                            <data_flow_direction>
2457                            input
2458                            </data_flow_direction>
2459                            <data_package_ref>
2460                            cdBox
2461                            </data_package_ref>
2462                        </data>
2463                    </task>
2464                    <task name="Produce_Box"
2465                    active_node_id="Local_ProB"
2466                    owner="SP2" executor="SP2">
2467                        <event
2468                        tag_name="Box_available"
2469                        var_id="Local_Box_available"
2470                        changeable="true"
2471                        enabled="enabled">false </event>
2472                        <data>
2473                            <data_flow_direction>
2474                            output </data_flow_direction>
2475                            <data_package_ref>Box
2476                            </data_package_ref>
2477                        </data>
2478                    </task>
2479                    <send_task name="Send_Box"
2480                    active_node_id="Local_SendB"
2481                    owner="VE" executor="SP2">
2482                        <data>
2483                            <data_flow_direction>output
2484                            </data_flow_direction>
2485                            <data_package_ref>Box
2486                            </data_package_ref>
2487                        </data>
2488                        <destination_URI>
2489    http://www.ve.com/receivePort.wsdl?operation=1111
2490                        </destination_URI>
2491                    </send_task>
2492                </sequence>
2493                <!-- end of seq within SP2 -->
2494            </process>
2495            <lifecycle_definitions>
```

```
                    <process_lifecycle>
                        <lifecycle_sequence>
                            <atomic_state
                            name="SP2_process_ready"
                            tag_name="ready"/>
                            <transition
                            name="SP2_process_start_enactment"
                            tag_name="start_enactment"/>
                            <atomic_state
                            name="SP2_process_enacting"
                            tag_name="enacting"/>
                            <transition
                            name="SP2_process_finish_enactment"
                            tag_name="finish_enactment"/>
                            <atomic_state
                            name="SP2_process_ended"
                            tag_name="ended"/>
                        </lifecycle_sequence>
                    </process_lifecycle>
                    <active_node_lifecycle>
                        <lifecycle_sequence>
                            <atomic_state
                            name="SP2_active_node_ready"
                            tag_name="ready"/>
                            <lifecycle_parallel_sync>
                                <lifecycle_sequence>
                                    <transition
                                    name="SP2_active_node_accept"
                                    tag_name="accept"/>
                                    <atomic_state
                                    name="SP2_active_node_executing"
                                    tag_name="accepted"/>
                                </lifecycle_sequence>
                                <lifecycle_sequence>
                                    <transition
                                    name="SP2_active_node_bypass"
                                    tag_name="bypass"/>
                                    <atomic_state
                                    name="SP2_active_node_bypassed"
                                    tag_name="bypassed"/>
                                </lifecycle_sequence>
                            </lifecycle_parallel_sync>
                            <transition
                            name="SP2_active_node_complete"
                            tag_name="complete"/>
                            <atomic_state
                            name="SP2_active_completed"
                            tag_name="completed"/>
                        </lifecycle_sequence>
                    </active_node_lifecycle>
                </lifecycle_definitions>
