Scalable Ontological EAI and e-Business Integration

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

von der Fakultät für
Wirtschaftswissenschaften
der Universität Karlsruhe (TH)

vorgelegte

DISSERTATION

von

Dipl.-Inform. Jens Lemcke

Tag der mündlichen Prüfung: 17. Juli 2009

Referent: Prof. Dr. Rudi Studer
Korreferent: Prof. Dr. Tiziana Margaria-Steffen

2009 Karlsruhe

Abstract

The integration of enterprise applications (EAI) and e-business integration are time-consuming and expensive procedures. On the one hand, high integration costs result from non-machine-interpretable information about supported behavior—i. e., communication protocols. Although that information exists at the development time of single enterprise applications, it is not accessible for the integration team. On the other hand, enterprise applications typically utilize different data exchange formats which complicates the identification and integration of identical business relevant object classes (for example, purchase order).

This thesis discusses an ontological approach to simplify the integration while turning its attention to the scalability of the involved algorithms. The ontological approach abstracts conceptually identical business relevant matters regardless of their different technical representation and facilitates capturing the behavioral knowledge. The first contribution of the thesis uniquely applies a pattern mining algorithm to determine identical object classes from hierarchical data exchange representations, such as the currently predominant XML schema definitions (XSD). The second contribution is a novel model-driven approach to formally capture and transform behavioral knowledge from application development time to integration time. The third contribution utilizes the identified object classes and the behavioral knowledge to enable the declaration of integration goals and automatically derive a correct integration more efficiently than existing approaches.

The pattern mining algorithm presented in this thesis was incorporated into the CCTS Modeler Warp 10 developed at SAP. That tool supports the incremental normalization of a repository of data exchange formats. In addition, it could be shown how SAP NetWeaver CE (composition environment)—so far a manual tool for modeling integration—can benefit in the future from the algorithm for deriving a correct integration from declared integration goals.

Contents

Abbreviations

Common abbreviations

AI	Artificial intelligence, p. 36
API	Application programming interface, p. 22
ASG	Adaptive services grid, p. 47
ASM	Abstract state machine, p. 48
BAPI	Business application programming interface, p. 22
BDI	Beliefs, desires, and intentions, p. 48
BM	Behavioral model , p. 89
BP	Business process, p. 12
BPEL	Web services business process execution language, p. 3
BPI	Business process integration, p. 24
BPM	Business process management, p. 24
BPMN	Business process model and notation, p. 3
BPMS	Business process management system, p. 25
CBP	Collaborative business process, p. 25
CCTS	Core components technical specification (ISO 15000-5), p. 149
CE	Composition environment, p. 185
CFIM	Closed frequent itemset mining, p. 42
CICS	Customer information control system, p. 22

CIDX	Chemical industry data exchange, p. 76
CORBA	Common object request broker architecture, p. 22
CRM	Customer relationship management, p. 49
CTL	Computation tree logic, p. 50
DAML-S	DARPA agent markup language for services, p. 45
DARPA	Defense advanced research projects agency, p. 45
DCOM	Distributed component object model, p. 22
DIP	Data, information and process integration with semantic Web services, p. 48
DL	Description logics, p. 45
DTD	Document type declaration, p. 20
D-U-N-S	Data universal numbering system, p. 30
EaGLe	EaGLe goal language, p. 50
EAI	Enterprise application integration, p. 13
e-Business	Electronic business, p. 12
e-Business XML	see ebXML, p. 19
ebXML	Electronic business using extensible markup language, p. 19
EIN	Employer identification number, p. 30
e-Procurement	Electronic procurement, p. 17
ERP	Enterprise resource planning, p. 17
e-Sales	Electronic sales, p. 17
ESR	Enterprise services repository, p. 141
FedEx	FedEx Corporation is a logistics services company, p. 28
F-logic	Frame logic, p. 45
GEM	Goal-driven enterprise management, p. 49
GUI	Graphical user interface, p. 144

HR	Human resources, p. 49
HTTP	Hypertext transfer protocol, p. 22
IBM	International Business Machines is a multinational computer, technology and IT consulting corporation, p. 12
ILOG	is an IBM company that creates enterprise software products in four broad areas: supply chain, business rule management, visualization, and optimization, p. 48
IOPE	Input, output, precondition, and effect, p. 45
IT	Information technology, p. 13
LCM	Linear time closed itemset miner, p. 67
Mbyte	Megabyte, p. 145
MDSD	Model-driven software development, p. 45
METEOR-S	Applying semantics in annotation, quality of service, discovery, composition, execution; follow-up of the managing end-to-end operations project (METEOR), p. 45
OWL	Web ontology language, p. 45
OWL-S	Web ontology language for Web services, p. 45
PLM	Product lifecycle management, p. 17
SAP	SAP AG is a multinational software development and consulting corporation, p. 141
SA-WSDL	Semantic annotations for WSDL, p. 23
SCM	Supply chain management, p. 17
SOA	Service-oriented architecture, p. 141
SOAP	Simple object access protocol, p. 22
STS	State transition system, p. 87
SWSC	Semantic Web services challenge, p. 49
Tuxedo	Transactions for Unix, extended for distributed operations, p. 22
UDDI	Universal description discovery & integration, p. 43

UML	Unified modeling language, p. 46
UML AD	UML activity diagram, p. 48
UN/EDIFACT	United Nations/electronic data interchange for administration, commerce, and transport, p. 19
UPS	United Parcel Service, Inc. is the world's largest package delivery company, p. 28
URI	Uniform resource identifier, p. 24
W3C	World Wide Web consortium, p. 20
WS	Web service, p. 22
WSDL	Web service definition language, p. 22
WSML	Web service modeling language, p. 45
WSML-DL	DL dialect of WSML, p. 45
WSML-Flight	F-logic dialect of WSML, p. 45
WSML-Rule	Rules dialect of WSML, p. 45
WSMO	Web service modeling ontology, p. 45
WSMX	Web service model execution environment, p. 45
XML	Extensible markup language, p. 20

Abbreviations introduced in this dissertation

APR	Approval, p. 157
CDT	Component design time, p. 14
CUST	Consumer view of the customer's provided behavioral model , p. 157
CUST^{-1}	Provider view of the customer's provided behavioral model , p. 158
DEL	Delivery, p. 159
DONE	Positive notification, p. 157
FAIL	Failure notification, p. 157
FIN	Invoicing, p. 160

HQ *or* H	Head quarter, p. 33
ID	Identification, p. 65
IDT	Integration design time, p. 14
INV	Invoice, p. 157
NO	ASM module NEXTNONDETOPTIONS, p. 129
NS *or* N	New school, p. 33
OCR	Order creation, p. 157
OS *or* O	Old school, p. 33
PO	Purchase order, p. 157
POM	Purchase order management, p. 157
PROD	Production, p. 160
REJ	Rejection, p. 157
RG	ASM module REACHGOAL, p. 129
RT	Run time, p. 15
RV	ASM module REACHVARIANT, p. 129
SELR	Consumer view of the seller's provided behavioral model , p. 161
SELR^{-1}	Provider view of the seller's provided behavioral model , p. 161
SO	Sales order, p. 159
SOM	Sales order management, p. 159
U	User, p. 125

Common mathematical symbols

	All, true, p. 51
	Nothing, false, p. 51
$=$	Equal, p. 78
$=$	Unequal, p. 78

$\sim$	Similar, p. 78
	Logical not, negation, p. 51
	Logical and, conjunction, p. 51
	Logical or, disjunction, p. 51

Chapter 1

Introduction

Communication amongst business partners is an inherent characteristics of performing business. Each business partner can be classified by roles as either seller, buyer, or market maker. A seller provides goods or services to be consumed by one or a multitude of buyers. A market maker creates a platform where buyers and sellers can meet to trade. The seller-buyer relationship holds between a producing company and its end customers as well as between a supplying and a compiling company. Often, a single individual—company or person—can exercise multiple roles at once.

With the increased use of computers, the proper execution of formerly entirely manually executed business procedures has become mechanized and standardized. In fact, many single steps of a business procedure may still be manual, such as the picking of goods from a warehouse and their packing to prepare for their transport, but the coordination of the tasks became automated. Mechanical execution of business procedures frees the human resources that were formerly needed for these pure organizational tasks that can now concentrate on the core business that creates revenue for a company.

The business procedures of a whole company usually span multiple departments. In large companies, the single departments use separate computer systems to manage the local fragments of the company-wide business procedures. Due to that reason, the integration of different computer systems in a company is necessary. That task is also called enterprise application integration (EAI).

Through the rise of the Internet, a new platform for business collaboration became available. The Internet provides a very efficient way to exchange information around the world at very low costs and high speeds compared to traditional services as, for example, postal mail. In particular, the Internet allows not only for the communication between humans, but especially between machines of different companies with humans or other machines. That is usually referred to as e-business.

The Internet thus delivers a huge potential for competitive benefits when used as a platform for the collaboration of companies. But also, with such a potential in the market, the Internet increases the pressure on each single company in the market to quickly and flexibly adapt their collaboration with business partners to gain competitive benefit. That results in a need for solutions that support business flexibility and especially scale

up to the size and complexity of real-life messages communicated and procedures performed (Papazoglou and Ribbers, 2006).

1.1 Problems

While communication is essential to perform business, it is at the same time difficult to be reached by computers. That is illustrated by the fact that nowadays about 40% of a company's IT budget are spent on integration (Kastner and Saia, 2006). The main reasons for the difficulties are listed below.

1. First, the same real-world artifacts often have different computer implementations if the software was produced independently. That is referred to as structural difference (see Rahm and Bernstein, 2001, Shvaiko and Euzenat, 2005).

2. Second, parties willing to integrate their computer systems have differing understandings of the same concepts of a domain. That is referred to as ontological difference (see Kalfoglou and Schorlemmer, 2005, Noy, 2004, Staab and Studer, 2004).

3. Third, knowledge about the IT systems that implement a business procedure fragment is often stored in natural language, if at all, and cannot be accessed at the time of designing a collaborative business procedure from the fragments. That is counterproductive as the behaviors a participant can exercise to fulfill the procedure fragment corresponding to its role in the collaboration is a direct excerpt from the behavioral capabilities of its IT system components—referred to as component model. A collaborative business process roughly consists to 80% of the participants' component models (see Küster et al., 2007).

4. Fourth, collaborative business procedures are complex and involve nondeterministic responses from fragment implementations. The manual creation of a transactionally correct coordination is expensive and error-prone (see Berardi et al., 2005, Pistore et al., 2005c).

1.2 Research questions

The questions that we deal with in this dissertation seek to improve the situation described in the previous paragraphs.

1. Can ontologically redundant, but structurally different artifacts involved in business collaboration be efficiently detected?

2. What is needed to store knowledge of the generally intended and supported behavior of a software component such that it can later be used for creating a partner's role model and a collaborative business process?

3. Can the design of a complex collaborative business procedure with real-world features be efficiently supported by an automated approach?

Different approaches have been proposed in the past that partly address our research questions.

1.2.1 Ontological redundancy

Current approaches for detecting ontological differences concentrate on a general problem such as alignment of arbitrary ontologies and produce rather complex algorithms that tend to be less efficient (see Kalfoglou and Schorlemmer, 2005, Noy, 2004). Existing approaches specially targeted at the domain of EAI and e-business, such as Dong et al. (2004), Wang and Stroulia (2003), perform clustering as opposed to pattern mining. Clustering is helpful to identify groups of objects or elements were not all elements necessarily share the same properties. The opposite is the case with pattern mining, where all elements of a group share the exact same properties. Clustering is rather useful for retrieval of matches upon a certain request. Pattern mining is rather useful for identifying common overlap.

As for detecting ontological differences, we restrict the problem to the area of e-Business and EAI using the currently predominant languages and methods that arose in the context of the Web services (WS) architecture (see Krafzig et al., 2004). Thus, we are able to use a tailored algorithm with linear complexity implementing a special type of pattern mining—the closed frequent itemset mining (CFIM). That technique is well known and studied. However, its application on the problem of detecting redundant artifacts involved in business collaboration in order to reach common understanding is unique.

1.2.2 Tracking procedural information

For tracking procedural information through the software lifecycle, two principal streams can be identified.

1. **Service and collaboration description approaches.** In the recent years, approaches were introduced that allow for the annotation of services, partially with a model-theoretic semantics. For collaboration languages, such as BPEL[1] and BPMN,[2] however, no model-theoretic semantic exists. That makes their interpretation ambiguous. Applications have focused on providing and directly consuming services. No work considered supporting the whole software development lifecycle with service and collaboration description approaches.

[1] `http://www.oasis-open.org/committees/wsbpel/`
[2] `http://www.bpmn.org/`

2. **Software development support tools.** Approaches like model-driven software development (MDSD) focus on the development lifecycle. However, the approaches in the domain of MDSD have not been applied to business process modeling for EAI and e-business.

We utilize an established, light-weight language for annotating procedural information to software components involved in EAI and e-business scenarios. During definition, we consider that the language used for component description should be deployable to existing execution infrastructures.

In addition, we provide an execution semantics for procedural information and for generated collaborative business procedures. We use the coherent semantics of component behavior, company's behavior, and collaboration behavior to directly execute an e-business coordination on the component interfaces, and bypass, yet not break, the company interfaces.

1.2.3 Designing complex collaborative business procedures

In the area of procedural interoperability there is much work done. However, most of the work neglects realistic features of single service providers and requirements to the orchestration—such as transactionality.

Two research groups (Berardi et al., 2005, Pistore et al., 2005c) have presented interesting results. However, both approaches utilize general-purpose tools and start from a problem definition that is unnecessarily complex for the setting of EAI and e-business. In consequence, the approach of Berardi et al. (2005) is of exponential complexity and the run time of the approach of Pistore et al. (2005c) ranges in the seconds and minutes for even smaller scenarios.

We argue again that a *simpli ed problem statement* can sufficiently deliver on the needs for EAI and e-business. That allows for the creation of a more efficient algorithm for complex-goal-based WS composition. Moreover, our algorithm to solve the procedural integration is *specially tailored*, as opposed to being general-purpose, which additionally reduces the run time of our approach compared to the existing approaches.

1.3 Summary of contributions

To sum up, we contribute to three topics in this thesis.

1. **Detect redundant interface objects.** Our CFIM of hierarchical types uniquely solves the task to detect redundant interface objects with linear space and time consumption.

2. **Track behavioral information through software design phases.** By our approach to transform behavioral models for EAI and e-business, we are able to track behavioral information through software design phases.

3. **Mechanically support exception handling.** Our complex-goal-based WS composition solves a limited problem, yet sufficient in the context of EAI and e-business, by a targeted algorithm in shorter time than the existing approaches.

1.4 Previous publications

The results of this thesis have been published previously.

In the following book chapter, we introduce our view on the interplay between data and process semantics with respect to the integration of complex processes. Although that work provides the basis for the closed frequent itemset mining (CFIM) and the complex-goal-based WS composition presented in this dissertation, we used schema matching techniques instead of CFIM and process integration did not yet consider failure of the integrated systems.

> *Book chapter:*
> Drumm, C., J. Lemcke, and D. Oberle. Business process management and semantic technologies. In J. Cardoso, M. Hepp, and M. D. Lytras, editors, *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond Computing for Human Experience*, pages 209–239. Springer, 2007a. ISBN 978-0-387-48531-7. URL `http://dx.doi.org/10.1007/978-0-387-48531-7_10`.

The journal article below advances the process integration presented in the book chapter above to the complex-goal-based WS composition of this thesis.

> *Journal article:*
> Lemcke, J. and A. Friesen. Considering realistic Web service features for semi-automatic composition. *Annals of Mathematics, Computing and Teleinformatics (AMCT)*, 1(5):26–35, 2007b.

As a second strand of research, we introduce our first formalization of a process mediator in the following article. The mediator we define is able to receive an initial request, transform it to a sequence of parallel sub-requests, forward the sub-requests to providers, collect and combine the results from the providers, and return the combined results to the initial requester. As the definition is formal, we are able to formally prove whether a requested interface can be met by a set of provided interfaces. The specification of the mediator in that article builds the basis for the definitions of behavioral models and the semantics of integration in this thesis.

> *Journal article:*
> Altenhofen, M., A. Friesen, J. Lemcke, and E. Börger. A high-level specification for Virtual Providers. *International Journal of Business Process Integration and Management (IJBPIM)*, 1(4):267–278, 2006b.

The following article extends the definition of mediators from the previous article to the formalization of component behavior as it is used in this thesis. With the extended definition, we were able to apply the complex-goal-based WS composition from Lemcke and Friesen (2007b) above, which is part of this thesis, to generate the mediators which were considered to be manually constructed in the previous paper.

> *Journal article:*
> Altenhofen, M., A. Friesen, and J. Lemcke. ASMs in service oriented architectures. *Journal of Universal Computer Science (JUCS)*, 14(12):2034–2058, 2008.

Finally, the following book chapter presents how closed frequent itemset mining (CFIM) and the model-driven approach to transform behavioral models—presented in this thesis—allow applying the previously presented complex-goal-based WS composition to reduce issues of software engineering. The topics of detecting ontological redundancy and of tracking procedural information of this thesis are covered in that chapter.

> *Book chapter:*
> Lemcke, J. Light-weight semantic integration of generic behavioral component descriptions. In G. Mentzas, T. Bouras, P. Gouvas, and A. Friesen, editors, *Semantic Enterprise Application Integration for Business Processes*. IGI Global, Harrisburg, PA, 2009.

Besides the named publications, we contributed with the research for this thesis to further workshops and symposia. Our full list of publications is given in Appendix B on page 207.

1.5 Thesis structure

The rest of this thesis is structured into four parts.

I **Foundations.** The first part contains an introduction to the areas of EAI and e-business integration in Chapter 2. Chapter 3 describes problems is these areas and derives requirements on a solution. Chapter 4 reviews the state of the art in addressing the identified problems.

II **Scalable Ontological EAI and e-Business Integration.** The second part details our solution to the problems identified in Part I. Chapter 5 starts with an overview how the parts of our solution play together. Chapter 6 explains how our CFIM of hierarchical types tackles the problem of identifying redundant artifacts. Chapter 7 explains how to transform behavioral models for EAI and e-business for tracking procedural information throughout the software lifecycle. Chapter 8 details our complex-goal-based WS composition, which semi-automatically generates transactionally correct business procedures.

III **Application and Evaluation.** The third part describes the application of the concepts developed in Part II to artifacts and procedures of the software vendor SAP. Chapter 9 applies our CFIM of hierarchical types on SAP's public interfaces. We demonstrate in Chapter 10 how our approach to transform behavioral models for EAI and e-business can be used to consistently create a business procedure from existing software components. In Chapter 11, we evaluate our complex-goal-based WS composition against existing work.

IV **Finale.** The final part contains our conclusions and outlook in Chapter 12.

Part I

Foundations

Chapter 2

EAI and e-Business Integration

This section introduces the fields of enterprise application integration (EAI) and e-business. The terms discussed in this chapter are collectively displayed in Figure 2.1.

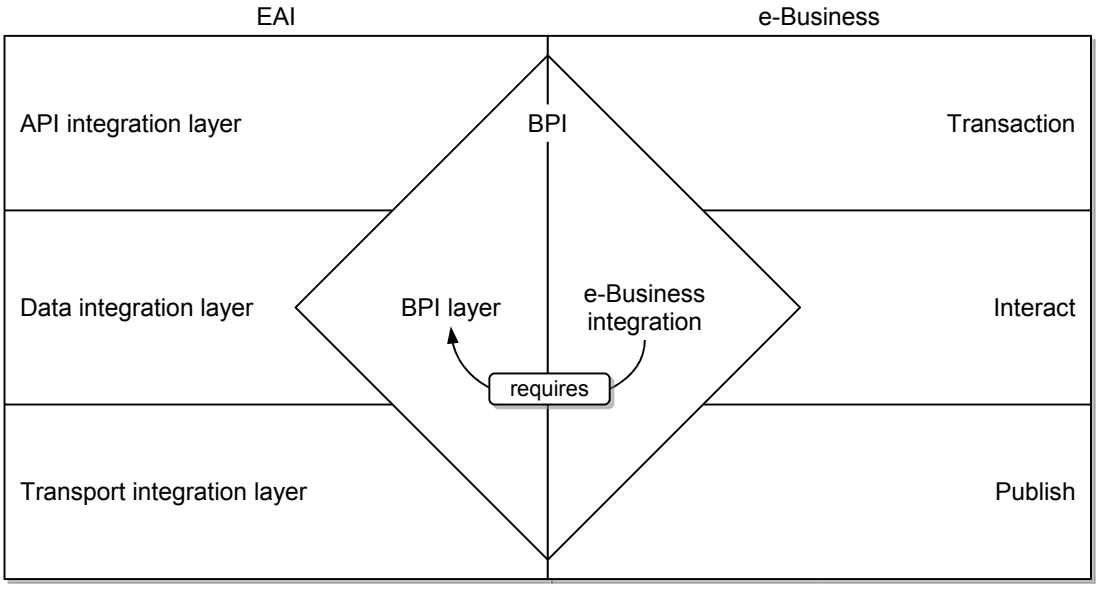


Figure 2.1: Relation of terms in EAI and e-business (suggestive).

This chapter is structured as follows. Since the type of e-business we are considering bases on EAI, we first introduce e-business in Section 2.1.1 before we explain the layers of EAI in Section 2.2. Each of the subsequent four sections focuses on one layer of EAI. While explaining a layer, each section also introduces the technology on that layer which is relevant for this dissertation. We begin with the lowest technical layer—the transport integration layer—in Section 2.3. Section 2.4 introduces the data integration layer. Section 2.5 describes the application programming interface (API) integration layer and Section 2.6 finalizes with the business process integration (BPI) layer.

The material contained in this chapter is based on Papazoglou and Ribbers (2006), Ruh et al. (2001), Serain (2002).

2.1 e-Business integration

In order to explain the term "e-business integration," we first present the definitions of e-business and business process integration (BPI) as found in the literature.

2.1.1 e-Business

The term e-business, or electronic business, was introduced in 1997 by IBM as part of a marketing campaign (Papazoglou and Ribbers, 2006). By definition, e-business means solving business problems or implementing functional tasks using Internet technologies.

According to Serain (2002), the Gartner Group classifies an e-business solution with respect to the business functions that can possibly be offered over the Internet with growing maturity into four categories (right side of Figure 2.1 on the preceding page).

1. **Information publication on the Web, or short "publish."** An e-business solution of that category allows a user—human or machine—to browse information provided by a company. The computer system providing the company's information on the Web is called a Web server.

2. **A client s interaction with an enterprise server, or short "interact."** Interacting with an enterprise server includes that users can engage with forums or place ticket reservations that do not involve payment, and thus do not constitute a business transaction.

3. **The ability to perform transactions, or short "transaction."** In that category, a user can engage with parts of an application, such as to select multiple goods, to order, and to pay for them via a Web site. The computer system backing the Web site is thus called application server.

4. **The integration of the Web server with the company s information structure, or short "integrate."** Information structure means in the context of that citation the technical infrastructure that manages a company's information. "Integrating a Web server with the company's information structure" means that a human or mechanical user can operate a set of applications using the same Web site. The definition implies that the different enterprise applications a user can interact with are integrated, which is also called enterprise application integration (EAI, left side of Figure 2.1).

Performing e-business facilitated by enterprise application integration is the maturity level of business functions we are concerned with in this dissertation.

2.1.2 Business process integration

A business process (BP) is an activity in a company that uses resources and can involve the activities of different departments. According to Papazoglou and Ribbers (2006),

> "business process integration (BPI) can be described as the ability to define a commonly acceptable process model that specifies the sequence, hierarchy, events, execution logic and information movement between systems residing in the same enterprise (viz. EAI) and systems residing in multiple enterprises (viz. e-Business integration)." (p. 519)

We conclude that business process integration (in the middle of Figure 2.1 on page 11) is part of the areas of EAI and e-business. In fact, as the following sections will reveal, business process integration between multiple parties (e-business) requires a business process view of each participating enterprise to exist. In the case that a participating enterprise employs EAI, the business process view of that company is established on the BPI layer of EAI. Details about EAI are given in the following sections.

e-Business integration can be seen as the intersection of e-business and business process integration. Thus, e-business integration as considered by this dissertation has the following properties.

- **Processes.** e-Business integration is concerned with the integration of business processes.
- **Internet.** e-Business integration is performed via the Internet. Thus, Internet technology is used for communication.
- **EAI.** Each participant of a concrete e-business integration uses EAI to integrate their individual systems as a prerequisite to integrate with other companies.

2.2 EAI: The basis for modern e-business

"Application integration means making independently designed systems work together."—Gartner Group

The demand for integration has grown with the demand for business flexibility. The basic driving force behind enterprise applications integration is the fact that a shorter time from an initial business idea to bringing it to the market directly translates to competitive advantage in the market economy. As detailed by Papazoglou and Ribbers (2006), a new business idea often is tightly coupled with a restructuring of the business itself.

2.2.1 Business software components

In order to support changing business needs and to provide flexibility in the IT infrastructure, business software vendors found it a good idea to separate their software to components—or enterprise applications—in preference to a single monolithic block

of software. In particular, the reasons for modularization from a software customer's perspective are the following.

First, mostly no one single application may serve all the IT needs of a company alone. Therefore, the different functionalities needed are usually collected from different vendors and are then integrated in order to work together.

Second, the initial buying or development cost may be too high for a completely integrated solution. It may be more convenient for a company to buy or produce the components of their IT system landscape one after the other. Thus, the integration of the different components grows over time becoming more and more expensive if no coherent integration mechanism is chosen.

And third, as market economy companies tend to continuously evolve by, for example, extending their business to new market areas, selling parts of their business, cooperating with new business partners to enlarge the offered service portfolio, or buying other businesses, they usually acquire the components they need for their operation as they grow.

Different software components are either bought at different times, thus they may be of different releases, acquired from different software vendors, or even be produced in-house.

2.2.2 Component-based software development

A company's IT system landscape supports business execution by storing business documents and the progress of business processes.

Business components are reusable components that can be assembled together to form a business application performing a business process. Therefore, the actual development of the product handling the business process of a customer of a business software vendor is split into the two main software development phases

- component design time and
- integration design time.

Component design time (CDT) means that part of the development lifecycle that can also be observed in traditional software development. Component design time is concerned with developing the components to be offered themselves. That process surely contains some deviation from classical software development because special attention has to be paid to the proper alignment of the components in order to cover the desired domain of functionality. Normally, a special governance process would accompany a component-based development methodology.

Integration design time (IDT) refers to the application of the components developed at component design time in a specific integration project of a customer. An integration

project is mostly concerned with implementing an existing business process of the customer using the software components of the business software vendor. That is the time when enterprise application integration, and especially business process management, is relevant.

After integration design time, the integrated software is deployed to the computing infrastructure of the customer. The phase when business software actually manages business processes is called run time (RT).

2.2.3 Challenge of modularization

Business activity falls into the two categories: performing the current business model and changing the business model. A business model can be changed by

- rearranging existing business areas,
- outsourcing existing business areas, or
- acquiring new business areas.

Change of the business model of a company requires changing the way the IT system facilitates the business procedures.

With modularization, one gains flexibility realigning the components of one or multiple vendors to implement new business scenarios. The drawback of components is that communication is needed between the components. Communication between multiple components causes three effects:

1. If a new business functionality is to be implemented involving n components, $n(n-1)$ relations have to be considered.
2. If an existing functionality is to be changed, first the relations being affected have to be identified among the $n(n-1)$ relations, and in the worst case $n(n-1)$ relations need consistent adaptation.
3. If a new participant should be integrated with a set of n participants, the costs for adding linearly increase with the original size of the community because that requires the construction of n new relations in the worst case.

Performing integration efficiently is desirable as, according to Kastner and Saia (2006), integration activity currently consumes about 40% of a company's IT budget.

2.2.4 Layers of EAI integration

In the previous section, we spoke about abstract relations between software components. More specifically, a relation between two components touches many aspects of integration on different levels that can be organized in a hierarchy (Figure 2.2 on the following page) including

1. transportation layer,
2. data integration layer,
3. application programming interface (API) integration layer, and
4. business process integration (BPI) layer.

Each of the layers addresses an increasingly complex level of integration and builds on top of the lower layers.

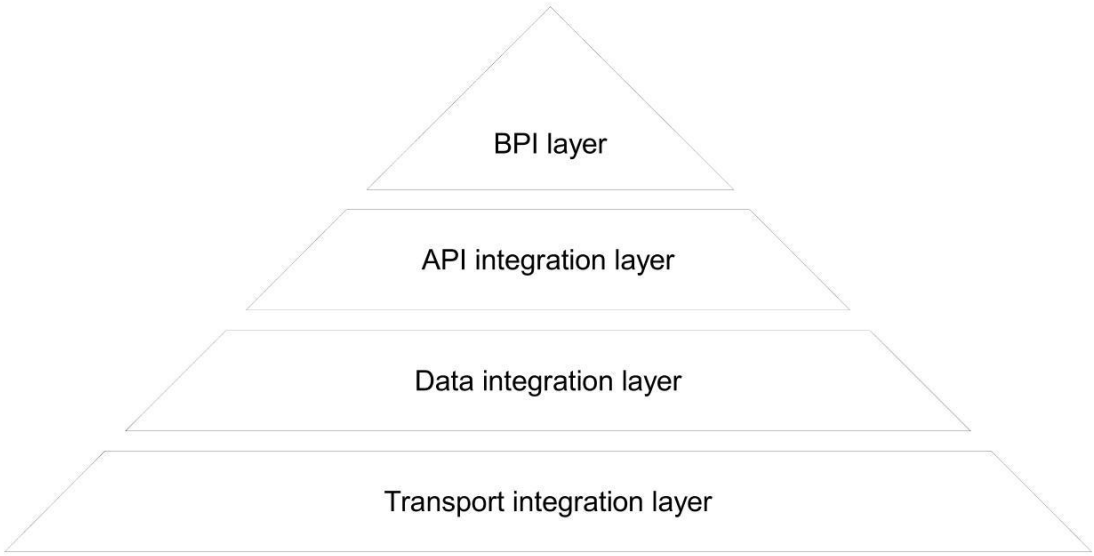


Figure 2.2: The EAI pyramid (from Papazoglou and Ribbers, 2006, p. 512).

2.3 Transport integration layer

The transport integration layer is concerned with the communication channels of the integration participants as displayed in Figure 2.3 on the next page. There exist three basic architectures to organize the transportation integration layer:

1. the peer-to-peer architecture,
2. the hub-and-spoke architecture, and
3. the publish-subscribe architecture.

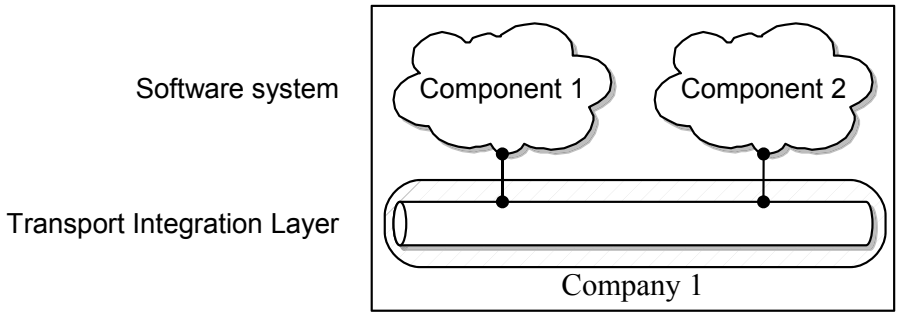


Figure 2.3: Transport integration layer.

2.3.1 The peer-to-peer architecture

The reasons for integration in Section 2.2.1 on page 13 have one commonality. In each of the described cases, it is difficult to plan ahead what the future integration need will be. In particular, it may be difficult to determine which concrete software pieces will be integrated in the future. It might even be hard to determine the vendor of business software components to buy before, as competitors may provide more attractive solutions over time.

The lack of the ability to plan the future integration needs has mostly resulted in a costly IT infrastructure in today's companies. The typical integration architecture emerging due to the above reasons is called peer-to-peer (see left-hand side of Figure 2.4). In a peer-to-peer architecture, every integration participant, be it a piece of internal business software, or the IT system of a collaborating business partner, is directly integrated with the integration participants it has to communicate to. In the worst case, there are $n(n-1)$ directed peer-to-peer connections in a setting with n integration participants that have to be built and maintained.

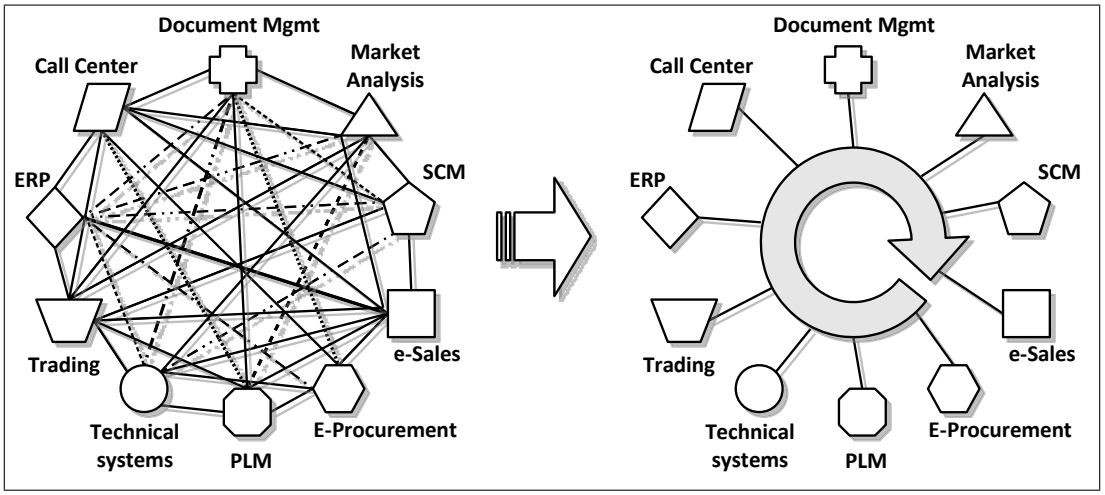


Figure 2.4: Peer-to-peer versus centralized approach.

2.3.2 Improving EAI

The major motivation to improve the situation of $n(n-1)$ individual directed connections of n systems is to lower the high costs of building and maintaining each connection itself. There are two architectures which may serve that purpose.

The hub-and-spoke architecture places a single information broker in the middle of the integration participants. It acts as a single communication partner for each participant and routes incoming requests appropriately to the respective receiver. The major characteristics of the hub-and-spoke architecture is that the integration broker contains business logic to orchestrate the participants, it is responsible for secure and reliable messaging and also for converting data between the different formats of the participants. The advantage of the hub-and-spoke architecture is that it acts as a centralized store of integration information, and that directed connections needed for the integration of n systems are reduced to $2n$. It should be noted that what was said only holds for the number of communication channels. The number of message mappings remains the same. The drawback of that architecture is that the information broker quickly becomes a bottleneck of the integrated system. In addition, that centralized approach seems to pose specific difficulties to globally distributed companies when they have to route all communication through the single broker.

The publish-subscribe, or shared bus, architecture tackles the integration challenge by connecting the integration participants on the technical level, only. The shared bus architecture requires each participant to handle messaging security and reliability as well as data conversion locally. Integration is helped by a potentially distributed shared bus system that the participants connect to on the messaging layer. Information providers publish their advertisement with the shared bus middleware and information consumers subscribe to advertisements. Upon sending data to the message bus, the middleware routes the data to the appropriate subscribers.

To conclude with a rule of thumb, the hub-and-spoke architecture may be rather suitable for the integration of fewer participants. Its maintenance is easier as the integration information resides on a central system. Hub-and-spoke is also more suitable for a setting with a central point of control, such as a powerful car manufacturer dominating its smaller suppliers. On the other hand, the shared bus architecture is more scalable and flexible for larger groups of integration participants. However, the maintenance of the system is rather difficult as integration information remains scattered across the single participants.

2.4 Data integration layer

As different IT systems and, in a large software company even different components of the same system, are developed and evolve to a large extent independently from each other, different applications are likely to rely on incoherent representations of conceptually the same real-life artifact. Different implementations of EAI postulate that the community needs a means to come up with a shared standard to communicate, and new integration participants joining the community must ensure their type definitions match with the standard chosen by the community (see Figure 2.5).

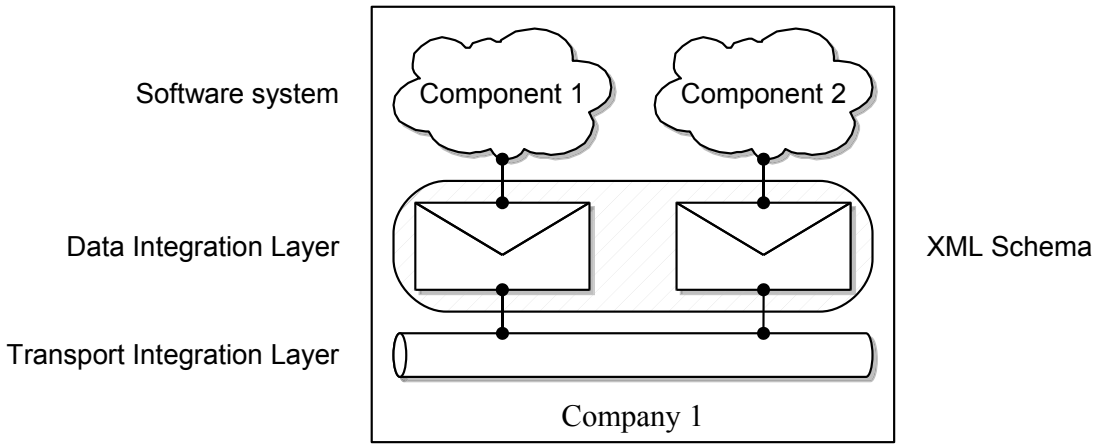


Figure 2.5: Data integration layer.

In principle, data heterogeneity can either be resolved by *changing the software* sending or receiving the data, or by mediating between participants through *changing the transmitted data*. Mediating data can either be done peer-to-peer, which requires $2n(n-1)$ adaptations for n participants. Data mediation can also be done centrally by each participant conforming to an agreed standard. That reduces the number of adaptations to $2n$.

Examples for data formats used for e-business today are "United Nations/Electronic Data Interchange For Administration, Commerce, and Transport" (UN/EDIFACT)[1] and "Electronic Business using eXtensible Markup Language" (e-business XML, or ebXML).[2] Both describe hierarchical data structures. This dissertation uses the eXtensible Markup Language (XML)[3] for representing data and the XML schema definition (XSD)[4] for describing an XML format. As XML is a model for hierarchical organization of data, the results of this dissertation can be transferred to other hierarchical data representations such as UN/EDIFACT or ebXML.

[1] `http://www.unece.org/trade/untdid`
[2] `http://www.ebxml.org/`
[3] `http://www.w3.org/XML/`
[4] `http://www.w3.org/XML/Schema`

2.4.1 XML: The enabling technology for e-business

The extensible markup language (XML) builds the foundation for e-business. XML describes a class of data objects called XML documents. An XML document consists of storage units, called entities, which can contain parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure (Bray et al., 2006).

Figure 2.6 contains an example of an XML document. The first few characters of an XML document must make up an XML declaration, for example, `<?xml version="1.0" encoding="UTF-8"?>`. The XML processing software determines, based on that declaration, how to proceed with the subsequent XML document. The second part of an XML document is a single element known as the document, or root element. A textual unit in an XML document is called element. An element is also a structural component. It can contain either text or further elements, or both. An element is enclosed within a start tag and a corresponding end tag, for example, `<tag>content</tag>`. The set of all elements in an XML document makes up its content. Another way of putting data into an XML document is to assign attributes to tags. An attribute is a key-value pair as in the example `<tag attribute="value">content</tag>`.

```
<?xml version="1.0" ?>
<note>
  <to>Dave</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body lang="en">Did you pay our bill yet?</body>
</note>
```

Figure 2.6: Example XML document.

An XML document has two central properties.

1. An XML document is well-formed, if its structure follows the production rules in Bray et al. (2006). A well-formed XML document is also called a data object.

2. An XML document is valid, if it further complies with an associated document type declaration.

There are multiple contemporary languages to specify a document type. Among them, the two W3C standards are the document type declaration (DTD)[5] and XML schema (XSD, Fallside and Walmsley, 2004), where XSD is the most prominent schema language today.

[5] `http://www.w3.org/TR/REC-xml/#dt-doctype`

2.4.2 XML schema

XML schema is a language to describe the structure of a set of XML documents that validate against the schema. An XML schema consists of the following:

- **Data types.** There are simple and complex data types. The XML schema standard defines a predefined set of simple types. An extension mechanism allows defining derived types that extend or restrict another type. Each type consists of a name. A complex type can further consist of attributes and a particle.

- **Attributes.** An attribute consists of a name and a simple type. The simple type determines the values the attribute can take in an XML document validated by the XSD.

- **Particles.** A particle either consists of exactly one "model group" or exactly one element definition.

- **Model groups.** A model group consists of a vector of particles. A model group can be either a choice or a sequence. Choice means that an XML document's structure can adhere to the definition of *any* of the particles in the vector. Sequence means that an XML document must contain *all* the definitions of the particles in the order of the vector. By the recursive definition, a specific mixture of choices and sequences can be defined for the complex type the model group is part of.

- **Element definitions.** Elements are the basic building blocks of XML documents. An element consists of a name and a type. The tags of an XML document validated by an XSD must adhere to the element names of the XSD, where the structure must correspond with the type definition of the element. Through the recursive definition of an element, the complex type that element is part of can be specified.

- **Constraints.** These are declarations about data types and elements. A constraint may, for example, define minimal, maximal, and default values.

- **Namespaces and import/include options.** Each type definition defined in XSD is associated to a target namespace. The target namespace can be declared globally for all type definition or individually for each type. Via the import and include options, an XSD may reference type definitions in other namespaces. That supports modularization of the XML schemas.

As the definition of XML schema is recursive, we refer to an XML document also as a hierarchical type. We understand a hierarchical type to be an abstraction of XML schema. A formal definition of a hierarchical type will be given later.

2.5 API integration layer

In contrast to the data integration layer concentrating on the seamless exchange of data, the application programming interface (API) integration layer concentrates on the sharing of business logic as displayed in Figure 2.7 on the facing page. In particular, rather than building the interface of two systems with their databases, the API integration layer builds the interface with the applications. An invoker may not only access an application's data, but also its methods. That means that APIs provide a hook to connect to an application via its business logic to retrieve its underlying data. Examples for APIs are

- component interfaces such as CORBA,[6] DCOM,[7] or JavaBeans,[8]
- transaction processing interfaces such as IBM's CICS[9] or Tuxedo,[10] and
- packaged application interfaces such as SAP's Business API (BAPI).[11]

On the interface integration layer, this dissertation uses Web service technology for interface descriptions. According to the W3C, a Web service

> "is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL).[12] Other systems interact with the Web service in a manner prescribed by its description using SOAP[13] messages, typically conveyed using HTTP[14] with an XML serialization in conjunction with other Web-related standards." (Booth et al., 2004, Sect. 1.4)

Web service technology comprises more than 100 languages and definitions handling different aspects of Web services. The Web service description language (WSDL) plays the central role among them.