            </process_section>
        </service>
    </exchanged_value>
```

```
2550      <exchanged_value>
2551          <financial_reward>
2552              <process_section>
2553                  <process tag_name="Name"
2554                  process_id="moneyflow">
2555                      <sequence>
2556                          <task active_node_id="RM"
2557                          name="Receive␣Money" owner="VE"
2558                          executor="VE">
2559                              <data>
2560                                  <data_flow_direction>
2561                                  input
2562                                  </data_flow_direction>
2563                                  <data_package_ref>
2564                                  BankStatement
2565                                  </data_package_ref>
2566                              </data>
2567                          </task>
2568                          <task active_node_id="DM"
2569                          name="Distribute␣Money"
2570                          owner="VE" executor="VE">
2571                              <data>
2572                                  <data_flow_direction>
2573                                  output
2574                                  </data_flow_direction>
2575                                  <data_package_ref>
2576                                  TransferBill
2577                                  </data_package_ref>
2578                              </data>
2579                          </task>
2580                      </sequence>
2581                  </process>
2582              </process_section>
2583          </financial_reward>
2584      </exchanged_value>
2585      <exchange_provisions/>
2586      <exchange_provisions/>
2587      <!--    end of What  concept -->
2588  </contract>
```

Bibliography

[1] ATHENA: Advanced Technologies for Interoperability of Heterogenous Enterprise Networks and their Application. http://www.athena-ip.org/index.php. pages 45

[2] CrossWork: Cross-Organisational Workflow Formation and Enactment, IST no. 507590. http://www.crosswork.info/. pages 14, 15, 32, 41, 166

[3] *CrossWork Deliverable D2.1: Final Report*. pages 134

[4] *CrossWork Deliverable D2.2 Final Report*. pages 134

[5] IBM MQSeries workflow. http://www-4.ibm.com/software/mqseries/workflow. pages 45

[6] IBM patterns for e-business. http://www-128.ibm.com/developerworks/patterns/. pages 48

[7] On compatibility of web services. *Petri Net Newsletter*, 65:12–20, 2003. pages 100

[8] W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996. pages 18, 22

[9] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997. pages 18, 21, 23

[10] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. pages 18, 21, 107, 109, 126

[11] W.M.P. van der Aalst. Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control? BETA Working Paper Series, WP 46, Eindhoven University of Technology, Eindhoven, 2000. pages 67

[12] W.M.P. van der Aalst. Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control? *Information Technology and Management Journal*, 2(3), 2002. pages vi, 17, 18, 19, 21, 22, 23, 24, 25, 27, 86, 87, 94, 97, 98, 100

[13] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Computing Science Reports 99/06, Eindhoven University of Technology, Eindhoven, 1999. pages 38

[14] W.M.P. van der Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002. pages 18, 26, 88

[15] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-Based Analysis of BPML (and WSCI). *QUT Technical report*, (FIT-TR-2002-05):487–531, 2002. pages 54

[16] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002. pages 3, 4, 17

[17] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns Home Page. http://www.workflowpatterns.com. pages 17, 48, 51, 54, 104, 125, 126, 181, 184

[18] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000. pages 35, 48, 54, 125

[19] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003. pages 35, 48, 54, 166

[20] W.M.P. van der Aalst and A. Kumar. XML Based Schema Definition for Support of Inter-organizational Workflow. *Information Systems Research*, 14(1):23–47, March 2003. pages 125, 129

[21] W.M.P. van der Aalst, H.M.W. Verbeek, and A. Kumar. XRL/Woflan: Verification of an XML/Petri-net based language for inter-organizational workflows (Best paper award). In K. Altinkemer and K. Chari, editors, *Proceedings of the 6th Informs Conference on Information Systems and Technology (CIST-2001)*, pages 30–45. Informs, Linthicum, MD, 2001. pages vi, 125, 126, 127, 129

[22] G. Alonso, SU. Fiedler, C. Hagen, A. Lazcano, H.Schuldt, and N. Weiler. WISE: business to business e-commerce. In *Proceedings of the 9th International Workshop on Research Issues on Data Engineering*, pages 132–139, Sydney, Australia, 1999. pages 45

[23] S. Angelov. *Foundations of B2B Electronic Contracting*. Dissertation, Technology University Eindhoven, Faculty of Technology Management, Information Systems Department, 2006. pages 15, 114, 143, 152, 157, 171, 174

[24] A. Barros and M. Dumas. The rise of web service ecosystems. *IT Professional*, 8:31–37, October 2006. pages 165, 167

[25] A. Barros, M. Dumas, and A.H.M. ter Hofstede. Service interaction patterns. In W.M. P. van der Aalst and F. Curbera B. Benatallah, F. Casati, editors, *Business Process Management: 3rd International Conference, BPM 2005*, number 3649

in Lecture Notes in Computer Science, pages 302–318, Nancy, France, 2005. Springer Verlag, Berlin. pages 48, 55, 104

[26] T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998. pages 27, 38

[27] T. Basten and W.M.P. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001. pages 18, 26, 28, 88

[28] BEA Systems, Intalio, SAP AG , Sun Microsystems. *Web Service Choreography Interface (WSCI) 1.0 Specification*. http://wwws.sun.com/software/xml/developers/wsci/, 2003. pages 126

[29] T. Bellwood, L. Clément, and D. Ehnebuske et al. *UDDI Version 3.0, Published Specification*. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, 2003. pages 6, 157

[30] G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987. pages 87

[31] V. Bjork and C.T. Davis. Data processing spheres of controldata, 1978. pages 158

[32] D. Box, D. Ehnebuske, and G. Kakivaya et al. *Simple Object Access Protocol (SOAP) 1.1*. http://www.w3.org/TR/SOAP/, 2003. pages 5

[33] BPML.org. *Business Process Modeling Language (BPML) version 1.0*. Accessed August 2003 from www.bpmi.org, 2003. pages 48, 126

[34] F. Cabrera, G. Copeland, and et al. B. Cox. *Specification: Web Services Transaction (WS-Transaction)*. citeseer.nj.nec.com/vanderaalst02yawl.html, 2002. pages 6

[35] L.F Cabrera, G. Copeland, M. Feingold, R.W. Freind, and T. Freund. *WS-Coordination 1.0*. ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf, 2005. pages 6

[36] M. Carpenter and A. Gledsonand N. Mehandjiev. Support for dynamic ontologies in open business systems. In *Agent-Oriented Information Systems Workshop AAMAS'04*. pages 13

[37] R. Chinnici, M. Gudgin, J.J. Moreau, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 1.2*. http://www.w3.org/TR/2003/WD-wsdl12-20030611, 2003. pages 6, 54

[38] Workflow Management Coalition. *XML Process Definition Language*. http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf, 2002. pages 126

[39] J.M. Colom and M. Silva. Improving the Linearly Based Characterization of P/T Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–146. Springer-Verlag, Berlin, 1990. pages 87

[40] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. *Business Process Execution Language for Web-Services 1.0*. http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003. pages 6, 48, 54, 101, 126

[41] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In S. Dustdar, J.L. Fiadeiro, and pages = A. Sheth, editors, *Business Process Management: 3rd International Conference, BPM 2006*. pages 166

[42] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993. pages 18, 21

[43] R. Eshuis and P. Grefen. Constructing Customized Process Views. BETA Working Paper Series, WP 197, Eindhoven University of Technology, Eindhoven, 2007. pages 166

[44] R. Eshuis, P. Grefen, and S. Till. Structured service composition. In S. Dustdar, J. Fiadeiro, and A.P. Sheth, editors, *4th International Conference, BPM 2006*, volume 4102 of *Lecture Notes in Computer Science*, pages 97–112, Vienna, Austria, September 2006. LNCS Springer. pages 2, 13, 109

[45] R. Eshuis, P.W.P.J. Grefen, and S. Till. Structured service composition. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *BPM*, volume 4102 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2006. pages 166

[46] R. Soley et al. *Unified Modelling Language*. http://www.uml.org, 2004. pages 49, 105, 115

[47] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison Wesley, Reading, MA, USA, 1995. pages 54

[48] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996. pages 26, 88