2.5.1 Structure of Web service descriptions

WSDL is an XML grammar for describing network services as collections of communication endpoints, called ports, capable of exchanging messages (see Christensen

[6] http://www.corba.org/
[7] http://msdn.microsoft.com/library/cc201989.aspx
[8] http://java.sun.com/products/javabeans/docs/spec.html
[9] http://www-306.ibm.com/software/htp/cics/library
[10] http://www.oracle.com/technology/products/tuxedo/index.html
[11] http://help.sap.com/saphelp_nw04/helpdata/en/e0/9eb2370f9cbe68e10000009b38f8cf/frameset.htm
[12] http://www.w3.org/TR/wsdl
[13] http://www.w3.org/TR/soap/
[14] http://www.w3.org/Protocols/

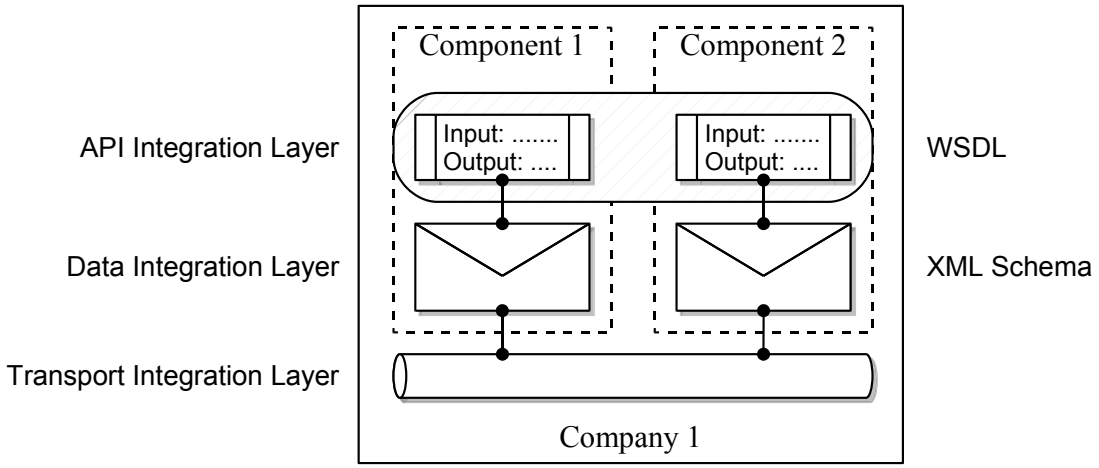


Figure 2.7: API integration layer.

et al., 2001). The abstract definition of endpoints and messages is separated from their concrete deployment in the network or data format bindings. That allows messages, port types, and operations to be defined *abstractly* and reused.

- A message is an abstract description of data being exchanged.
- A port type is an abstract set of operations supported by one or more endpoints.
- An operation is an abstract definition of an action supported by the service.

In addition, a WSDL definition consists of the following *concrete* elements.

- A type is a container for data type definitions using some type system, for example, XSD.
- A binding is a concrete protocol and data type specification for a particular port type.
- A port is a single endpoint defined as a combination of a binding and a network address.
- A service is a collection of related endpoints.

2.5.2 Semantic annotation for Web service descriptions

SA-WSDL[15] is an extension of WSDL by semantic annotations. The structure of an SA-WSDL document is very similar to the structure of a WSDL document. SA-WSDL extends WSDL by additional attributes that can be attached to some of the original WSDL tags. In particular, the attribute `modelReference` defined in the namespace

[15] `http://www.w3.org/2002/ws/sawsdl/`

`http://www.w3.org/ns/sawsdl` may be added to an element definition in an SA-WSDL file. The attribute's value is an arbitrary string that can be used for annotation. It is recommended by the SA-WSDL working group that the string contains an URI identifying a concept of some taxonomy or other term definition system. The interpretation of the annotation is left open by SA-WSDL with the purpose of flexibility in adapting the appropriate term definition system for the actual application. Thus, the interpretation depends on the implementation utilizing the annotation.

2.6 Business process integration layer

The business process integration layer, displayed in Figure 2.8, is primarily concerned with allowing business processes to bridge multiple applications in a consistent way. BPI solutions allow enterprises to leverage their investments in legacy systems by automating and managing the business processes that span these systems. Existing systems may be interconnected without writing code that replicates existing functionality.

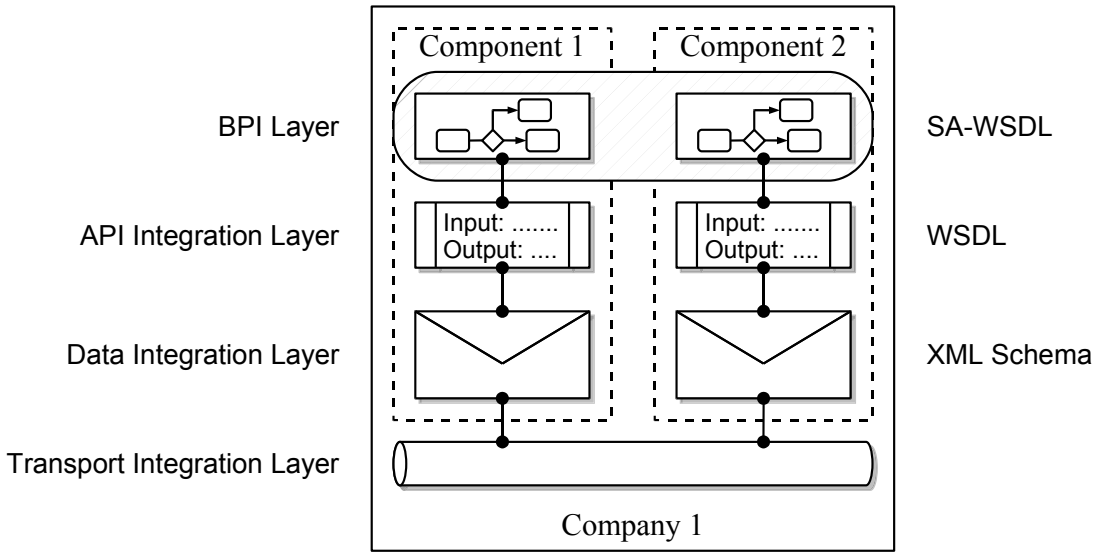


Figure 2.8: Business process integration layer.

Business process integration requires dividing the software into front-office and back-office. Front-office is exposed to users and utilizes back-office functionality. That leads to a loose coupling of the applications. A critical element in BPI is the human involvement. Business processes therefore consist of manual and automatic tasks jointly managed.

Business process management (BPM)

> "extends BPI techniques by including process analysis, process definition and redefinition, resource allocation, scheduling, measure-

> ment of process quality and efficiency, and process optimization." (Papazoglou and Ribbers, 2006, p. 539)

Business process management is sometimes seen as a means to reducing the gap between the business user and the technician. Although it can be said that organizations have always been using BPM, new impetus was brought by business process management systems (BPMS), i. e., software tools that allow for the direct execution of the business processes without a costly and time intensive development of the required software. Conventional BPMS are primarily designed for intra-enterprise process management. They are hardly used to handle processes with tasks and data separated by enterprise boundaries, for reasons such as security or privacy. However, the nature of most business processes involves the cooperation of two or more roles to achieve a specific task. Hence, newer BPMS are increasingly enhanced by corresponding functionality to manage also collaborative business processes (CBP) that integrate multiple companies and thus reach to the area of e-business.

Chapter 3

Problems and Requirements

After introducing the fields of e-business and enterprise application integration (EAI) in the previous chapter, this chapter identifies problems of EAI with respect to business process integration in e-business. In particular, the problematic situations identified in this chapter are

1. free definition of interface objects,
2. behavioral information only in experts' heads, and
3. manual exception handling.

This chapter is structured as follows: Section 3.1, Section 3.2, and Section 3.3 describe the three problematic situations listed above. In Section 3.4, we derive requirements for a solution of the three situations.

3.1 Free definition of interface objects

Starting to implement after designing has the highest probability to end up in a high-quality implementation. However, a thorough design step has disadvantages. It is costly, and it is not always possible to foresee all potential uses of the artifacts modeled. Practitioners differ from theoreticians in that they start implementing before a design is worked out to its ultimate completion. That has been the driver for many years of research in software engineering which provides tools and procedures to target the lowest achievable *combined* effort of design and implementation as a trade-off between the two. Still, the result are artifacts that are not sufficiently structured to optimally allow later reuse, and maybe even redundant implementations of the same concepts in slightly different ways.

While redundancy is not as problematic during software design—except labor that could have been spared,—the bigger problem starts with using the redundant system, especially when business partners—customers or other collaborators—heavily rely on its continuous functioning such as in the area of enterprise software. Albeit many

development support tools, software maintenance is still a big cost driver in IT (see Mens and Tourw, 2004). We now examine why a redundant system—i. e., a system containing redundant artifacts—increases the cost of software maintenance.

3.1.1 Maintenance effort

From the release, a software product undergoes two kinds of change. First, bugs are being fixed, and second, missing features are being added on. Both are problematic in a redundant system.

Change impact

Adapting a redundant system causes an unnecessary multiplication of the cost of the change. There are two main reasons for that. Adapting a redundant system yields the potential of

1. inconsistently changing the redundant assets, and
2. overseeing some assets that also need change.

Reuse

Adding to an existing system raises the similar question

- whether to reuse some existing artifacts, or
- whether to create new artifacts from scratch.

The reuse of artifacts is difficult if there are multiple, obviously related—i. e., redundant—versions of the same artifacts because one would like to pick the most relevant, up-to-date, and appropriate artifact for reuse.

3.1.2 Interrelation of redundancy levels

The redundancy problem gets more complex when we consider that redundancy can occur at different types of artifacts. Let's consider Figure 3.1 on the facing page as an example for a system containing redundant artifacts. The level of IT system interfaces is the level business partners use to integrate. That means, once an interface of a provider (for example, a shipper like Amazon[1]) is exposed to be used by requesters (for example, a carrier like FedEx,[2] or UPS[3]), it must be maintained throughout all subsequent releases. That becomes an unnecessary burden if the same functionality

[1] `http://www.amazon.com/`
[2] `http://www.fedex.com/`
[3] `http://www.ups.com/`

1. is presented via alternative interfaces (the code fragment 1 provides the redundant interfaces 1 and 2 in Figure 3.1), or even
2. is redundantly implemented (the redundant code fragments 1 and 2 provide the redundant interfaces 1 and 2 in Figure 3.2 on the next page), or moreover
3. is implemented using different internal data types (the redundant code fragments 1 and 2 provide the redundant interfaces 1 and 2, and use the redundant types 1 and 2 in Figure 3.3 on the following page).

A change of the functionality reflects in the need to unnecessarily adapt

- interface 2 in the first case,
- interface 2 and code fragment 2 in the second case, and
- interface 2, code fragment 2, and type 2 in the third case.

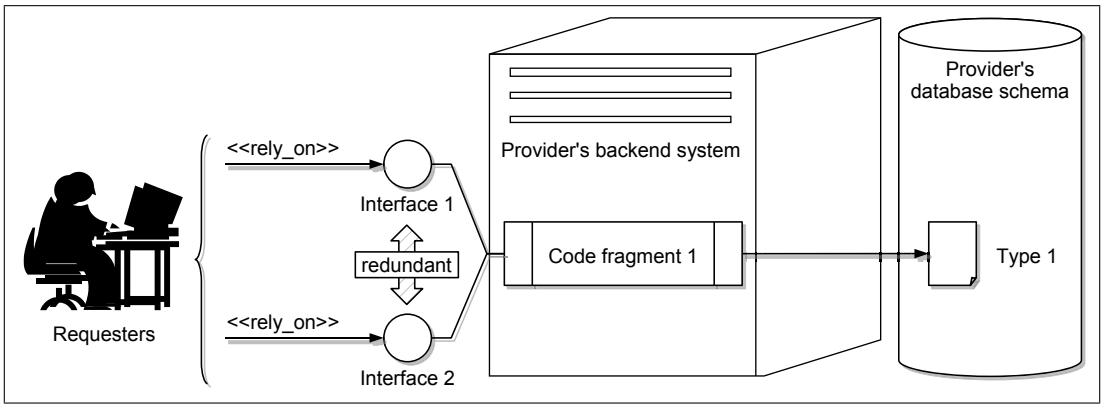


Figure 3.1: System with redundancy caused on interface level.

3.1.3 Summary

The main challenge on the data integration layer (see Figure 2.2 on page 16) for enterprise integration in e-business is that different enterprises, and even different applications within the same enterprise, may use different data types for the same real-life object they refer to. In a community of willing integration participants, the problem multiplies. On the API integration layer (see Figure 2.2 on page 16), different Web services and operations may have been defined to access the same, equivalent, or similar implementations.

In addition to syntactic mismatches like different languages used to name tags, there are semantic mismatches. Two tags carrying the same name may indeed mean different things in the different contexts they are used. A very unnatural, but unfortunately

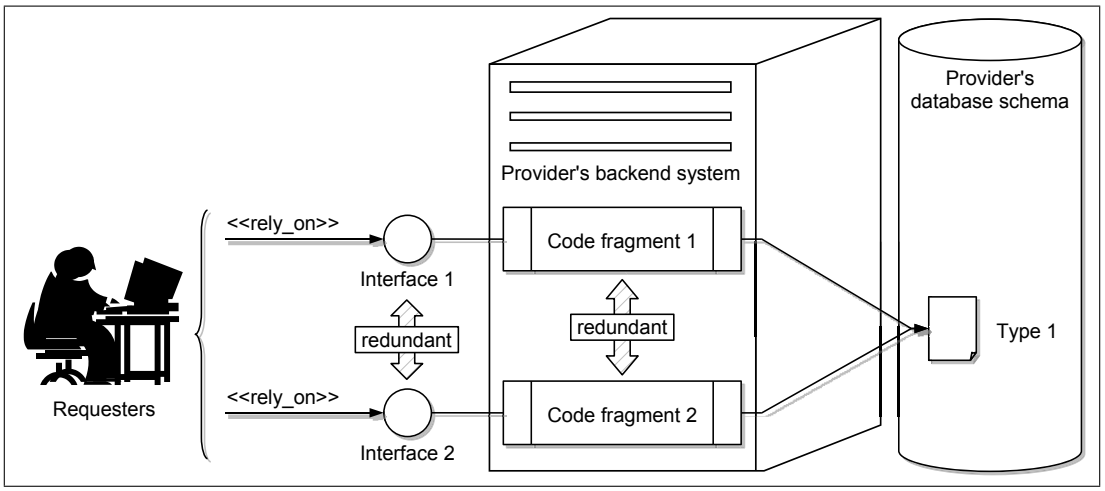


Figure 3.2: System with redundancy caused on implementation level.

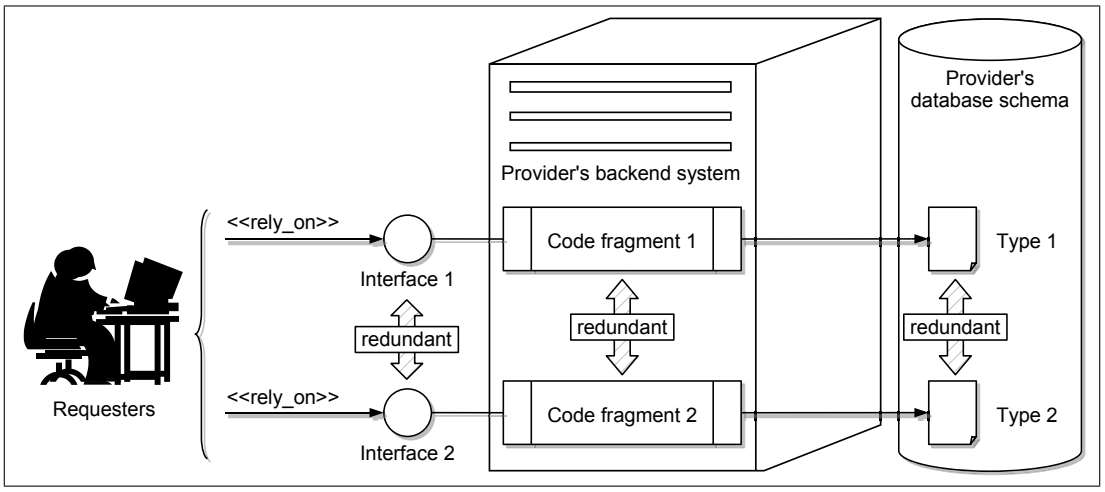


Figure 3.3: System with redundancy caused on data type level.

very common semantic mismatch is the misuse of a field by filling it with data of a completely disjoint domain. That mostly happens due to the lack of a field of the needed domain.

The third form of mismatches concerns implementation-level mismatches. The most prominent cause for implementation-level mismatches are different value lists. For example, in the United States, mostly the employer identification number (EIN)[4] is used for referring to an institution, whereas in the European area more commonly the data universal numbering system (D-U-N-S)[5] is used. An employer active in both markets may have a different reference number depending on the context where it is referred.

[4] `http://www.irs.gov/businesses/small/article/0,,id=98350,00.html`
[5] `http://www.dnb.com/us/duns_update/index.html`

3.2 Behavioral information only in experts heads

There are two main challenges with contemporary Web service descriptions: the gap between technical and ontological representation, and a lack of information on operation interdependencies. Both can be subsumed under unmodeled information.

3.2.1 Technical representation vs. ontological meaning

A Web service description as provided by the prominent technology WSDL describes what is needed to technically invoke a Web service. Although a WSDL document describes the types of operations, its parameters and their detailed structure, it lacks to state the different potential meanings of an operation for a business process. It is very common that such information is hidden in specific values of a message. Thus, a different instantiation of the abstract message type "purchase order" may actually once be a purchase order request, another time a purchase order acceptance, and a third time a purchase order cancellation, depending on, for example, a type code field which might be part of the purchase order message (Fensel and Bussler, 2002).

3.2.2 Potential communications

A prerequisite for complex workflow integration is enumerating the potential communications between the IT systems to be integrated. That is one of the two by far most work-intense tasks in IT system integration besides the actual design of the collaborative business process (Küster et al., 2007, Pistore et al., 2005c). That information is normally not stored. It resides in the heads of experts who know what the potential connections are, or they detect them from looking at the message types.

3.2.3 Operation interdependencies

The second challenge of Web service descriptions for e-business is that there is no machine-interpretable information on how different operations included in the description may depend on each other. For example, an online purchasing system may provide three Web services: login, search, and order. Whereas the owner of the system does not care whether the customer directly orders without searching for products, they may indeed care that a customer is logged into the system before they place an order.

The current take on the matter is that Web services are said to be stateless. That means, that a Web service may be invoked at any time without breaking the system. However, that definition does not say how a Web service may *successfully* be invoked. In practice, a wrong execution order of Web service operations results in a failure response. Receiving a failure response at run time is neither helpful for an automated system nor an unskilled human trying to generate a meaningful execution sequence of the Web service's operations at design time.

From an architectural point of view, a Web service is an interface to manipulating data objects in the background. Thus, the operations of a Web service are lifecycle methods of the underlying data object. The lifecycle, and thus the sequencing of Web service operations, is constrained by the following three factors:

1. An obvious reason for the need for sequencing is a message dependency of two operations. If an operation o_{receive} requires an input which can only be provided by another operation o_{send}, then o_{send} must be executed some time before o_{receive}.

2. Business requirements may be the reason for a mandatory operation sequencing although technically a concurrent execution would be possible. For example, a buying service may state that it would give its credit card information only after an offer was made and accepted. Another example is the two-step authorization procedure where a purchase order, before being issued, first needs to be checked by a representative of the financial department and subsequently, depending on the amount, also by the representative's manager. The business reason for sequencing is clearly to reduce the amount of authorization requests that hit the financial department representative's manager.

3. Finally, the most profound reason for the need to sequence at the interface level is that a Web service may be implemented in a way that the invocation of operations in the wrong order will always fail. There does not even need to be any reason for that—it suffices that the implementation cannot cope with any other invocation order. Such an inflexible implementation may be due to a poor software design which may be due to cost constraints during software development.

A practical observation is that about 80% of integration code manages the sequentially correct invocation of lifecycle methods of the underlying data objects, and only 20% are actual business logic. A company's IT system's lifecycle information is being established at component design time and must be observed for each integration with partners. That information needs to be re-engineered for each integration project at integration design time, if no proper reuse mechanism is in place.

Example

Let's consider the example of a consortium of schools managed by a common head quarter. The collaborative business process to compose is the transfer of a student from one to another school in the consortium. The desired process depicted in Figure 3.4 on the next page involves the retrieval of active student enrollment information from the old school, the matriculation at the new school, and the removal from the register of students at the old school. Now assume, that upon removal from the register of students, the old school actually deletes the student's record. Thus, any subsequent invocation of the retrieval operation will fail. The knowledge that retrieval must always be invoked before deletion is established during component design time. In addition, we observe that the student matriculation service at the new school also accesses student

information. Since old and new school however play different roles in the student transfer process, the invocation of the enrollment operation of new school is completely independent of the retrieval and deregistration at the old school. The information that student information is managed at two independent locations, and thus creation at new school can be invoked after deletion at old school is only available at integration design time.

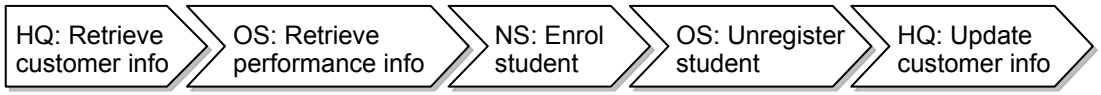


Figure 3.4: Desired business process. HQ: Head quarter, OS: Old school, NS: New school.

3.3 Manual exception handling

As elaborated before, EAI faces the challenge to overcome semantic, syntactic and implementation heterogeneities. Business process management systems support the design, execution and monitoring of business processes. The only manual part is the design of business processes, which in fact is a relevant research topic as a multitude of research publications in the area of process integration indicate. The challenges of business process design involve multiple integration participants, whose IT systems must coherently evolve during the execution of a business process. The core is that all participants coherently have to pursue some common goal in the collaborative process. Such a goal may either exclusively concentrate on the desired outcome of the process, or also state acceptable fall-back outcomes.

3.3.1 Desired outcome as a goal

Commonly pursuing a goal means that all participants have to cooperate in a way that activities being required by a later process step have been executed by a participant in a former process step. Here, activities includes operation interdependencies discussed before with the addition that such interdependencies can now span multiple participants, and with the difference that only interdependencies that affect the business goal need to be considered. Cross-participant operation interdependencies are inherent to business integration, as the reason for cooperation is to achieve a common business goal by combining the capabilities of the partners. A collaborative business process must have the potential to achieve the business goal when executing the process.

3.3.2 Goal includes fall-back outcomes

Correctly treating fall-back outcomes becomes important as soon as operations in a process have side effects, which is the standard case in business processes. For example,

consider a process step that takes money from a customer's bank account and books a flight ticket for the customer. That shall, for example, only happen if another process step successfully rented a hotel room for the customer. Here, it is important that the participating process steps are either both completed or none of them. That behavior is also known as transactionality. We are able to specify transactionality by explicitly stating the desired and the acceptable outcomes of a business process. That definition includes that no other outcomes are ever produced by the business process.

A subproblem of transactionality is called exception handling. Exception handling is ensuring transactionality in the presence of undesired IT system responses that may hinder the achievement of the primary business goal. Undesired system responses may result from

non-deterministically observable system implementations, such as a flight booking service may either respond positively or negatively;

faulty implementation exceeding the system's interface contract, for example, a system proposing to always accept a reservation, but at execution time responds with a decline message; or

system failure on a lower level, for example, the invoker of a flight-booking service does not receive its answer in the case the network connection dropped due to hardware failure or break-down of a network connection.

Common practice proposes to handle issues on the level they occur. As business process management relies on the contracts Web service descriptions provide, the issues of faulty implementation and system failure are out of the scope for this dissertation.

3.4 Deriving requirements

As discussed in this chapter, the causes for challenges of business process integration reach down to the layers of business process, API, and data integration. In particular, we identified the challenges of

- free definition of interface objects,
- behavioral information only in experts' heads, and
- manual exception handling.

A free definition of interface objects yields the risk of inconsistency among *different* object types that are used for the *same* purpose. In the case of change, the redundant object types have to be altered correspondingly. That makes their maintenance expensive as explained in Section 3.1 on page 27.

Information residing only in the heads of experts, and thus especially being not accessible for machines, tends to be forgotten. That results in redundant behavioral

models that require expensive maintenance. The later in the software production process an error is detected after the introduction of the error, the more costly is the correction.

Manual exception handling is a difficult job to perform, and therefore may yield the risk of a low-quality implementation that can, of course, be countered by more expenses for the development or testing. However, some failures will only show up at run time, which reduces customer satisfaction.

From the challenges identified, we derive the following requirements for our solution:

- detect redundant interface objects,
- track behavioral information through software design phases, and
- mechanically support exception handling.

The challenges, risks, problems, and requirements named in this section are also collectively displayed in Table 3.1.

Table 3.1: Requirements

Situation	**Risk**	**Problem**	**Requirement**
Free definition of interface objects	Inconsistency through redundancy	Expensive maintenance	Detect redundant interface objects
Behavioral information only in experts' heads	Forgetting knowledge & redundant models	Expensive reengineering & late corrections	Track behavioral information through software design phases
Manual exception handling	Low quality of implementation	Development expensive, failures only at run time	Mechanically support exception handling

In the rest of this section, we further detail the requirements. The elaboration brings together the facts from Chapter 2 with the challenges from the beginning of this chapter.

3.4.1 Detect redundant interface objects

Interface objects relevant for business process integration can be grouped into

- data types and
- Web service definitions.

Both groups have in common that they are hierarchical types. In particular, a data type can contain complex elements that are defined as data types themselves. A Web service definition consist of operations. Operations consist of messages. Messages are structured according to data types.

Using the available information, detecting redundant interface objects means to search for overlap in the technical definitions of the interface object types. How that is related to finding ontological redundancy will be detailed later. Different redundant interface object types can be uniquely identified by their overlap. As the task is to identify redundant interface object types among a possibly large set of interface object types, a solution is sought that computes all unique overlap groups at once as that is expected to yield a better performance over a one-to-one comparison and later integration of the single results.

Reviewing the literature on methods in artificial intelligence (AI) reveals that the method of closed frequent itemset mining (CFIM) solves a very similar task. Therefore, the more refined problem of this dissertation is to utilize and adapt CFIM to detect redundant interface object types. Details on the technique follow in a separate chapter on the structural alignment part of our solution.

3.4.2 Track behavioral information through software design phases

The challenge with behavioral information only in experts' heads is

1. to cover the information in a form that it can be further used by a machine and
2. to keep and transform the information over the different phases of component-based software development.

Covering the behavioral information means to introduce a model that is capable of storing or deducing

- ontologically different outcomes of a service,
- operation sequences that must be followed, and
- potential data flow between different services.

In addition, the model of behavioral information must be mapped to the definition of the respective Web service operation.

Keeping and transforming behavioral information during the software development lifecycle means to have a representation for behavioral information during

- component design time,
- integration design time, and

- run time.

The model at component design time contains a description of the general back-office capabilities of a software component. The integration design time model uses a fragment of the behavioral model at component design time and may introduce new behavioral constraints to the model. However, no operation execution sequences that were not allowed in the component design time model must be allowed in the integration design time model.

As an integration design time model can be understood as a view of the component design time model, it is desired that there is no duplicate implementation of functionality in the integration design time that existed in the component design time model.

The behavioral model at run time in the Web service architecture is an orchestration of Web services. Again, the orchestration is a view of a set of participating behavioral models. It should thus not exceed the restrictions imposed by the integration design time models. In the best case, the orchestration is directly executable to minimize consistency problems that would arise if executable code was built based on the orchestration model, which could then separately being altered.

As integration design time and run time models are views, an ideal solution would produce a directly executable orchestration that crosses the integration design time model and directly utilize the preferably executable component design time model.

3.4.3 Mechanically support exception handling

Realistic Web services have alternative outcomes. For example, approval or denial may be communicated via the same response message type by setting a flag inside the concrete message object at run time. However, a human process designer starts not with looking at particular Web services, but thinking of the business goal a business process should pursue. Based on that goal, the human process designer seeks for services that contribute to the goal. After identification, the Web services need to be ordered according to their behavioral requirements. When a preliminary sequencing is done, the used Web services are checked whether their contract, or their Web service description, contain information about responses alternative to the responses desired for pursuing the business process goal. If such responses exist, the human business process designer would create compensation Web service operation sequences that handle the exception and bring the running business process instance to a controlled, consistent end. Building the compensation sequences may affect the initial sequencing that was chosen for the other Web service operations. In conjunction with realigning the whole business process, also the data flow between the Web service operations needs to be defined.

With the behavioral information described in Section 3.4.2 on the facing page, an automatic proposal for the sequencing should be feasible. In addition, the CFIM-like message type analysis from Section 3.4.1 on page 35 could be used to propose potential communications. In an ideal solution, the human would need to define the

process goal with respect to the behavioral model and check and correct the potential communications found from the message type analysis. A semi-automatic process designer would prepare a business process proposal that the human process designer could again check and adapt. Additionally, an ideal solution would only allow those adaptations by the human that conform with the behavioral models of the participating operations.

Chapter 4

Shortcomings of Existing Approaches

In this thesis, we consider the three challenges of free definition of interface objects, behavioral information only in experts' heads, and manual exception handling. We start with a restatement of the requirements on the properties an ideal solution would have. Afterwards, we assess the current state of the art with respect to the requirements.

An ideal solution to the support of business process integration should address the issues discussed in the previous chapter. That is,

1. detect redundant interface objects, or otherwise ontologically and technically tackle the free definition of interface objects on the data and API integration layer,
2. track behavioral information through software design phases, or make behavioral information only in experts' heads otherwise manageable, and
3. mechanically support exception handling, or otherwise support manual exception handling, including
    - respecting each partner's operation invocation sequencing requirements,
    - ensuring that the desired outcome may be achieved, and
    - ensuring that only desired or acceptable outcomes are the result of every execution.

We structure this chapter based on the requirements listed above. Section 4.1, Section 4.2, and Section 4.3 discuss existing approaches that work toward the listed requirements. Section 4.4 summarizes the existing approaches and states how our solution builds upon the existing work.

4.1 Detect redundant interface objects

The closest existing, related work to detecting redundant interface objects can be found in the areas of

- XML schema definition and Web service mining,
- software restructuring,
- schema matching and ontology alignment,
- clustering, and
- data mining.

4.1.1 XML schema definition and Web service mining

There exists work for the explicit mining of XML schema definitions (see Dong, 2005, Termier, 2004). However, that work considers the tree structure of XML during mining. We argue that the approach is too fine-grained for an ontological analysis as the structuring of certain elements may be for the whole sake of grouping and does not contribute to the meaning of the single sub-elements. Moreover, although efficient algorithms exist, mining trees adds computational expense (Zaki, 2002). As our focus is both on an ontological and a scalable approach, we will rather analyze the substructure as a plain set as we will detail later.

In Web service mining, approaches are to our knowledge either based on some distance calculation (see, for example, Wang and Stroulia, 2003), thus rather belong to the clustering domain, or perform an analysis of operation and parameter names (see, for example, Dong et al., 2004) without considering the data structures.

As we could not identify appropriate methods to address our requirement to detect redundant interface objects in its natural domain, we widen our view to other domains of computer science in order to find related concepts.

4.1.2 Software restructuring

In the field of software restructuring and refactoring, formal concept analysis is used for the mining of redundancies in data types (see Mens and Tourw, 2004). Formal concept analysis bases on the lattice theory performing graph operations to identify concepts out of sets of objects that share common attributes (see Ganter and Godin, 2005, Snelting and Tip, 1998, Stumme et al., 2002). Work in that area looks promising from the performance perspective. There was also research performed using clustering for data reorganization (see McCormick et al., 1972). However, as concluded in a more recent work, clustering is different from frequent itemset mining and less appropriate for the task of detecting redundant interface objects (see van Deursen and Kuipers, 1999).

4.1.3 Schema matching and ontology alignment

There is a lot of work done in the areas of schema matching (see Rahm and Bernstein, 2001, Shvaiko and Euzenat, 2005) and ontology alignment (see Kalfoglou and Schorlemmer, 2005, Noy, 2004). A rather new approach is a combination of both, which is concerned with using an ontology to improve schema matching, performed by Drumm et al. (2007b).

The main setting for schema matching and ontology alignment is the respective existence of two schemas or ontologies that are being tried to align using an automated procedure. First approaches on schema matching focused on seeking structural similarities of the two schemas. Later, the labels of schema entities were considered as well. Ontology alignment expands the similarity measures by using ontology reasoning.

The contemporary schema matching tools follow the same procedure:

1. A similarity matrix is defined as the cross product of the set of all schema entities with itself.

2. A set of matchers is employed. Each matcher creates values for the similarity matrix.

3. The different values of the similarity matrix that originated from the different matchers are combined using an aggregation function, such as the average function.

The aggregated similarity matrix is the result of the schema matching and presented to the user. The result of a schema matcher must always be checked by a human since the matcher results represent probability values that suggest the similarity of the entities.

However, work in that area may not solve the problem of free definition of types or the requirement of detecting redundant interface objects. The reason is that the approaches in the area of schema matching consider two interface objects at a time and deeply analyze their similarity. However, for solving the named problem, it is important to identify a common overlap of multiple interface objects. A one-to-one comparison cannot directly serve that purpose. It could, of course, be used to compare each pair of messages which adds up to $\frac{n(n-1)}{2}$ computations for n message types. However, the complexity of the matcher adds to the complexity which makes the approach less feasible.

4.1.4 Clustering

Clustering relies on a similarity measure between pairs of entities that is used to group entities into clusters so that entities within a cluster have high similarity in comparison to one another, but are very dissimilar to entities in other clusters (Han and Kamber, 2006). However, the contributions in the area of clustering cannot be applied to the challenge of detecting redundancy as a similarity measure considers the properties of

single entities, but not properties of clusters. We demonstrate the difference with the following example.

Let's consider the sets $S_1 = \{a, b, e, f\}$, $S_2 = \{a, b, c, d\}$, and $S_3 = \{c, d, e, f\}$. A similarity measure d for clustering sets with redundant elements could be defined as the size of the intersection of two sets. Calculating the similarity measures for all pairs of sets yields

$$\begin{aligned} d(S_1, S_2) &= |S_1 \setminus S_2| = |\{a, b\}| = 2, \\ d(S_2, S_3) &= |S_2 \setminus S_3| = |\{c, d\}| = 2, \text{ and} \\ d(S_1, S_3) &= |S_1 \setminus S_3| = |\{e, f\}| = 2. \end{aligned}$$

As all pairs have the same similarity measure, which is also larger than 0, all sets—$S1$, $S2$, and $S3$—are determined equally similar to each other and would therefore be clustered together in one cluster in our example. That is however not the desired result because for detecting redundancy, it is important to discover that S_1 and S_2 contain $\{a, b\}$; S_2 and S_3 contain $\{c, d\}$; and S_1 and S_3 commonly contain $\{e, f\}$. Instead of a similarity value calculated in a pairwise manner, the exact commonality of the entities should be the measure used to form groups in order to be useful for detecting redundancy. That type of analysis is natively achieved by closed frequent itemset mining, which is presented in the following section.

4.1.5 Closed frequent itemset mining

A lot of work has been carried out in the area of pattern mining. The technique most appropriate for the requirement to detect redundant interface objects is closed frequent itemset mining because it returns a maximum overlap between flat structures. The problem is defined as follows:

Transactions and items. A set of transactions $\mathcal{P}$ and a set of items $\mathcal{A}$ are given.

Itemsets. Each transaction $P \in \mathcal{P}$ is defined as a subset of $\mathcal{A}$. A subset of $\mathcal{A}$ is also called an itemset.

Occurrences. Every transaction P containing the itemset A is called an occurrence of A. The set of occurrences of A is denoted by $\mathcal{P}(A)$.

Frequency and support. An itemset $A \subseteq \mathcal{A}$ is frequent, if there are at least ϕ transactions in $\mathcal{P}$ that subsume A. We call ϕ the minimum support.

Closed frequent itemset. An itemset $A \subseteq \mathcal{A}$ is a frequent closed itemset if there is no other A' with $\mathcal{P}(A) = \mathcal{P}(A')$ and $A \subset A'$.

Closed frequent itemset mining. Closed frequent itemset mining discovers all closed frequent itemsets of a given set of transactions $\mathcal{P}$.

However, the traditional closed frequent itemset mining does not exactly meet the requirement. Part of the reason is that its original purpose was market basket analysis. Market basket analysis mines the selling transactions of a retail store. It reports patterns in the purchasing behavior of the store's customers. A result of market basket analysis might, for example, be that 70% of the customers have bought bread, butter, and cheese together. If there was an online store of that company, it could use that information by proposing a customer to add butter to their cart, if the cart just contained bread and cheese. That application is very targeted and does, for example, neglect structured transactions.

Fortunately, closed frequent itemset mining was deeply studied during the Internet boom in the early 2000s and an abstract description can be found in several textbooks Berry and Linoff (2004), Han and Kamber (2006), Petersohn (2005), Witten and Frank (2005). The naming of its elements as transactions and items still reminds of its origin.

Efficient algorithms exist for the mining of closed frequent itemsets. For example, *formal concept analysis* uses lattice theory, a special graph theory, for the representation of common attributes of objects. It has led to the development of efficient closed frequent itemset mining algorithms (Zaki et al., 2005).

4.2 Track behavioral information through software design phases

Research for the purpose of e-business in the recent years focused on creating a holistic approach to tackle the challenges jointly under a common umbrella—a framework. Two major kinds of framework can be identified. The semantic Web services frameworks and the grid frameworks. Both share the common target of increasing automation of main tasks of the Web service lifecycle.

- Automated discovery should improve the features of today's Web service registries, particularly UDDI (Clement et al., 2004), by going beyond keyword-based discovery.

- Automated selection should support choosing one Web service for invocation of a set of functionally equivalent services. Thus, properties considered during selection are non-functional.

- Automated composition should simplify the procedural connection of different business partners.

- Automated execution and automated monitoring should improve the running and maintenance of IT systems by reusing gathered information for standard activities.

4.2.1 Semantic Web services frameworks

Semantic Web services is an area of research triggered by the research area of the semantic Web as the application of semantic Web concepts in the area of Web services. The idea behind semantic Web services is to add further information to the information available in industrial Web service descriptions as, for example, in WSDL. A number of competing frameworks has been proposed: OWL-S, WSMO, and METEOR-S.

Ontologies: The foundation of the semantic Web

The term ontology has been widely used by many communities using different definitions. The origin of the term ontology is in philosophy. Here, ontology means the science of being (see Rosenkrantz, 1998). Ontology discovers approaches to sort out what things and kinds of things exist. The first to perform ontology research was Aristotle who introduced the notion of 10 different categories of being, namely, substance, quantity, quality, relation, place, time, posture, possession/habit, action, and passion (receiving). In computer science, the term ontology is used to refer to one understanding of the things in the world. As one computer program is a model, by definition, it only works on a subset of the world (see Stachowiak, 1973). Therefore, the ontology it utilizes needs only describe parts of the whole set of existing things. Other programs may describe a different set of things needed for their purposes, and again other programs may use a different conceptualization to categorize existing things as there were more than one basic category proposals in philosophy as well. For the sake of computing with ontologies, Gruber (1995), Staab and Studer (2004) introduce a definition of ontology as "a formal, explicit specification of a shared conceptualization."

In this thesis, we use the weaker, more abstract, definition of the term ontology as

> "a set of vocabulary definitions that expresses a community's consensus knowledge about a domain. This body of knowledge is meant to be stable over time and can be used to solve multiple problems." (Gruber, 1995)

That definition contains two features:

1. Community. The term "community's consensus knowledge" refers to a group of users of the ontology that agree upon the assertions the ontology makes.

2. Definitions. An ontology constitutes a specification that is binding to the members of the community.

In contrast to the original definition of Gruber (1995), Staab and Studer (2004), the definition of this thesis does not include a formal model-theoretic semantics of the ontology and no methodologically well-designed conceptualization of the matter of discourse. Whenever we use the term ontology in conjunction with external work, the definition of Gruber (1995), Staab and Studer (2004) is meant. Whenever we use the term ontology in the context of the contribution in this dissertation, the definition of this thesis is meant.

OWL-S

The Web ontology language for Web services (OWL-S)[1] originated from the DARPA[2] agent markup language for services (DAML-S).[3] OWL-S is an ontology expressed in the Web ontology language (OWL).[4] OWL-S describes three facets of Web services: profile, process, and binding. The profile contains general input, output, precondition, and effect (IOPE) definitions of a Web service. In addition, a category and a function can be annotated to a Web service by linking it to one concept of any OWL ontology. The process describes how the IOPEs distribute over the single operations of the Web service. The binding describes how the Web service endpoint connected to the semantic Web service description can be invoked.

WSMO

The Web service modeling ontology (WSMO)[5] provides a general, conceptual model for specific Web service models, a concrete language for Web service descriptions (the Web service modeling language WSML),[6] and the reference implementation WSMX (the Web service model execution environment).[7] There are different dialects of WSML, among them are WSML-DL, which bases on description logics (DL, see Baader et al., 2007), WSML-Flight, which bases on F-logic (Kifer and Lausen, 1989), and WSML-Rule, which allows for the specification of rules executed similar to the Rete algorithm (Lausen et al., 2005).

METEOR-S

Unlike OWL-S and WSMO, which were motivated top-down from a conceptual model toward the concrete implementation, the approach of the METEOR-S framework (Verma et al., 2005) is to start from widely accepted languages, such as WSDL, and add semantic annotation only where needed. One of the results of that approach is semantically annotated WSDL (SA-WSDL). SA-WSDL is WSDL plus the possibility to add an annotation to each part of the WSDL description.

4.2.2 Model-driven software development approaches

The need for appropriate documentation of software behavior is well-known in the field of software engineering. However, documentation produced throughout software design is stored in natural language. The model-driven software development (MDSD)

[1] http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
[2] http://www.darpa.mil/
[3] http://www.daml.org/services/
[4] http://www.w3.org/TR/owl-features/
[5] http://www.wsmo.org/
[6] http://www.wsmo.org/wsml/
[7] http://www.wsmx.org/

approach aims at producing machine-interpretable documentation in the form of models which can then be transformed to other models or executable code (Stahl and Voelter, 2006). Unfortunately, many different models addressing different purposes exist in different phases in the development process, for example, use case, component, object, or collaboration models (Booch et al., 2005). Furthermore, many of the popular models in software engineering, as, for example, UML[8] activity and class diagrams, have no well-defined semantics, which makes the interpretation of their integration difficult. MDSD rather focuses on helping the software engineer with maintaining consistency between the models instead of performing higher-order services, such as generating transactionally correct orchestrations, as we do.