[49] P. Grefen. Towards dynamic interorganizational business process management [keynote speech]. In *Proceedings 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. pages v, 2, 3, 5

[50] P. Grefen. Service-Oriented Support for Dynamic Interorganizational Business Process Management. to appear, 2006. pages 15, 32, 33, 153, 154

[51] P. Grefen, H. Ludwig, and S. Angelov. A Three-Level Framework for Process and Data Management of Complex E-Services. *International Journal of Cooperative Information Systems*, 12(4):487–531, 2003. pages 33, 40, 43, 52, 84, 93

[52] P. Grefen, H. Ludwig, A. Dan, and S. Angelov. An Analysis of Web Services Support for Dynamic Business Process Outsourcing. *Information and Software Technology*, 48(11):1115–1134, 2006. pages 5

[53] P. Grefen, N. Mehandjievb, G. Kouvasc, G. Weichhart, and R. Eshuis. Dynamic Business Network Process Management in Instant Virtual Enterprises. BETA Working Paper Series, WP 198, Eindhoven University of Technology, Eindhoven, 2007. pages 15

[54] A.R. Hevner, S. Ram, S.T. March, and J. Park. DESIGN SCIENCE IN INFORMATION SYSTEM RESEARCH. *MIS Quarterly*, 28(1):75–105, 2004. pages v, 6, 7

[55] IBM. *Web Service Flow Language (WSFL) 1.0 Specification*. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2003. pages 126

[56] G. Dessler J. Reinecke and W. Schoell. *Introduction to Business - A Contemporary View*. Allyn and Bacon, 1989. pages 114

[57] J. Saint-Blancat (Editor). *CrossFlow Deliverable D16: Final Report*. pages 2

[58] D. Jordan, J. Evdemon, A. Alves, and A. Arkin. *Business Process Execution Language for Web-Services 2.0*. http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm, 2007. pages 120

[59] D. Jordan, J. Evdemon, A. Alves, and A. Arkin. *Web Services Choreography Description Language 1.0*. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 2007. pages 120

[60] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Queensland University of Technology, Brisbane, Australia, 2002. pages 125

[61] B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, March 2003. pages 125

[62] E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, June 1999. pages 107

[63] E. Kindler, J. Billington, and S. Christensen et al. The petri net markup language: Concepts, technology, and tools. In W.M.P. van der Aalst and E. Best, editors, *Proceedings of the 24th International Conference,ICATPN 2003*, number 2679 in Lecture Notes in Computer Science, pages 483–505, Eindhoven, The Netherlands, 2003. Springer Verlag, Berlin. pages 126, 128

[64] E. Kindler and M. Weber et al. *Petri Net Kernel (PNK) Home Page*. http://www.informatik.hu-berlin.de/top/pnk/, 2003. pages 109

[65] E. Kindler and M. Weber et al. *Petri Net Markup Language (PNML) Home Page*. http://www.informatik.hu-berlin.de/top/pnml/, 2003. pages 54, 126

[66] E. Kindler, A. Martens, and W. Reisig. Inter-operability of workflow applications: Local criteria for global soundness. *Lecture Notes in Computer Science: Business Process Managements - models, techniques and empirical studies*, 1806:235–253, 2000. pages 100

[67] L. Kutvonen, J. Metso, and T. Ruokolainen. Inter-enterprise Collaboration Management in Dynamic Business Networks. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, page 593Ű611, Agia Napa, Cyprus, October 2005. LNCS Springer. pages 152, 153, 154

[68] A. Lazcano, H. Schuldt, G. Alonso, and H Schek. WISE: Process Based E-Commerce. *IEEE Data Engineering Bulletin*, 24(1), 2001. pages 45

[69] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall, N.J., 2000. pages 4, 158

[70] N. Lohmann. *BPEL2oWFN: Translating BPEL Processes into Open Workflow Nets*. http://www.gnu.org/software/bpel2owfn/, 2006. pages 101

[71] L.A. Maciaszek. *Requirements Analysis and System Design. Developing Information Systems with UML*. Addison Wesley, 2001. pages 105

[72] A. Martens. On usability of web services. In Coral Calero, Oscar Díaz, and Mario Piattini, editors, *Proceedings of 1st Web Services Quality Workshop (WQW 2003)*, Rome, Italy 2003. pages 100

[73] A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2003. pages 100

[74] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, and S. McIlraith. *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/, 2004. pages 33

[75] P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. In *2nd South-East European Workshop on Formal Methods 2005 (SEEFM05)*, Ohrid, Republic of Macedonia, 2005. pages 100

[76] P. Massuthe, W. Reisig, and K. Schmidt. An operating guideline approach to the soa. Ohrid, Republic of Macedonia, 2005. pages 101

[77] P. Massuthe and K. Schmidt. Operating Guidelines for Services. In Karsten Schmidt and Christian Stahl, editors, *12. Workshop "Algorithmen und Werkzeuge für Petrinetze" (AWPN 2005), Proceedings*, pages 78–83. Humboldt-Universität zu Berlin, September 2005. pages 100

[78] P. Massuthe and D. Weinberg. *Fiona*. http://www.informatik.hu-berlin.de/top/tools4bpel/fiona, 2006. pages 101

[79] C. Mohan. Dynamic e-business: Trends in web services. In Alejandro P. Buchmann, Fabio Casati, Ludger Fiege, Mei-Chun Hsu, and Ming-Chien Shan, editors, *TES*, volume 2444 of *Lecture Notes in Computer Science*, pages 1–5, Hong Kong, China, August 2002. LNCS Springer. pages 5

[80] Frederic Montagut and Refik Molva. Augmenting web services composition with transactional requirements. *icws*, 0:91–98, 2006. pages 158

[81] M.Reichert and P.Dadam. ADEPTflex - Supporting dynamic changes of workflow without loosing control. *Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management*, 10(2):93–129, 1998. pages 13

[82] A. Norta. XRL Home Page. http://is.tm.tue.nl/research/xrl/. pages 115, 125, 129

[83] A. Norta. XRL/flower Home Page. http://is.tm.tue.nl/research/xrl/flower. pages 109, 128

[84] A. Norta and P. Grefen. Discovering Patterns for Inter-Organizational Business Collaboration in a Top-Down Way. BETA Working Paper Series, WP 163, Eindhoven University of Technology, Eindhoven, 2006. pages 51, 142

[85] Alex Norta. *eSourcing: electronic Sourcing for business to business*. http://is.tm.tue.nl/research/eSourcing, 2006. pages 15, 51

[86] OASIS. *eXtensible Markup Language (SOAP) 1.1*. http://www.xml.org/, 2006. pages 5

[87] M.P. Papazoglou. Web Services and Business Transactions. *World Wide Web: Internet and Web Information Systems*, 6:49–91, 2003. pages 158

[88] M.P. Papazoglou and B. Kratz. A business-aware web service transaction model. In A. Dan and W. Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006, 4th International Conference*, volume 4294 of *Lecture Notes in Computer Science*, pages 352–364, Chicago, USA, December 2006. LNCS Springer. pages 159

[89] M.P. Papazoglou and P.M.A Ribbers. *e-Business: organizational and technical foundations*. John Wiley & Sons, Ltd., 2006. pages 3

[90] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998. pages 7, 17, 18

[91] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998. pages 7, 17, 18

[92] W. Reisig, K. Schmidt, and C. Stahl. Kommunizierende Workflow-Services modellieren und analysieren. *Informatik - Forschung und Entwicklung*, pages 90–101, October 2005. pages 100

[93] IBM Research. Crossflow architecture description, Technical report, ESPRIT Crossflow EP 28653, 1999. pages 45

[94] G. Rozenberg and J. Engelfriet. Elementary net systems. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:12–121, 1998. pages 20

[95] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. *QUT Technical report*, (FIT-TR-2004-01), 2004. pages 13, 40, 48, 54, 104, 145, 150

[96] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Resource Patterns. *BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven*, 2004. pages 13, 40, 48, 54, 104

[97] W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, Berlin, 1999. pages 14

[98] K. Schmidt. Lola: A low level analyser. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474, Aarhus, Denmark, June 2000. LNCS Springer. pages 101

[99] K. Schmidt. *LoLA: A Low Level Analyser*. http://www.informatik.hu-berlin.de/top/lola/doku.ps, 2004. pages 101

[100] K. Schmidt. Controllability of Open Workflow Nets. In Jörg Desel and Ulrich Frank, editors, *Enterprise Modelling and Information Systems Architectures*, volume P-75 of *Lecture Notes in Informatics (LNI)*, pages 236–249, Bonn, 2005. Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH. pages 100

[101] K.A. Schulz and M.E. Orlowska. Facilitating cross-organizational workflows with a workflow view approach. *Data & Knowledge Engineering*, 51:109–147, 2004. pages 45

[102] Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 1999. pages 54

[103] S. Thatte. *XLANG: Web Service for Business Process Design*, 2003. pages 126

[104] J.D. Tygar. Atomicity in electronic commerce. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 8–26, New York, 1996. pages 158