4.3 Mechanically support exception handling

Different approaches have been proposed to support procedural interoperability. The approaches range from no automation to mechanically support exception handling of complex services with complex goal requirements (see Rao and Su, 2004).

4.3.1 Manual approaches

The majority of contemporary BPMS provides no mechanic means to support the design of collaborative business processes. They rather allow asserting activities, data transformations, and data and control flow. Research in the area of manual approaches keeps the human task of manually sequencing the business process constituents, but supports that task by

- utilizing patterns or templates (see Benatallah et al., 2002, Karastoyanova and Buchmann, 2004, van der Aalst et al., 2003),
- utilizing knowledge from existing processes via recombination (see Bernstein et al., 1999), or
- allowing specification and verification of interesting properties of the objects involved in a business process and on the business process itself (see Gerede and Su, 2007, Salaun et al., 2006).

The clear advantage of the manual approach is that human users are kept tightly in the modeling loop and have a good chance to trust the result as they observed the whole manual creation procedure.

The disadvantage of the manual approach is that the human user has no support in assessing whether the created business process correctly handles all known error situations, or each execution always arrives at some acceptable outcome.

[8] http://www.uml.org/

4.3.2 Assuming singular Web service response

The first kind of automatic approaches assumes that a Web service only responds in one possible way when invoked. Approaches generating business processes from Web services under that assumption mostly make use of goal-driven AI planning to plan for a simple goal to achieve. Examples for that approach are e-workflow of Cardoso and Sheth (2003), the ASG composer from Meyer et al. (2005), and others (see Doshi et al., 2004, Kim et al., 2004, Mayer et al., 2003).

AI planning needs the following prerequisites:

- A semantic description of an operation consists of its preconditions and effects. Preconditions and effects are characteristics of the world before and after the execution of the operation.
- The current situation is a description of the current state of the world of discourse.
- The goal describes the characteristics of a desired world.

The task of AI planning is to find such a partial-order sequence, or plan, of instantiated operations, called actions, that, when executed, transforms the current world into a world meeting the goal description.

The advantage of primary goal approaches is that the generated action sequence by construction is correct and consistent in that it is guaranteed to reach the desired final state if the operations work as described.

However, for planning with IT systems, managing exceptional situations due to a non-deterministic answer of a service must be taken into consideration. That is dealt with by approaches presented in the following section.

4.3.3 Considering alternative Web service responses

Further work considers that a Web service call can have a multitude of alternative responses. As invokers of a Web service usually do not know which of the alternative kinds of responses they will receive, we call the Web service behavior non-deterministic. Automatic approaches to create a business process considering non-deterministic Web service behavior differ in whether they try to reach a simple goal or a complex goal.

Simple-goal-based approaches

Simple-goal-based approaches can be further split to design-time and run-time approaches based on the time when a composition is generated.

Design-time approaches. Among the design time approaches, one set of approaches assumes an annotation attached to each operation of each participant that states its relation to other operations. These approaches add rules or preconditions and effects to

atomic tasks which express certain dependencies between the single tasks (see Chun et al., 2002, Laukkanen and Helin, 2003).

The second set of approaches assumes an explicit workflow representation of each participant's software system. The composer from ILOG[9] presented by Albert et al. (2005a;b) is an example of the second group.

The ILOG composer origins from the DIP[10] project which operates upon the WSMO language. Although WSMO uses abstract state machines (ASM, Börger and Stärk, 2003) to express one participant's behavioral interface, ILOG's approach foots on a translation from ASM to UML activity diagrams (UML AD). The ILOG composer takes two or more UML ADs and combines them to a unified UML AD containing the process steps of all participants' behavior descriptions.

The composition is performed by reducing the composition problem to a configuration problem. That is done in such a way that a configuration engine performs a backward chaining of operations based on their inputs and outputs. An output can be assigned to an input if both are annotated by the same WSMO concept. Thus, a semantic alignment has to be done upfront.

Run time process generation. The research area of business process generation during run time can be further separated into approaches which create partial action plans and steer the process based on the plan and approaches that create no plans, but rather align the separately executing business processes without intervention.

The work of Cimpian (2007) is an example for run time process generation without intervention. The approach bases on the WSMO framework. The business process is generated by observing the messaging of the business process at each participant's system and routing the messages appropriately. The drawback of that approach is that the potential of alignment that can be achieved is limited because routing decisions earlier in the process execution can not be made considering routing options later in the process execution.

The paradigm of software agents, especially the beliefs, desires, and intentions agents (BDI agents), can be understood as an example for run time process generation with intervention. In that paradigm, a software agent is an autonomous entity that observes its environment, pursues goals, changes the environment through actions, and communicates with peer agents or humans. A BDI agent is imbued with mental attitudes.

- Beliefs represent the agent's view of the world.
- Desires represent the goals an agent may pursue in general.
- Intentions are goals the agent has committed to pursue.
- Plans are action sequences achieving goals by changing the environment.

[9] `http://www.ilog.com/`
[10] `http://dip.semanticweb.org/`

A plan may contain further goals for which another plan needs to be built as soon as execution reaches such a step. The BDI agent thus computes a plan at run time which may be incomplete. The agent executes its plan and thus steers the business process. The BDI agent may react to non-deterministically behaving IT systems through its observations and may adapt its plan accordingly.

We have participated in an effort to use a technology inspired by the BDI agent paradigm to perform flexible and consistent enterprise management—called "goal-driven enterprise management" (GEM, see Kaiser, 2007, Kaiser and Lemcke, 2008, Kubczak et al., 2008, Lemcke et al., 2008). In the GEM approach, each software agent has a clearly defined scope to observe and influence the environment. For enterprise management, the scope should be matched with the business areas of a company, such as CRM, SCM, and HR. That way, the maintenance of the semantic declarations remains restricted to a manageable size and the planning remains feasible due to the restricted set of operations an agent may utilize. GEM deals with non-deterministic systems by replanning. That method does not ensure that an acceptable outcome may always be reached. For these cases, forward conflict resolution must be performed which is common to today's business software. Although we could show that GEM may manage the near-real-life systems at the semantic Web services challenge (SWSC, see Kubczak et al., 2008, Lemcke et al., 2008), the run time approach yields the disadvantages we discuss in the following section.

Discussion. The design time and run time approaches differ in when they construct the business process. Using the design time approach, the business process is generated before being executed. The advantage is that all non-deterministic branches could be commonly evaluated—although not done by the approaches presented in this section. Thus, the composition algorithm could deal with all known exceptions consistently. Consequently, once a business process was created, it is known to consistently deal with all situations arising from non-determinism.

On the other hand, the design time approach needs an extra mechanism to deal with unknown exceptions. Additionally, a change in business policies or operation definitions needs the explicit recreation of all touched business processes at once. That may be unfortunate in an environment that changes very often.

These shortcoming are at the same time the advantages of the run time approach. As it does not compute business processes upfront, changes of the environment—policies and operations—are incorporated without extra effort in newly requested business processes. Unknown exceptions need no extra treatment compared to known exceptions. The disadvantage of the run time approach is that upon occurrence of an exception, it is not guaranteed that a solution can be found at all that leads to an acceptable final state.

The solution we are going to present in the following chapters is a design time approach following the main ideas of the work from Pistore et al. (2005c). In contrast to the top-down approach from Pistore et al. (2005c), we use a minimalistic approach starting bottom-up by only adding features definitely needed to tackle the procedural

integration problem. These solutions belong to the complex-goal-based approaches, which we review in the following.

Complex-goal-based approaches

For dealing with fall-back goals, two systems have been proposed in the literature. Berardi et al. (2005) propose a complex-goal-based Web service composition that allows a rich description of the participating components including internal state transitions of components in addition to the globally observable behavior. In addition, a building block principle is incorporated that allows building a behavioral model out of common atomic services. Due to the complex description of services and orchestration requirements, the composer described by Berardi et al. (2005) runs in exponential time.

As we seek for a scalable solution, we abandon the approach of Berardi et al. (2005) and have a closer look on a composition approach developed by Pistore et al. (2005c), whose evaluations suggest to run in polynomial time. We highlight its architecture and features in order to give an assessment of its capabilities.

General set-up. The composition system of Pistore et al. (2005c) starts from a set of behavioral descriptions of Web services described using abstract BPEL. The result of their algorithm is a new BPEL file that orchestrates the given Web services according to their behavioral description and a complex goal formulated by the integration expert.

Architecture. The algorithm sequentially undergoes two major computations: (1) The abstract BPEL descriptions of the participating Web services are transformed to labeled state transition systems $\Sigma_{\mathrm{W}1}, \ldots, \Sigma_{\mathrm{W}n}$. A combined state chart Σ is generated that joins the state transition systems. That expresses all potential executions of the combined system. As the behavioral descriptions of the Web services are assumed to include information internal and external to the Web service, the externally observable behavior of the Web services has to be extracted before the integration can be started. For that purpose, the labeling function of Σ is adapted so that the label of each state of Σ is merged with the labels of all externally observable states of each Web service—also called "belief"—that are reachable from that state. (2) Σ and the orchestration requirements—expressed as an "EaGLe" formula g, which is explained below—are fed to a model-based planner. The planner is an extended model checker used to identify a fragment of Σ whose belief adheres to g. In the first versions, there was a strict separation of the two phases. Thus, the algorithm of Pistore et al. (2005c) had to compute a complete joined state machine upfront which costs much computation time. During further research of Bertoli et al. (2006), the upfront state combination became interlinked with the composition procedure. However, extracting the belief information for the joined state transition system is still expensive.

Features. The desired and acceptable outcomes of the integration are specified using the EaGLe language by Lago et al. (2002). EaGLe is motivated as an extension of CTL.

A formula consists of propositional formulas $p :=$ b $p\ p$ $p\ p$ p, where b is a basic proposition. The propositional formulas are connected to other propositional formulas via extended goals (g) as follows.

$$g := p\ \ g\ \mathbf{And}\ g\ \ g\ \mathbf{Then}\ g\ \ g\ \mathbf{Fail}\ g\ \ \mathbf{Repeat}\ g$$
$$\mathbf{DoReach}\ p\ \ \mathbf{TryReach}\ p\ \ \mathbf{DoMaint}\ p\ \ \mathbf{TryMaint}\ p$$

The specialty of EaGLe is the **Fail** goal constructor, which allows recovering from failure, and the distinction between the prefixes **Do** ("strong") and **Try** ("weak"). In combination, the constructors allow for the flexible definition of complex goals. For example, the goal $\mathbf{TryReach}\ a\ \mathbf{Fail}\ \mathbf{DoReach}\ b$ creates a plan that first tries to achieve a. During the execution of the plan, a state may be reached from that it is not possible to reach a. For such a state, $\mathbf{TryReach}\ a$ fails and $\mathbf{DoReach}\ b$ is considered.

The remaining task is the definition of potential output and input matches. The mapping from an output to an input message can be simple or complex as well. A simple mapping directly assigns the value of the output to the input. A complex mapping uses some transformation function to compute the input from the assigned output. Examples for predefined transformation functions are the extraction of message parts and the merge of multiple outputs to one input. Hoffmann et al. (2007) propose to tackle the message assignment problem by using a planner.

Discussion. In this thesis, we seek for an integrated solution for business process integration that is able to track behavioral information through software design phases and mechanically support exception handling. As the work of Pistore et al. (2005c) is promising, we are going to reuse their ideas of how a complex goal should be specified.

However, the existing work expects initial process descriptions to exist, which need to be created from scratch in reality although 80% of the behavioral information is already known at integration design time. We do not attempt using abstract BPEL to cover the existing behavioral information, as the existing work does, due to the lack of formal semantics of abstract BPEL. The lack of formal semantics implies that different users of the models could have different interpretations of their meaning. Therefore, the work cannot be directly applied in our solution as we argue to reuse behavioral information from an earlier development phase, and only a view of the behavior exists at integration design time. An executable orchestration must be mapped back to the earlier behavioral models, which cannot be done by the solution of Pistore et al. (2005c).

In addition to the practical constraints named above, the way the existing approach describes behavior is rather complex which results in a slower run time of that approach compared to the solution we will describe in this dissertation. A performance evaluation will follow in the third part of this dissertation.

4.4 Summary

Existing approaches provide interesting ideas to address the requirements expressed in Table 3.1 on page 35, but the existing approaches do not meet the requirements satisfactorily as discussed in this chapter. We summarize how we advance the state of the art in the following.

Ontology research creates rich languages and general purpose reasoners which are too complex and slow for mass-data handling in e-business. We use the ontology notion of a shared, explicit, central representation of common understanding as introduced in Section 4.2.1 on page 44 and apply more targeted high-performance algorithms.

Semantic Web services research did not achieve automation, but they understand the need for more machine-interpretable descriptions. We use light-weight semantic annotations for WSDL (SA-WSDL) and different kinds of transformable models for the different stages of the software development lifecycle to achieve reuse and facilitate the semi-automatic generation of reliable business processes.

Pistore et al. (2005c) could compose Web services, but they use general-purpose model checking implying an exploding joined state space and an overly complex goal definition, which is not necessary in practice. We use a targeted composition algorithm operating on less expressive descriptions of composition goals and yield a reasonable speed-up.

Our solutions are presented in the following chapters. We begin with the presentation of how our solutions interact in the subsequent section.

Part II

Scalable Ontological EAI and e-Business Integration

Chapter 5

Solution Overview

In this chapter, we introduce the overall concept of how the parts of our solution play together. With the uniform framework presented here, we also address parts of the requirement to track behavioral information through software design phases.

Roughly, our CFIM of hierarchical types facilitates that the understanding and technical representation of the integration participants converge on the data integration layer and the API integration layer of the EAI pyramid (see Figure 2.2 on page 16). The convergence is a prerequisite for our complex-goal-based WS composition to mechanically support exception handling for business process integration. CFIM of hierarchical types and complex-goal-based WS composition are embedded in a common approach to transform behavioral models for EAI and e-business. Table 5.1 summarizes the solutions we provide for the requirements defined in Chapter 3 on page 27.

Table 5.1: Solutions

Requirement	**Solution**
Detect redundant interface objects	CFIM of hierarchical types
Track behavioral information through software design phases	Transform behavioral models for EAI and e-business
Mechanically support exception handling	Complex-goal-based WS composition

This chapter is structured as follows: Section 5.1 partitions our solution to the perspectives taken by different peers involved in business process integration. Section 5.2 details the types of activities that are performed on the different perspectives. As the same types of activities take place on different perspectives, Section 5.3 provides the complete picture by presenting the perspectives and activities in the chronological order they would occur naturally.

5.1 Structuring

For the solution of the integration problem on the levels of EAI and e-business integration,

1. we extend the way software components are described at component design time by a formal representation of behavioral information—the behavioral models—which will be described in Chapter 7,

2. we introduce new, model-driven transformations to utilize behavioral models in the different stages of the integration design time without loosing or violating any behavioral information of earlier stages, and

3. we utilize closed frequent itemset mining (CFIM) to support the human task between the transformations of identifying ontologically equivalent objects that are represented differently.

For our approach, we need to take a more fine-grained view on the business process integration (BPI) layer than we did in Chapter 2 on page 11. Figure 5.1 on the facing page displays the four perspectives of how people who are involved with component-based software development come in touch with behavioral information about components. We detail these perspectives in the following. Thereby, we address four phases of component-based software development:

1. component design,

2. EAI,

3. e-business integration, and

4. execution.

The two hatched boxes in company 1 and company 2 represent the BPI layer of EAI as it was introduced in Section 2.6 on page 24. In the BPI layer, we distinguish the component perspective and the EAI perspective.

- **Component perspective.** The component perspective consists of the message types, Web service definitions, and information about operation dependencies that describe the interface of one software component. We can say that the information contained in the component perspective describes a component's basic capabilities. That perspective is created during component design time and it is the basis for integration at EAI time.

- **EAI perspective.** The EAI perspective contains the integrated view over all components in a company. It is used as a basis to start EAI at integration design time. However, that view is not just the sum of the information on all components because it is usually the case that a company—for example, company 1—does

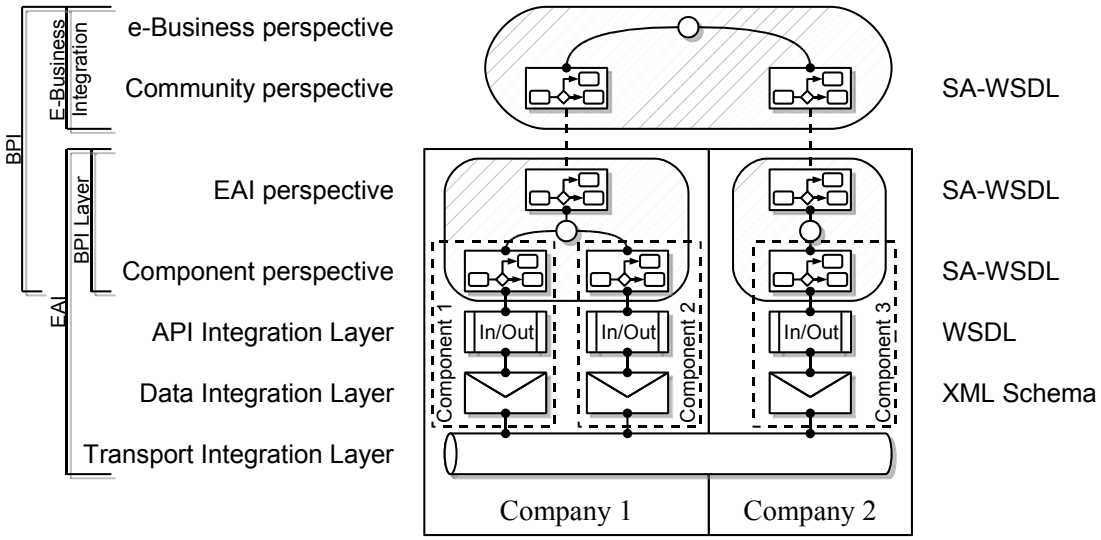


Figure 5.1: Integration layers with detailed BPI layer.

not use all the capabilities of each component—for example, of component 1 in Figure 5.1. Rather, it is likely that company 1 only uses that part which is needed to implement its specific business procedures. Furthermore, company 1 may also choose to restrict any flexibility of Web service operation sequences to the very specific sequence that is sufficient or required for its procedures. The restriction is also called an orchestration of the components—or, more specifically, of the Web services providing the components' functions. The orchestration is illustrated in Figure 5.1 by the circle connecting component perspective and EAI perspective artifacts.

Finally, the EAI perspective contains also a description of the behavior that it expects business partners to follow when performing e-business. That is also called the public interface of the company's IT systems. It is obvious that the public interface is closely connected to the integrated components as public interface and components have to interact to implement the business procedures of, for example, company 1. The public interface is represented as the box in the EAI perspective in Figure 5.1.

The hatched box in the upper part of Figure 5.1 denotes the level of e-business integration, which we refine to the community perspective and the e-business perspective:

- **Community perspective.** The community perspective is used during e-business integration time. The community perspective is the union of all potential integration participants' EAI perspectives looked at from the perspective of all participants as a community of potential collaborators. Companies participating in that perspective do not necessarily have to build a collaborative business process together. That perspective rather groups companies with the potential to interact. That perspective can also be seen as a market of available functionality.

If a company needs some services to engage in a new business process, they would search for the available functions on that market and start integration.

- **e-Business perspective.** The e-business perspective is created during e-business integration time. It is concerned with the particular integration goal that a subset of the potential integration participants in the community perspective share. The target of that perspective is to produce a concrete collaboration materializing in an executable orchestration of the collaborative business process.

  Furthermore, the orchestration generated as a result of e-business integration is also part of that perspective. Therefore, that perspective is also active at run time.

All perspectives and their relevance at certain times in the development lifecycle are displayed in Figure 5.2. The picture also includes the activities that we present in this thesis to support the different phases of software and integration development. The activities are explained in the following section.

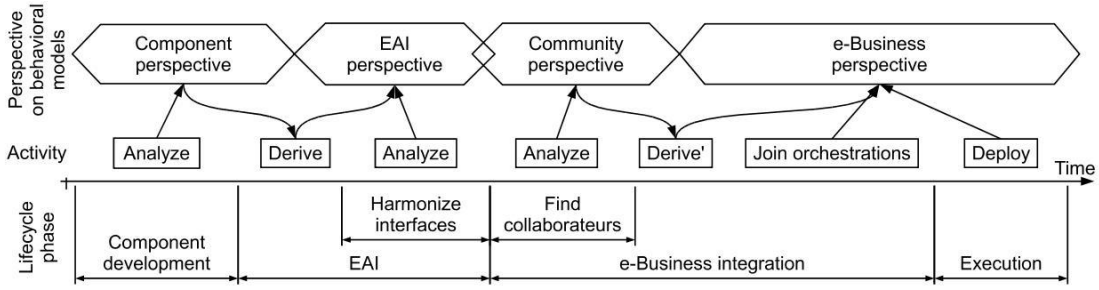


Figure 5.2: Activities and perspectives mapped on development lifecycle.

5.2 Activities

In our solution, three basic activities are performed on the perspectives introduced above. The activities are

- analyze,
- derive, and
- join orchestrations.

Each activity is supported by one of the solutions, or techniques, presented in this thesis. The mapping between activities and techniques is shown in Figure 5.3 on the facing page, where nesting boxes denote sub-activities.

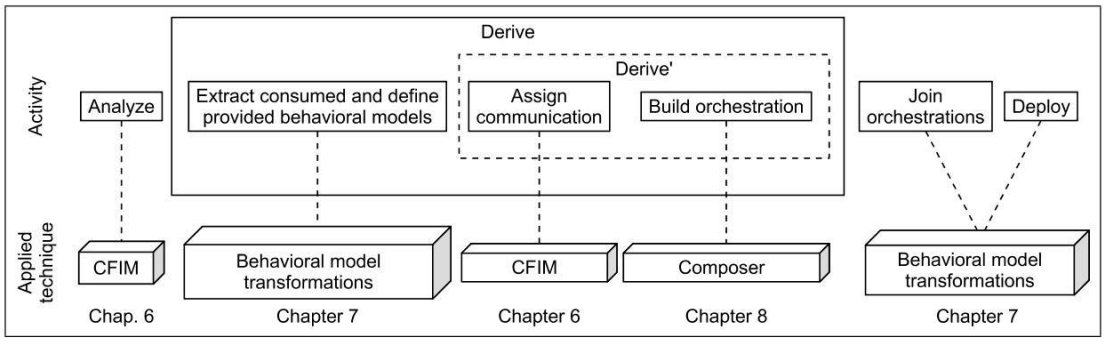


Figure 5.3: Overview of activities.

5.2.1 Analyze

The analyze activity can be performed on the data and message types and Web service definitions that appear in the behavioral models in the component perspective, EAI perspective, and e-business perspective. The analyze activity utilizes the CFIM of hierarchical types to reduce redundancy in the different perspectives. That activity will be explained in Chapter 6.

5.2.2 Derive

The derive activity is used to connect the behavioral models between different perspectives. The derive activity produces an orchestration that consumes the behavioral models of the lower perspective and provides a behavioral model to the superior perspective. The derive activity consists of the following three sub-activities:

1. **Extract consumed and define provided behavioral models.** The consumed behavioral models are extracted from behavioral models of the lower perspective. The provided behavioral model is to be defined by the human process designer. As these steps are part of transforming behavioral models for EAI and e-business, they will be explained in Chapter 7.

2. **Assign communication.** That activity supports the human process designer in identifying potential communication by again utilizing the CFIM of hierarchical types explained in Chapter 6.

3. **Build orchestration.** That activity involves the definition of a complex business process goal by a human integration expert and the automated creation of an orchestration considering consistent transactional compensation. That activity uses the complex-goal-based WS composition which will be explained in Chapter 8.

The first activity is only relevant when deriving the EAI perspective from the component perspective because only in that derivation step, the consumed behavioral model does not describe a single transaction. When deriving the e-business perspective from the

community perspective, the behavioral model of each EAI perspective in the community perspective constitutes a transaction with all or nothing semantics.

5.2.3 Join orchestrations

That activity is concerned with combining the orchestrations generated in the community perspective and the e-business perspective. The requirement to track behavioral information through software design phases is tackled by transforming behavioral models for EAI and e-business. That includes jointly executing the orchestrations to achieve model-based e-business interoperability relying on component Web service interfaces only, and not requiring a redundant implementation of the EAI perspective interfaces. Our approach to transform behavioral models for EAI and e-business is described in Chapter 7.

5.3 Schematic run-through

This section brings together perspectives and activities. That is done by walking through the different perspectives and by mapping the activities to the respective perspective according to our solution (see Figure 5.2 on page 58).

5.3.1 Component perspective

In the component perspective, the focus lies on one company and the software development activities inside the company. Heterogeneous definitions of the same

- data type,
- message type, or
- operation definition

complicate the reuse and maintenance of the artifacts.

The activity performed on the component perspective is to analyze the interface objects used inside one company in order to hint at redundancy. The analysis thus helps to reduce redundancy and therefore improves the decision support for reuse of interface objects and to lower maintenance costs as in the ideal case one functionality is implemented exactly once. That activity fulfills the requirement to detect redundant interface objects for the component perspective.

5.3.2 From component perspective to EAI perspective

Only a part of the interface objects in the component perspective are available to collaborators in realistic settings. Also, an enterprise may provide a set of business processes that base on the same component perspective interface objects. To address

the separation of component perspective and EAI perspective behavioral models, the behavioral model of the component perspective is used to derive a behavioral model in the EAI perspective. The derivation of an EAI perspective behavioral model from component perspective behavior may be expensive as it is concerned with correctly handling a business transaction including manual exception handling. Therefore, we use the complex-goal-based WS composition to fulfill the requirement to mechanically support exception handling for the transition from the component perspective to the EAI perspective. The derivation implicitly defines the set of interface objects that is exposed to partners.

5.3.3 EAI perspective and community perspective

The EAI perspective consists of the behavioral model and thus the interface objects of an enterprise exposed to integration partners. The community perspective is the union of all participants' EAI perspectives. Therefore, the same interface objects appear in the EAI perspective and the community perspective.

In our approach, the activity performed on the EAI perspective and community perspective is an analysis of the interface objects. The analysis of the interface objects pursues different goals for both perspectives.

Goals of analyzing the EAI perspective

Analyzing the EAI perspective means to compare interface objects in order to detect redundancy. Redundancy of exposed interfaces means increased maintenance effort, as in the event of a functionality change, all redundant interfaces offering the same functionality have to be adapted, or even multiple redundant implementations of the same functionality must be adapted correctly and consistently. The effects of redundancy were detailed in Section 3.1 on page 27.

Goals of analyzing the community perspective

Analyzing the community perspective means to increase collaboration potential for the members of the community perspective. The goals of the analysis are different for

- data and message types and
- operations.

If the data and message types of two or more members can be identified to be ontologically equivalent, the partners may work toward making the interface objects also technically equivalent to enable cooperation.

Identifying ontologically equivalent operations in the community perspective discovers competitors providing the same or similar functionality. If both competitors provide the same ontological functionality, each provider may choose to align their interfaces

in order to actually compete for the whole customer base. Despite their similarities, a distinction of the competitors still exists due to the potentially different behavioral models that guide their operations. Thus, the sequencing of operations required before or following the operation in question may be different.

In the case of similar but not equivalent functionality of two competitors, both providers may offer the same service with slight variations in some options of the service. In that case, the operation interfaces have to remain as they are. The analysis helps requesters in that case to identify providers with similar offerings.

The analysis activities discussed in this section address the requirement to detect redundant interface objects in the EAI perspective and community perspective.

5.3.4 From community perspective to e-business perspective

The e-business perspective is fed from the community perspective when establishing a collaborative business process. Members provide their EAI perspective behavioral models, which have to be combined with other members' behavioral models. The integration of the behavioral models must consider transactional properties because a collaborative business process establishes a business transaction.

With the derivation process, we fulfill the requirement to mechanically support exception handling by using the complex-goal-based WS composition for the community perspective.

5.3.5 e-Business perspective

The e-business perspective contains the orchestration of the EAI perspective behavioral models of the integration participants. As the connection between the behavioral models in the component perspective and the e-business perspective is kept, the execution of the collaborative business process can directly utilize the original Web service interfaces in the component perspectives of the participants. That means that the functionality to implement the Web service definitions of the EAI perspective does not need to be implemented as a whole. As the implementation usually consists to 80% of existing components' lifecycle information, which is covered in the component perspective behavioral models, an implementation would largely be redundant with the components' implementation. Through directly accessing the generated orchestration of the component perspective Web services in the orchestration in the e-business perspective, our approach avoids redundancy in implementation, which would otherwise lead to higher maintenance costs. Transforming behavioral models for EAI and e-business addresses the requirement to track behavioral information through software design phases.

The following three chapters detail each part of our solution. As sketched in this chapter, each part of the solution supports one of the activities shown in Figure 5.3 on page 59.

Chapter 6

CFIM of Hierarchical Types

In this chapter, we apply closed frequent itemset mining (CFIM) introduced in Section 4.1.5 on page 42 to hierarchical types, in particular data types and message types on the data integration layer, and Web service definitions on the API integration layer of the EAI pyramid (see Figure 2.2 on page 16). As discussed in the previous chapter, CFIM appears in many places in our solution. However, the core algorithm is always the same.

After a motivating example in Section 6.1, this chapter starts with utilizing CFIM of hierarchical types for exploring structural overlap of data type definitions in Section 6.2. Section 6.3 extends our approach to also mine message type and Web service definitions. Section 6.4 maps our solution for discovering structural overlap to exploring ontological similarities.

6.1 Motivating example

Table 6.1 shows a set of types as it could be used in a company to describe the structure of their data storage or messages exchanged.

Table 6.1: Sample types

Type	**Elements**
BuyerParty	ID, Phone, Address
Partner	ID, Phone, DateOfBirth, Location
Person	ID, Address, DateOfBirth

Each of the types has certain elements. On a first glance, one recognizes some overlap in the element sets of the different types. The task of CFIM is to identify these redundancies. Table 6.2 on the next page shows the redundancy, i. e., all sets of common elements of the given types.

Table 6.2: Redundancy groups determined from sample types

Redundancy group	Redundant types	Common elements
g_1	BuyerParty, Partner, Person	ID
g_2	BuyerParty, Partner	ID, Phone
g_3	BuyerParty, Person	ID, Address
g_4	Partner, Person	ID, DateOfBirth

Based on that analysis, one could improve the design of the types in a couple of ways: For example, one could build a new type containing the common elements of BuyerParty and Partner and only refer to that new type in BuyerParty and Partner. Or, one could find some redundancy by looking at the non-common elements of redundancy group members (see Figure 6.1). Thus, a human could find out that Address and Location are two different representations of spatial information. The types could then be redesigned such that address information will be equally represented in BuyerParty and Partner.

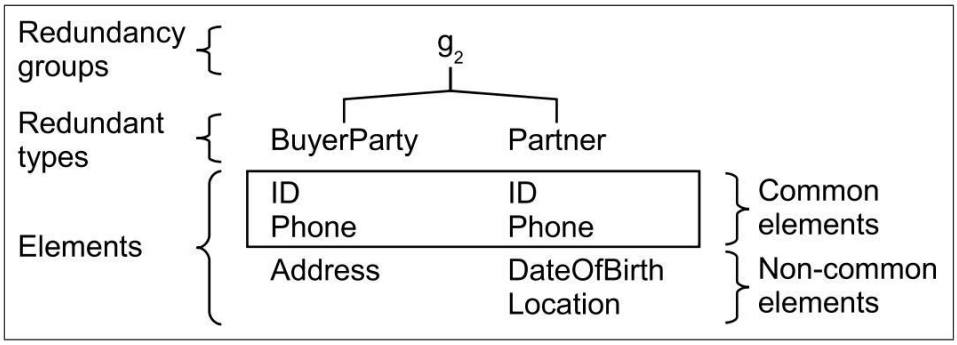


Figure 6.1: Detailed view of redundancy group g_2.

Besides containing some interesting redundancies, the members of the result set in Table 6.2 are not equally interesting. The degree of interest depends on domain-dependent facts and on generic facts: As an example for a generic fact, consider the first redundancy group which states that all types share a field ID. That is not as interesting as, for example, the redundancy group g_2 because it contains more common elements, however, shared by less types.

In a large result set of a CFIM run on real-world data there may be a lot of interesting, but also many less interesting redundancy groups. Therefore, as a service to the user, it is desirable to rank potentially interesting results—like the redundancy group g_2—higher than others—like the redundancy group g_1. For that purpose, we will provide a rank that considers three properties of redundancy groups later in this chapter.

6.2 Identifying structural similarities using the miner

In this section, we reduce the problem of detecting redundant interface objects to CFIM of hierarchical types. In particular, we adopt the LCM algorithm from Uno et al. (2004a;b). The problem reduction consists of two steps:

1. create itemsets from structural types and
2. interpret the mining result.

Before we start with the problem reduction, we give a definition of hierarchical types that we derive from the definition of XML schema. Although the problem reduction is applicable to hierarchical types in general, we take the XML schema definition again as an example in the subsequent sections. The adoption of the concepts presented in this section to further hierarchical types is shown in Section 6.3.

6.2.1 Definition of a hierarchical type

We now develop a conceptual definition of hierarchical types as an abstraction of XML schema, which was explained in Section 2.4.2 on page 21. We introduce the following abstractions:

1. As the import and include of namespaces are a mechanism for ease of use, but not conceptually necessary for our purposes, we abstract from namespaces. That is, we assume that all definitions reside in the same namespace.
2. It is a common practice for XML documents transferred in EAI settings to only specify sequences and neglect choices in the complex type definitions. The reason is that business software is commonly based on databases with fix database schemas. Therefore, we abstract from choices.
3. Finally, our approach does not consider differences of constraints and different orderings of the sequences of different schemas. Thus, we abstract from constraints and sequences as well.

The remaining conceptual structure consists of a hierarchy of elements of either simple or complex type having names. We formalize that as follows:

Definition 6.1 (Hierarchical type schema)**.** Let CT be a set of complex types, ST a set of simple types, N a set of names, Δ a relation of types and elements, where the type is said to directly contain the element, then a hierarchical type schema $\mathcal{H}$ is a tuple defined as follows:

$$
\begin{aligned}
&\mathcal{H} := \ \mathrm{CT}, \mathrm{ST}, \mathrm{N}, \Delta \\
&\mathrm{T} \ = \ \mathrm{CT} \quad \mathrm{ST} \ \text{(hierarchical types)}, \quad \mathrm{CT} \setminus \mathrm{ST} = \varnothing \\
&\mathrm{E} \ = \ \mathrm{N} \times \mathrm{T} \ \text{(elements)} \\
&\Delta\text{:} \quad \mathrm{T} \ \times \mathrm{E}
\end{aligned}
$$

Definition 6.2 (Sub-element)**.** Let $\mathcal{H} = \ \mathrm{CT}, \mathrm{ST}, \mathrm{N}, \Delta \ $ be a hierarchical type schema. An element $c = (n_c, t_c)$ is called a direct sub-element or direct child of an element $e = (n_e, t_e)$ with respect to $\mathcal{H}$, iff

$$(t_e, (n_c, t_c)) \quad \Delta \quad (n_e, n_c \quad \mathrm{N},\ t_e \quad \mathrm{CT},\ t_c \quad \mathrm{CT} \quad \mathrm{ST}).$$

6.2.2 Create itemsets from structural types

For creating itemsets from structural types, the transformation mainly consists in reverting the natural "top-down" representation of hierarchical types to the inverse "bottom-up" representation of the vertical itemset mining format required by LCM. We start with the basic transformation and handle further aspects thereafter.

Basic transformation

The vertical data structure for a structured XSD element e containing the child elements $c_1, \ldots, c_i, \ldots, c_n$, where $1 < i < n$ (see left-hand side of Figure 6.2 on the next page), is

$$(c_1^{\mathrm{I}},\ e^{\mathrm{T}}\ ), \ldots, (c_i^{\mathrm{I}},\ e^{\mathrm{T}}\ ), \ldots, (c_n^{\mathrm{I}},\ e^{\mathrm{T}}\ ),$$

illustrated in the form of a table,

c_1^{I}	$\ldots$	c_i^{I}	$\ldots$	c_n^{I}
e^{T}	$\ldots$	e^{T}	$\ldots$	e^{T}

,

where e^{T} denotes the *transaction* that corresponds to element e and c^{I} denotes the *item* that corresponds to element c. If another XSD element f becomes added to the same vertical data structure with the child elements $c_i, \ldots, c_n, \ldots, c_m$, where $n < m$ (see right-hand side of Figure 6.2), the data structure is

$$\begin{aligned}(c_1^{\mathrm{I}},\ e^{\mathrm{T}}\ ), \ldots, (c_{i-1}^{\mathrm{I}},\ e^{\mathrm{T}}\ ),& \\ (c_i^{\mathrm{I}},\ e^{\mathrm{T}}, f^{\mathrm{T}}\ ), \ldots, (c_n^{\mathrm{I}},\ e^{\mathrm{T}}, f^{\mathrm{T}}\ ),& \\ (c_{n+1}^{\mathrm{I}},\ f^{\mathrm{T}}\ ), \ldots, (c_m^{\mathrm{I}},\ f^{\mathrm{T}}\ ).&\end{aligned}$$

In the form of a table:

c_1^{I}	$\ldots$	c_{i-1}^{I}	c_i^{I}	$\ldots$	c_n^{I}	c_{n+1}^{I}	$\ldots$	c_m^{I}
e^{T}	$\ldots$	e^{T}	$e^{\mathrm{T}}, f^{\mathrm{T}}$	$\ldots$	$e^{\mathrm{T}}, f^{\mathrm{T}}$	e^{T}	$\ldots$	e^{T}

.

In addition to the basic transformation presented, further aspects need to be considered:

- mapping between elements, items, and transactions,
- domain of analysis, and
- analysis range.

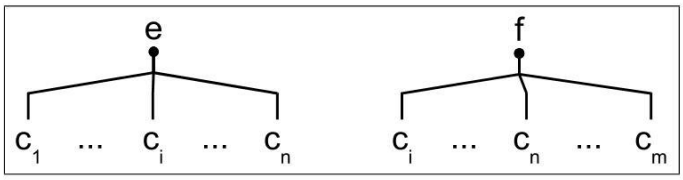


Figure 6.2: Sample abstract hierarchical data structures.

Mapping between elements, items, and transactions

For the basic transformation presented above, we have neglected the question of how exactly an element relates to an item or a transaction. That question is important because elements have both a name and a type. Let n_x be the name, for example, "TaxableAmount," and t_x the type of an element e_x, for example, "AmountType." There are obviously three ways to map a complex element e_x to mining items or transactions. If e_x appears as item, the item may be defined as

1. the name: $e_x^{\mathrm{I}} = n_x$, for example, "TaxableAmount,"
2. the type: $e_x^{\mathrm{I}} = t_x$, for example, "AmountType," or
3. a combination of name and type: $e_x^{\mathrm{I}} = (n_x, t_x)$, for example, "TaxableAmount-AmountType,"

of the corresponding element. Likewise, if e_x appears as transaction, the transaction may be defined as the name, the type, or a combination of both. In our solution, we only use the element's type if an element is mapped to a transaction. The reason is that the transaction is not subject to analysis, but the transaction needs to unambiguously identify a set of items. Only the type—and not the name—of a complex element of hierarchical data structures identifies the set of its child elements unambiguously.

To give an example for the relevant mapping alternatives, a complex XML schema element e named n of type t may contain the sub-elements $c_1, c_2, \ldots, c_i$, where c_x is named n_x and of complex type t_x, for all $x = 1, \ldots, i$. The corresponding vertical data structures would be

$$
\begin{aligned}
(n_1,\ t\ ), (n_2,\ t\ ), \ldots, (n_i,\ t\ ) &\quad \text{for alternative 1,} \\
(t_1,\ t\ ), (t_2,\ t\ ), \ldots, (t_i,\ t\ ) &\quad \text{for alternative 2, and} \\
((n_1, t_1),\ t\ ), ((n_2, t_2),\ t\ ), \ldots, ((n_i, t_i),\ t\ ) &\quad \text{for alternative 3.}
\end{aligned}
$$

The choice above determines when the miner would treat two elements appearing as items to be equal. The most precise estimation is surely performed by alternative 3 because the most information is used for the determination of equality. Therefore, alternative 3 might be a good choice in an already very much aligned repository. Contrarily, alternative 1 may even find matches in not so much aligned repositories as just names, which may be given arbitrarily by a human, are considered for matching.

The precision of alternative 2 lies in between the ones of alternative 1 and 3 because elements of the same type are equal per definition. However, they may be used in different contexts which may be differentiated by the elements' names, which is not considered in alternative 2.

Analysis range

In the paragraphs above, we have explained how a single complex type may be transformed to the vertical data structure. One more decision needs to be taken about the set of types to be analyzed—the analysis range. The alternatives are whether to consider

1. only the complex top-level elements of the given XML schemas, or

2. the top-level plus all contained complex elements.

The rationale for the first option is to only compare the given types. The rationale for the second is that it may detect similar complex types that were used to build up the given XML schemas. We call the second option for that reason **building block analysis**.

Let's consider the complex type s consisting of the complex elements e_1 and e_2, where e_1 and e_2 are complex, and e_1 consists of c_1 and c_2, and e_2 consists of c_3 and c_4. Let's further consider that another complex type t consists of c_1, c_3, and c_4. Both s and t are displayed in Figure 6.3.

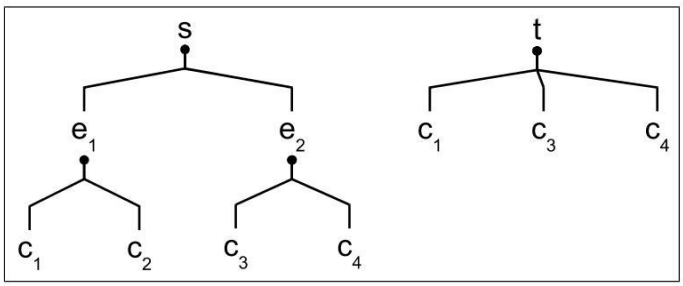


Figure 6.3: Further sample abstract hierarchical data structures.

Building the vertical data structure for option 1 considers the structures circled in Figure 6.4 on the next page.

The vertical data structure for option 1 is

$$(e_1^{\mathrm{I}},\ s^{\mathrm{T}}\ ),(e_2^{\mathrm{I}},\ s^{\mathrm{T}}\ ),(c_1^{\mathrm{I}},\ t^{\mathrm{T}}\ ),(c_3^{\mathrm{I}},\ t^{\mathrm{T}}\ ),(c_4^{\mathrm{I}},\ t^{\mathrm{T}}\ ), \tag{6.1}$$

and in tabular form

e_1^{I}	e_2^{I}	c_1^{I}	c_3^{I}	c_4^{I}
s^{T}	s^{T}	t^{T}	t^{T}	t^{T}

.