[105] W.J. van den Heuvel and S. Artyshchev. Developing a three-dimensional transaction model for supporting atomicity spheres. Proceedings of NetObjectDays 2002 vol. 2, 2002. pages 158

[106] H.M.W. Verbeek and W.M.P. van der Aalst. Woflan Home Page, Eindhoven University of Technology, Eindhoven, The Netherlands. http://www.tm.tue.nl/it/woflan. pages 90, 96, 99

[107] H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000. pages 99

[108] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001. pages 14, 28

[109] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes Using Woflan. *The Computer Journal, British Computer Society*, 44(4):246–279, 2001. pages 99, 109, 126

[110] H.M.W Verbeek, A. Hirnschall, and W.M.P. van der Aalst. XRL/Flower: Supporting inter-organizational workflows using XML/Petri-net technology. In C. Bussler, R. Hull, S. McIlraith, M.E. Orlowska, B. Pernici, and J. Yang, editors, *Web Services, E-Business, and the Semantic Web*, CAiSE 2002 International Workshop, WES 2002, Toronto, Canada, pages 93–109. LNCS Springer, May 2002. pages 126, 130

[111] J. Vonk and P.W.P.J. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2):137–172, 2003. pages 13

[112] T. Wang, P. Grefen, and J. Vonk. Abstract transaction construct: Building a transaction framework for contract-driven, service-oriented business processes. In A. Dan and W. Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006, 4th International Conference*, volume 4294 of *Lecture Notes in Computer Science*, pages 352–364, Chicago, USA, December 2006. LNCS Springer. pages 159

[113] M. Weber and E. Kindler. The petri net markup language. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Petri Net Technology for Communication-Based Systems Advances in Petri Nets*, number 2472 in Lecture Notes in Computer Science, page 455 p. Springer Verlag, Berlin, 2003. pages 54, 126

[114] D. Weinberg. Reduction Rules for Interaction Graphs. Informatik-Berichte 198, Humboldt-Universität zu Berlin, February 2006. pages 101

[115] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996. pages v, 4

[116] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, number 2813 in Lecture Notes in Computer Science, pages 200–215, Chicago, Illinois, 2003. Springer Verlag, Berlin. pages 54

[117] J.M. Zaha, A. Barros, M. Dumas, and A. H.M. ter Hofstede. Let's dance: A language for service behavior modeling. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, and ODBASE*,

volume 4276 of *Lecture Notes in Computer Science*, Montpellier, France, October 2006. LNCS Springer. pages 167

[118] L. Zhao, N. Mehandjiev, and L. Macaulay. Agent roles and patterns for supporting dynamic behavior of web service applications. In *AAMAS'04 Workshop on Web Services and Agent-Based Engineering (WSABE)*. pages 13

Index

Curriculum Vitae

Alex Norta (née Hirnschall) was born in 1972 in the city of Pretoria, South Africa. He graduated from grammar school in Linz Austria and after completing military service, he commenced to study business informatics at the Johannes Kepler University of Linz. In 1992/93 he was a student at the School of Management of the University of Manchester Institute of Science and Technology.

From 1998 till 2000 he performed his master-thesis project as part of a funded research project about active object-oriented database systems and was at the same time a student assistant and tutor at the department of Data- and Knowledge Engineering of the Johannes Kepler University. In 2000/01 he moved on to the department of Information Systems as a scientific programmer in the Co-flow project to implement a Smalltalk-based workflow management system, in cooperation with PSE Siemens Vienna. In 2001 he obtained a M.Sc. in business informatics.

From 2001 until 2006 he was a Ph.D. student at the Information Systems department of the TU-Eindhoven, The Netherlands. Embedded in the EU research project CrossWork, he focused on developing a concept for dynamic inter-organizational business process collaboration. His research work led to a number of international publications. During his Ph.D. studies, he was actively involved in teaching activities. In 2007 he joined the Computer Science department of the Helsinki University in Finland, to work for the SOAMeS project on an investigation of service-oriented architectures for enterprize computing and inter-enterpriser computing.

Since 2003 he is happily married to Kaisa Lotta Norta and a proud father of Elias Onni Alexander and Saga Lotta Amelie, with whom he lives in Espoo, Finland.