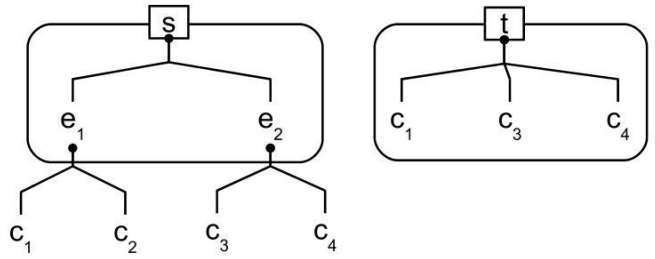


Figure 6.4: Structures considered without building block analysis.

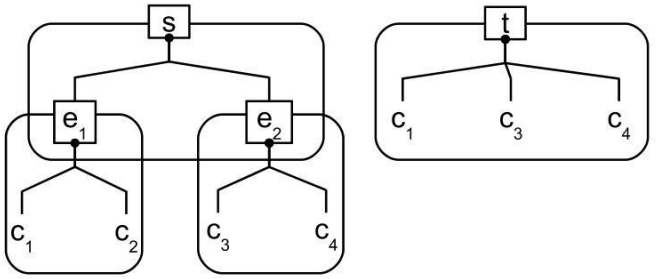


Figure 6.5: Structures considered for building block analysis.

Since no two pairs contain s and t together, the miner will not discover that related complex types were used to construct the both—as long as e_1, e_2, c_1, c_3, and c_4 have different types and names. In contrast, building the vertical data structure for option 2 considers the structures circled in Figure 6.5.

The vertical data structure for option 2 is

$$(e_1^{\mathrm{I}},\ s^{\mathrm{T}}\ ), (e_2^{\mathrm{I}},\ s^{\mathrm{T}}\ ), (c_1^{\mathrm{I}},\ e_1^{\mathrm{T}}, t^{\mathrm{T}}\ ), (c_2^{\mathrm{I}},\ e_1^{\mathrm{T}}\ ), (c_3^{\mathrm{I}},\ e_2^{\mathrm{T}}, t^{\mathrm{T}}\ ), (c_4^{\mathrm{I}},\ e_2^{\mathrm{T}}, t^{\mathrm{T}}\ ),$$

and in tabular form

e_1^{I}	e_2^{I}	c_1^{I}	c_2^{I}	c_3^{I}	c_4^{I}
s^{T}	s^{T}	$e_1^{\mathrm{T}}, t^{\mathrm{T}}$	e_1^{T}	$e_2^{\mathrm{T}}, t^{\mathrm{T}}$	$e_2^{\mathrm{T}}, t^{\mathrm{T}}$

.

In that case, the miner will discover that both e_2 and t share c_3 and c_4 and thus are related. The miner will also conclude that e_1 and t are related as they share c_1.

Domain of analysis

Another variation of transforming multiple complex element definitions to the vertical data structure is the completeness their substructure is being considered—the domain of the analysis. The two options are to consider

1. just the directly contained subelements, and
2. all directly and indirectly contained subelements.

The rationale for the first option is to perform a faster mining as less detail is being considered. That may be useful when the types to analyze are already strongly aligned. The rationale for the second is that it may detect complex type definitions with different granularity. We therefore call the second option **granularity-agnostic analysis**.

Let's consider the same example as before. The vertical data structure for option 1 is shown in (6.1) above. In contrast, building the structure for option 2—without building block analysis—considers the structures circled in Figure 6.6.

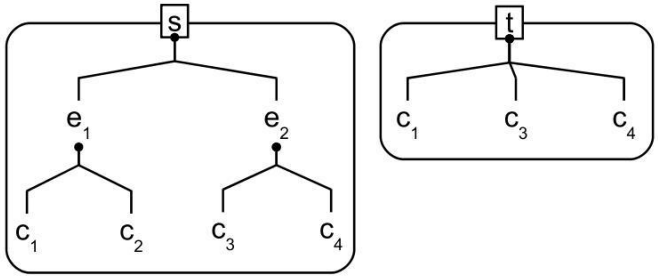


Figure 6.6: Structures considered for granularity-agnostic analysis.

The data structure for option 2 is

$$(e_1^{\mathrm{I}},\ s^{\mathrm{T}}),(e_2^{\mathrm{I}},\ s^{\mathrm{T}}),(c_1^{\mathrm{I}},\ s^{\mathrm{T}},t^{\mathrm{T}}),(c_2^{\mathrm{I}},\ s^{\mathrm{T}}),(c_3^{\mathrm{I}},\ s^{\mathrm{T}},t^{\mathrm{T}}),(c_4^{\mathrm{I}},\ s^{\mathrm{T}},t^{\mathrm{T}}),$$

and in tabular form

e_1^{I}	e_2^{I}	c_1^{I}	c_2^{I}	c_3^{I}	c_4^{I}
s^{T}	s^{T}	$s^{\mathrm{T}},t^{\mathrm{T}}$	s^{T}	$s^{\mathrm{T}},t^{\mathrm{T}}$	$s^{\mathrm{T}},t^{\mathrm{T}}$

.

In that case, the miner would determine that schema s and t are similar as they share c_1, c_3, and c_4.

The granularity-agnostic analysis is especially useful when different partners used different approaches to structure the same kind of data. For example, one partner may have chosen to create a type "PurchaseOrderType" that contains ordered goods, taxable and total gross amounts, contact and delivery addresses without any substructure directly below the top-level element (see left-hand side of Figure 6.7 on the facing page). Another partner may have chosen to create a similar type "OrderType" with the same data, but grouped the data into the complex subelements "Goods," "Amounts," and "Addresses" (right-hand side of Figure 6.7). Using granularity-agnostic analysis, "PurchaseOrderType" and "OrderType" can be determined to be similar.

6.2.3 Interpret the mining result

The result of the mining is a set of redundancy groups. Each redundancy group consists of a set of types that have some overlap. After mining, the computer has to present the redundancy groups to the user. As there are more interesting and less interesting results, we introduce a generic rank that gives a measure for the confidence that can be derived from the overlap that a specific redundancy group contains redundant types.

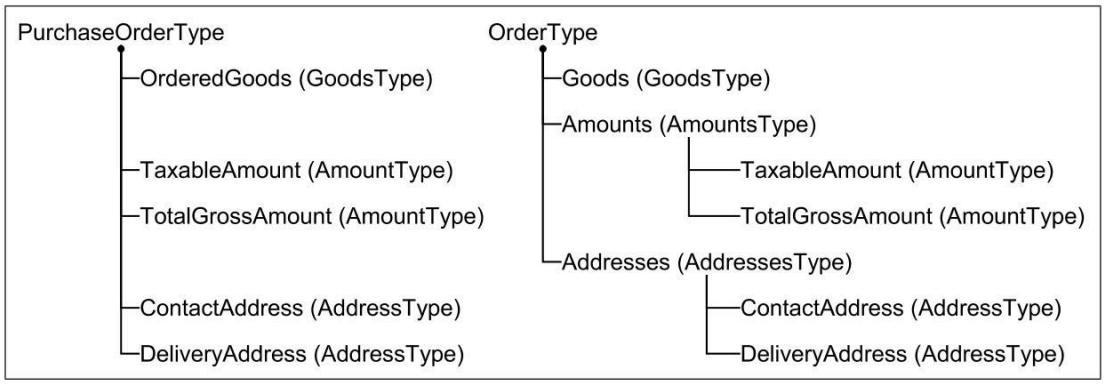


Figure 6.7: Advantage of granularity-agnostic analysis.

Rank

We now develop a rank that takes into account properties of a redundancy group that make it likely to contain redundant elements. Throughout this section, we use the notation of transactions and items introduced in Section 4.1.5 on page 42.

It follows from the definition of redundancy that we are looking for a redundancy group that at the same time is large in the number of types $\tau = T(A)$ as well as in the number of common elements $\kappa = A$. Let's now consider two redundancy groups g_1 and g_2 with equal number of types and common elements $\tau_{g_1} = 2, \kappa_{g_1} = 3$, and $\tau_{g_2} = 2, \kappa_{g_2} = 3$. Considering these two measures, both groups would have the same rank. However, if every type of g_1 contains more elements than every type of g_2, we would like to rank g_2 higher because its ratio of potentially redundant to potentially unique elements is higher. That situation is depicted in Figure 6.8 on the following page. For comparing redundancy groups' sizes, we look at the average number of elements of all types of $T(A)$, denoted by $\alpha = \overline{T(A)}$. Putting κ in relation to α yields the ratio of common elements to the average number of elements $\upsilon = \frac{\kappa}{\alpha}$.

Definition 6.3 (Interesting redundancy groups)**.** We define a set of redundancy groups $\mathrm{G}_1 = g_{1,1}, \ldots, g_{1,n_1}$ to be a more interesting redundancy group than $\mathrm{G}_2 = g_{2,1}, \ldots, g_{2,n_2}$ if

1. G_1 is larger than G_2, $\tau_1 > \tau_2$
2. G_1's types have absolutely more common constituents, $\kappa_1 > \kappa_2$; and
3. G_1's redundancy groups contain a higher average percentage of common constituents, $\upsilon_1 > \upsilon_2$.

We calculate a redundancy group's rank ρ as a combination of the three components τ, κ, and υ. To be comparable, we normalize each component by its maximum value among all redundancy groups, respectively denoted by $\hat{\tau}$, $\hat{\kappa}$, and $\hat{\upsilon}$. It follows that each component $\frac{\tau}{\hat{\tau}}$, $\frac{\kappa}{\hat{\kappa}}$, and $\frac{\upsilon}{\hat{\upsilon}}$ is in range $[0, 1]$. For combining the components, we use the

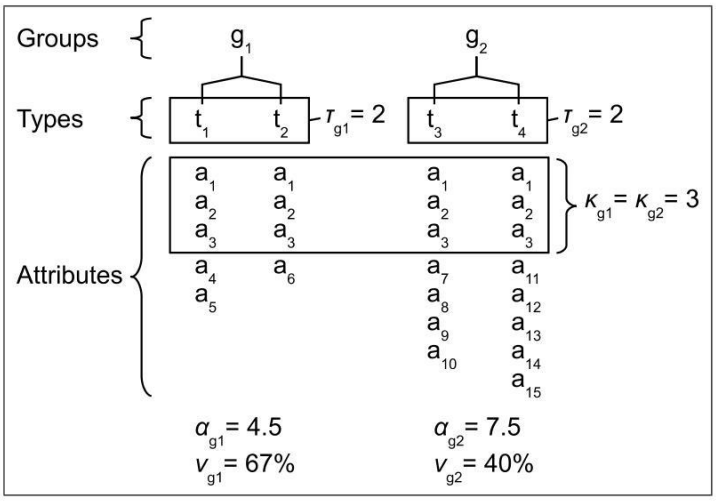


Figure 6.8: Example for rank calculation.

product operation which produces a higher rank the more the components' values are balanced as opposed to the sum operation. Let's consider the two redundancy groups g_1 with $\frac{\tau}{\hat{\tau}} = \frac{10}{10}, \frac{\kappa}{\hat{\kappa}} = \frac{2}{10}, \frac{\upsilon}{\hat{\upsilon}} = \frac{3}{10}$, and g_2 with $\frac{\tau}{\hat{\tau}} = \frac{5}{10}, \frac{\kappa}{\hat{\kappa}} = \frac{5}{10}, \frac{\upsilon}{\hat{\upsilon}} = \frac{5}{10}$. Both would be treated the same using the sum operation: $\frac{10+2+3}{10} = \frac{5+5+5}{10} = \frac{15}{10}$. However, g_1 is a rather large group with a relatively and absolutely rather small number of common elements. On the other hand, g_2 is not as large as g_1, but has a relatively and absolutely larger core than g_1 which makes it more likely to contain redundancy. Combining the components of the rank via a product, we boost g_2 over g_1, as its properties are more balanced: $\frac{10}{10}\ \frac{2}{10}\ \frac{3}{10} = \frac{60}{1000} < \frac{5}{10}\ \frac{5}{10}\ \frac{5}{10} = \frac{125}{1000}$. The interrelation between the balance of two components and the resulting product is depicted in Figure 6.9 on the next page. After scaling the product of the components by factor σ, rank ρ is in range $[0, \sigma]$.

$$\rho = \sigma\left(\frac{\tau}{\tau}\ \frac{\kappa}{\kappa}\ \frac{\upsilon}{\upsilon}\right)$$

Theorem 6.4 (Correctness of rank)**.** *The formula for ρ produces a larger rank for a redundancy group G_1 than for G_2 if G_1 is more interesting than G_2 as de ned in De nition 6.3.*

Proof. We must prove $\sigma\left(\frac{\tau_1}{\hat{\tau}}\ \frac{\kappa_1}{\hat{\kappa}}\ \frac{\upsilon_1}{\hat{\upsilon}}\right) > \sigma\left(\frac{\tau_2}{\hat{\tau}}\ \frac{\kappa_2}{\hat{\kappa}}\ \frac{\upsilon_2}{\hat{\upsilon}}\right)$, which is equivalent to $\frac{\tau_1}{\tau_2}\ \frac{\kappa_1}{\kappa_2}\ \frac{\upsilon_1}{\upsilon_2} > 1$. That is true when $\tau_1 > \tau_2$, $\kappa_1 > \kappa_2$, and $\upsilon_1 > \upsilon_2$, which is the definition of a more interesting redundancy group. □

Determining the redundancy

In principle, redundancy can be determined by analyzing the

- common, and the

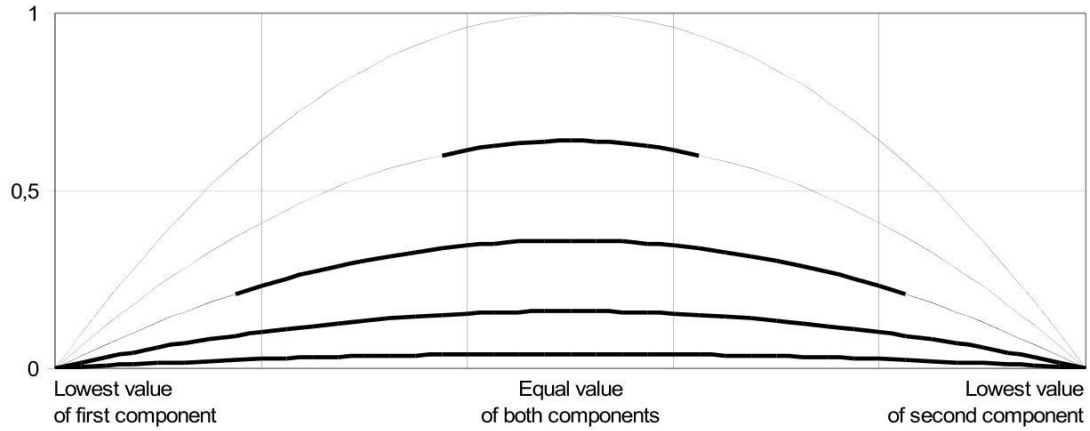


Figure 6.9: Product of two factors with constant sum.

- uncommon items in a redundancy group.

Obviously, the common elements identified by the miner point out some definite redundancy of the elements of the redundancy group.

But also the uncommon elements of a redundancy group provide an interesting insight. Let's assume that two business partners are willing to agree on common data types to exchange. In that setting, both business partners may contribute some kind of a business partner type to the community. If these partners worked together before, some of the data types deeper in the nesting will already be aligned. Consequently, the miner will be able to determine that match based on the common, already aligned elements. As, in that setting, both partners really talk about the same kind of business partner type, the fields not known for the miner to be common may indeed be just different representations of the same things. Therefore, it might be interesting for the human integration experts to also look at the uncommon elements as these may contain matching pairs.

6.2.4 Discussion

In order for the algorithm to find redundancy groups, we assume that the data types fed to the miner are aligned to a certain extent already. That means that two conceptually equivalent data types also carry the same label. That assumption can be made because, when acting in the realm of a single company, at least some basic data types will be consistent with a common model that other data types are built upon. In SAP, for example, data type labels conform to the naming conventions of the core component technical specification (CCTS). That might be a difficult assumption when dealing with multiple companies in a community. The best thing our solution can contribute here is to support the process of aligning the technical structures of multiple companies by hinting at definitions that have been manually aligned to a certain extent, but need further alignment work. Such a situation is actually likely to be found as communities

form in order to perform similar tasks, for example, shippers and carriers will probably talk about location, package types, and dates. If some members of a community already worked together, they might even have aligned their types with existing standards for communication, such as RosettaNet or CIDX.[1] However, a well-known difficulty with such industry standards is the high degree of freedom the standards allow for. That means that although two partners implement, for example, RosettaNet, they may still have different custom extensions of the standard. In that very common situation, the algorithm is expected to perform well due to the existing alignment.

6.3 Application on further hierarchical types

The mining described above can not only be applied to XML schema, but any other hierarchical data structure, in particular WSDL's

- message type definitions and
- Web service definitions.

In order to make closed frequent itemset mining applicable to message type and Web service definitions, it suffices to explain how message type and Web service definitions are hierarchical types.

6.3.1 Message type definitions

Conceptually, a message type definition in WSDL consists of a name and a set of parts. Each part consists of a name and an XML schema element. Therefore, the application of closed frequent itemset mining to message type definitions is quite straightforward. The name of the message type itself is used to identify a transaction. Each of the following becomes an item of the transaction:

1. each part's name,
2. each part's element name, and
3. all XML schema elements, which are handled as described for XML data types above.

6.3.2 Web service definitions

Conceptually, a Web service interface contains a set of operations. Each operation can communicate at most one input, one output, and a fault message. Each message type consists of a set of parts as explained above. In general, there are two options to apply closed frequent itemset mining on Web service definitions. One could look for

[1] `http://www.cidx.org/`

1. Web service *operations* having redundant *signatures* or
2. Web service *de nitions* having redundant *operations*.

Web service operations with redundant signatures

The analysis of Web service operations having redundant signatures is relevant because actually not Web services, but Web service operations are the entities to be orchestrated in business process integration. For applying closed frequent itemset mining to Web service operations, the name of each Web service operation becomes a transaction. The items of a transaction are made up of

1. the name of each message potentially communicated by the operation
2. the constituents of a message as explained above.

Web service definitions with redundant operations

Web service definitions are used to structure operations of related functionality together. Therefore, it is reasonable to mine for redundant Web service definitions as they could hint on redundant functionality bundles exposed to partners.

For mining Web service definitions, the name of a Web service definition becomes a transaction. The following makes up the items of a transaction:

1. the name of each operation in the Web service definition and
2. the constituents of an operation, which are handled as described above.

6.4 Using CFIM for ontological alignment

Although ontological alignment brings many advantages, it is very rarely reached in computer science. In contrast, we repeatedly observe certain situations of lacking alignment in integration scenarios. In the following sections, common situations of misalignment are systematically presented based on the concept of the semiotic triangle (Ogden and Richards, 1923).

6.4.1 Semiotic triangle

At least part of the reason for ambiguity is that things of common interest materialize as different objects (for example, as t, u) and are referred to via some terms, or denotations (such as l, m). That facilitates misunderstanding of the objects, or materializations, because there is actually no direct connection between a term and an object as the research fields of linguistics and semiotics have revealed. Rather, both are connected via a concept (such as c) that the term stimulates in the imagination of a human peer,

or in the implementation of a computer. These relations, shown on the left-hand side of Figure 6.10, are called the semiotic triangle. Whereas a term can be used to refer to multiple types of objects and an object may be referred to by multiple terms, a concept is always unique.

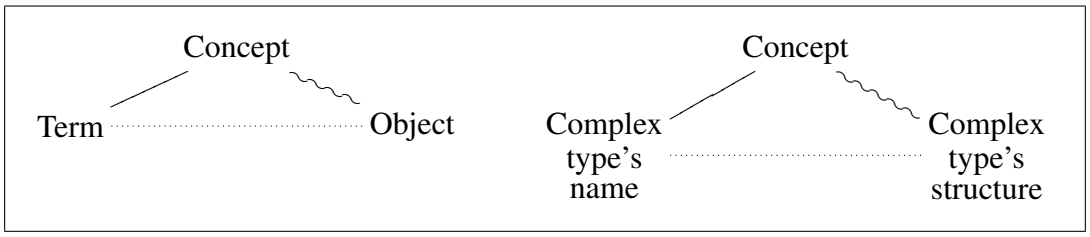
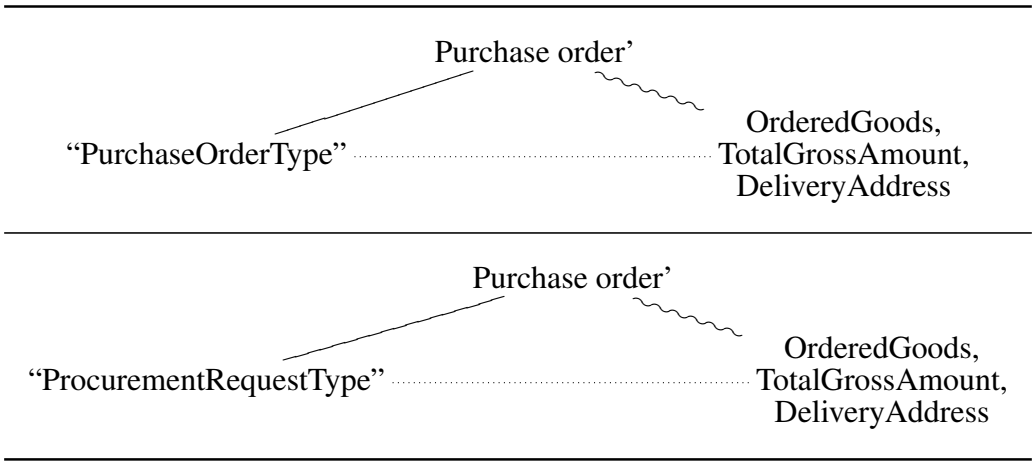


Figure 6.10: The semiotic triangle and its application to complex types.

In this thesis, we use the semiotic triangle to investigate the theoretically possible reasons for the misalignment of complex types (see right-hand side of Figure 6.10). For that purpose, we compare the semiotic triangles of two different types with each other. The potential interrelations between objects, terms, and concepts of two types are depicted in Figure 6.11 on the next page. In order to cope with the misalignment, we are going to separately address all these cases. But before, we give a systematic overview over the situations of lacking alignment. During the rest of this chapter, we use single quotes (. . . ') to refer to concrete concepts and double quotes (". . . ") to refer to concrete terms.

Redundancy / synonym vs. parallel

A known type t once called l may be referred to as m at another time or in another place. That may be an indication that l and m are just two synonymous denotations for the same concept (6.11c), such as "purchase order" and "procurement request."



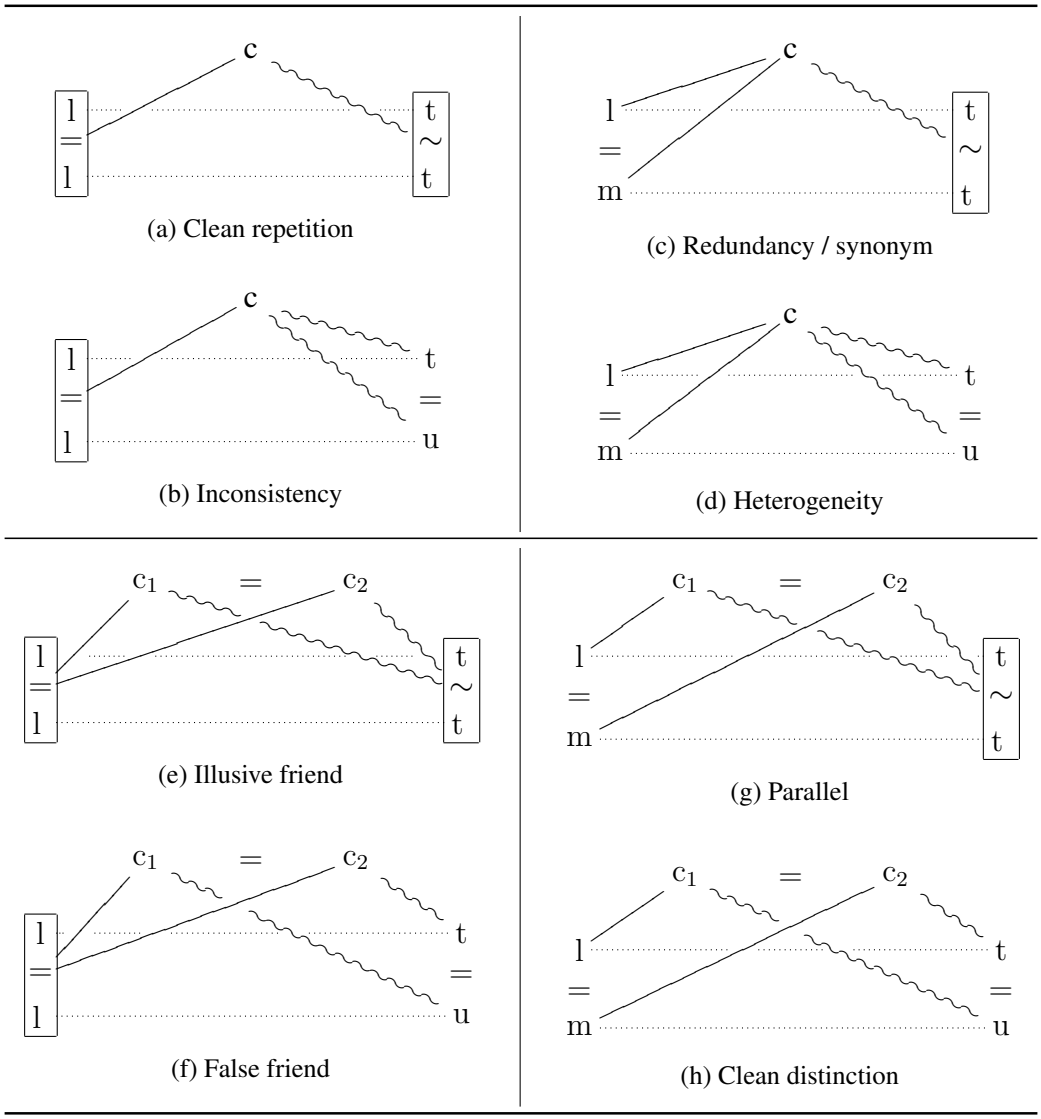
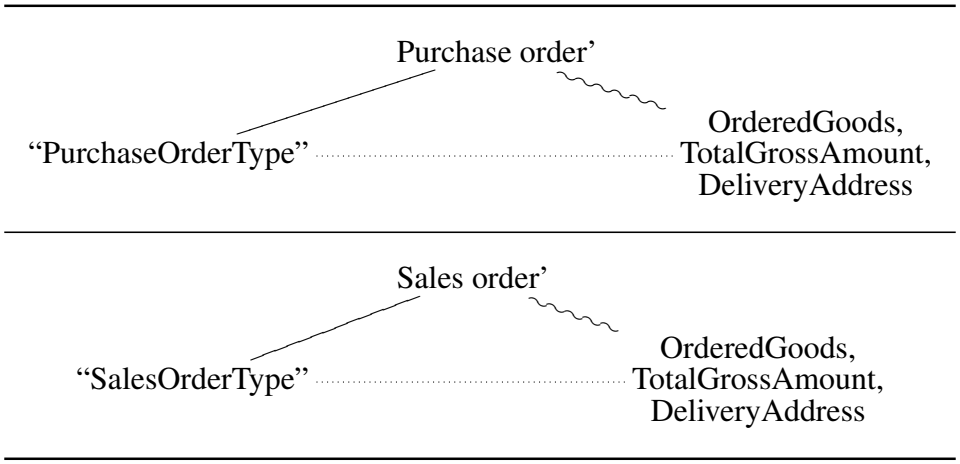


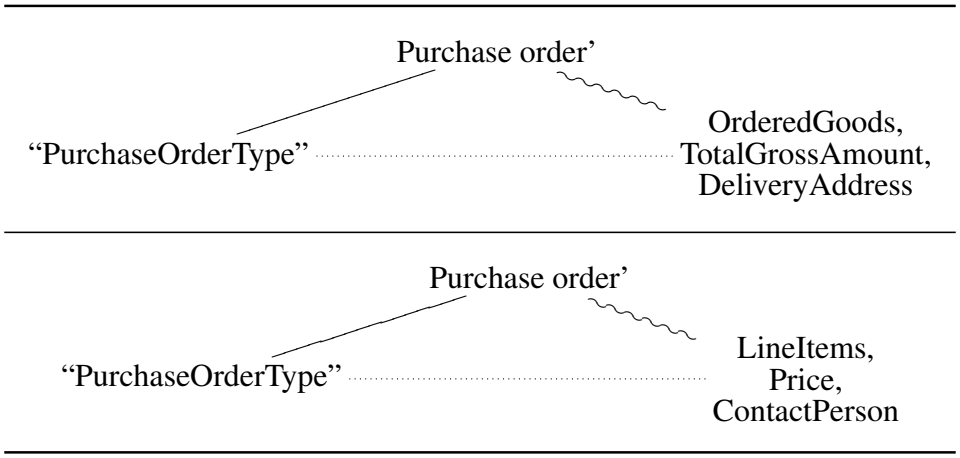
Figure 6.11: Potential semiotic triangles for two repetitive observations.

Vice versa, l and m may indeed denote disjoint concepts and just happen to be materialized by a type t similar to t (6.11g), such as ‹purchase order› and ‹sales order.› As a sales order is the seller's view on the buyer's purchase order, both types are likely to have a similar structure in current systems.



Inconsistency vs. false friend

We could furthermore come across two situations where the term l is used the first time to address a type t and a second time l , equivalent to l, refers to a type u. Again, it might be the case that t and u are two different materializations of l in different contexts (6.11b) as for two different representations of purchase order.'



Conversely, that case might be an indication that l is inconsistently used: once to refer to t and another time to refer to u (6.11f).

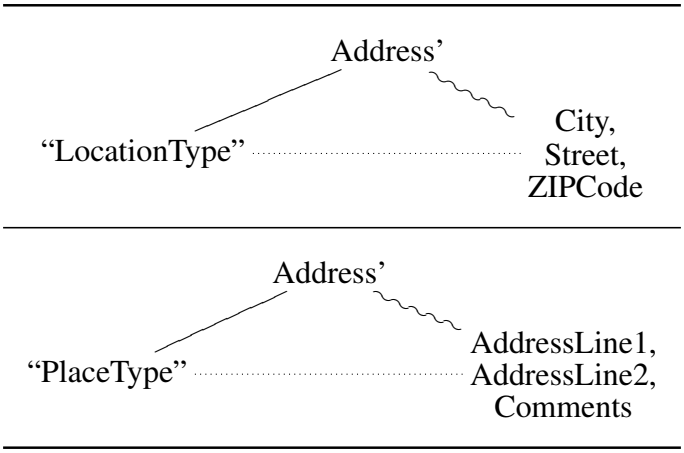
Clean repetition vs. illusive friend

Finally, we observe a very cumbersome situation when a type t is once called l and at another time or in another place a type t , similar to t, is also called l , equal to l. That might of course indicate that t was referred to as l with the same background as when

t was called l (6.11a). However, it might also be the case that in the first place term l was used to reference t, but the second time just the same term l was used to refer to an equally appearing type t which indeed might be totally different from t (6.11e) in its use or in its meaning in the second context.

Clean distinction vs. heterogeneity

We may also observe the opposite to the previous when in the first place l is used to refer to t, and in the second place u is called m. On the one hand, that might indicate that t and u are completely different and thus the different denotations l and m are used to refer to them (6.11h). On the other hand, it might also be the case that l and m are just different terms for the same thing, which, unfortunately, is heterogeneously materialized at the two distinct observations as t and u (6.11d), such as "LocationType" and "PlaceType" could both stand for 'address.'



6.4.2 Identifying ontological similarities using CFIM

In the realm of closed frequent itemset mining,

- a transaction is a denotation and
- the items of a transaction are a materialization.

When comparing transactions in the mining database, a transaction and its items can be understood as an observation of a denotation and a materialization in terms of the semiotic triangle. Therefore, the cases depicted in Figure 6.11 on page 79 can be applied to every two transactions in the mining database. The different cases of misalignment are all handled by the way closed frequent itemset mining is applied on hierarchical types

- during populating the mining database and
- during performing the mining.

Populating the mining database

When types are going to be analyzed, they are first loaded to the mining database. Various options during that process were described in Section 6.2.2 on page 68. Now we focus on cases of existing alignment or misalignment.

During population, more and more transactions are loaded to the mining database. A case of existing alignment or misalignment is observed when a transaction to be loaded is already contained in the mining repository. Therefore, the cases with similar denotations, i. e.,

- clean repetition (6.11a),
- inconsistency (6.11b),
- illusive friend (6.11e), and
- false friend (6.11f),

can be addressed during the population of the mining database (left-hand side of Figure 6.11 on page 79).

To determine one of the above cases, we only need to compare the name of the new transaction to the names of existing transactions. As the transactions in the mining database are per definition organized as a set, the addition of an existing transaction is actually a violation to the definition of the closed frequent itemset mining.

For the resolution of these cases, it is important to differentiate the cases depending on whether the two transactions with similar denotations, i. e., complex types with same names, stem from

- the same or
- different partners.

Same partner. In the case that the conflicting types stem from the same partner, we assume that one partner always uses the same denotation (term) to refer to the same concept (upper left part of Figure 6.11 on page 79). In that case, only one version should be kept in the data structure because either

1. the structure of both types is equal (clean repetition, 6.11a), or
2. an inconsistency in the partner's own data model was discovered (inconsistency, 6.11b).

In the first case, it is irrelevant which type is added to the structure. In the second case, the partner should first resolve their inconsistency internally, before trying to align with other partners.

Different partners. In the case the conflicting types s and s stem from different partners, we have to assume that same denotations (terms) are potentially used for different things (lower left part of Figure 6.11 on page 79). Both types should be added to the structure as the mining procedure should determine whether both schemas are really already aligned or actually different. Therefore, we transform the case

- illusive friend (6.11e) to parallel (6.11g) and
- false friend (6.11f) to clean distinction (6.11h).

In other words, the equal transaction representations s^{T} and s^{T} must be made unequal before being added to the mining database. We propose to add a unique distinguisher to s^{T} and to perform the transformation to the item representation afterwards as described in the previous sections.

It is to be noted that the procedure only applies to conflicting transactions, not to their constituents with equal item representation as equal item representations are used to determine the similarity of the transactions.

After populating the mining database, it only contains the cases of the right-hand side of Figure 6.11 on page 79, i. e.,

- redundancy / synonym (6.11c),
- heterogeneity (6.11d),
- parallel (6.11g), and
- clean distinction (6.11h).

Performing the mining

The closed frequent itemset mining determines the similarity of materializations in the mining database. A redundancy group stands for a set of mutually similar types. According to Figure 6.11 on page 79, two observations may either

1. belong to the same concept, i. e., have the same ontological meaning (redundancy / synonym, 6.11c), or
2. belong to distinct concepts, i. e., the similar structures are used to materialize ontologically different things (parallel, 6.11g).

The distinction must clearly be made by a human utilizing the closed frequent itemset mining.

Same concepts. In the first case, it is advisable that the differently named, but structurally similar types become named the same. In addition, it should be a goal to not only homogenize the names, but also make both types use the same complex type definition. As the miner detects technical overlap, the decision is not necessarily correct for the other members of the redundancy group. In order to support the homogenization, further closed frequent itemset mining runs can be applied on only the members of the redundancy group. That serves two purposes:

First, further detection of ontologically equivalent types that appear as subtypes in the elements of the redundancy group.

Second, if no more types are ontologically equivalent, technically equivalent types can be identified.

The first case would facilitate the alignment of terms and thus bring forward a common understanding. The second case would foster the alignment of type structures and therefore lower efforts for later maintenance.

Distinct concepts. In the case that two observations belong to different concepts, the analysis revealed that although being ontologically distinct, the materializations have some commonality. By further closed frequent itemset mining runs on the members of the redundancy group, it should be determined, whether more similar subtypes of the members can be identified whose definitions can be merged.

The remaining cases of misalignment in Figure 6.11 on page 79, namely

1. heterogeneity (6.11d) and
2. clean distinction (6.11h),

are not detected by the closed frequent itemset mining. For the second case, that is perfectly fine as a clean distinction (6.11h) actually contains no redundancy. Thus, the only case which can not correctly be handled is heterogeneity (6.11d). The reason is that there is no information at all which a machine could use in order to draw any conclusions on the potential similarity in that case. That is the spot where always human intelligence is needed in alignment. Due to that finding, a solution using closed frequent itemset mining to detect redundant interface objects should always allow a human in the loop to manually point out similarity. We facilitate that in our solution as the closed frequent itemset mining only supports the human analysis of the type repository. Our solution seeks to point out the most that could possibly be done automatically and always leaves room for human adaptation.

Chapter 7

Transforming Behavioral Models for EAI and e-Business

In Section 3.4.2 on page 36, an ideal solution to the business process integration problem was described as a directly executable orchestration that circumvents the integration design time model and directly utilizes the executable component design time model. In this section, we describe how our solution can integrate component design time and integration design time models and, based on that, run the executable orchestration of the e-business perspective using the executable models in the component perspective.

As described in Section 5.2 on page 58 and displayed in Figure 7.1 on the following page, the relevant activities to track behavioral information through software design phases are the following:

1. **Extract consumed and define provided behavioral models** is a sub-activity of the derive activity when derive is applied on the component perspective. That activity consists of the following three steps:

   (a) **Define consumer view of provided behavioral model.** The behavioral model that should be provided on the superior perspective needs to be manually defined as a requirement for building an orchestration. The behavioral model must follow the definition for behavioral models given later in this chapter.

   (b) **Convert provided behavioral model to provider view.** The provided behavioral model is initially modeled from the consumer view in step (a). However, for building an orchestration, the provider view is needed.

   (c) **Excerpt consumed behavioral model fragment.** That activity is used to specify the part of a consumed behavioral model that is used in the derived model.

2. **Join orchestrations**.

This chapter is structured as follows: First, we establish the connection between Web service definitions of the API integration layer with behavioral models of the BPI

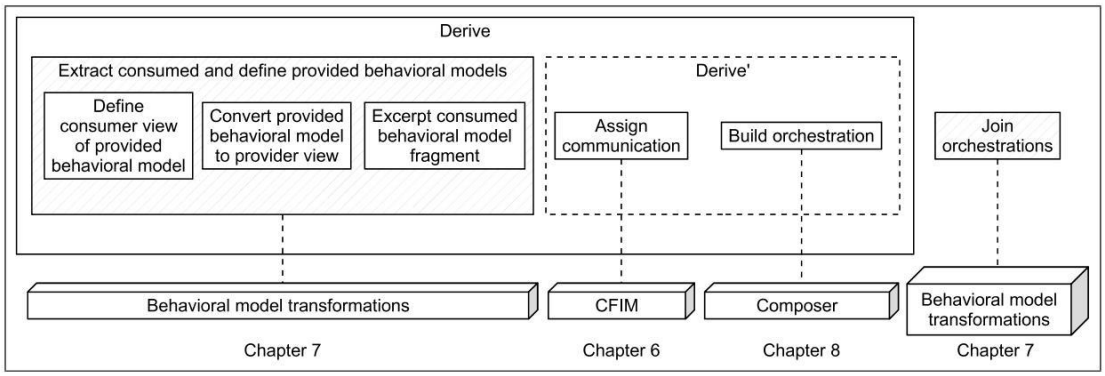


Figure 7.1: Detailed view of activities transforming behavioral models.

layer in Section 7.1. In Section 7.2, we formally introduce behavioral models. That is the basis for step (a) above. In Section 7.3, we formally define our requirement to track behavioral information through software design phases. The following three sections detail the remaining steps and activities. Namely, Section 7.4 tackles the activity to convert a provided behavioral model to the provider view (step b). Section 7.5 presents an algorithm to excerpt a consumed behavioral model fragment (step c) and Section 7.6 explains how orchestrations are joined in our solution (activity 2 above). Section 7.7 names the conditions under which the identified requirement can be fulfilled. We conclude in Section 7.8.

7.1 Web service descriptions and behavioral models

We introduce a behavioral model as an abstraction of Web service operations. As introduced in Section 2.5.1 on page 22, a Web service operation consists of at most

- one input,
- one output, and
- one fault message.

However, the Web service definition does not mention the intent of the Web service provider about the process-relevant semantics an invocation of a Web service operation may have. For example, a purchase order notification message may contain a whole purchase order document for reference plus a code identifying the acceptance status of the purchase order. In that case, only the acceptance status contains the data relevant for the subsequent flow. In addition, no assertion is made about sequencing of operations that may be required to perform successfully. That information is today normally given as natural language documentation.

In our behavioral models, we allow a Web service provider to specify their intention regarding potential process flows as a set of potential outcomes of an operation. In

addition, a pre-state is assigned to an operation and a posterior state to each operation outcome.

In our solution, a behavioral model (BM) is described by SA-WSDL. We use the semantic annotation hooks—the `modelReference` attributes—in SA-WSDL to add the state information to the operations. As SA-WSDL allows for the same operation types as WSDL, multiple outcomes of an operation can also not directly be specified in SA-WSDL. We overcome that limitation be redefining the interpretation of SA-WSDL in the following way:

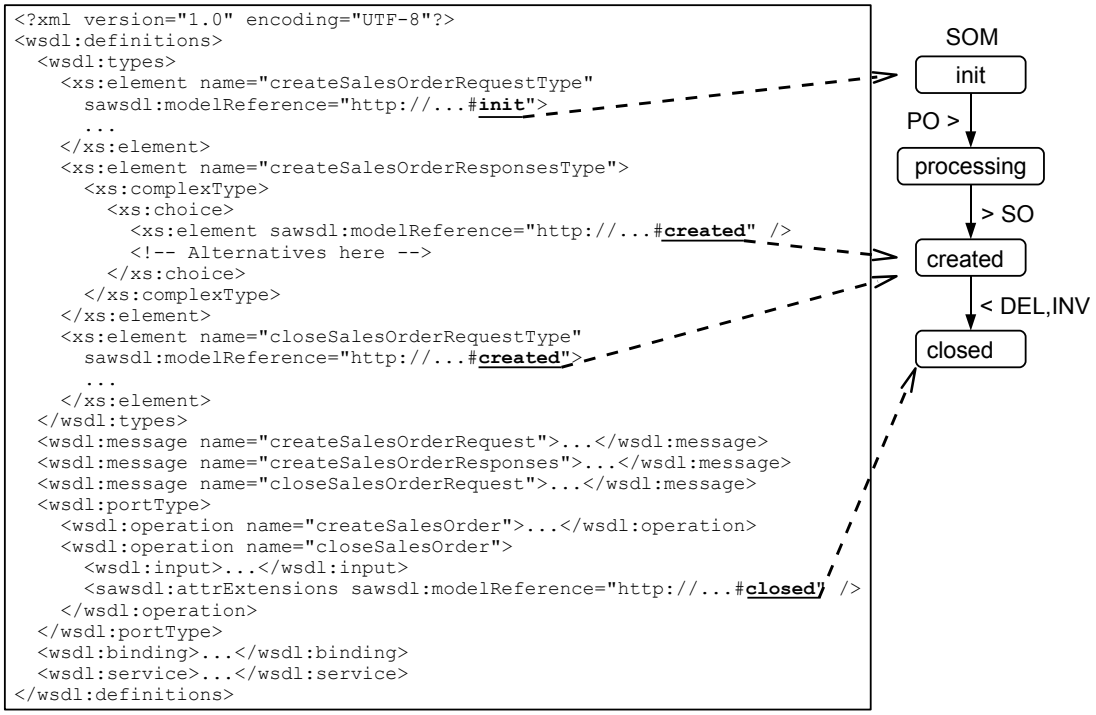
- An operation's output must consist of a special complex type—the container type—which we use to group and attach alternative outcomes to an operation.
- A container type contains one choice particle. Each element of the choice is annotated with the posterior state of the respective outcome.
- The type of a choice's element defines the message type that is communicated by the operation for the particular outcome.

It should be noted that an SA-WSDL file does not *refer* to an external state transition system (STS). Rather, the SA-WSDL *contains* the state transition system.

The use of SA-WSDL to define operation semantics is practical because it can be used in all places where WSDL can be used, for example, in the definition of business processes using contemporary business process definition languages such as BPEL or BPMN.

In Figure 7.2 on the following page, we introduce a shorter graphical representation for an SA-WSDL file containing a state transition system. We use here for illustration parts of the example which will be given in detail in Chapter 10. The text on top of the graphical representation is used to name the component the behavioral model belongs to. Each box in the graphical representation is a state and each arrow is a state transition. We model a request-response operation via two transitions: the first for receiving the input—for example, from state init to processing in Figure 7.2—and the second for sending the output—for example, the transition from processing to created in the figure. The newly introduced state—"processing" in the example—is only implicitly defined in SA-WSDL. Although these states do not exist in SA-WSDL, we name them in the graphical representation to ease the reading of graphical behavioral models. Text next to an arrow denotes the message type communicated by the state transition. The communication mode may be either input or output. Input is denoted by a smaller or greater than relation symbol with the closed end pointing toward the arrow. Output is denoted the opposite way. In the rest of this thesis, we use the graphical representation of SA-WSDL and their contained behavioral models for brevity.

In our solution, the SA-WSDL does not only play the role of capturing process-relevant semantics of Web service operations. In addition, an SA-WSDL interface is




```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
  <wsdl:types>
    <xs:element name="createSalesOrderRequestType"
      sawsdl:modelReference="http://...#init">
      ...
    </xs:element>
    <xs:element name="createSalesOrderResponsesType">
      <xs:complexType>
        <xs:choice>
          <xs:element sawsdl:modelReference="http://...#created" />
          <!-- Alternatives here -->
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="closeSalesOrderRequestType"
      sawsdl:modelReference="http://...#created">
      ...
    </xs:element>
  </wsdl:types>
  <wsdl:message name="createSalesOrderRequest">...</wsdl:message>
  <wsdl:message name="createSalesOrderResponses">...</wsdl:message>
  <wsdl:message name="closeSalesOrderRequest">...</wsdl:message>
  <wsdl:portType>
    <wsdl:operation name="createSalesOrder">...</wsdl:operation>
    <wsdl:operation name="closeSalesOrder">
      <wsdl:input>...</wsdl:input>
      <sawsdl:attrExtensions sawsdl:modelReference="http://...#closed" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding>...</wsdl:binding>
  <wsdl:service>...</wsdl:service>
</wsdl:definitions>
```

Figure 7.2: A graphical notation for SA-WSDL containing a behavioral model.

expected to be implemented in a specific way: We assume that the implementation makes use of the WSDL interface that is originally provided for the described software component. We call the implementation an adapter. The connection between Web service descriptions in the API integration layer and behavioral models in the BPI layer is depicted in Figure 7.3.

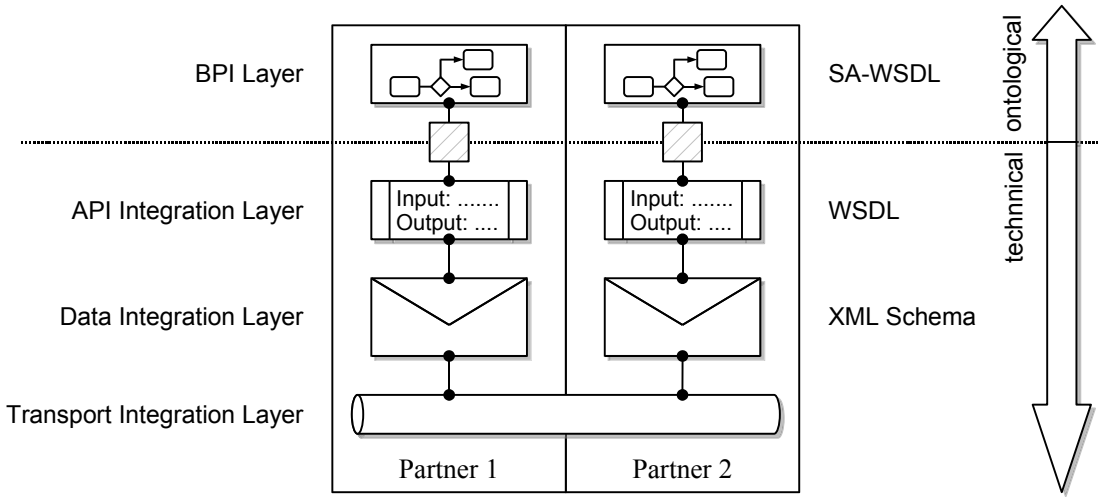


Figure 7.3: Integrating technical and ontological level.

7.2 Behavioral models

In order to be able to identify a single behavioral model in a set of behavioral models, we introduce a set of identifiers (ID).

Definition 7.1 (Behavioral model identifiers)**.** Each identifier from the set ID identifies exactly one behavioral model.

$$\mathrm{ID} \ \ldots \text{ set of unique identifiers for behavioral models}$$

We define a behavioral model via a set of states (Q), a set of possible input and output messages, referred to as input and output variables (Σ and Z), and a state transition function (δ). The definition includes the behavioral information via the state transition function. We restrict δ to describe a bipartite graph. Bipartite means that input and output transitions alternate. An exemplary behavioral model is shown on the right-hand side of Figure 7.2 on the facing page.

Definition 7.2 (Behavioral model)**.**

$$\begin{aligned}
\mathcal{BM} :=\ & \mathrm{Q}, \Sigma, \mathrm{Z}, \delta, q_0 \\
q_0 \quad & \mathrm{Q} \\
\delta \ = \ & \delta_{\mathrm{in}} \quad \delta_{\mathrm{out}} \\
\delta_{\mathrm{in}} \ : \ & \mathrm{Q} \times (2^{\Sigma} \quad \varnothing \ ) \quad \mathrm{Q} \\
\delta_{\mathrm{out}} \ : \ & \mathrm{Q} \times (2^{\mathrm{Z}} \quad \varnothing \ ) \quad \mathrm{Q}
\end{aligned}$$

We use the superscript notations Q^{bm}, Σ^{bm}, Z^{bm}, q_0^{bm}, δ^{bm}, $\delta_{\mathrm{in}}^{bm}$, and $\delta_{\mathrm{out}}^{bm}$ to refer to the components of the behavioral model identified by $bm \quad \mathrm{ID}$ when reference otherwise would be ambiguous.

To give an example, we express the behavioral model from Figure 7.2 using the notation above (compare with the behavioral model on the right-hand side of Figure 7.2 on the preceding page):

$$\begin{aligned}
\mathrm{Q}^{\mathsf{SOM}} &= \mathsf{init}, \mathsf{processing}, \mathsf{created}, \mathsf{closed} \\
\Sigma^{\mathsf{SOM}} &= \mathsf{PO}, \mathsf{DEL}, \mathsf{INV} \\
\mathrm{Z}^{\mathsf{SOM}} &= \mathsf{SO}
\end{aligned}$$

$\delta_{\mathrm{in}}^{\mathsf{SOM}} =$

q_1	$\mathrm{Q}^{\mathsf{SOM}}$	inputs	q_2	$\mathrm{Q}^{\mathsf{SOM}}$
init		PO	processing	
created		DEL, INV	closed	

$\delta_{\mathrm{out}}^{\mathsf{SOM}} =$

q_1	$\mathrm{Q}^{\mathsf{SOM}}$	outputs	q_2	$\mathrm{Q}^{\mathsf{SOM}}$
processing		SO	created	

$$q_0^{\mathsf{SOM}} = \mathsf{init}$$

We refer to a state with at least one leaving output transition or with at least one entering input transition as operation. Therefore, processing and closed would be the two states that represent operations in the example.

Multiple input transitions leaving the same state stand for the capability of the system to receive any of the inputs. Thus, the decision how to proceed the execution is determined by the communication partner of the system. Therefore, we talk of *deterministic* behavior.

Contrarily, multiple output transitions leaving the same state enumerate the set of possible responses of the system at a specific time of execution. Thus, the decision how to proceed the execution is determined by the system, and not by the communication partner of the system. Therefore, we talk in that respect of *non-deterministic* behavior.

Please note the difference of that intuition from the interpretation of a state transition system as a finite state machine.

We explicitly refuse business logic encoded into behavioral models because that is an implementation detail which should only appear in the implementation of a component, but not in its behavioral model. That means that the behavioral model does not contain the information how the implementation decides which of the output alternatives will be given for a specific input. In related workflow specification languages like BPEL or BPMN, explicit conditions can be specified that examine data contained in messages. In our approach, we explicitly forbid the examination of messages because that would make the behavioral model dependent on the complete message type. Instead, we only allow the behavioral model to refer to an abstraction of the data transported in a message by allowing the explicit output alternatives defined in SA-WSDL to be differentiated.

As behavioral models do not contain business logic, messages whose content influences subsequent behavioral models must be classified into different variables. We are hence only interested in a variable's status rather than its value and introduce the $\mathrm{varState}$ function.

Definition 7.3 (Variable status)**.**

$$\begin{array}{rrl} \mathrm{varState}: & (\,bm, v\,) & status: \; bm \quad \mathrm{ID}, v \quad (\Sigma^{bm} \quad \mathrm{Z}^{bm}), \\ & status & \mathrm{undef}, \mathrm{initialized}, \mathrm{processed} \end{array}$$

Before we define an execution semantics for behavioral models expressed as described above, we introduce the abstract state machine (ASM) formalism, which we will utilize for defining the execution semantics.

7.2.1 ASM: A software system specification approach

Abstract state machines (ASM Börger and Stärk, 2003) is a formalism for the specification of software systems and a method for software system design. We chose ASM to

express the operational semantics of behavioral models because of the following two features:

1. The ASM notation is intuitive because it reads like pseudo-code over abstract data.

2. An ASM specification is theoretically executable due to its formal semantics. Concrete implementations exist to simulate ASM specifications—for example, CoreASM (Farahbod et al., 2007).

The main concept of an ASM is the transition rule of the form **if** <condition> **then** <updates> which transforms abstract states. The condition is an arbitrary predicate logic formula without free variables whose interpretation evaluates to true or false. The updates are a set of assignments of the form $f(p_1, \ldots, p_n) := v$ that take place concurrently. An update is to be understood as the definition of the value of the function f with the parameters $p_1, \ldots, p_n$ to v. $f(p_1, \ldots, p_n)$ is also called location. The set of all locations makes up the abstract state of an ASM. The condition of a transition rule checks for properties of the current state of the ASM. A rule is called active if its condition evaluates to true. The run of an ASM is defined as the parallel firing of all active rules. Firing means a state transition where the following state is equivalent to the current state after applying the update assignments of the active rules.

There are two additional constructs to group rules of common form. The first states the concurrent execution of multiple rules: **forall** <x> **with** <ψ> **do** <R>, where usually x has unbound occurrences in R. The second states the non-deterministic execution of one rule out of R: **choose** <x> **with** <ψ> **do** <R>.

The complete, formal execution semantics of ASM may be found in Börger and Stärk (2003). Due to the execution semantics, the readable ASM pseudo-code may be used for formal software specification as it allows proving interesting properties of the specified software system. The ASM method supports rigorous software design as its formal grounding allows for the formal definition of refinement as detailed in Börger (2003).

7.2.2 Behavioral model execution semantics

We now define the execution of behavioral models presented in Section 7.2 on page 89 in terms of ASMs. The involved information artifacts of the structural and execution world and their relations are depicted in Figure 7.4 on the next page.

A Web service interface is abstracted by the ASMs SEND and RECEIVE. The contained ASMs INVOKEWS and WSRESPONSE access the adapter that connects the ontological definitions with the technical WSDL interface and thus with the component's implementation.

The definition of the SEND machine is parameterized by the ID of a behavioral model. We use the set notation to express that the SEND machine of a behavioral model consists of one **if-then** rule per input transition of the behavioral model.

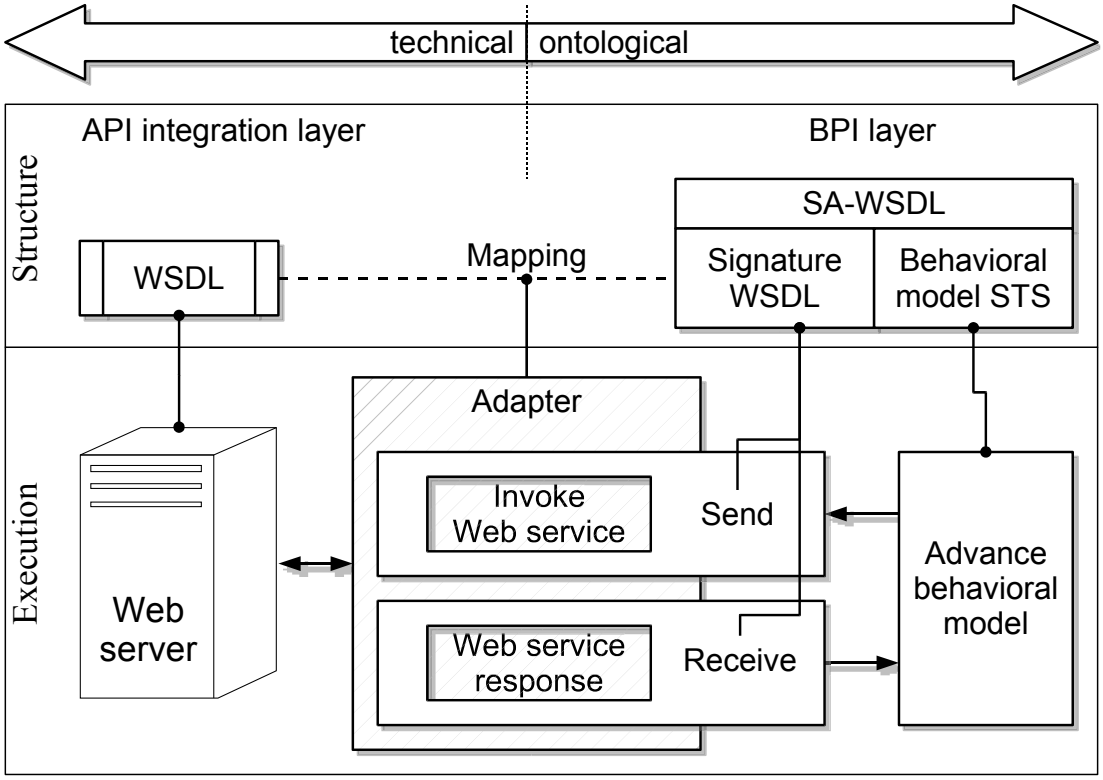


Figure 7.4: Structure and execution of the behavioral model.

$\textsc{Send}(bm) \equiv$

if $\mathrm{bmState}(bm) = q_{\mathrm{post}}$
and $\mathrm{varState}(bm, i_1) = \mathrm{initialized}$ **and** ...
and $\mathrm{varState}(bm, i_I\ ) = \mathrm{initialized}$
then
$\textsc{InvokeWS}(bm, I)$
forall $i \quad I$ **do** $\mathrm{varState}(bm, i) := \mathrm{processed}$

:

$(\ q_{\mathrm{pre}},\ I,\ q_{\mathrm{post}}\ ) \quad \delta_{\mathrm{in}}^{bm},$
$i_x \quad I, \quad x = 1\ ..\ I$

To give an example, we explicate $\textsc{Send}(\mathsf{SOM})$ below.

$\textsc{Send}(\mathsf{SOM}) \equiv$
if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{processing}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{PO}) = \mathrm{initialized}$
then
$\textsc{InvokeWS}(\mathsf{SOM},\ \mathsf{PO})$

forall i $\quad$ PO $\quad$ **do** $\mathrm{varState}(\mathsf{SOM}, i) := \mathrm{processed}$

if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{closed}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{DEL}) = \mathrm{initialized}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{INV}) = \mathrm{initialized}$
then
INVOKEWS(SOM, DEL, INV)
forall i $\quad$ DEL, INV $\quad$ **do** $\mathrm{varState}(\mathsf{SOM}, i) := \mathrm{processed}$

The activation of a rule of the SEND machine depends on the current state of the execution of the behavioral model—namely, after an input transition—and on the availability of the necessary input data. The availability of input data is handled by the orchestration, which we will explain later. Once the input data for the current state is available, SEND fires and calls the component's implementation through the adapter via INVOKEWS(SOM, ...). At the same time, SEND marks the input data sent to the component as processed in order to not fire over and over again.

In contrast, the RECEIVE machine handles the response of the component's implementation.

RECEIVE(bm) $\equiv$

if $\mathrm{bmState}(bm) = q_{\mathrm{pre}}$
and $\mathrm{varState}(bm, o_1) = \mathrm{initialized}$ **and** ...
and $\mathrm{varState}(bm, o_O) = \mathrm{initialized}$
and WSRESPONSE(bm, O)
then
forall o $\quad O$ **do** $\mathrm{varState}(bm, o) := \mathrm{initialized}$
:
$(q_{\mathrm{pre}}, O, q_{\mathrm{post}})$ $\quad \delta^{bm}_{\mathrm{out}}$
$o_x \quad O, \quad x = 1 \,..\, O$

RECEIVE fires when the execution of the behavioral model is before an output transition and when the component's implementation has responded, which is recognized via WSRESPONSE($bm, \ldots$) $= true$. After firing a rule in RECEIVE, the data received from the component are initialized. That means, they are from now on available to other components in the orchestration. We also list RECEIVE(SOM) here as an example.

RECEIVE(SOM) $\equiv$
if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{processing}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{SO}) = \mathrm{initialized}$
and WSRESPONSE(SOM, SO)
then
forall o $\quad$ SO $\quad$ **do** $\mathrm{varState}(\mathsf{SOM}, o) := \mathrm{initialized}$

As we have seen, SEND and RECEIVE expect the state of the behavioral model to be either after an input or before an output transition. SEND and RECEIVE only change the state of variables, but not the state of the behavioral model. That is done by the ADVANCEBM machine.

ADVANCEBM(bm) $\equiv$

if $\mathrm{bmState}(bm) = q_{\mathrm{pre}}$
and $\mathrm{varState}(bm, v_1) = \mathrm{initialized}$ **and** ...
and $\mathrm{varState}(bm, v_V) = \mathrm{initialized}$
then
$\mathrm{bmState}(bm) := q_{\mathrm{post}}$

:

$(q_{\mathrm{pre}}, V, q_{\mathrm{post}}) \quad \delta^{bm}$,
$v_x \quad V, \quad x = 1 \,..\, V$

ADVANCEBM advances the state of a behavioral model according to the availability of data—either made available by the orchestration or a component's response—and to the state transition function of the behavioral model. The concrete machine ADVANCEBM(SOM) is given below as an example.

ADVANCEBM(SOM) $\equiv$
if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{init}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{PO}) = \mathrm{initialized}$
then
$\mathrm{bmState}(\mathsf{SOM}) := \mathsf{processing}$

if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{processing}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{SO}) = \mathrm{initialized}$
then
$\mathrm{bmState}(\mathsf{SOM}) := \mathsf{created}$

if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{created}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{DEL}) = \mathrm{initialized}$
and $\mathrm{varState}(\mathsf{SOM}, \mathsf{INV}) = \mathrm{initialized}$
then
$\mathrm{bmState}(\mathsf{SOM}) := \mathsf{closed}$

We illustrate the interplay of SEND, RECEIVE, and ADVANCEBM below by presenting their activity sequence when executing the behavioral model of SOM.

ASM	bmState SOM	varState PO	varState SO	varState DEL	varState INV
(orchestration)	init	initialized	(unset)	(unset)	(unset)
ADVANCEBM	processing	initialized	(unset)	(unset)	(unset)
SEND	processing	processed	(unset)	(unset)	(unset)
RECEIVE	processing	processed	initialized	(unset)	(unset)
ADVANCEBM	created	processed	initialized	(unset)	(unset)
(orchestration)	created	processed	processed	initialized	initialized
ADVANCEBM	closed	processed	processed	initialized	initialized
SEND	closed	processed	processed	processed	processed

The SOM behavioral model is triggered when it is in its initial state and the PO has been made available by the orchestration. That is recognized by the ADVANCEBM machine which advances SOM's state. In the new state, processing, the SEND machine becomes active as the execution is behind an input transition and the data has not yet been sent to the component. After that is done, the RECEIVE machine awaits an answer from the component and marks the variables representing the received data as available, or initialized. ADVANCEBM recognizes the completion of the communication with the component and advances the behavioral model's state to created. Now, the control is again at the orchestration, which will be focus of the following section. When the orchestration is ready to perform SOM's final operation, it initializes the variables DEL and INV. That is again recognized by ADVANCEBM, which advances SOM's state to closed. Finally, the SEND machine recognizes that there is data to be forwarded to the component, which is finished by marking the variables DEL and INV as processed.

7.2.3 Communication execution semantics

In the previous section, we introduced a mathematical definition of behavioral models. As we finally target to describe orchestrations of multiple behavioral models, we describe in this section, how we represent communication between different behavioral models. The definition of communication will be the basis for the definition of an orchestration.

Consequently, we now have to refer to behavioral models and their variables in a global manner. Thus, we need to add a unique identification to behavioral models and variables. Therefore, we introduce two sets that uniquely identify states and variables of specific behavioral models.

Definition 7.4 (Global states and variables)**.**

$$\begin{aligned} \mathcal{B} &:= \quad bm \quad bmState : bm \quad \mathrm{ID}, bmState \quad \mathrm{Q}^{bm} \\ \mathrm{Variable} &:= \quad (bm, v) : bm \quad \mathrm{ID}, v \quad (\Sigma^{bm} \quad \mathrm{Z}^{bm}) \end{aligned}$$

We would now, for example, refer to the states and variables of SOM as

$$\begin{aligned} &(\mathsf{SOM}, \mathsf{init}), (\mathsf{SOM}, \mathsf{processing}), (\mathsf{SOM}, \mathsf{created}), (\mathsf{SOM}, \mathsf{closed}) \quad \mathcal{B} \\ &\qquad (\mathsf{SOM}, \mathsf{PO}), (\mathsf{SOM}, \mathsf{SO}), (\mathsf{SOM}, \mathsf{DEL}), (\mathsf{SOM}, \mathsf{INV}) \quad \mathrm{Variable} \end{aligned}$$

Correspondingly, we define a possible communication as a tuple of a globally unique output variable of one behavioral model and a globally unique input variable of another behavioral model.

Definition 7.5 (Communication)**.**

$$\begin{aligned} \mathcal{K} &:= \quad ((bm_1, o), (bm_2, i)) : bm_1, bm_2 \quad \mathrm{ID}, \\ &\qquad bm_1 = bm_2, o \quad \mathrm{Z}^{bm_1}, i \quad \Sigma^{bm_2} \end{aligned}$$

When we consider the three behavioral models in Figure 7.5, we can write down the following potential communications.

$$\begin{aligned} &1 :((\mathsf{SOM}, \mathsf{SO}), (\mathsf{PROD}, \mathsf{SO})), \qquad 4 :((\mathsf{PROD}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV})), \\ &2 :((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{FIN}, \mathsf{DEL})), \qquad 5 :((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{SOM}, \mathsf{DEL})), \\ &3 :((\mathsf{PROD}, \mathsf{INV}), (\mathsf{FIN}, \mathsf{INV})), \qquad 6 :((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV})) \end{aligned}$$

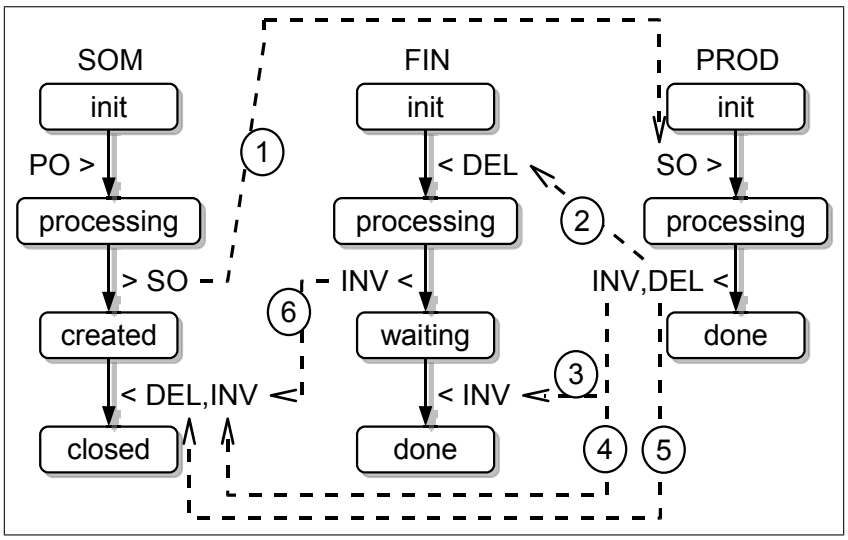


Figure 7.5: Three behavioral models with potential communications.

Please note that not all potential communications necessarily have to participate in an orchestration. Which of the technically possible communications actually take place in an orchestration depends on the specific integration purpose the orchestration is built for. In Chapter 8, we explain when and how that decision is made.

Based on the definitions of state and communication, we now define a copy rule being a communication (see Definition 7.5) linked to a specific time when executing an orchestration—identified via states of the behavioral models (see Definition 7.4). We chose the name "copy rule" because the execution of a copy rule stands for assigning, or copying, the value of a variable of one behavioral model to a variable of another behavioral model. We call it a rule because the state information can be interpreted as a guard that enables the enactment of the communication.

Definition 7.6 (Copy rules)**.**

$$\mathrm{C} := \; S, K \;\; , \; S \subseteq \mathcal{B} \;\; , \; K \subseteq \mathcal{K}$$

The following are examples for copy rules for communications appearing in Figure 7.5 on the preceding page.

$$\begin{aligned} c_1 = (\;\; & (\mathsf{SOM}, \mathsf{created}), (\mathsf{FIN}, \mathsf{init}), (\mathsf{PROD}, \mathsf{init}) \;\; , \\ & ((\mathsf{SOM}, \mathsf{SO}), (\mathsf{PROD}, \mathsf{SO})) \;\;) \\ c_2 = (\;\; & (\mathsf{SOM}, \mathsf{created}), (\mathsf{FIN}, \mathsf{waiting}), (\mathsf{PROD}, \mathsf{done}) \;\; , \\ & ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV})) \;\;) \end{aligned}$$

The first copy rule c_1 states that the value of SO should be copied from SOM to PROD when the behavioral model of SOM is in the state created and the other behavioral models are in their initial states. Thus, c_1 assigns a concrete time in the execution of the three behavioral models to communication 1 above. The second copy rule c_2 assigns another execution time point to communication 6.

It shall be noted that the copy rules are not meant to be created manually in our approach. That is the task of our complex-goal-based WS composition which will be presented in Chapter 8. Also, enumerating potential communications is not an entirely manual task, but supported by our CFIM of hierarchical types presented in Chapter 6 on page 65.

After defining copy rules, we now specify their execution semantics. As said, a copy rule can be understood as a rule which becomes active when the time point in the execution—represented by states of each behavioral model—is reached. In addition, we have to check whether the variables that should be read from are initialized and the variables that should be written to are not yet initialized. As denoted, the execution of

a copy rule corresponds to changing the states of the variables involved. Since a copy rule contains multiple assignments of variables, that may trigger the further sending of multiple messages via the respective SEND ASMs as explained above by marking the input data as available.

COPY $\equiv$

if $\mathrm{bmState}(bm_1) = s_1$ **and** ...
  and $\mathrm{bmState}(bm_{\ states}) = s_{\ states}$
  and $\mathrm{varState}(bm_{\mathrm{out}_1}, o_1) = \mathrm{initialized}$ **and** ...
  and $\mathrm{varState}(bm_{\mathrm{out}_{\ comms}}, o_{\ comms}) = \mathrm{initialized}$
  and $\mathrm{varState}(bm_{\mathrm{in}_1}, i_1) = \mathrm{initialized}$ **and** ...
  and $\mathrm{varState}(bm_{\mathrm{in}_{\ comms}}, i_{\ comms}) = \mathrm{initialized}$
then
  do forall $(\ (\ bm_{\mathrm{out}}, o\ ), (\ bm_{\mathrm{in}}, i\ )\ )$ $\quad comms$
    $\mathrm{varState}(bm_{\mathrm{out}}, o) := \mathrm{processed}$
    $\mathrm{varState}(bm_{\mathrm{in}}, i) := \mathrm{initialized}$

:

$(\ states, comms\ )$ Rules,
$(\ bm_k, s_k\ )$ $\quad states, \quad k = 1\ ..\ states$
$(\ (\ bm_{\mathrm{out}_n}, o_n\ ), (\ bm_{\mathrm{in}_n}, i_n\ )\ )$ $\quad comms, \quad n = 1\ ..\ comms$

To give an example, we explicitly formulate the concrete COPY machine for the set $Rules = \ c_1$ (see above).

COPY $\equiv$
  if $\mathrm{bmState}(\mathsf{SOM}) = \mathsf{created}$
    and $\mathrm{bmState}(\mathsf{FIN}) = \mathsf{init}$
    and $\mathrm{bmState}(\mathsf{PROD}) = \mathsf{init}$
    and $\mathrm{varState}(\mathsf{SOM}, \mathsf{SO}) = \mathrm{initialized}$
    and $\mathrm{varState}(\mathsf{PROD}, \mathsf{SO}) = \mathrm{initialized}$
  then
    do forall $(\ (\ bm_{\mathrm{out}}, o\ ), (\ bm_{\mathrm{in}}, i\ )\ )$ $\quad (\ (\mathsf{SOM}, \mathsf{SO}\ ), (\ \mathsf{PROD}, \mathsf{SO}\ )\ )$
      $\mathrm{varState}(bm_{\mathrm{out}}, o) := \mathrm{processed}$
      $\mathrm{varState}(bm_{\mathrm{in}}, i) := \mathrm{initialized}$

The rule contained in that machine becomes activated when the behavioral model of SOM is in the state created, both FIN and PROD are in their initial states, and the value of variable SO in the behavioral model SOM is available and was not yet copied to variable SO of PROD. When fired, the rule marks SO in SOM as processed and SO in PROD as available, or initialized, in order to not fire over and over again and to trigger subsequent ASMs.

The interplay of the ASMs representing behavioral models—i. e., SEND, RECEIVE, and ADVANCEBM—and their communication—i. e., COPY—for the sake of forming an orchestration is summarized in the following section.

7.2.4 Orchestration execution semantics

In the former sections, we have presented mathematical models for behavioral models and their communication. An orchestration of a set of behavioral models can be understood as a set of communications happening at specific time points. Therefore, we use copy rules to express an orchestration.

The ASMs involved in defining the execution semantics of one behavioral model were displayed in Figure 7.4 on page 92. In order to express an orchestration of two behavioral models, we have to, informally speaking, duplicate Figure 7.4 and combine the two ADVANCEBM machines via a COPY machine. That is displayed in Figure 7.6.

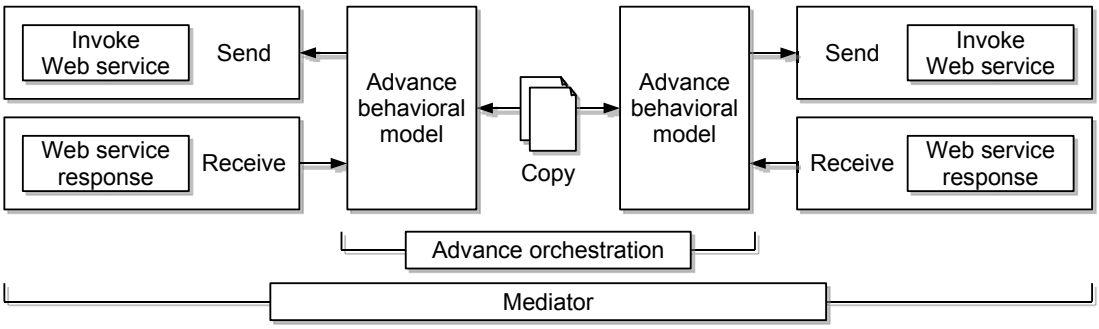


Figure 7.6: ASMs involved in executing an orchestration.

Although the picture shows the interplay of just two behavioral models, one COPY machine can cope with an arbitrary number of parallelly evolving executions of behavioral models.

We have already defined all ASMs participating in the execution of an orchestration. What is missing is a machine that invokes one ASM after the other. However, instead of providing one such machine, we incrementally combine the ASMs in two machines. We make that distinction because it allows us to extend our approach from EAI to e-business later in this chapter.

The incremental combination of machines is connotated at the bottom of Figure 7.6. First, we bundle all ADVANCEBM and the COPY ASM together. The new machine invoking the ADVANCEBM and COPY machines is the ADVANCEORCHESTRATION machine.

$$\begin{array}{l}\textsc{AdvanceOrchestration}(IDs) \equiv \\ \quad \textbf{choose} \quad M : \; M \equiv \textsc{AdvanceBM}(bm) \quad M \equiv \textsc{Copy}, bm \quad IDs \\ \quad\quad M\end{array}$$

The ADVANCEORCHESTRATION machine chooses an ASM from a set of ASMs that contains an activated rule—i. e., a rule that may fire. That is done by the non-deterministically **choose** construct in the ASM formalism.

In the second step, we bundle the ADVANCEORCHESTRATION machine with all SEND and RECEIVE machines to be alternatingly invoked by the MEDIATOR machine.

MEDIATOR $\equiv$
  choose $M : M \equiv$ ADVANCEORCHESTRATION(ID)
    $M \equiv$ SEND(bm) $M \equiv$ RECEIVE(bm), bm ID
  M

In this section, we defined a notation for behavioral knowledge that

- is connected to technical Web service descriptions—and thus can be applied on existing Web service infrastructure,
- is capable to express operation dependencies in a formal way which is done today ambiguously in the form of natural language,
- can be interlinked to form orchestrations, and
- can be executed by simulation engines such as CoreASM.

In the rest of this chapter, we present a method for transforming behavioral knowledge in the form of behavioral models between the phases of software development to benefit from the formal explication. For that purpose, our method involves storing behavioral models when components are developed and adapting the stored behavioral models to the needs of integration experts for the two integration stages EAI and e-business integration without losing any of the behavioral knowledge on the way—even not when the final orchestration is deployed for execution.

7.3 Formalizing requirement

Before we present our method to track behavioral information through software design phases, we formalize that requirement. In the definition, we use a transitive closure of edges in a behavioral model. Therefore, we first define the transitive closure.

Definition 7.7 (Transitive closure)**.** Let Q be a set of states of a behavioral model, L a set of labels. Let δ be a set of labeled transitions of the form $\delta : \mathrm{Q} \times \mathrm{L}$ Q connecting the states. Then, the transitive closure of all edges $\mathcal{T}(\delta)$ is defined as follows:

$$(q_1, l, q_2) \quad \delta \qquad q_1, q_2 \quad \mathcal{T} \quad (\delta)$$

$$q_1, q_2 \quad \mathcal{T} \quad (\delta) \qquad q_2, q_3 \quad \mathcal{T} \quad (\delta) \qquad q_1, q_3 \quad \mathcal{T} \quad (\delta)$$

In Section 3.4.2 on page 36, we stated as a requirement to track behavioral information through software design phases that no operation execution sequences that were not allowed in the component design time model must be allowed in the integration design time model. We formally express that condition as follows.

Definition 7.8 (Correct overall sequencing)**.** Whenever a sequence of operation invocations (o_1, o_2) is observed during the execution of an orchestration of copy rules, then the operations should appear in the respective community perspective behavioral model $\mathcal{C}$ in the same order on some path calculated using the transitive closure of all edges of the community perspective behavioral model: $o_1, o_2 \ \mathcal{T} \ (\delta^{\mathcal{C}})$.

In order to check for the proper sequencing of operations in the orchestration and in the community perspective behavioral model, we need to check that property on all intermediate model transitions. That is done in the following sections.

7.4 Convert provided behavioral model to provider view

As depicted in Figure 7.7, every interface can in general be seen from the

- consumer view and the
- provider view.

The peer implementing an interface takes the provider view and a peer using an interface takes the consumer view. For example, if a piece of software makes use of an interface by reading information from the interface (consumer view), then some piece of software implementing the interface must send that information to the interface (provider view). Vice versa, if a piece of software makes use of an interface by writing data to the interface (consumer view), then some piece of software implementing the interface must be ready to absorb and process the data (provider view). In that sense, the consumer view of an interface is the opposite of the provider view.

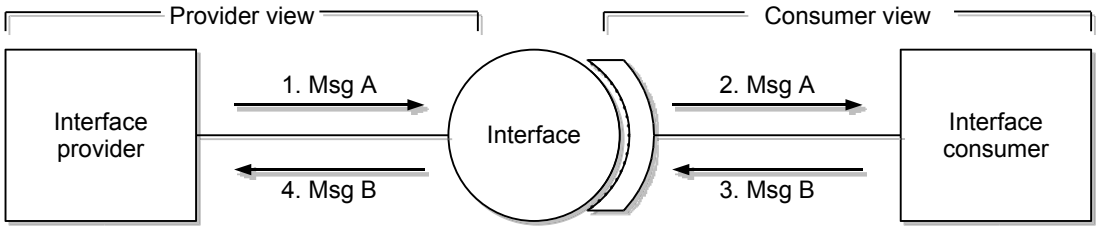


Figure 7.7: Interface provider and consumer perspectives. Provider and consumer have opposite behavior. The provider first sends then receives. The consumer first receives then sends.

In other words, the concept of the opposite of an interface allows us to connect the consumer and the provider view. The opposite of an interface acts as a gateway between the implementation of the provider and the consumer. That is the key idea behind our solution transforming behavioral models for EAI and e-business.

Definition 7.9 (Opposite behavioral model)**.** We define the opposite behavioral model $\mathcal{BM}^{-1}$ of an original behavioral model $\mathcal{BM} = (\Sigma, \mathrm{Z}, \mathrm{Q}, q_0, \delta)$ as

$$\mathcal{BM}^{-1} := (\mathrm{Z}, \Sigma, \mathrm{Q}, q_0, \delta) .$$

The only difference between a behavioral model and its opposite is the mutual substitution of inputs and outputs. As an example, we show the provided behavioral model of a selling company in Figure 7.8. As said, the provided behavioral model can be seen from the consumer view (SELR)—which is used for e-business by business partners like CUST to connect—and the provider view (SELR^{-1})—which is used for EAI to integrate with the components of the seller (i. e., SOM, FIN, and PROD).

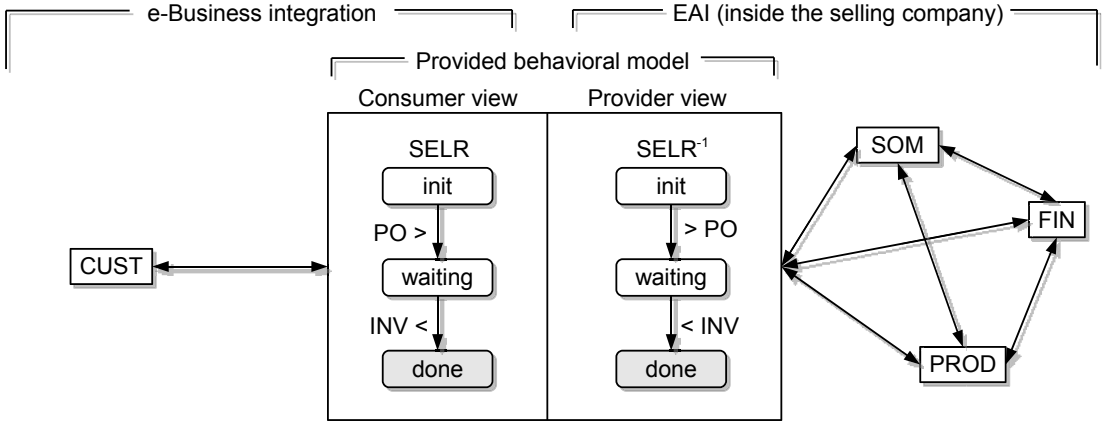


Figure 7.8: Interface of the seller.

In order to demonstrate the difference, we present the behavioral models in formal notation below.

$\mathrm{Q}^{\mathsf{SELR}} = \mathsf{init}, \mathsf{waiting}, \mathsf{done}$

$\Sigma^{\mathsf{SELR}} = \mathsf{PO}$

$\mathrm{Z}^{\mathsf{SELR}} = \mathsf{INV}$

$\delta^{\mathsf{SELR}} =$

q_1	I/Os	q_2
init	PO	waiting
waiting	INV	done

$q_0^{\mathsf{SELR}} = \mathsf{init}$

$\mathrm{Q}^{\mathsf{SELR}^{-1}} = \mathsf{init}, \mathsf{waiting}, \mathsf{done}$

$\Sigma^{\mathsf{SELR}^{-1}} = \mathsf{INV}$

$\mathrm{Z}^{\mathsf{SELR}^{-1}} = \mathsf{PO}$

$\delta^{\mathsf{SELR}^{-1}} =$

q_1	I/Os	q_2
init	PO	waiting
waiting	INV	done

$q_0^{\mathsf{SELR}^{-1}} = \mathsf{init}$

It is obvious that creating the opposite of a behavioral model does not violate the proper sequencing of operations as was demanded in Section 7.3 on page 100.

Lemma 7.10 (Proper sequencing of opposite behavioral models)**.** *Two operations that appear in a behavioral model $\mathcal{BM}$ appear in the same order in the opposite behavioral model $\mathcal{BM}^{-1}$.*

Proof. That is trivial because $\mathrm{Q}^{\mathcal{BM}} = \mathrm{Q}^{\mathcal{BM}^{-1}}$. □

7.5 Excerpt consumed behavioral model fragment

The behavioral model of a software component in the component perspective describes

- the potential operations the component may perform,
- potential alternative outcomes of an operation invocation, and
- sequencing constraints that are required for a successful operation invocation.

However, the operations are general and need to be applied in the context of a business process in order to perform some meaningful task. For that purpose, a business process expert may define a view of the components to participate in the enterprise-internal business process (EAI). As not the full functionality of the components may be needed in the context of the business process, the business process expert first has to define the relevant fragment of the component perspective behavioral models. Of course, the relevant fragment must be in concordance with the component perspective behavioral models to meet the requirement of preserving operation sequences stated in Section 7.3 on page 100.

Before we can present an algorithm how to construct a concordant fragment from a behavioral model, we introduce some preliminaries.

The first is a notational convention which we introduce to differentiate between states of a component perspective behavioral model $\mathcal{C}$ and a fragment behavioral model $\mathcal{F}$. We represent a state of the component perspective behavioral model using superscript as $q^{\mathcal{C}}$ and a state of the fragment behavioral model as $q^{\mathcal{F}}$.

Second, we define a relation between states of a component perspective behavioral model and states of a fragment—the "originating state"—to enable later comparison of the two behavioral models.

Definition 7.11 (Originating state)**.** We denote the originating component perspective behavioral model state $q^{\mathcal{C}}$ for a fragment state $q^{\mathcal{F}}$ as

$$q^{\mathcal{C}} = \sigma(q^{\mathcal{F}}).$$

As an example, let's assume the left-hand side of Figure 7.9 on the following page would be the component perspective behavioral model of the seller's finance component ($\mathsf{FIN}^{\mathcal{C}}$) and the right-hand side would be the derived fragment $\mathsf{FIN}^{\mathcal{F}}$. The component perspective behavioral model describes two operations: First, an operation to create an invoice (INV) from a delivery (DEL). The operation is represented by the

state $\mathsf{processing}^{\mathcal{C}}$. Second, an operation to pay an invoice (INV) without any output, represented by the state $\mathsf{init}^{\mathcal{C}}$.

In the fragment behavioral model, the loops in the state transition system are unfolded. Here, the first operation is represented by state $\mathsf{processing}^{\mathcal{F}}$, the second by $\mathrm{done}^{\mathcal{F}}$.

Between both behavioral models, the originating state function is displayed by the dashed arrows (Figure 7.9) which we formally explicate below.

$$\begin{aligned}\mathsf{processing}^{\mathcal{C}} &= \sigma(\mathsf{processing}^{\mathcal{F}})\\ \mathsf{init}^{\mathcal{C}} &= \sigma(\mathsf{init}^{\mathcal{F}})\\ \mathsf{init}^{\mathcal{C}} &= \sigma(\mathsf{waiting}^{\mathcal{F}})\\ \mathsf{init}^{\mathcal{C}} &= \sigma(\mathsf{done}^{\mathcal{F}})\end{aligned}$$

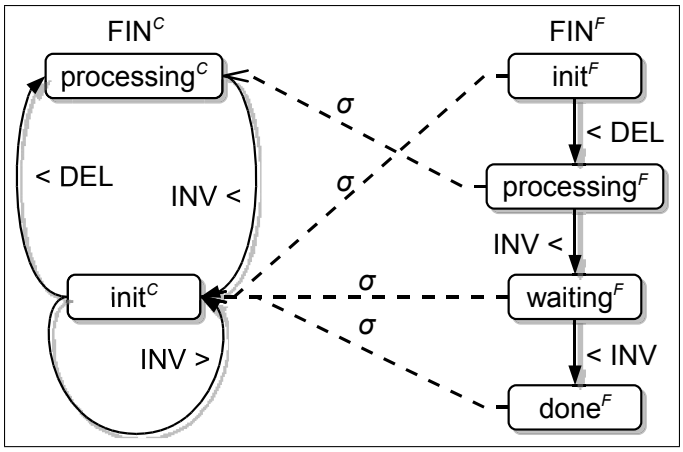


Figure 7.9: Originating states of the seller's financial component.

Third, we pose the restriction on fragment behavioral models to be trees only, which allows us later—in Chapter 8—to define an efficient complex-goal-based WS composition. Despite the restriction to trees, we will show in Chapter 10 that our fragments still allow expressing real-life business processes.

A handy property of trees is that one can always name the parent node of each node except for the root node. We are interested in that property to define the "enabling state" of an operation.[1]

Definition 7.12 (Enabling state)**.** For every fragment behavioral model $\mathcal{F}$, $\delta^{\mathcal{F}}$ forms a rooted tree. Thus, we can uniquely name the enabling state $\phi_{o^{\mathcal{F}}}$ at which the specific operation $o^{\mathcal{F}}$ can be invoked.

[1]Operation was defined in Section 7.2 on page 90 to be a state with at least one leaving output transition or with at least one entering input transition.

To give an example, we enumerate the enabling states of both behavioral models in Figure 7.9 on the preceding page.

$$\phi_{\mathsf{processing}^{\mathcal{F}}} = \mathsf{init}^{\mathcal{F}}$$
$$\phi_{\mathrm{done}^{\mathcal{F}}} = \mathsf{waiting}^{\mathcal{F}}$$

Finally, we need to define what it means to preserve the knowledge of proper operation sequencing between component perspective behavioral models and fragments. Whenever we observe two operations o_1, o_2 on any path in a fragment $\mathcal{F}$ calculated using the transitive closure ($\mathcal{T}$) of all edges, i. e.,

$$\mathrm{A} = \ o_1, o_2 \ \mathcal{T} \quad (\delta^{\mathcal{F}}),$$

and these operations appear in any order on a common path in the respective component perspective behavioral model $\mathcal{C}$, i. e.,

$$\mathrm{B} = \ \sigma(o_1), \sigma(o_2) \ \mathcal{T} \quad (\delta^{\mathcal{C}}) \quad \text{or}$$
$$\mathrm{C} = \ \sigma(o_2), \sigma(o_1) \ \mathcal{T} \quad (\delta^{\mathcal{C}}),$$

then they should appear in the same order in $\mathcal{C}$ as they appear in $\mathcal{F}$:

$$\mathrm{B} = \ \sigma(o_1), \sigma(o_2) \ \mathcal{T} \quad (\delta^{\mathcal{C}}).$$

Thus, the following should hold at all times:

$$\mathrm{A} \quad (\mathrm{B} \quad \mathrm{C}) \quad \mathrm{B}$$

We can transform the formula using the rules of propositional logic:

$$\begin{array}{l} \mathrm{A} \quad (\mathrm{B} \quad \mathrm{C}) \quad \mathrm{B} \\ \mathrm{A} \quad (\ \mathrm{B} \quad \mathrm{C}) \quad \mathrm{B} \\ \mathrm{A} \quad [\ (\ \mathrm{B} \quad \mathrm{B}) \quad (\ \mathrm{C} \quad \mathrm{B})\ ] \\ \mathrm{A} \quad \mathrm{C} \quad \mathrm{B} \\ \mathrm{A} \quad \mathrm{C} \quad \mathrm{B} \end{array}$$

We can now introduce a definition for the "accordance of excerpted and original behavioral model."

Definition 7.13 (Accordance of excerpted and original behavioral model)**.** Let $\mathcal{C}$ be a component perspective behavioral model, $\mathcal{F}$ a fragment behavioral model excerpted from $\mathcal{C}$, $o_1, o_2 \quad \mathrm{Q}^{\mathcal{F}}$ operations of $\mathcal{F}$, and $\mathcal{T}$ the transitive closure. Then,

$$o_1, o_2 \ \mathcal{T} \quad (\delta^{\mathcal{F}}) \quad \sigma(o_2), \sigma(o_1) \ \mathcal{T} \quad (\delta^{\mathcal{C}}) \quad \sigma(o_1), \sigma(o_2) \ \mathcal{T} \quad (\delta^{\mathcal{C}}).$$

We can say that two behavioral models are accordant, if for every two operations in the excerpted behavioral model whose originating states appear in a different order in the original behavioral model, the originating states must be also in the order of the excerpted behavioral model. That is, both orders must be allowed in the original behavioral model.

As an example, let's check whether the two behavioral models in Figure 7.9 on page 104 are in accordance. For that, we first list their transition functions.

$\delta^{\mathsf{FIN}^{\mathcal{C}}}$

q_1	$\mathrm{Q}^{\mathsf{FIN}^{\mathcal{C}}}$	I/Os	q_2	$\mathrm{Q}^{\mathsf{FIN}^{\mathcal{C}}}$
	$\mathsf{init}^{\mathcal{C}}$	DEL		$\mathsf{processing}^{\mathcal{C}}$
	$\mathsf{processing}^{\mathcal{C}}$	INV		$\mathsf{init}^{\mathcal{C}}$
	$\mathsf{init}^{\mathcal{C}}$	INV		$\mathsf{init}^{\mathcal{C}}$

$\delta^{\mathsf{FIN}^{\mathcal{F}}}$

q_1	$\mathrm{Q}^{\mathsf{FIN}^{\mathcal{F}}}$	I/Os	q_2	$\mathrm{Q}^{\mathsf{FIN}^{\mathcal{F}}}$
	$\mathsf{init}^{\mathcal{F}}$	DEL		$\mathsf{processing}^{\mathcal{F}}$
	$\mathsf{processing}^{\mathcal{F}}$	INV		$\mathsf{waiting}^{\mathcal{F}}$
	$\mathsf{waiting}^{\mathcal{F}}$	INV		$\mathsf{done}^{\mathcal{F}}$

Second, we compute the transitive closure of their transition functions.

$$\begin{aligned}
\mathcal{T}(\delta^{\mathsf{FIN}^{\mathcal{C}}}) = \quad & \left\langle \mathsf{init}^{\mathcal{C}}, \mathsf{processing}^{\mathcal{C}} \right\rangle, \quad \left\langle \mathsf{processing}^{\mathcal{C}}, \mathsf{init}^{\mathcal{C}} \right\rangle, \\
& \left\langle \mathsf{init}^{\mathcal{C}}, \mathsf{init}^{\mathcal{C}} \right\rangle, \quad \left\langle \mathsf{processing}^{\mathcal{C}}, \mathsf{processing}^{\mathcal{C}} \right\rangle \\
\mathcal{T}(\delta^{\mathsf{FIN}^{\mathcal{F}}}) = \quad & \left\langle \mathsf{init}^{\mathcal{F}}, \mathsf{processing}^{\mathcal{F}} \right\rangle, \quad \left\langle \mathsf{processing}^{\mathcal{F}}, \mathsf{waiting}^{\mathcal{F}} \right\rangle, \\
& \left\langle \mathsf{waiting}^{\mathcal{F}}, \mathsf{done}^{\mathcal{F}} \right\rangle, \quad \left\langle \mathsf{init}^{\mathcal{F}}, \mathsf{waiting}^{\mathcal{F}} \right\rangle, \\
& \left\langle \mathsf{init}^{\mathcal{F}}, \mathsf{done}^{\mathcal{F}} \right\rangle, \quad \left\langle \mathsf{processing}^{\mathcal{F}}, \mathsf{done}^{\mathcal{F}} \right\rangle
\end{aligned}$$

Third, we find a sequence of operations in $\mathsf{FIN}^{\mathcal{F}}$:

$$\left\langle \mathsf{processing}^{\mathcal{F}}, \mathsf{done}^{\mathcal{F}} \right\rangle \ \mathcal{T} \ (\delta^{\mathsf{FIN}^{\mathcal{F}}}).$$

Fourth, the originating states of $\mathsf{FIN}^{\mathcal{F}}$ were shown above. We can now identify the opposite order of o_1 and o_2 in the component perspective behavioral model $\mathsf{FIN}^{\mathcal{C}}$:

$$\left\langle \mathsf{init}^{\mathcal{C}}, \mathsf{processing}^{\mathcal{C}} \right\rangle \ \mathcal{T} \ (\delta^{\mathsf{FIN}^{\mathcal{C}}})$$

Finally, we need to check whether the reverse order also appears in $\mathsf{FIN}^{\mathcal{C}}$.

$$\left\langle \mathsf{processing}^{\mathcal{C}}, \mathsf{init}^{\mathcal{C}} \right\rangle \ \overset{!}{\mathcal{T}} \ (\delta^{\mathsf{FIN}^{\mathcal{C}}})$$

As that is the case, we have shown that the behavioral models $\mathsf{FIN}^{\mathcal{C}}$ and $\mathsf{FIN}^{\mathcal{F}}$ are in accordance, and thus $\mathsf{FIN}^{\mathcal{F}}$ is a proper excerpt of $\mathsf{FIN}^{\mathcal{C}}$.

However, the above was just an example of a behavioral model and one possible excerpt. We are now prepared to describe the algorithm a business process expert should have performed to excerpt $\mathsf{FIN}^{\mathcal{F}}$ from $\mathsf{FIN}^{\mathcal{C}}$ to ensure by construction that both behavioral models are in accordance.

1. An excerption step begins with picking an operation $o^{\mathcal{C}}$ from $\mathcal{C}$.

2. (a) If $\mathcal{F}$ does not contain any states yet, $o^{\mathcal{C}}$ must have an incoming transition from $q_0^{\mathcal{C}}$, or $o^{\mathcal{C}}$ must be the initial state $o^{\mathcal{C}} = q_0^{\mathcal{C}}$. The new states $q_0^{\mathcal{F}}$ with $q_0^{\mathcal{C}} = \sigma(q_0^{\mathcal{F}})$ and $o^{\mathcal{F}}$ with $o^{\mathcal{C}} = \sigma(o^{\mathcal{F}})$ build the first operation in $\mathcal{F}$.

   (b) i. If $\mathcal{F}$ already contains operations, the new state $o^{\mathcal{F}}$ with $o^{\mathcal{C}} = \sigma(o^{\mathcal{F}})$ becomes integrated into $\mathcal{F}$. That is done by unifying $o^{\mathcal{F}}$'s enabling state $\phi_{o^{\mathcal{F}}}$ with an existing state s in $\mathcal{F}$ that has the same origin in $\mathcal{C}$ as $\phi_{o^{\mathcal{F}}}$, i. e.,
   $$\sigma(s) = \sigma(\phi_{o^{\mathcal{F}}}).$$

   ii. If the former step is not possible, $o^{\mathcal{C}}$ cannot be added at this time.

Whenever an operation $o^{\mathcal{C}}$ is added to a fragment $\mathcal{F}$, $o^{\mathcal{C}}$ becomes unfolded. That means that the state $o^{\mathcal{F}}$ and all states that $o^{\mathcal{F}}$'s output transitions directly link to are new, separate states in $\mathcal{F}$.

Lemma 7.14. *The described construction procedure only produces trees which are in accordance with their respective community perspective behavioral model.*

Proof of trees. It is exactly one tree generated due to two reasons: First, the enabling state $\phi_{o^{\mathcal{F}}}$ of a newly added operation $o^{\mathcal{F}}$ unifies with an existing state in $\mathcal{F}$ if $o^{\mathcal{F}}$ is not the first operation. Second, all states except for $\phi_{o^{\mathcal{F}}}$ are new states that were not present in $\mathcal{F}$ before. □

Proof of accordance. Let's assume the negation of Definition 7.13.

$$\begin{array}{cccccccccc} & & & & & (\,\mathrm{A} & \mathrm{C} & \mathrm{B}\,) = \mathrm{A} & \mathrm{C} & \mathrm{B} \\ = & o_1, o_2 & \mathcal{T} & (\delta^{\mathcal{F}}) & & \sigma(o_2), \sigma(o_1) & \mathcal{T} & (\delta^{\mathcal{C}}) & \sigma(o_1), \sigma(o_2) & \mathcal{T} \quad (\delta^{\mathcal{C}}) \end{array}$$

The formula states that in $\mathcal{C}$, there is a connection from the originating state of o_2 to the originating state of o_1, but not vice versa. Additionally, in $\mathcal{F}$, o_2 is behind o_1. If we can show that our algorithm cannot produce such a situation, we have proven that it only creates according excerpts.

In our algorithm, operations are being added one after the other to $\mathcal{F}$. Therefore, we must differentiate between two cases:

1. o_1 is added first, o_2 afterwards. With regards to our algorithm, an operation can only be added if its enabling state is already in $\mathcal{F}$. However, if o_2 is going to be added in the same path below o_1, a state $s^{\mathcal{F}}$ must be created in $\mathcal{F}$ whose

originating state is at the same time behind the originating state of o_1 and before the originating state of o_2:

$$\begin{array}{l} \left\langle \sigma(o_1), \sigma(s^{\mathcal{F}}) \right\rangle \ \mathcal{T} \ (\delta^{\mathcal{C}}) \\ \left\langle \sigma(s^{\mathcal{F}}), \sigma(o_2) \right\rangle \ \mathcal{T} \ (\delta^{\mathcal{C}}). \end{array}$$

That implies $\sigma(o_1), \sigma(o_2) \ \mathcal{T} \ (\delta^{\mathcal{C}})$, which contradicts the assumption.

2. o_2 is added first, o_1 afterwards. If o_1 is added after o_2 then all the states directly following o_1 are new states in $\mathcal{F}$ which can never be linked to operations already existing in $\mathcal{F}$. Thus, for every state $s^{\mathcal{F}}$ with

$$\begin{array}{l} \left\langle o_1, s^{\mathcal{F}} \right\rangle \ \mathcal{T} \ (\delta^{\mathcal{F}}) : \\ \left\langle s^{\mathcal{F}}, o_2 \right\rangle \ \not\mathcal{T} \ (\delta^{\mathcal{F}}). \end{array}$$

Therefore, $o_1, o_2 \ \not\mathcal{T} \ (\delta^{\mathcal{F}})$, which contradicts the assumption.

By contradiction we have shown that our algorithm preserves operation order. □

7.6 Join orchestrations

Let's assume that there is an orchestration O_{ebus} of the e-business perspective. Let the set of identifiers of behavioral models in O_{ebus} be PID. Let's further assume that for each behavioral model $\mathcal{BM}$ in O_{ebus} with identifier bm there exists an orchestration O_{bm} of the EAI perspective that contains $\mathcal{BM}^{-1}$. In order to deliver on the promise to track behavioral information through software design phases, we have to define a composition of orchestrations. We define the joined orchestration O_{join} of O_{ebus} and all O_{bm} by joining their copy rules.

Definition 7.15 (Joined orchestration)**.**

$$O_{\text{join}} := \text{Rules}^{O_{\text{ebus}}} \bigcup_{bm \ \text{PID}} \text{Rules}^{O_{bm}}$$

The execution semantics of the joined orchestration illustrated in Figure 7.10 on the next page is a slight modification to the MEDIATOR ASM.

MEDIATOR* $\equiv$
  choose M : $M \equiv$ ADVANCEORCHESTRATION(ID PID)
    $M \equiv$ SEND(bm) $M \equiv$ RECEIVE(bm), bm ID
  M

The parameters of ADVANCEORCHESTRATION differentiate MEDIATOR* from MEDIATOR: In the joined orchestration, the behavioral models of all Web services and all behavioral models in the e-business perspective orchestration have to be advanced.

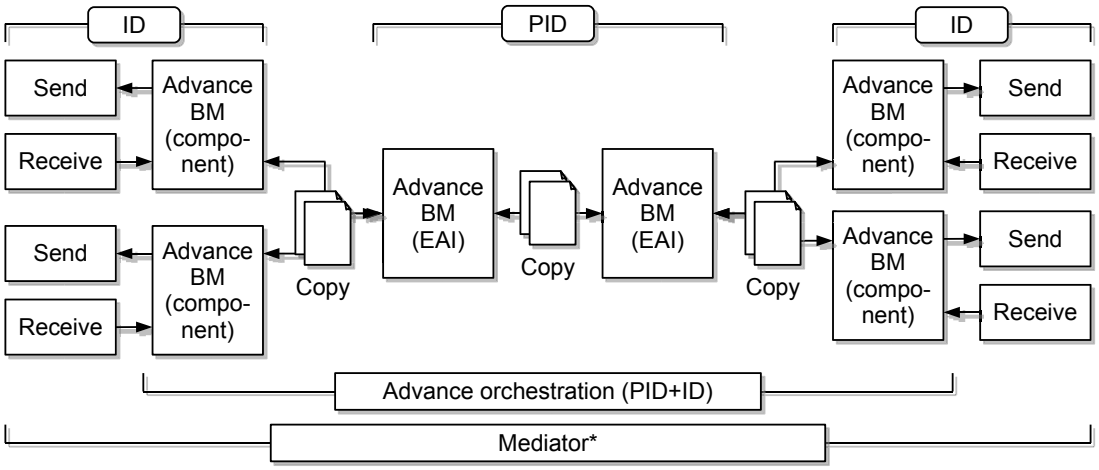


Figure 7.10: ASMs involved in execution in the e-business perspective.

Lemma 7.16. *Let o be an orchestration consisting of a set of copy rules $\mathcal{C}$. Let o be an orchestration that consists of two sets of copy rules $\mathcal{C}$ and $\mathcal{C}$ that are joined by $\mathcal{C}$. Let o be an orchestration consisting of $\mathcal{C}$. Two operations that are triggered to be executed in a sequence by advancement of $\mathcal{C}$ through o can also be triggered in sequence by o.*

Proof. As we have defined, each copy rule c of an orchestration p contains only states of behavioral models that participate in p and for each behavioral model that participates in p, there is exactly one state in the state component of c. Further, two orchestrations p and p are joined when they share exactly one participating behavioral model.

The COPYASM performing o from the lemma executes $\mathcal{C}$, which is contained in o, as if $\mathcal{C}$ was the single set of copy rules of an orchestration, such as in o. Thus, every pair of operations triggered in a sequence by o would also be triggered in the same sequence by o. From the perspective of $\mathcal{C}$, the joining orchestration o just appears as a participant of o. □

7.7 Proving requirement

We can now prove the requirement defined in Section 7.3 on page 100. The proof directly follows from the lemmas that were presented in this chapter under two circumstances:

1. The transformations from this chapter are employed based on the procedure described in Chapter 5 on page 55.

2. The orchestration established through the copy rules is correct in that it guarantees the same sequence of operation invocations as defined in each community perspective behavioral model. That property will be proved in the following chapter.

Theorem 7.17. *Under the named conditions, each sequence of operation invocations* (o_1, o_2) *observed during the execution of an orchestration, consisting of a set of copy rules, also appears in a community perspective behavioral model* $\mathcal{C}$ *in the same order* $(o_1^{\mathcal{C}}, o_2^{\mathcal{C}})$ *on some path, calculated using the transitive closure (*$\mathcal{T}$*) of all edges* $\langle o_3^{\mathcal{C}}, o_4^{\mathcal{C}} \rangle$ $\mathcal{T}(\delta^{\mathcal{C}})$.

Proof. Follows from the lemmas in this chapter. □

7.8 Summary

In this chapter, we have detailed the derive and join orchestrations activities that were first introduced in the overview in Chapter 5 on page 55. With the techniques presented here, it is now possible to express behavioral knowledge at component design time and use it during integration design time through model-driven transformations. That is a benefit compared to what needs to be done currently to achieve process integration:

First, the capabilities in terms of operation dependencies are today not expressed formally—as, for example, using our behavioral models,— but in a natural language documentation. That yields much room for interpreting the natural language documentation when an integration team works on integrating different components.

Second—in addition to interpreting natural language documentation,—the integration team also has to restrict the potentially very flexible capabilities of the components to the operations and the flows that are actually needed to implement a business process. That—in our solution supported by the derive activity—is today done manually, mostly together with the first step—the interpretation of the natural language documentation. Performing that task manually can not ensure that the behavioral constraints given by the capabilities of the component are actually observed in the restricted behaviors that are used to build the integration.

Third, the integration team has to combine the restricted capabilities of the components to a single orchestration. That includes not only combining the components to achieve a desired outcome. Also, undesired responses from components must be managed consistently across all components. That error-prone, today manual task is called exception handling. Our behavioral models and their transformations build the basis to apply our complex-goal-based WS composition—which will be presented in Chapter 8—to support that task with formal methods to increase consistency.

Fourth, in large integration projects, business processes are modeled first using pen and paper or using business process modeling tools. As these models are mostly informal, it is not guaranteed that the process steps can directly be mapped on, for example, Web service operations. When modeling has finished, the—informal—models are handed over to developers who perform that mapping. The developers may recognize technical difficulties connecting the operations as depicted in the process models. Therefore, developers have to take decisions or ask modelers back for resolution. Consequently, the media break between models and implementation is likely to result in a divergence between modeled and implemented process. Our solution overcomes

that media break as behavioral model and copy rules have operational semantics which can be either directly simulated or transformed to an implementation, such as BPEL.

Fifth, after integration has been achieved internally (EAI), a company might decide to engage in e-business. Again, a collaborative business process needs to be built which considers and does not break the partial business procedures of the partners. Apparently, ensuring consistency in e-business integration strongly depends on the EAI of each partner. As today, the business process a company reveals to its partners is, if at all, documented in natural language, manually integrating different partners (e-business integration) is as error-prone and costly as manually integrating enterprise applications inside one company (EAI). Fortunately, the integration of enterprise applications using formal methods explicates the capabilities of each company. Having that, our same tools can support e-business integration similar to supporting EAI. Our activity "join orchestration" is the key enabler for the formal integration of different orchestrations.

And finally, it is to be remarked that all the steps explained above are today often considered as one single step. One whole step from existing component's Web service interfaces and natural language documentation to an executable integration, expressed, for example, in BPEL. Therefore, today all the different aspects above have to be considered together. With our approach, we raise awareness of the single aspects. Moreover, we pursue the divide-and-conquer approach in that we separate the different aspects to single steps which can be solved one after the other with the support of formal methods to increase understanding, maximize reuse of existing knowledge, and to ensure overall consistency.

Chapter 8

Complex-goal-based WS Composition

In this chapter, we refine the “derive” activity explained in Section 5.2.2 on page 59, in particular the “build orchestration” sub-activity, which was not addressed in this thesis so far (see Figure 8.1). The purpose of the build orchestration activity is to mechanically support exception handling. That is done by our complex-goal-based WS composition.

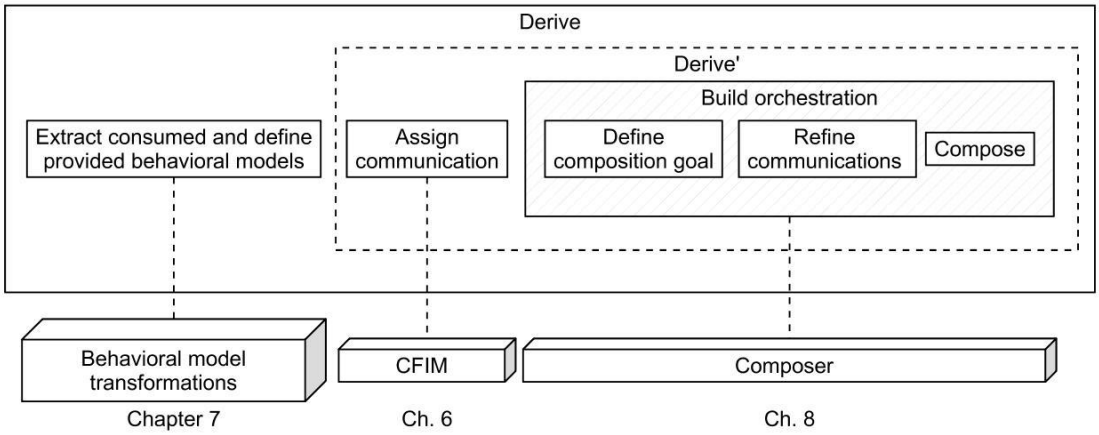


Figure 8.1: Detailed view of the build orchestration sub-activity.

As depicted in Figure 8.2 on the next page, the core of the complex-goal-based WS composer takes as input:

- a set of behavioral models,
- a composition goal, and
- a set of variable assignments.

In addition to the core algorithm, which we call compose, the two preparatory steps involving human intervention

1. define composition goal and

2. refine communications

are necessary. These steps transform the results from the "excerpt consumed behavioral model fragment" activity (sub-activity of "extract consumed and define provided behavioral models," see Chapter 7 on page 85) and the assign communication sub-activity to the inputs of compose—namely, the sets of behavioral models and communications.

The output is an orchestration represented via a set of copy rules as introduced in Section 7.2.3 on page 95. Depending on whether the orchestration was created for EAI or e-business integration, it is—as described in Chapter 5 on page 55—either joined with other orchestrations by the "join orchestrations" activity or directly deployed and executed.

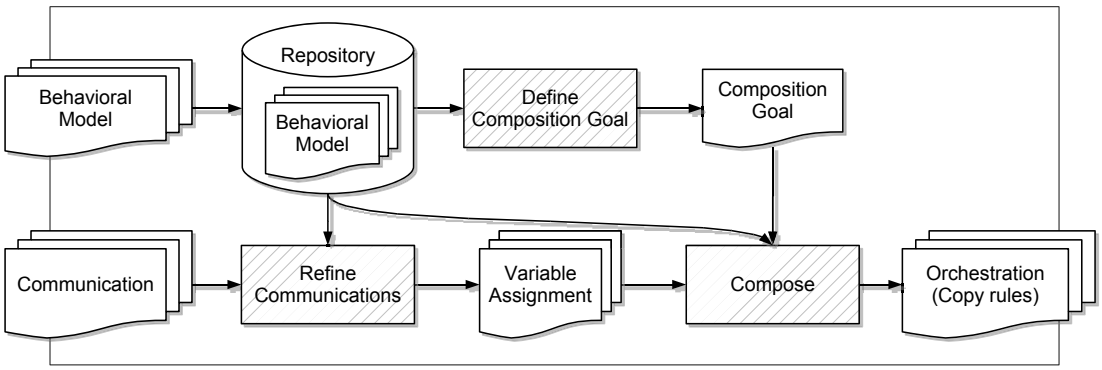


Figure 8.2: Architecture of complex-goal-based WS composition.

The central element of the complex-goal-based WS composition is a repository to store the set of behavioral models to be composed. The repository assigns a unique ID to each behavioral model contained.

Definition 8.1 (Repository)**.**

$$\text{Repository} := \text{ID} \quad \mathcal{BM}$$

Figure 8.3 on the next page shows a set of behavioral models that make up an exemplary repository. We choose an example here to demonstrate the computation of the composer which is different from the one used in Chapter 7 because it is more complex. We introduced the student transfer example in Section 3.2.3 on page 32. The collaborative business process to compose is the transfer of a student from one to another school in a consortium of schools which are managed by a common head quarter.

The rest of this chapter is structured as follows: Section 8.1 explains the behavioral model of the student transfer in more detail. Section 8.2 describes how the requirement of correct exception handling can be formally stated. In Section 8.3 to Section 8.6, we present the main features of a composition algorithm that is able to generate an orchestration that guarantees correct exception handling. Its detail is given in Appendix A. Section 8.7 proves the properties of the composer.

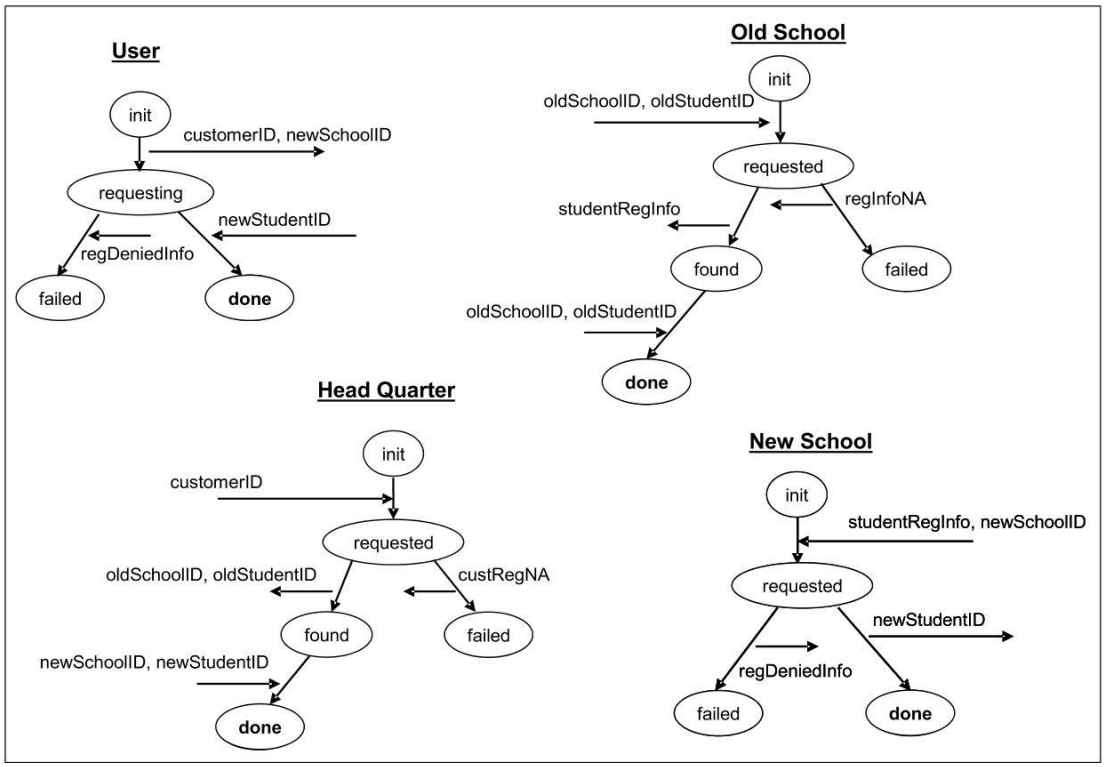


Figure 8.3: Behavioral models for student transfer example.

8.1 Student transfer example

To explain the behavioral models involved in the student transfer process, we list the states representing operations[1] and describe the respective operation verbally below.

User:

- init. That is a notification operation which starts the execution of the student transfer process by sending a transfer request consisting of customer ID and new school ID.
- failed. That is a one-way operation which receives the reason for failure of the process.
- done. That one-way operation is called to inform of successful transfer by reporting the new student ID.

Head quarter:

[1]Operation was defined in Section 7.2 on page 90 to be a state with at least one leaving output transition or with at least one entering input transition.

- requested. That request-response operation returns the enrollment information—school ID and student ID—about where the user (customer) is currently enrolled.
- done. That one-way operation updates the records of the customer to their new school ID and student ID.

Old school:

- requested. That request-response operation returns the detailed registration information about the courses the student is enrolled in at the old school. That operation may fail non-deterministically for the invoker if the student is not know at that school.
- done. That one-way operation unregisters the student from the old school.

New school:

- requested. That request-response operation registers a student at the new school. The operation may fail non-deterministically if, for example, the school does not offer the courses the student is currently enrolled in.

8.2 Refining requirements

In Section 3.4.3 on page 37, it was stated that a correct orchestration should ensure consistent transactional compensation. The correctness of a composition can be defined based on the states that all participating behavioral models can potentially reach in the end of the execution of the orchestration. Such a set of states is called Goal. We differentiate between primary goals ($\mathcal{P}$) and recovery goals ($\mathcal{R}$). Both types of goals are used to describe the requirements of a correct orchestration (Γ).

Definition 8.2 (Correct orchestration)**.** We define an orchestration to be correct if and only if it has the following properties:

- Each execution results in a system state that is part of the composition goal.
- There must be a theoretic execution that leads to a system state defined as one of the primary goals.

By that definition, we ensure transactionality of the behavioral models. One thus has the possibility to specify that either all behavioral models have to reach a successful state or no behavioral model must reach a successful state. For our student transfer example, it would be bad if the Old School successfully unregistered a student, but the New School failed in registering the student.

Definition 8.3 (Goals)**.**

$$
\begin{aligned}
\Gamma &:= \mathcal{P}\ , \mathcal{R} \\
\mathcal{P} &\subseteq \mathrm{Goal} \quad \ldots \text{ set of primary goals} \\
\mathcal{R} &\subseteq \mathrm{Goal} \quad \ldots \text{ set of recovery goals} \\
\mathrm{Goal} &:= \quad goal : \ goal = \quad (\ bm, \mathrm{bmState}\ ) : \ bm \quad \mathrm{ID},\ \mathrm{bmState} \quad \mathrm{Q}^{bm} \\
&\qquad\quad bm \quad \mathrm{ID} : \ (\ bm, \mathrm{bmState}\ ) \quad goal, \\
&\qquad\quad goal \ = \ \mathrm{ID}
\end{aligned}
$$

We illustrate the goal definition by giving possible primary and recovery goals for the behavioral models of our example repository in Table 8.1 on the next page.

The only desired state is that all participants can successfully finish their transactions. That is, the student (user) is told their new student ID, the old school successfully unregistered the student, the new school successfully registered the student, and the head quarter record contains the up-to-date student enrollment information.

The combination of all other non-operation states, initial states, and leaf states makes up the recovery goals (please note that only an excerpt of the recovery goals is listed in Table 8.1 to limit the table's size). Let's consider the example of the old school. The following old school states appear in the recovery goals:

- init**.** A recovery goal containing that state defines an outcome of an execution to be acceptable—though not desirable—if no operation of the old school has been triggered at all.

- failed**.** A recovery goal containing that state defines an outcome to be acceptable if the old school does not have the student on record and no deregistration operation was executed for that student.

- found**.** A recovery goal containing that state defines an outcome to be acceptable if the old school could retrieve student information, but it is not deleted. That state is not desired as it did not complete the student transfer, but it is okay to leave the old school in that state where only information has been requested, but no deregistration has been performed. Having that state in the recovery goal allows for the failure of other participants without ending in an unacceptable state. For example, the new school could deny to register the student. In that case, the orchestration our complex-goal-based WS composition generates would leave old school in the state found and would not unregister the student from the old school.

In addition to the goals, also the set of communications to be considered is input to the composer and thus belongs to the composition requirements. We call the set of communications to be considered by the composer variable assignments. Variable assignments are a subset of communications (see Definition 7.5 on page 96).

Table 8.1: Exemplary goals.

ID	User	OldSchool	HeadQuarter	NewSchool
pg_1	done	done	done	done
rg_8	failed	init	failed	init
rg_{10}	failed	failed	failed	init
rg_{12}	failed	found	failed	init
rg_{16}	failed	failed	found	init
rg_{26}	failed	init	failed	failed
rg_{28}	failed	failed	failed	failed
rg_{30}	failed	found	failed	failed
rg_{34}	failed	failed	found	failed
rg_{36}	failed	found	found	failed

Definition 8.4 (Variable assignment)**.**

$$\mathcal{A} \sqsubseteq \mathcal{K}$$

According to the approach of this thesis, the modeler is supported in identifying possible communications during the assign communication activity (see Figure 8.1 on page 113) by our CFIM of hierarchical types described in Chapter 6. During complex-goal-based WS composition, in particular during the refine communications activity (see Figure 8.2 on page 114), the set of possible communications is reduced to the set of allowable variable assignments for the orchestration. As the naming of the allowable communications depends on the concrete purpose, the orchestrated business process should serve, that reduction is a manual task which can only be done by a human integration expert.

We would like to note that through our approach, we reduced the manual effort by proposing potential communications using our CFIM of hierarchical types to a minimum.

8.3 Structure of the complex-goal-based WS composer

The complex-goal-based WS composer implements the compose box displayed in Figure 8.2 on page 114. The general idea of the composer is to perform a backward chaining of matching inputs and outputs.

The initial backward chaining starts from a primary goal, for example, pg_1 from Table 8.1 on the preceding page. If the chaining was successful, the generated copy rules are simulated.

If the simulation detects a situation where one behavioral model has non-deterministic output alternatives, for example, when the operation represented by the requested state of the head quarter is invoked, the algorithm performs a backward chaining for any goal—primary or recovery goal—that is reachable on that alternative path. That is, the composer tries to reach the state failed of the head quarter in our example. Therefore, it tries to chain backwards for the goals in Table 8.1 on the facing page that contain the state failed of the head quarter.

Chaining the alternative path also produces copy rules which are merged with the copy rules from the chaining for the primary goal. The algorithm recurses as long as non-deterministic output alternatives are detected either in the chain for the primary goal or in any chain that needed to be generated for an output alternative.

The composition algorithm is described by a set of ASMs.[2] Each ASM represents a module of the algorithm. The dependencies of the modules are depicted in Figure 8.4 on the following page. The figure also groups the modules with respect to their purposes.

Based on the grouping, we structure the following sections. Section 8.4 reduces the problem of composing complex behavioral models to the composition of smaller units. The smaller units are then composed by the core composition algorithm highlighted in Section 8.5. We give only as much detail as needed to understand the idea. The details of the core composition algorithm are described in Appendix A. Section 8.6 explains how the algorithm ensures a correct composition by properly ordering the composition of the smaller units. The modules residing in the utility group of the picture are explained upon their first usage.

8.4 Dividing the composition problem

In this section, we show how to break down the composition problem into smaller pieces. The definition of these pieces bases on the different execution paths each behavioral model can possibly take to reach a goal. We call the set consisting of exactly one potential execution path of each participating behavioral model a variant. Figure 8.5 on page 121 displays a variant which leads to pg_1. We can define a domain to store variants as follows.

[2] ASMs were introduced in Section 7.2.1 on page 90.

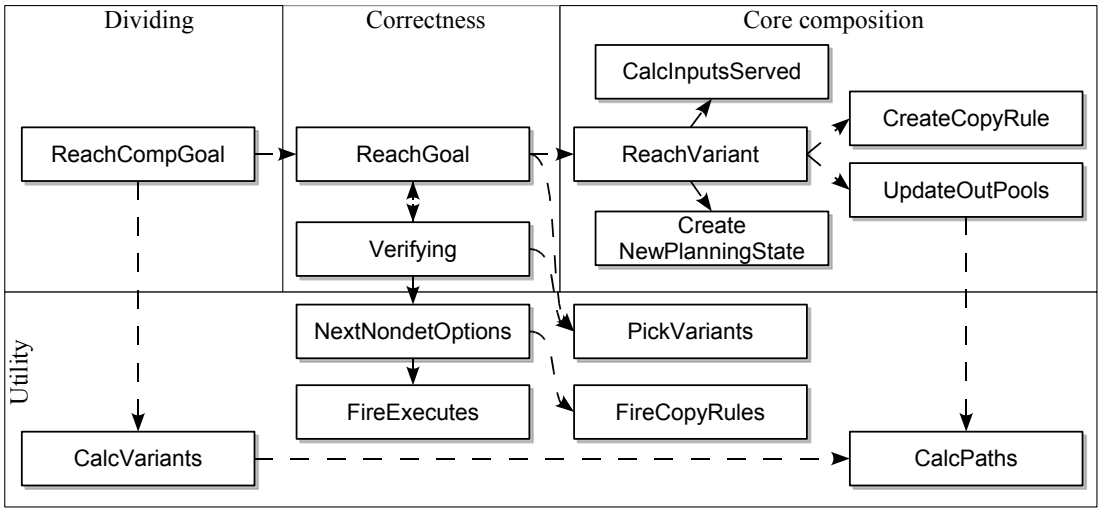


Figure 8.4: Invocation dependencies of ASMs. Each box represents an ASM, each arrow denotes the dependency of an ASM from another. The underlying boxes group the modules based on their purposes.

Definition 8.5 (Variant)**.**

$$
\begin{aligned}
\text{Variant} := \quad & variant: \\
& goal\ \mathcal{P}\ \mathcal{R} \quad , \Gamma = \mathcal{P}\ , \mathcal{R}\ , \\
& variant = \ (\ bm, Transs\ ):\ bm \quad \text{ID},\ Transs \subseteq \delta^{bm}, \\
& \quad (\ q_0^{bm},\ V_1,\ q_1\ ) \quad Transs, \quad \ldots\ (q_0^{bm} \text{ is initial state}) \\
& \quad (\ q_2,\ V_2,\ q_{\text{goal}}\ ) \quad Transs,\ (\ bm, q_{\text{goal}}\ ) \quad goal, \\
& \quad\ (\ q_3,\ V_3,\ q_4\ ) \quad Transs: \quad (\ q_4,\ V_4,\ q_5\ ) \quad Transs \quad = 1, \\
& \quad\ (\ q_7,\ V_5,\ q_8\ ) \quad Transs: \quad (\ q_6,\ V_6,\ q_7\ ) \quad Transs \quad = 1, \\
& \quad s \quad \text{Q}^{bm}:\ (\ \ q,\ V_7,\ q_9\ ) \quad Transs \leq \ 1, \\
& \qquad\qquad\quad (\ \ q_{10},\ V_8,\ q\ ) \quad Transs \leq \ 1 \\
& \quad , \\
& bm \quad \text{ID}:\ (\ bm, Transs\ ) \quad variant, \\
& variant \ = \text{ID}
\end{aligned}
$$

Please note that a variant always leads to exactly one goal.

The ASM starting the composition is called REACHCOMPGOAL. The purpose of the REACHCOMPGOAL machine is to initialize our composition algorithm. First, it identifies possible behavioral model executions (CALCVARIANTS) and hands them over to REACHGOAL. Second, it defines that the composition can only be successful if at least one primary goal can be achieved by providing the primary goals as the second

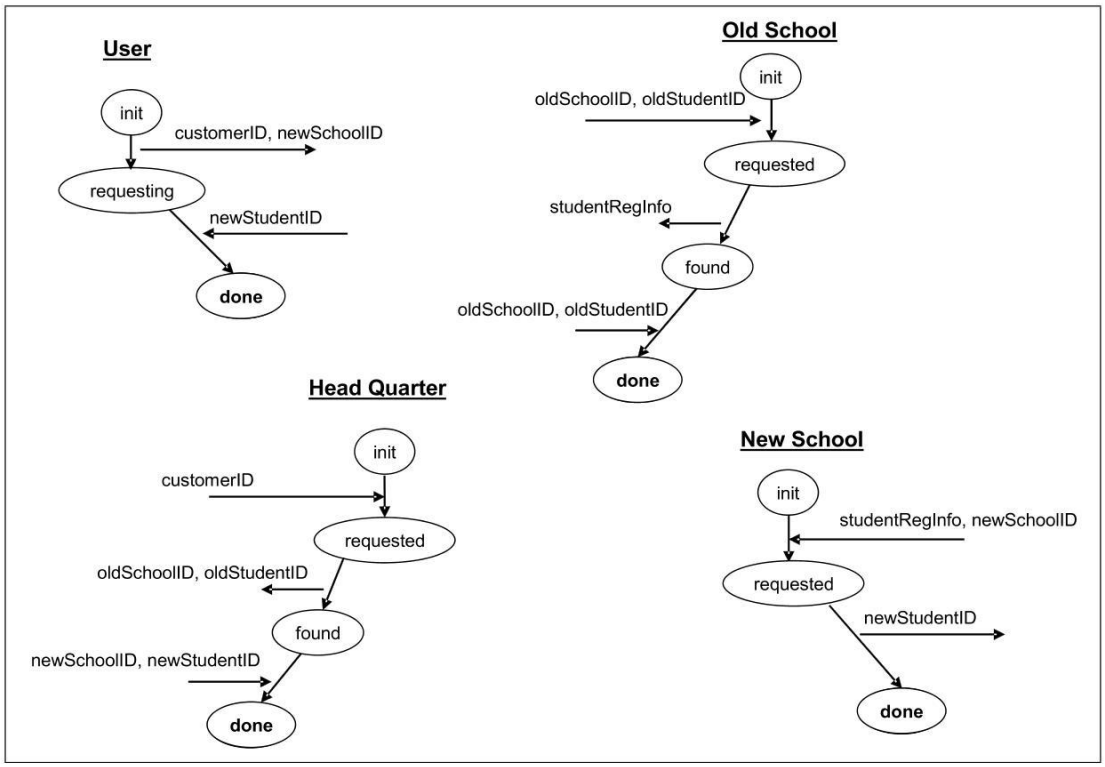


Figure 8.5: Variant leading to pg_1 in the student transfer example.

parameter to REACHGOAL. It also ensures that every possible execution of the resulting copy rules ends in one of the composition goals ( ) by assigning $allowedGoals$ as the fourth parameter to REACHGOAL. A more detailed examination of the parameters of REACHGOAL follows in the next section. If it is not possible to generate a correct orchestration for any of the primary goals, the result is the empty set. Third, since REACHGOAL is invoked recursively for some kind of simulation that is introduced later on, we need to keep track of the current state of the simulation and thus introduce the simulation state (SimState).

Definition 8.6 (Simulation state)**.**

$$\text{SimState} := 2^{\mathcal{B}}$$

The computation is started by calling REACHGOAL with the initial states of all behavioral models as a starting point ($initialSs$).

REACHCOMPGOAL(cg , A $\mathcal{A}$)
return $copyRules$ **in** **let**
$pgs = \text{primaryGoals}(cg)$,
$initialSs =$ bm $s :$ bm $\text{ids}(pg)$, pg pgs, $s = q_0^{bm}$
$allowedGoals = \text{primaryGoals}(cg)$ $\text{recoveryGoals}(cg)$

$$vnts = \bigcup_{goals \quad allowedGoals} \text{CALCVARIANTS}(goals) \textbf{ in}$$
$$(\, fail,\, copyRules \,) :=$$
$$\text{REACHGOAL}(vnts,\, pgs,\, initialSs,\, allowedGoals,\, A, initialSs)$$

The first activity of REACHCOMPGOAL is the calculation of all variants. A variant is computed as the cross product ($crossProduct$) of all possible execution paths of the behavioral models involved in a goal (g) to reach g.

$$\text{CALCVARIANTS}(g \quad \text{Goal}) \equiv$$
$$\textbf{return } crossProduct \textbf{ in } \textbf{ seq}$$
$$\textbf{forall } (\, bm,\, s \,) \quad g \textbf{ do } \text{paths}(bm) := \text{CALCPATHS}(\delta^{bm},\, s)$$
$$crossProduct \;:= \times_{bm \quad \text{ids}(g)} [\quad bm \times \quad \text{paths}(bm) \;]$$

The possible execution paths to reach a specific state (s) of a behavioral model (bm) are computed as follows: In the end, the paths location will contain a set of paths, where each path is a set of transition rules. In the beginning, the paths location is initialized with exactly one path consisting of the one transition that directly leads to the specified state. Now, those transitions directly leading to an existing transition in a $path$ in paths are iteratively added to that $path$. The calculation is performed as long as some paths grow. Therefore, we store the overall size of all paths (calcSumOfLengths) during the preceding iteration in oldSumOfLengths.

$$\text{CALCPATHS}(T \subseteq \delta^{bm},\, s \quad \text{Q}^{bm}) \equiv$$
$$\textbf{step}$$
$$\text{oldSumOfLengths} := 0$$
$$\text{paths} := \quad (\, q_{\text{pre}},\, V,\, q_{\text{post}} \,) \quad T :\; q_{\text{post}} = s$$
$$\textbf{step} \quad \textbf{while } \text{calcSumOfLengths(paths)} > \text{oldSumOfLengths } \textbf{do}$$
$$\text{oldSumOfLengths} := \text{calcSumOfLengths(paths)}$$
$$\textbf{forall } path \quad \text{paths } \textbf{do}$$
$$\textbf{step } \text{paths} := \text{paths} \quad path$$
$$\textbf{step} \quad \textbf{do} \quad \textbf{forall } rule \quad (\, q_{\text{pre}},\, V_1,\, q_{\text{pre}_{\text{ex}}} \,) \quad T :$$
$$(\, q_{\text{pre}_{\text{ex}}},\, V_2,\, q_{\text{post}_{\text{ex}}} \,) \quad path$$
$$\text{paths} := \text{paths} \quad path \quad rule$$
$$\textbf{step} \quad \textbf{result} \;:= \text{paths}$$
$$\textbf{where}$$
$$\text{calcSumOfLengths(paths)} \equiv$$
$$path_1 \;+\; path_2 \;+ \ldots +\; path_{\,\text{paths}}$$
$$path_x \quad \text{paths}, \quad x = 1 \,..\, \text{paths}$$

8.5 Core composition algorithm

In this section, we give a high-level explanation of the core composition algorithm (upper right part of Figure 8.4 on page 120). Due to its length, the formal ASM description of the core composition algorithm and all nested modules is not given here, but in Appendix A on page 201.

The core composition algorithm works iteratively from the final states of each behavioral model to their initial states. Therefore, we need to keep track of the current state of the backchaining and thus introduce the planning state ($\mathrm{PlState}$).

Definition 8.7 (Planning state)**.**

$$\mathrm{PlState} := 2^{\mathcal{B} \quad IN,OUT,\mathrm{undef}}$$

The planning state consists of a set where each member consists of two components: a behavioral model state ($\mathcal{B}$) and a mode (IN, OUT, undef). The state component can only contain a behavioral model state directly behind an output transition. The planning state referred to by that representation is the state of the behavioral model state component in the case of mode OUT, and the directly preceding state in the case of mode IN. The states in an exemplary behavioral model and in the corresponding $\mathrm{PlState}$ are displayed in Figure 8.6.

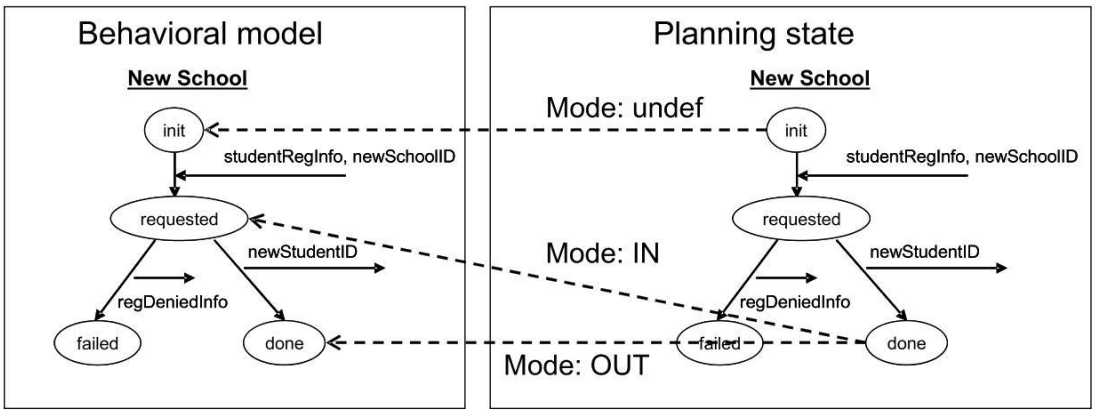


Figure 8.6: Planning state and corresponding state of the behavioral model.

An exception to the above is made for the final state of a behavioral model. The final state may as well be the content of the behavioral model state component. The mode in that case is OUT if the directly preceding transition is an output transition. The mode is IN in the case of a directly preceding input transition.

Another exception is that the behavioral model state component can take the initial state of a behavioral model (q_0^{bm}). In that case, the mode is undef. The planning state for a behavioral model represented by the behavioral model state component and the mode component equals the behavioral model state component in the exceptional cases.

The composition algorithm takes the following inputs:

- A variant of the possible behavioral model executions, i. e. a specific execution path for each participating behavioral model. For example, the variant shown in Figure 8.5 on page 121.
- An initial planning state derived from the given goal, for example, from pg_1.
- A set of possible variable assignments.

In our example, we used equal variable names in the different behavioral models to denote types that can be mapped on each other. Thus, for example,

$$(\,(\,\mathsf{User},\,\mathsf{customerID}\,),\,(\,\mathsf{HeadQuarter},\,\mathsf{customerID}\,)\,)$$

would be one of the variable assignments in Figure 8.5 on page 121. We explicitly list the two exceptions where we did not follow the convention of equal names.

$$(\,(\,\mathsf{HeadQuarter},\,\mathsf{custRegNA}\,),\,(\,\mathsf{User},\,\mathsf{regDeniedInfo}\,)\,)$$
$$(\,(\,\mathsf{OldSchool},\,\mathsf{regInfoNA}\,),\,(\,\mathsf{User},\,\mathsf{regDeniedInfo}\,)\,)$$

The general idea of the composition is to create copy rules for matching outputs and inputs of different behavioral models in the current planning state (ps) and to add them to the set $copyRules$ (CREATECOPYRULE). So, for example, in the initial call, the planning state corresponds to the primary goal pg_1. In that state, the copy rule c_{pg_1} is generated—a more exact description of generating copy rules follows later. After that has been done, the planning state proceeds toward the initial states of the behavioral models (CREATENEWPLANNINGSTATE)—let's call that state ps_1—and the algorithm reiterates. Thus, a copy rule c_{ps_1} is generated for ps_1, c_{ps_2} for ps_2, and so on.

The composition of a variant is aborted if no valid composition could be achieved ($fail$), the planning state consists of only initial states (done), or the composition came to a dead end, i. e., the planning state remained the same for two iterations. The latter case may occur if not all output variables of a behavioral model are consumed by other behavioral models. During composition, such a behavioral model's planning state will not proceed any further toward its initial state.

For the creation of the copy rules in the current planning state as highlighted above, some preliminary calculations have to be performed.

First, we identify all output variables of all behavioral models that are available for that variant ($\mathrm{outPool}$). As an example, we assume that planning is still in at the state corresponding to the primary goal pg_1. We give the $\mathrm{outPools}$ of the variant in Figure 8.5 on page 121 below.

Behavioral model	$\mathrm{outPool}$
User	customerID, newSchoolID
OldSchool	studentRegInfo
HeadQuarter	oldSchoolID, oldStudentID
NewSchool	newStudentID

Second, we identify all input transitions of all behavioral models that directly lead to the current planning state ($\mathrm{adjInTrans}$).

> Note that for one behavioral model there is exactly one such transition because the calculation bases on a variant.

For the planning state corresponding to the primary goal pg_1, the direct input transitions are the following.

	$\mathrm{adjInTrans}$				
Behavioral model	q_1	Q^{bm}	inputs	q_2	Q^{bm}
User	requesting		newStudentID	done	
OldSchool	found		oldSchoolID, oldStudentID	done	
HeadQuarter	found		newSchoolID, newStudentID	done	
NewSchool			$\varnothing$		

Third, we match all inputs of the identified input transitions with available outputs (CALCINPUTSSERVED). The correspondences for that matching are taken from the given, possible variable assignments (A). In our example, all input variables in $\mathrm{adjInTrans}$ can be served. Therefore, we create the copy rule below. As a state component of the copy rule, we take the states from the current planning state—in our case, the planning state corresponds to pg_1—with the exception that for the behavioral models whose planning state was directly behind an input transition, we take the state before the input transition. For brevity, we refer to the names of behavioral models by their first letter, only.

$$
\begin{aligned}
c_{pg_1} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{done})\ , \\
& ((\mathsf{N}, \mathsf{newStudentID}), (\mathsf{U}, \mathsf{newStudentID})), \\
& ((\mathsf{H}, \mathsf{oldSchoolID}), (\mathsf{O}, \mathsf{oldSchoolID})), \\
& ((\mathsf{H}, \mathsf{oldStudentID}), (\mathsf{O}, \mathsf{oldStudentID})), \\
& ((\mathsf{U}, \mathsf{newSchoolID}), (\mathsf{H}, \mathsf{newSchoolID}))\ ), \\
& ((\mathsf{N}, \mathsf{newStudentID}), (\mathsf{H}, \mathsf{newStudentID}))
\end{aligned}
$$

The rationale for the computation of the state component is that the state in the copy rule now describes the global system state at which the variables are actually copied during execution. Namely, execution of the copy rule must occur

- *after* all outputs are available—therefore, we take the planning state when it is behind an output transition—and
- *before* inputs can be received—therefore, we take the state before the planning state when it is behind an input transition.

After creating the copy rule, the new planning state is computed based on which output still need to be consumed. The planning state for input transitions is always advanced toward the initial state. Therefore, the new planning state ps_1 now refers to the following behavioral model states.

(U, requesting), (O, found), (H, found), (N, requested)

Finally, we update the $\mathrm{outPools}$ in order to only contain all output variables that will be consumed at a later stage of the composition—i. e., by inputs before the current planning state. We give the updated $\mathrm{outPools}$ for our example below.

Behavioral model	$\mathrm{outPool}$
User	customerID, newSchoolID
OldSchool	studentRegInfo
HeadQuarter	oldSchoolID, oldStudentID
NewSchool	$\varnothing$

The subsequent iterations of the composition would, for our example, result in the following copy rules before the composer reaches all behavioral models' initial states.

$$
\begin{aligned}
c_{ps_1} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{O}, \mathsf{studentRegInfo}), (\mathsf{N}, \mathsf{studentRegInfo})), \\
& ((\mathsf{U}, \mathsf{newSchoolID}), (\mathsf{N}, \mathsf{newSchoolID})) \ ); \\
c_{ps_2} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{H}, \mathsf{oldSchoolID}), (\mathsf{O}, \mathsf{oldSchoolID})), \\
& ((\mathsf{H}, \mathsf{oldStudentID}), (\mathsf{O}, \mathsf{oldStudentID})) \ ); \\
c_{ps_3} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{init}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{U}, \mathsf{customerID}), (\mathsf{H}, \mathsf{customerID})) \ )
\end{aligned}
$$

Please note that the four generated copy rules already constitute an orchestration for the behavioral models in Figure 8.3 on page 115 for the case that all operations respond as depicted in Figure 8.5 on page 121.

We manage the cases when not all operations respond as desired in the following section.

8.6 Computing correct orchestrations

For one variant, the creation of copy rules can be achieved by our core composition algorithm (REACHVARIANT), which was explained in the previous section. The copy

rules created by REACHVARIANT ensure that the given goal can be reached in that variant. As we have demonstrated with our example, the first call to REACHVARIANT creates a set of copy rules to orchestrate the behavioral models if their executions would behave according to the variant shown in Figure 8.5 on page 121.

Due to potential non-deterministic behavior of the participating behavioral models, it may happen that the execution of the orchestration leaves one of the behavioral models' path along the variant, or even leave the path to its final state that is part of the defined goal. For example, the customer ID sent from the user to the head quarter could be declined be the head quarter.

The result of our composition has to ensure that in such a case an alternative path is taken that leads to any other desired final state. That would, for example, include informing the user of the failure. That is ensured by VERIFYING. With the high-level understanding, we step one level up and go in detail through the implementation of REACHGOAL (compare Figure 8.4 on page 120), which internally calls REACHVARIANT described in the last section. Second, we explain VERIFYING—another constituent of REACHGOAL—and third, we detail the simulation of the created copy rules that is part of VERIFYING.

8.6.1 Reach goal

The aim of REACHGOAL is to return copy rules ensuring a correct orchestration for at least one of the given $goals$ only considering the given variants ($vnts$). For that, it first identifies all variants ($goalVnt$) that lead to the $goals$ (CALCVARIANTS). Second, it tries to compose each of the variants (REACHVARIANT). That results in some copy rules (regCopyRules). Third, the algorithm creates copy rules (altCopyRules) for each non-deterministic branch in the theoretic execution of regCopyRules (VERIFYING). The created copy rules either provide a correct orchestration of that branch, or VERIFYING fails (altFail). If a correct orchestration could be generated for at least one variant in the end, the corresponding copy rules ($oneVariantCopyRules$) are finally returned.

REACHGOAL($vnts \subseteq$ Variant, $mandatGoals \subseteq$ Goal, ss SimState,
$allowedGoals \subseteq$ Goal, $A \subseteq \mathcal{A}$, $startstate$ SimState) $\equiv$ **return**
($fail$, $copyRules$) **in**
if $mandatGoals = \varnothing$ **then** $fail :=$*false* **par** $copyRules := \varnothing$
else
step $fail :=$*true* **par** $copyRules := \varnothing$ **par** variantCopyRules $:= \varnothing$
step do forall $goalVnt$ PICKVARIANTS($vnts$, ss, $mandatGoals$)
step (regFail, regCopyRules) :=
REACHVARIANT($goalVnt$, finState($goalVnt$), A, $startstate$)
step if not regFail **then**
step (altFail, altCopyRules) :=
VERIFYING($vnts$, $goalVnt$, regCopyRules, $startstate$, $allowedGoals$, A)
step if not altFail **then**

$fail := false$
variantCopyRules := variantCopyRules
filterRules(regCopyRules , altCopyRules)
step **if** variantCopyRules = ∅ **then**
choose $oneVariantCopyRules$ variantCopyRules
$copyRules := oneVariantCopyRules$
where
filterRules(C_1, C_2) := c : c_1 C_1, c_2 C_2,
$$c = \begin{cases} c_2, & c_1 : \text{states}(c_1) = \text{states}(c_2) \\ c_1, & \text{otherwise} \end{cases}$$
finState(vnt Variant) := (bm, s) : (q_{pre}, V, q_{post}) T,
(bm, T) vnt, (q_{post}, V, s_{next}) T

As an optimization, not all variants ($vnts$) are considered during computation, but only those ($pickedVnts$) that pass the given state (ss) and lead to the given $goal$.

PICKVARIANTS($vnts \subseteq$ Variant, ss SimState, $goals \subseteq$ Goal) ≡
return $pickedVnts$ **in**
$pickedVnts$:= vnt $vnts$: $path$ vnt, $bm = id(path)$,
$F = \text{transs}(path)$, (q_{pre_1}, V_1, q_{post_1}) F, (bm, q_{pre_1}) ss,
(q_{pre_2}, V_2, q_{post_2}) F, (bm, q_{post_2}) $goal$, $goal$ $goals$

Both REACHVARIANT and VERIFYING are contained in REACHGOAL, which was just presented. We discussed REACHVARIANT as part of Section 8.5 on page 123. In the following section, we present VERIFYING.

8.6.2 Verifying

Through REACHVARIANT in REACHGOAL, we ensure that a composition can be generated that steers the execution along the specific variant as discussed. However, the path of execution may depend on the non-deterministic behavior of other behavioral models that cause a deviation from the path. For that case, VERIFYING ensures that there exists a successful composition for each non-deterministically deviating path. The result of VERIFYING is either the set of copy rules that ensure the successful composition or a notification of failure if no successful composition exists for all non-deterministic deviations.

We now detail the functioning of VERIFYING. In order to give the full picture, we link our description to REACHGOAL where necessary. The overall process is schematically depicted in Figure 8.7 on the next page.

First, REACHGOAL tries to reach a variant (REACHVARIANT).

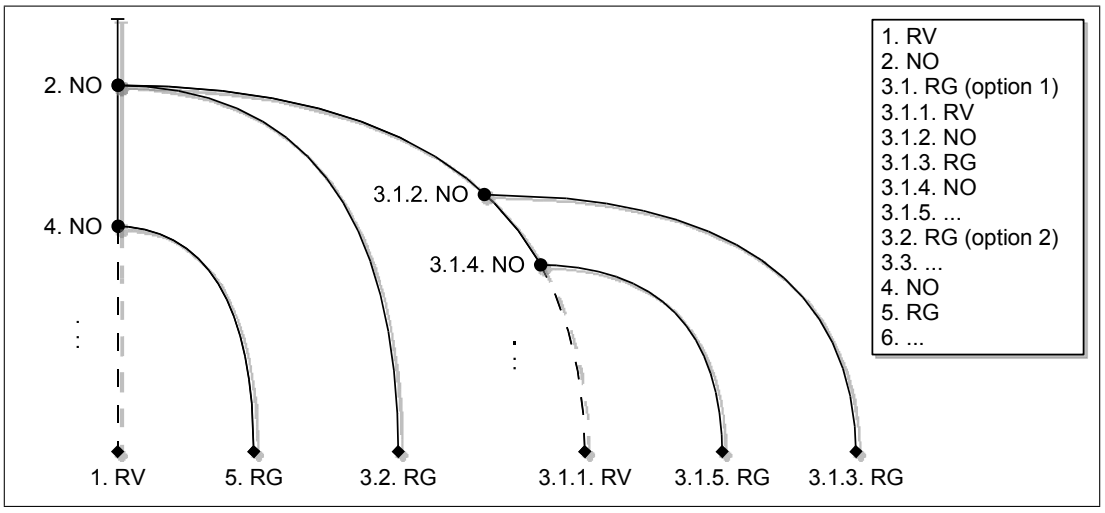


Figure 8.7: Recursive computation of REACHGOAL. The abbreviations represent the modules REACHVARIANT (RV), NEXTNONDETOPTIONS (NO), and REACHGOAL (RG). A circle or diamond next to an abbreviation denotes the simulation state that was the input or the output of the respective module. The perpendicular line end denotes the initial simulation state. Each circle denotes a simulation state prior to some non-deterministic options. Each diamond denotes a goal. A solid line represents copy rules ensuring a correct, partial orchestration from its upper to its lower simulation state. A dashed line stands for copy rules to be elaborated on. The legend on the right-hand side presents the sequence of module invocations corresponding to the picture.

Second, VERIFYING simulates the execution of the given copy rules (cr) starting from the given state ss (NEXTNONDETOPTIONS). The simulation stops at the first point of non-determinism and returns all different, non-deterministic options that can occur at the current point of execution ($\mathrm{options}$).

Simulating the copy rules generated in our example, we find out that the first non-determinism occurs in the head quarter's behavioral model after executing c_{ps_1}. In particular, the operation represented by the state requested of head quarter in Figure 8.3 on page 115 may result in fail instead of the desired state found. Thus, the non-deterministic option is

$$(\ \mathsf{U},\ \mathsf{requesting}\ ),\ (\ \mathsf{O},\ \mathsf{init}\ ),\ (\ \mathsf{H},\ \mathsf{failed}\ ),\ (\ \mathsf{N},\ \mathsf{init}\ ).$$

Third, our objective implies that there must be a successful composition for each of the options. The different options are depicted in Figure 8.7 by the multiple lines leaving "2. NO." Since each option may be reached through different variants ($\mathrm{optionVnts}$), we need to ensure that there exists a successful composition for at least one of the variants for each option ($optionVnt$).

Ensuring successful composition for a variant is exactly the objective of the ASM REACHGOAL presented in Section 8.4 on page 119. Such a call is represented by "3.2. RG" in the figure. In contrast to the initial call of REACHGOAL, we now only care about reaching one of our $allowedGoals$. We therefore provide $allowedGoals$ as second and fourth parameter of REACHGOAL. In our example, the allowed goals reachable from

(U, requesting), (O, init), (H, failed), (N, init)

are rg_8, rg_{10}, rg_{12}, rg_{26}, rg_{28}, and rg_{30}. These are all goals from Table 8.1 on page 118 which are "execution-wise behind" the option and where head quarter is failed.

$allowedGoal_1 = rg_8 =$ (U, failed), (O, init), (H, failed), (N, init)
$allowedGoal_2 = rg_{10} =$ (U, failed), (O, failed), (H, failed), (N, init)
$allowedGoal_3 = rg_{12} =$ (U, failed), (O, found), (H, failed), (N, init)
$allowedGoal_4 = rg_{26} =$ (U, failed), (O, init), (H, failed), (N, failed)
$allowedGoal_5 = rg_{28} =$ (U, failed), (O, failed), (H, failed), (N, failed)
$allowedGoal_6 = rg_{30} =$ (U, failed), (O, found), (H, failed), (N, failed)

Also, as an optimization, we want to restrict the variants to be considered by REACHGOAL to the variants relevant for the current option ($\mathrm{optionVnts}$). Finally, we provide the simulation state of the non-deterministic option ($option$) as the starting state for REACHGOAL.

For our example, REACHGOAL finds out that composition is possible only for rg_8. The respective variant is shown in Figure 8.8 on the next page. We present the resulting copy rules for the variant below.

$c_{rg_8} = ($ (U, requesting), (O, init), (H, failed), (N, init) ,
((H, custRegNA), (U, regDeniedInfo)));
$c_{ps_4} = ($ (U, requesting), (O, init), (H, init), (N, init) ,
((U, customerID), (H, customerID)))

Please note that the call to REACHGOAL is recursive. That is depicted by the labels starting with "3.1." in Figure 8.7.

For our example, the simulation of the copy rules above reveals no more non-determinism.

In the case that REACHGOAL was successful, we collect the copy rules generated ($\mathrm{optionCopyRules}$) in the return variable $copyRules$.

Fourth, we continue the original simulation up to the next point of non-determinism deviating from the original variant (vnt). That would, in our example, be the operation

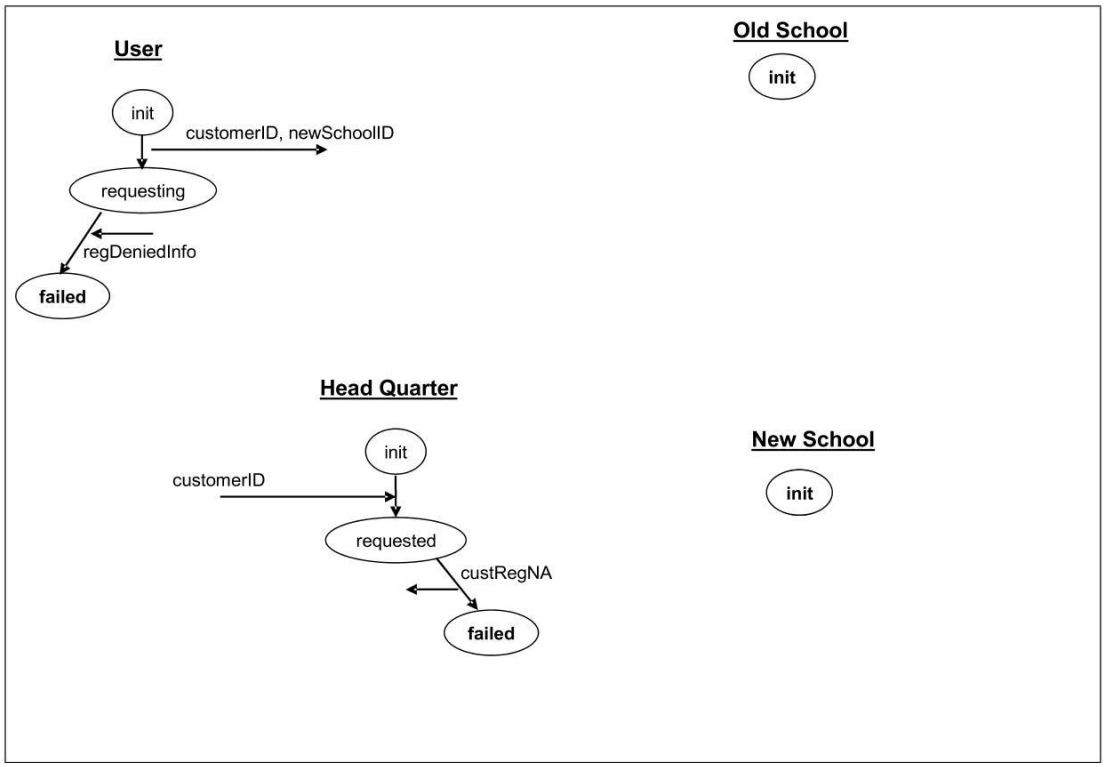


Figure 8.8: Variant leading to rg_8 in the student transfer example.

represented by state requested of old school because our copy rules for the primary goal—presented in Section 8.5 on page 123—first ask the head quarter for the customer data and second gather enrollment information of that customer from old school. The respective non-deterministic option is

$$(\,\mathsf{U},\,\mathsf{requesting}\,),\,(\,\mathsf{O},\,\mathsf{failed}\,),\,(\,\mathsf{H},\,\mathsf{found}\,),\,(\,\mathsf{N},\,\mathsf{init}\,)\ .$$

The allowed, reachable goals are rg_{16} and rg_{34}.

$$\begin{aligned} allowedGoal_7 = rg_{16} = &\ (\,\mathsf{U},\,\mathsf{failed}\,),\,(\,\mathsf{O},\,\mathsf{failed}\,),\,(\,\mathsf{H},\,\mathsf{found}\,),\,(\,\mathsf{N},\,\mathsf{init}\,) \\ allowedGoal_8 = rg_{34} = &\ (\,\mathsf{U},\,\mathsf{failed}\,),\,(\,\mathsf{O},\,\mathsf{failed}\,),\,(\,\mathsf{H},\,\mathsf{found}\,),\,(\,\mathsf{N},\,\mathsf{failed}\,) \end{aligned}$$

Fifth, each non-determinism found is elaborated as described before. The process continues for all non-deterministic deviations from the original variant.

From the above goals in our example, only rg_{16} can be reached. The respective variant is displayed in Figure 8.9 on the next page. We give the respective copy rules

below.

$$c_{rg_{16}} = (\ \ (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{failed}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init})\ \ ,$$
$$((\mathsf{O}, \mathsf{regInfoNA}), (\mathsf{U}, \mathsf{regDeniedInfo}))\ \ );$$
$$c_{ps_5} = (\ \ (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init})\ \ ,$$
$$((\mathsf{H}, \mathsf{oldSchoolID}), (\mathsf{O}, \mathsf{oldSchoolID})),$$
$$((\mathsf{H}, \mathsf{oldStudentID}), (\mathsf{O}, \mathsf{oldStudentID}))\ \ );$$
$$c_{ps_6} = (\ \ (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{init}), (\mathsf{N}, \mathsf{init})\ \ ,$$
$$((\mathsf{U}, \mathsf{customerID}), (\mathsf{H}, \mathsf{customerID}))\ \ )$$

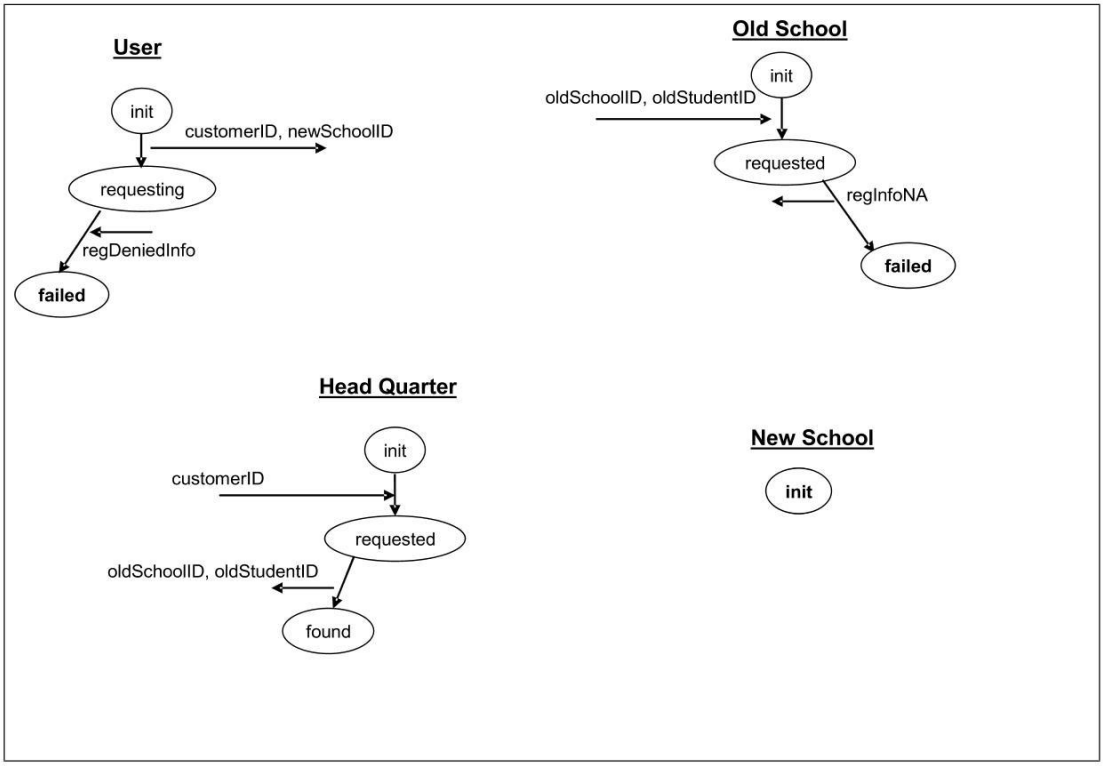


Figure 8.9: Variant leading to rg_{16} in the student transfer example.

Simulating the copy rules above yields no more non-determinism. Thus, we continue the simulation of the original copy rules and find the last non-deterministic option

$$(\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{failed})\ .$$

The only reachable, allowed goal is rg_{36}. We give the copy rules resulting from its composition below. The variant is displayed in Figure 8.10 on the facing page. The copy rules for that option do not contain any new non-determinism.

$$allowedGoal_9 = rg_{36} = \ \ (\mathsf{U}, \mathsf{failed}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{failed})$$

$$
\begin{aligned}
c_{rg_{36}} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{failed}) \ , \\
& ((\mathsf{N}, \mathsf{regDeniedInfo}), (\mathsf{U}, \mathsf{regDeniedInfo})) \ ); \\
c_{ps_7} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{found}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{O}, \mathsf{studentRegInfo}), (\mathsf{N}, \mathsf{studentRegInfo})), \\
& ((\mathsf{U}, \mathsf{newSchoolID}), (\mathsf{N}, \mathsf{newSchoolID})) \ ); \\
c_{ps_8} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{found}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{H}, \mathsf{oldSchoolID}), (\mathsf{O}, \mathsf{oldSchoolID})), \\
& ((\mathsf{H}, \mathsf{oldStudentID}), (\mathsf{O}, \mathsf{oldStudentID})) \ ); \\
c_{ps_9} = (\ & (\mathsf{U}, \mathsf{requesting}), (\mathsf{O}, \mathsf{init}), (\mathsf{H}, \mathsf{init}), (\mathsf{N}, \mathsf{init}) \ , \\
& ((\mathsf{U}, \mathsf{customerID}), (\mathsf{H}, \mathsf{customerID})) \ )
\end{aligned}
$$

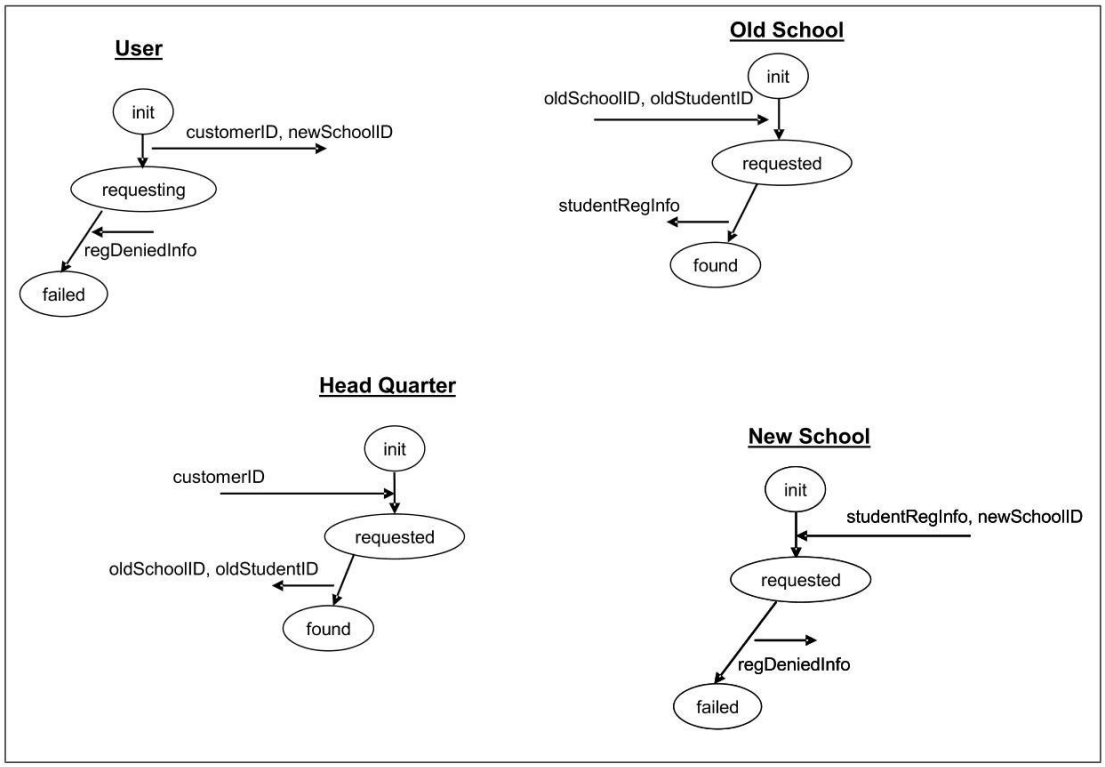


Figure 8.10: Variant leading to rg_{36} in the student transfer example.

Finally, the verifying is done when the simulation stagnates ($\mathrm{oldss} = ss$) or the generation of alternative copy rules fails ($\mathrm{globalFail}$). Stagnation may happen when simulation reached an $allowedGoal$. In that case, we return the collected $copyRules$. In every other case of stagnation and in any case of failure, we return an empty set of $copyRules$ and a failure notification ($fail$).

```
VERIFYING(vnts ⊆ Variant, vnt  Variant, cr ⊆ C, ss  SimState,
    allowedGoals ⊆ Goal, A ⊆A  ) ≡  return ( fail, copyRules ) in
  step
    oldss := ∅
    globalFail :=false
  step  while oldss = ss and  not globalFail do
    step
      oldss := ss
      ( beforeOpts, ss, options ) := NEXTNONDETOPTIONS(vnt, cr, ss)
    step  if oldss = ss then  do  forall option  options
      step optionVnts := PICKVARIANTS(vnts, option, allowedGoals)
      step
        if optionVnts = ∅ then
          step  ( optionUncomposable, optionCopyRules ) :=
              REACHGOAL(optionVnts, allowedGoals, option,
                  allowedGoals, A, beforeOpts)
          step
            copyRules := copyRules  optionCopyRules
            if optionUncomposable then globalFail :=true
        else globalFail :=true
  step
    if  not globalFail and ss  allowedGoals then fail :=false
    else
      fail :=true
      copyRules := ∅
```

At that stage, the algorithm has ensured that the primary goal for the example (pg_1) could be reached and there exist deterministic resolutions for each non-deterministic deviation from the intended execution path to an allowed recovery goal. Therefore, we can claim that the example can be successfully composed. The copy rules returned by our algorithm contain the copy rules for reaching the primary goal and for all non-deterministic deviations from the intended path, i. e., all copy rules shown in this section.

8.6.3 Simulation

The NEXTNONDETOPTIONS machine performs a simulation of the current copy rules in order to determine the next non-determinism in their application on a real, collaborative behavioral model execution starting from simulation state ss. That functionality was assumed in the previous section, but not explained in detail.

The next non-determinism is identified by simulating the firing of all ADVANCEBM machines of all behavioral models alternated with applying the copy rules (FIRECOPYRULES).

NEXTNONDETOPTIONS(vnt Variant, $cr \subseteq$ C,
ss SimState) $\equiv$ **return** (beforeOpts, ss, options) **in**
step oldss := ss **par** options := $\varnothing$
step (beforeOpts, ss, $dummy$) :=
FIREEXECUTES(vnt, ss, cr, $\varnothing$) // start initiator
step **while** oldss = ss **and** options = $\varnothing$ **do**
oldss := ss
step $inputs$:= FIRECOPYRULES(ss, cr) // these should be deterministic
step (beforeOpts, ss, options) := FIREEXECUTES(vnt, ss, cr, $inputs$)

The main task when simulating copy rules is to identify the inputs that are served by the copy rules applicable in the current simulation state (ss).

FIRECOPYRULES(ss SimState, $cr \subseteq$ C) $\equiv$ **return** $inputs$ **in**
$inputs$:= (bm_{in}, i) : ((bm_{out}, o), (bm_{in}, i))
assignments($copyRule$), ss = states($copyRule$), $copyRule$ cr

During simulating the ADVANCEBM rules of all behavioral models, we identify non-deterministic options as follows: First, we advance the simulation state for all input transitions with all inputs served. Second, we examine all the states (S $ndStates$) for each behavioral model with non-deterministic branches that can be directly reached at the current state of simulation (ss) and that do not appear in the transitions (T) of the current variant (vnt). Third, we collect the states of the behavioral models without non-determinism ($detStates$). Fourth, we calculate all non-deterministic deviations from the current variant (vnt) by creating the cross product of the states in $ndStates$ and $detStates$. Finally, we calculate the simulation state (ss) for each behavioral model that has to be evaluated by the verification after the non-deterministic options were checked. We set the next state to the directly following simulation state (q_{post}) of the current variant (vnt) if there are no active transition alternatives directly following the current simulation state. A transition is active in the following cases:

- It is an output transition.
- It is an input transition and all of its input variables can be served.

FIREEXECUTES(vnt Variant, ss SimState,
$cr \subseteq$ C, $inputs \subseteq$ Variable) $\equiv$ **return** (beforeOpts, ss, $options$) **in**
step ss := (bm, s) : (bm, q_{ss}) ss, (bm, T) vnt, t T,
t = (q_{ss}, V, q_{post}),
$$s = \begin{cases} q_{\text{post}}, & V \subseteq \Sigma^{bm}, \quad i \quad V : (\, bm, i\,) \quad inputs \\ q_{\text{ss}}, & \text{otherwise} \end{cases}$$
step
$ndStates$:= (bm, S) : (bm, q_{ss}) ss, (bm, T) vnt,

$t_{\text{vnt}}\quad T, t_{\text{nd}}\quad \delta^{bm},$
$t_{\text{vnt}} = (\, q_{\text{ss}}, O_{\text{vnt}}, q_{\text{post}_{\text{vnt}}}\,), t_{\text{nd}} = (\, q_{\text{ss}}, O_{\text{nd}}, q_{\text{post}_{\text{nd}}}\,),$
$q_{\text{post}_{\text{vnt}}} = q_{\text{post}_{\text{nd}}}, q_{\text{post}_{\text{nd}}}\quad S, O_{\text{vnt}}, O_{\text{nd}}\quad \text{Z}^{bm}$
$detStates :=\quad (\, bm, S\,) : (\, bm, q_{\text{ss}}\,)\quad ss, (\, bm, T\,)\quad vnt,$
$t_{\text{vnt}}\quad T, t_{\text{vnt}} = (\, q_{\text{ss}}, V_{\text{vnt}}, q_{\text{post}_{\text{vnt}}}\,),$
$[\, V_{\text{vnt}}\quad \Sigma^{bm}, q_{\text{ss}}\quad S\,]\quad [\, V_{\text{vnt}}\quad \text{Z}^{bm},\quad t_{\text{nd}}\quad \delta^{bm},$
$t_{\text{nd}} = (\, q_{\text{ss}}, O_{\text{nd}}, q_{\text{post}_{\text{nd}}}\,), q_{\text{post}_{\text{vnt}}} = q_{\text{post}_{\text{nd}}}, q_{\text{post}_{\text{vnt}}}\quad S\,]$

step

$options := \times_{(\, bm, S\,)}\quad (ndStates\quad detStates)\,[\quad bm \times\quad S\,]$
$ss :=\quad (\, bm, s\,) : (\, bm, q_{\text{ss}}\,)\quad ss, (\, bm, T\,)\quad vnt, t\quad T,$
$t = (\, q_{\text{ss}}, V, q_{\text{post}}\,), s = \begin{cases} q_{\text{post}}, & V \subseteq \text{Z}^{bm} \\ q_{\text{ss}}, & \text{otherwise} \end{cases}$
$\text{beforeOpts} :=\quad (\, bm, s\,) : (\, bm, q_{\text{ss}}\,)\quad ss, (\, bm, T\,)\quad vnt, t\quad T,$
$t = (\, q_{\text{ss}}, V, q_{\text{post}}\,),$
$s = \begin{cases} q_{\text{post}}, & V \subseteq \text{Z}^{bm}\quad (\, bm, S_{\text{det}}\,)\quad detStates \\ q_{\text{ss}}, & \text{otherwise} \end{cases}$

8.7 Proving requirements

In this section, we prove the requirements defined in Section 8.2 on page 116 and in Section 7.7 on page 109.

Theorem 8.8 (Termination)**.** *The composition terminates.*

Proof. The composer works on a set of trees. The planning state is initialized as a set of states, one in each tree. For each tree, the "initial state" is assumed to be some state on the path from the root to the state that is part of the planning state. In each composition step, at least one tree's planning state is advanced toward the root state of that tree. Otherwise, the composition terminates.

When the composer reaches or passes the initial state, it terminates. When the composer has reached the initial state, composition was successful. If the composer passes the initial state, composition failed. Thus, a composer run that does not hit non-deterministic branching points terminates.

When reaching a non-deterministic branching point, the composer recurses. During recursion, only branches that were not considered so far are considered in the recursive call. As the considered state transition systems are finite, there is a finite number of non-deterministic branches. Therefore, each recursion reduces the number of branches to consider. A recursion that only has to consider a single branch terminates as stated above. Thus, the complete algorithm terminates. □

Theorem 8.9 (Correctness)**.** *The composer is correct. Correctness consists of three sub-properties:*

1. *Each orchestration contains one path to a primary goal.*
2. *Each orchestration only reaches primary or recovery goals.*
3. *The composer maintains the sequence of operations as de ned in the participating behavioral models.*

Proof for primary goal. The composition is initialized to start from the primary goal and generates copy rules while proceeding backwards until reaching or passing the "initial state." The composition only terminates when the initial state is reached or passed.

In the case of passing without reaching, the composition fails and does not generate an orchestration. In the case of reaching the initial state, the composer succeeds and outputs the set of copy rules that were generated during computation. As the composer started from a primary goal, the so far generated copy rules describe a path to the primary goal. Thus, each generated orchestration always contains a path to a primary goal. □

Proof for always reaching some goal. As proven above, an orchestration always contains a path to a primary goal when the composer succeeds. When a non-deterministic branch is detected that would be hit by the so far generated orchestration, the composer recurses. For recursion, the composer is configured as for reaching a primary goal, but the parameter that was formerly initialized with the primary goal is in the case of recursion the set of all primary and recovery goals. Therefore, as before, a recursion will either fail if no orchestration could be found for the recursion or the generated copy rules will describe an execution to reach one of the given primary or recovery goals. Thus, when the composer generates an orchestration, each execution always either reaches a primary or recovery goal under the assumption that the participating implementations follow the contract of their respective behavioral model. □

Proof for proper sequencing. The composer operates on a set of given behavioral models. Whenever the planning state is advanced during composition, each behavioral model is traversed toward its root. Each generated copy rule always contains a state for each behavioral model according to the current planning state. Therefore, each set of copy rules generated guarantees a proper sequencing of operations belonging to the same behavioral model. □

Part III

Application and Evaluation

Chapter 9

Determining Redundancy of SAP ESR Message Types

As expressed in Table 9.1, we prove the applicability of the mining component to an industrial setting by exposing it to the enterprise SOA landscape of SAP.

Table 9.1: Proof of concepts

Requirement	**Solution**	**Proof of concept**
Detect redundant interface objects	CFIM of hierarchical types	CFIM of message types exposed by SAP

This chapter is structured as follows: Section 9.1 introduces SAP's implementation of SOA—called enterprise SOA,—which is the basis for our evaluation. Section 9.2 highlights the objectives of governance within SAP's enterprise SOA. The challenges of governance in enterprise SOA to be tackled by our CFIM of hierarchical types are explained in Section 9.3. Section 9.4 describes how our solution reduces the described challenges and Section 9.5 assesses the quality of CFIM of hierarchical types.

9.1 SAP s enterprise SOA

In 2004, SAP committed to deliver on its enterprise SOA roadmap. The basis for enterprise SOA is the business process platform. The business process platform consists of the technology platform and the application platform. The technology platform provides a technical infrastructure to manage a service-oriented architecture (SOA). The application platform provides predefined business content. Such content are best practice building blocks to define custom business processes. The interface objects used in the application platform are centrally managed in the enterprise service repository (ESR). The enterprise services repository consist of a design time and a run time

repository. The design time repository contains the interface objects needed to create Web service interfaces. The run time repository is a UDDI repository. As we are mostly concerned with the design time view, we use the term enterprise service repository also to refer to the design time repository.

On top of enterprise SOA, a specialized business solution may be developed to address the very specific needs of one company or a set of companies in a specific business domain. Such a solution is SAP Business ByDesign, which is targeted to mid-size companies. As one of its specialties, SAP Business ByDesign hides much of the capability actually provided by the business process platform and thus reduces the complexity the SAP Business ByDesign customer has to cope with.

9.2 Governance

With the development of enterprise SOA, SAP has established an internal governance process for the application platform to

1. ensure that newly added interface objects do not resemble existing objects, and

2. detect overlapping interface objects in the repository and propose a realignment to the development organization.

Governance is only relevant to parts of a software that end up in the application platform. Therefore, in the beginning of every software development project, it needs to be decided whether the software contains parts that are relevant for more than one customer. If so, the relevant parts become integrated into the application platform, and thus have to pass the governance process.

The governance process is based on the exchange of governance documents. For example, there is a separate governance document for creating

- a data type and

- a Web service operation.

The developer creates a governance document and hands it over to the governance team. The governance team responds within a couple of weeks with an approval or a request for change.

Among the two initially identified activities of the governance process, our approach supports the second, i. e., “detect overlapping interface objects.” Therefore, we explain both activities in little more detail—with an emphasis on the second activity,—and we highlight its challenges before we present how our approach helps to reduce the burden of the challenges.

9.3 Challenges of realignment

The decision of the governance team depends on the similarity of the interface object described in the governance document to existing interface objects in the enterprise service repository. Today, that is a manual endeavor. With their expertise in their mind, the governance expert browses the enterprise service repository with the aim to find out that either

- some existing interface object has some overlap with the proposed one or
- the proposed interface object is distinct from all others.

That decision is taken on both the ontological and the technical level. If the proposed interface object is distinct from all others, the governance expert acknowledges the application. If there is some overlap, they either

- request the interface object to be changed or
- acknowledge the interface object application and request the overlapping interface object to be changed.

The decision whether other interface objects than the one applied for need change is complicated. The reason is that not only the interface object that was applied for and the one it overlaps with influence the decision, but also other relevant interface objects in the repository need to be kept consistent with the, for example, new modeling convention just introduced. We describe how our approach supports that task in the following section.

9.4 Reducing the challenges using our approach

For the proof of concept, we implemented a prototype for structural analysis. The prototype consists of the components

- WSDL exporter,
- WSDL loader,
- Mining configuration, and
- Frequent itemset mining for structural types.

The WSDL exporter connects to the enterprise service repository of an SAP system and reads the WSDL files of the public Web service interfaces from the repository.

The WSDL loader is capable of reading WSDL files and creating a first internal representation. For the tests, the WSDL loader was configured to read the message types from the WSDLs.

The miner configuration component presents the user with a graphical user interface to pick from options that influence the conversion from the internal representation to the mining database.

The closed frequent itemset mining component bases on the concepts described by Uno et al. (2004a;b). The mining component takes multiple XML schema definitions as input and creates a set of redundancy groups as a result.

9.4.1 Evaluation configurations

The miner configuration component allows for configurations of three categories (see GUI in Figure 9.1 on the facing page):

1. WSDL files to analyze,
2. granularity of analysis, and
3. populating the mining database.

All WSDL files to be analyzed must reside in a specific directory on the computer. The choice in the first category determines whether

- all files will be analyzed at once or
- subsequent mining runs will use more and more files starting from one file in the first run and ending with all available files in the final run.

Through configuration in the granularity of analysis category, one can determine

- an initial minimum support,
- a final minimum support, and
- an increment on the minimum support between subsequent runs.

In addition, the minimum support may be defined as an absolute integer or as a percentage ranging from 0% for the lowest minimum support of 2 and 100% for a minimum support equal to the number of transactions in the mining database.

In the category populating the mining database, the user may choose how the first internal model is transformed to the mining database. The two orthogonal options, already explained in Section 6.2.2 on page 68, are

1. the analysis range and
2. the domain of analysis.

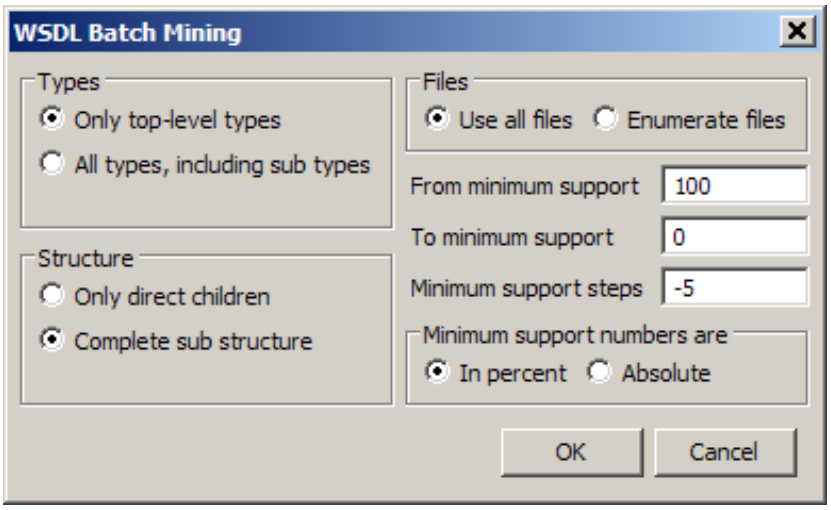


Figure 9.1: Miner configuration.

9.4.2 Evaluation runs

The purpose of the evaluation is to determine whether our concepts can be applied in an industrial context. In the context of SAP, it is important to determine ontological and technical overlap of exposed interface objects. Therefore, we performed test runs using the following configurations.

No	Range	Domain	Minimum support	Mining DB size
1	Top-level	All subelements	10%	increasing
2	Top-level	All subelements	decreasing	fix
3	All types	Direct subelements	decreasing	fix

The first two test settings aim at exploring ontological overlap because all given (top-level) types are evaluated based on their whole substructure.

In order to assess the performance of the mining component, a set of test runs was performed with increasing size of the mining database and a fix minimum support ratio of 10% of the current test's database size. The result is depicted in Figure 9.2 on the following page. The same runs were used to plot the time used for mining against the number of redundancy groups generated by the miner in Figure 9.3 on the next page. An interpretation of the results follows in the subsequent section.

The second test setting was chosen to assess the performance with respect to the desired granularity of the results—the minimum support. Therefore, the minimum support was decreased in the subsequent evaluation runs on a fix mining database. The mining database was generated using the 689 message types that appear in Web service definitions exposed by the SAP's enterprise SOA. The 689 message types are provided via 389 WSDL files with a total size of 55.4 Mbyte which makes about 1.77 messages

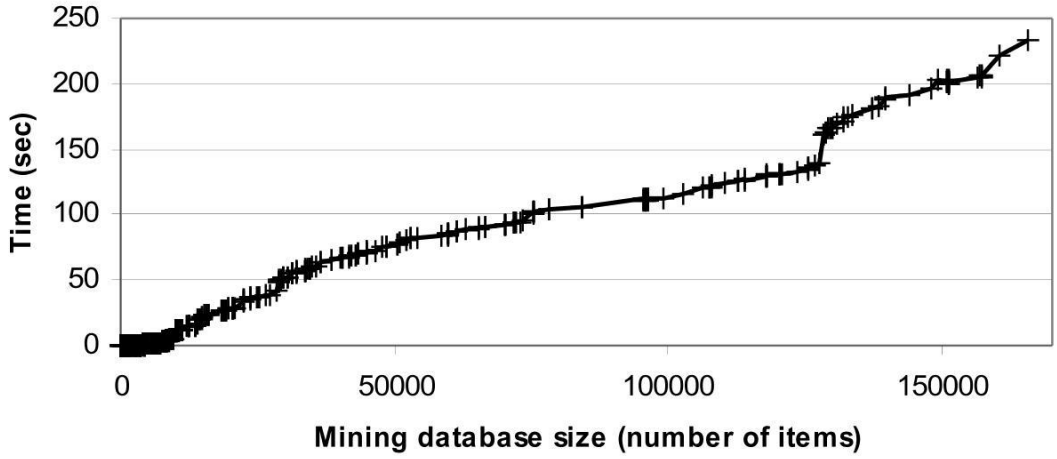


Figure 9.2: Mining time per input size.

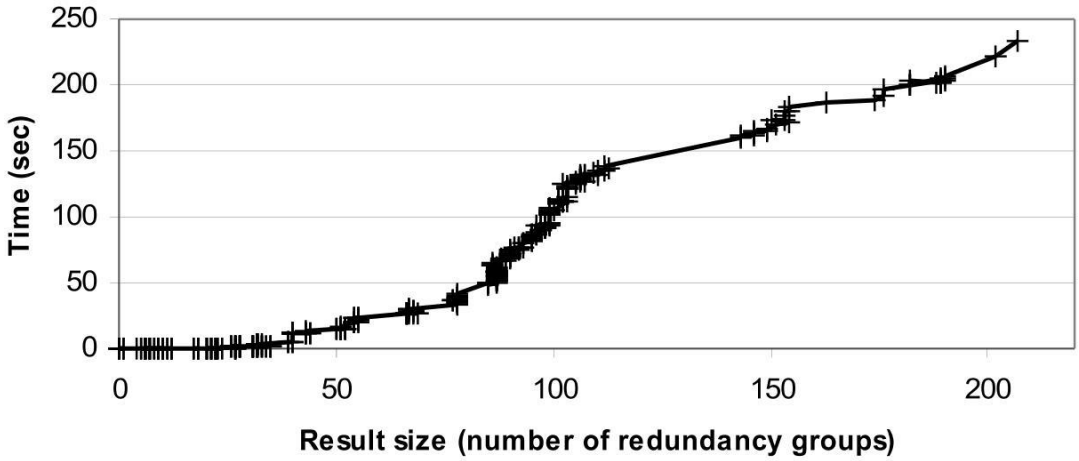


Figure 9.3: Mining time per output size.

per WSDL file. The resulting mining database contains 165783 items. We can compute from that that each message type contains about 239.61 fields. For the classical task of market basket analysis, these are rather large transactions. The 165783 items of the mining database consist of 5526 unique items. For a minimum support of 13%, 891 redundancy groups were found. The mining times for the second test setting are depicted in Figure 9.4 on the facing page together with the result size on the secondary axis.

The third evaluation setting aims at exploring technical overlap of message types in the enterprise service repository. Therefore, all complex types used in the definitions of the message types build up the mining database. As again all types are used in the mining database, it again consists of 165783 items. There are 10054 unique items in 5220 transactions because each complex type now appears as an own transaction. The average transaction size is now much smaller with about 31.8 items. For a minimum

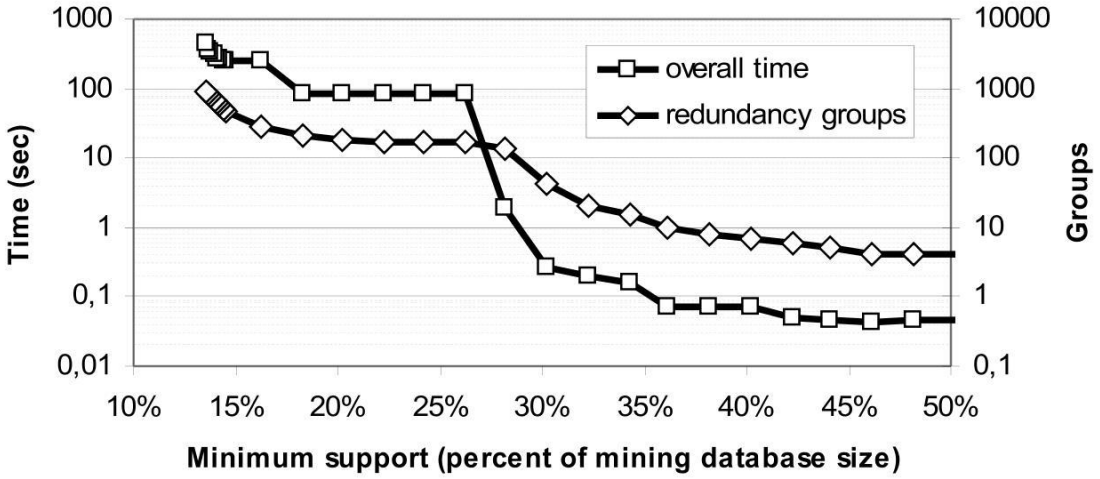


Figure 9.4: Top-level, all subelements mining.

support of 4 (0.08% of 5220 transactions), 5389 redundancy groups were found. The mining times together with the result size of that setting are depicted in Figure 9.5.

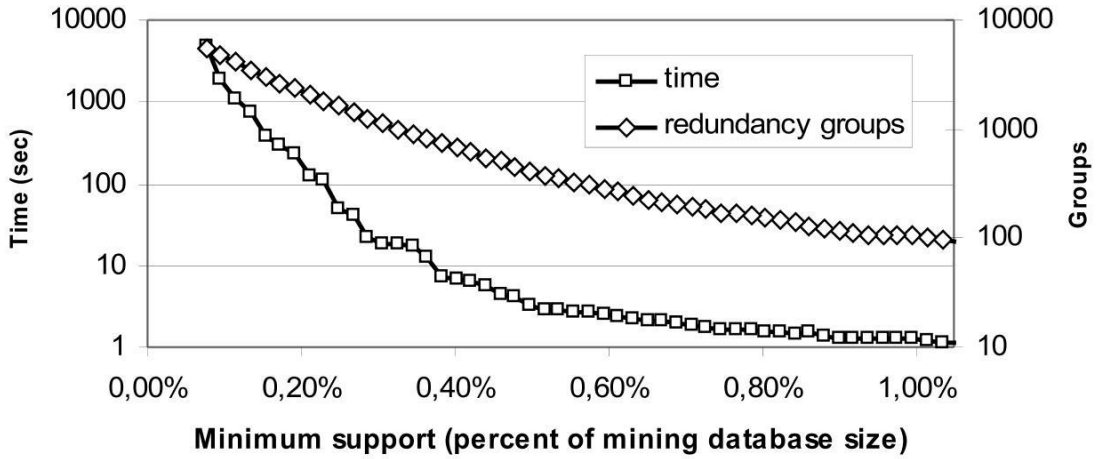


Figure 9.5: All types, direct subelements mining.

9.4.3 Discussion of the results

The first performance evaluation in Figure 9.3 on the facing page shows that the implementation of the frequent itemset mining has linear complexity with the size of the result. That feature was proven for the original algorithm by Uno et al. (2004a;b). The algorithm also yields linear performance with respect to the input size as can be interpreted from Figure 9.2 on the preceding page.

These results imply that a quickly growing result size also impacts the run time of the mining. The minimum support can be understood as a granularity cut-off for

the results. A lower minimum support yields more results. Therefore, the test runs in the second setting, depicted in Figure 9.4 on the preceding page, take longer with a decreasing minimum support. That is unfortunate in an industrial setting where large messages are handled, which means that the mining database contains large transactions. The evaluations in the related work, such as Uno et al. (2004b), mainly focus on the range from 90% to 20%. With large database sizes, a minimum support of 20% (i. e., 138 of 689 transactions) is still very rough because 138 message types would have to contain the same overlap. We expect more interesting results in the lower minimum supports. However, the mining result with minimum support of 93 (i. e., 13% of 689 transactions) can be obtained in fewer than an hour.

In contrast to mining large transactions, multiple transactions of smaller size, as in the third evaluation setting, can be mined much faster. In that setting, the miner takes longer than 1 minute the first time for a minimum support of 12 (i. e., 0.23% of 5220 transactions).

As the discovery of ontological and technical overlap provides support for decisions on a rather strategical level, the time needed to obtain the mining result is not critical. As the interfaces in an enterprise are not expected to change very quickly, the mining in a realistic setting would also be rather infrequent which makes it feasible to wait for the mining result. The only remaining important issue of the mining is that it can scale up well to large mining database sizes. Therefore, the linear time and space complexity of the algorithm as presented in Figure 9.2 on page 146 and Figure 9.3 on page 146 and proven by Uno et al. (2004a;b) is essential and makes closed frequent itemset mining fit well to detecting redundant interface objects in realistic settings.

9.5 Assessing the quality of the mining results

In this section, we investigate the quality of the mining results. For that purpose, we utilize the two measures precision and recall:

1. **Precision.** That measure assesses the exactness or fidelity of a method. In information retrieval, the precision is the relation of the number of retrieved relevant documents to the number of all retrieved documents. A method achieves the ideal value of 1.0 when it retrieves no irrelevant documents. However, the precision measure does not state whether a method fails to retrieve some further relevant documents.

2. **Recall.** That measure assesses the completeness of a method. Recall is defined in information retrieval as the relation of the number of retrieved relevant documents to the number of all relevant documents. An ideal value of 1.0 states that all relevant documents were retrieved. The recall measure however neglects how many irrelevant documents were also retrieved together with all relevant documents.

In order to make use of precision and recall for the assessment of our CFIM of hierarchical types, we need to define the notion of relevant document in the context of this work.

9.5.1 Adapting precision and recall for CFIM of hierarchical types

The result of the CFIM of hierarchical types is given as a ranked list of redundancy groups. A redundancy group is to be interpreted by the user as a set of redundant elements. We therefore need to define relevancy based on the structuring of elements in redundancy groups. In particular, it is desirable that in one redundancy group only ontologically similar elements reside.

Ontological relatedness

For the definition of ontological similarity, we use the fact that we are working on a data set of SAP which follows the strict naming convention of the core component technical specification (CCTS).[1] In CCTS, a type's name can be constructed from four components:

1. **Object class term.** That term specifies the logical data grouping or aggregation to which a property belongs, for example, `Person`.

2. **Property term.** That component represents the distinguishing characteristic of the object class, for example, `Residence`.

3. **Representation term.** That term describes the form in which the type is represented, for example, `Address`.

4. **Qualifier term.** That term can be given to further differentiate a type from other types.

In SAP's exposed interface objects, the components of CCTS type names are represented in camel case, for example, `PersonResidenceAddress`.

The definition of a type's name is subject to the governance process at SAP. We can assume that a type's name consists of meaningful parts due to the special care that is taken in defining and approving the definition of names and their parts. Examples for parts are `Property`, `Inventory`, `Lead`, `Payment`, `Withholding`, `Transportation`, `Item`, and `Date`.

The special care taken for name and parts definition reflects in the fact that only 546 unique parts are used in the 689 examined message type names. From analyzing the message names we found that on average, 4.98 parts make up one name. If one wanted to define type names of 4.98 parts per name without repeating any parts in two

[1] `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41022`

names, one could only build 109 names instead of 689. We conclude that the examined message type names share certain parts. We assume that two types sharing the same part are ontologically related.

Precision based on ontological relatedness

When we analyze the elements of a redundancy group, we may find that some elements are ontologically related as defined above. That is, the elements' names share some common parts. If not all elements of a redundancy group are ontologically related, one may identify multiple disjoint sets whose member are

- ontologically related with the other members of the set and
- not ontologically related with any member of any other set.

As we expect a redundancy group to only contain ontologically related elements, we define the precision measure as the ratio of the size of the largest set of related elements to the size of the redundancy group. We can simply calculate the measure by counting for each part how many of the redundancy group's element names contain the part. The largest number divided by the size of the redundancy group yields the precision measure.

With that definition, we are able to calculate the precision for one redundancy group. In order to assess the evolution of the precision when viewing the ranked list of redundancy groups from the highest to the lowest rank, we additionally introduce a window of redundancy groups. A window always includes the redundancy group g_{top} with the highest rank. Furthermore, a window is a set of redundancy groups that are adjacent in the ranked list of redundancy groups. Thus, each redundancy group g uniquely identifies a window, namely the window ranging from g to g_{top}.

We calculate the precision of a window as the average of the contained redundancy groups' precision values. The precision evolution for the windows of the second and third mining setting described in the previous section are depicted in Figure 9.6 on the next page and Figure 9.7 on the facing page. A discussion of the graphs follows later.

Recall based on ontological relatedness

Assessing the recall means to evaluate the ratio of the retrieved relevant documents to all relevant documents. In our evaluation environment, we start analyzing the mining database for ontological similarities. That means, we analyze the parts of the names of all message types that are fed to our mining database. Message types whose names share parts are said to be ontologically related. We expect that our CFIM of hierarchical types finds all sets of ontological related types. For simplicity, we define sets of ontologically related types based on single parts their names consist of.

Concretely, we define a set of ontologically related types as all types whose names share a specific part. It follows that each part uniquely identifies a set of ontologically related types.

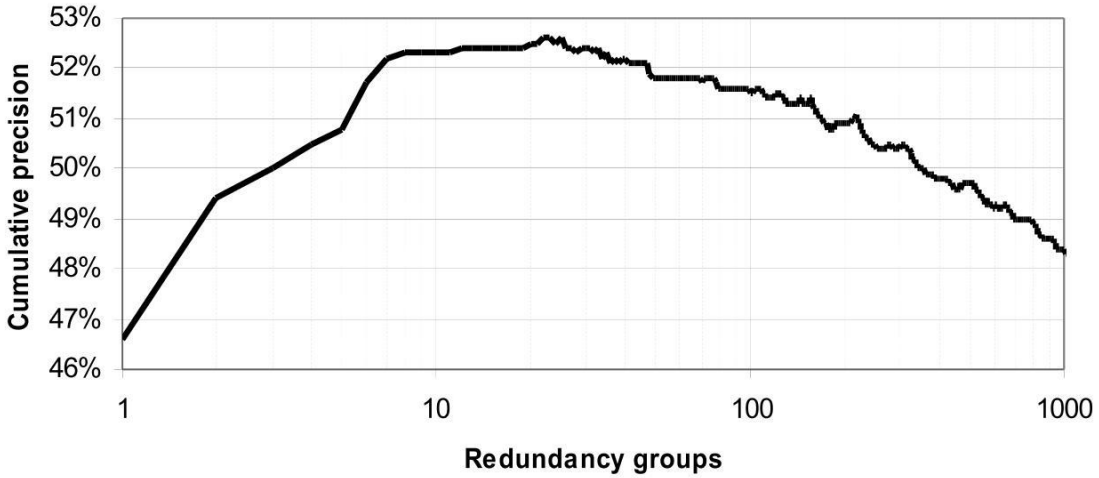


Figure 9.6: Precision evolution for mining with high minimum support of 13%.

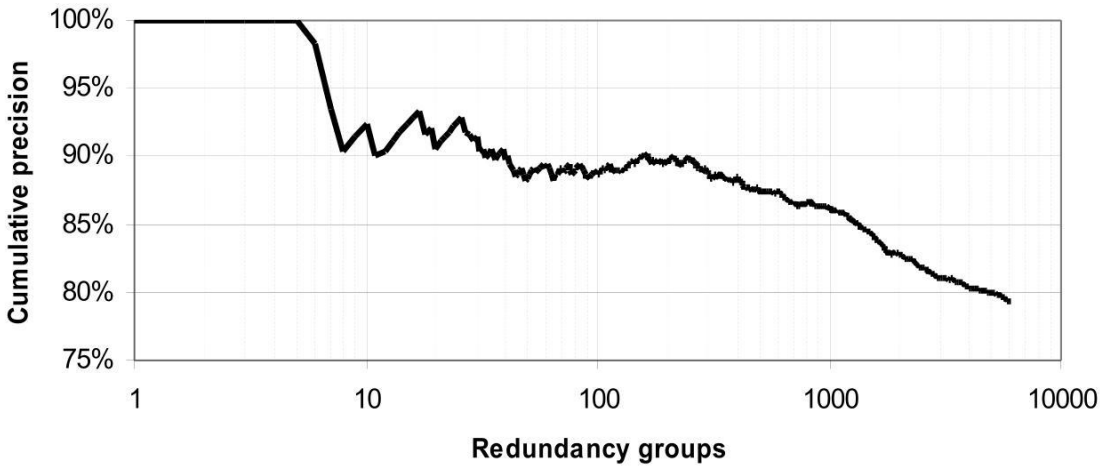


Figure 9.7: Precision evolution for mining with low minimum support of 0.23%.

We define the recall measure based on the degree to which a redundancy group g contains a set of ontologically related types t identified by part p. We thus first count the number n of elements in g whose name contains p. Second, we divide n by the size of t. That yields the degree to which g contains t. We repeat the computation for all parts contained in a redundancy group's elements' names.

The recall of a redundancy group g is the average of the degrees to which g contains a part p, for every part p. Parts that are not contained in g's elements' names contribute 0 to the average.

The recall of a window is defined in the same way except for the computation of the degree to which a redundancy group contains a set of ontologically related types: For the computation of the degree, the redundancy group with the largest degree is chosen from the redundancy groups in the window. Intuitively, one could say that in a window,

we can pick for each part p the redundancy group that contains most of the ontologically related types for p.

The evolution of recall is depicted in Figure 9.8 and Figure 9.9 for the second and third mining setting.

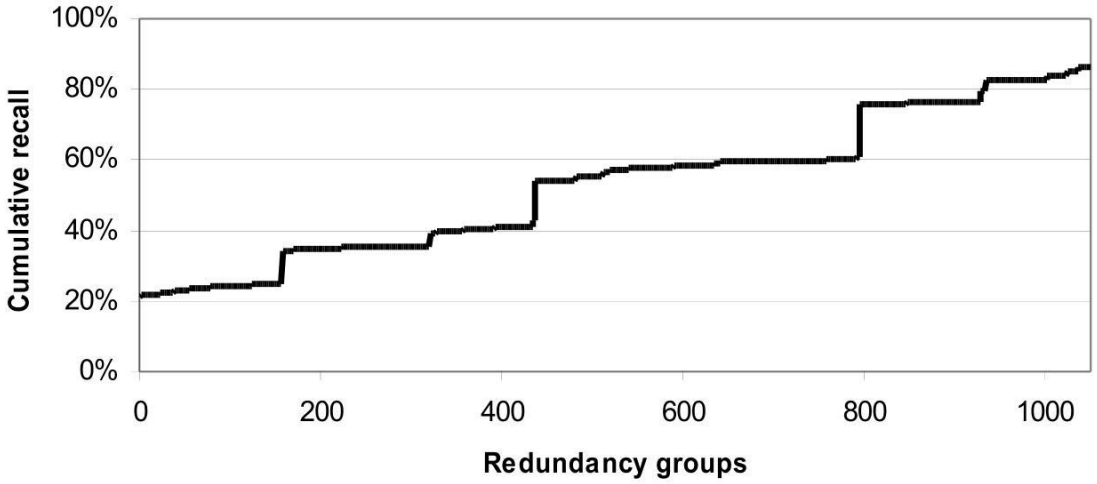


Figure 9.8: Recall evolution for top-level, all subelements mining.

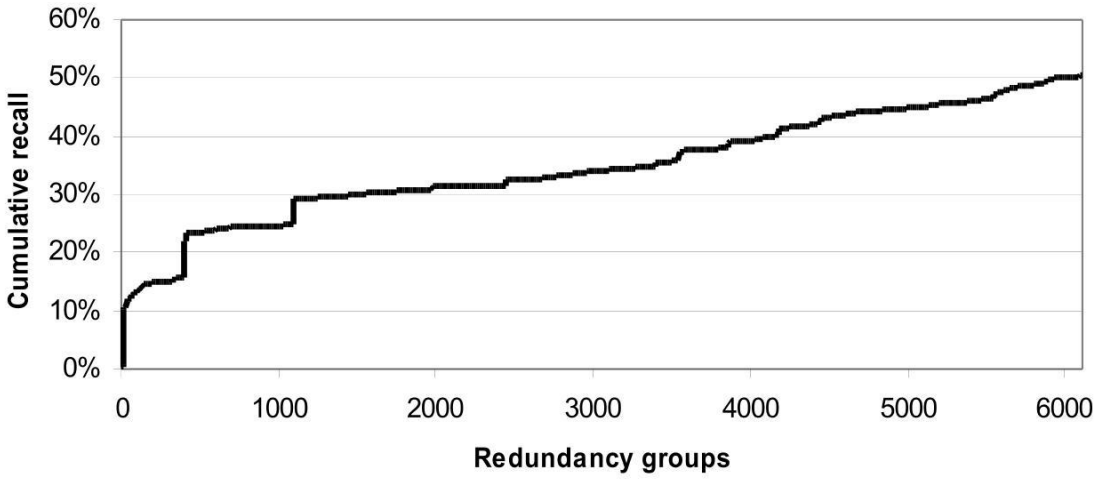


Figure 9.9: Recall evolution for all types, direct subelements mining.

9.5.2 Discussion

Evaluating the precision diagrams Figure 9.6 and Figure 9.7 on the previous page has two major purposes: assessing the approach of CFIM of hierarchical types in general and assessing the ranking function in particular.

For a positive assessment of CFIM of hierarchical types, we would expect that the precision our approach can achieve closely reaches 100% as occurs in the case of mining

SAP's interface objects considering all types but only direct subelements (Figure 9.7). The precision of mining only top-level types but all subelements, as presented in Figure 9.6, reaches just 52.6% for the window identified by the 23rd redundancy group. That is because the result was taken for a minimum support of 13%, which is, as already denoted in Section 9.4.3 on page 147, still too rough to produce results of acceptable precision. However, it is to be noted that in the case where mining was performed with a lower minimum support, a precision can be reached by our CFIM of hierarchical types that is above 79.3% for all redundancy groups as depicted in Figure 9.7. Thus, it turns out that CFIM of hierarchical types with only considering direct subelements seems to be a good method when time is an issue and precise results are desired. Based on our definitions above, a lower precision of about 50% means that a redundancy group consists to 50% of ontologically related types. It is unknown whether the remaining 50% belong to one or more other sets of related types. That situation may be desirable for an integration expert who would like to perform a deeper manual analysis and who appreciates a wider range of potentially matching types. In any case, it is ensured that all types of a redundancy group share a technical commonality which provides valuable input when technical harmonization of the types is aspired.

For a positive assessment of the ranking function, we would expect to see a rather high precision in the beginning of a graph which may decrease toward its end. That is the case in Figure 9.7 where mining was performed with a low minimum support. For the mining with high minimum support where the precision is always in the range of 46.6% and 52.6%, our ranking seems to perform not ideally as the peek precision is only reached for the window including the 23rd redundancy group of 1050 in total. It seems to be a good rule of thumb to consider the first 2–3% of the returned redundancy groups as rather relevant results. Indeed, the precision leaves its higher-level plateau after about 45 redundancy groups (4.3% of 1050) in Figure 9.6 and 180 (2.9% of 6113) in Figure 9.7. That is a very good finding as with the creation of our rank, we intended to present the relevant results the expert should look at upfront in the results list.

Looking at the recall diagrams in Figure 9.8 and Figure 9.9 on the preceding page, we make the interesting observation that, although the minimum support for the mining in Figure 9.9 was low, only 50.4% of the ontological information could be discovered. Surprisingly, the mining with a relatively high minimum support in Figure 9.8 could discover 86.3% of the ontological information. Obviously, the reason for the higher recall is our specific approach to consider mining the whole substructure of hierarchical types even when an unfortunate, relatively high minimum support is given. That finding justifies to state that structural mining is capable of identifying ontological redundancy.

As a general conclusion, we have identified in this section, as expected, that mining with a lower minimum support consumes more time but yields a higher precision and vice versa. Orthogonally, mining with a limited consideration of substructure still yields ontologically correct results in a shorter time, though with a lower recall. If a higher recall is desired and time is not an issue, as, for example, in offline mining, the complete substructure should be considered by choosing the respective option for the CFIM of hierarchical types.

Chapter 10

Facilitating Interoperability of SAP Business Partners

In this chapter, we implement SAP's best practice business process "cross-company-code sales order processing" using our solution to transform behavioral models for EAI and e-business and the complex-goal-based WS composition as denoted by Table 10.1. Parts of the example have been used to demonstrate the concepts of Chapter 7 on page 85. SAP's best practices[1] are a collection of standard configurations for SAP systems. The aim of the best practices is to shorten the time needed to implement an SAP solution in an enterprise that performs mostly standard processes.

Table 10.1: Proof of concepts

Requirement	**Solution**	**Proof of concept**
Track behavioral information through software design phases	Transform behavioral models for EAI and e-business	Model-driven derivation of a standard business process from SAP's best practice library and executing the generated orchestration

This chapter is structured as follows: In Section 10.1, we introduce the integration scenario including the participating roles and their respective components. In Section 10.2, we assume the existence of component perspective behavioral models describing the components and derive EAI perspective behavioral models from the component perspective behavioral models to participate in the e-business perspective business process. In Section 10.3 we integrate the EAI perspective behavioral models into an e-business perspective orchestration. Section 10.4 presents the copy rules of the generated e-business perspective orchestration and exemplarily executes the orchestration directly utilizing the exposed components step by step. Section 10.5 concludes this chapter.

[1] `http://www.sap.com/bestpractices`

10.1 Cross-company-code sales order processing

The e-business process "cross-company-code sales order processing" depicted in Figure 10.1 consists of two participants:

1. **Customer.**
2. **Seller.** The seller is further split to its divisions
   - sales center and
   - production plant.

In business terms, each division has a unique company code. Therefore, cross-company-code sales order processing means the selling procedure involving multiple departments.

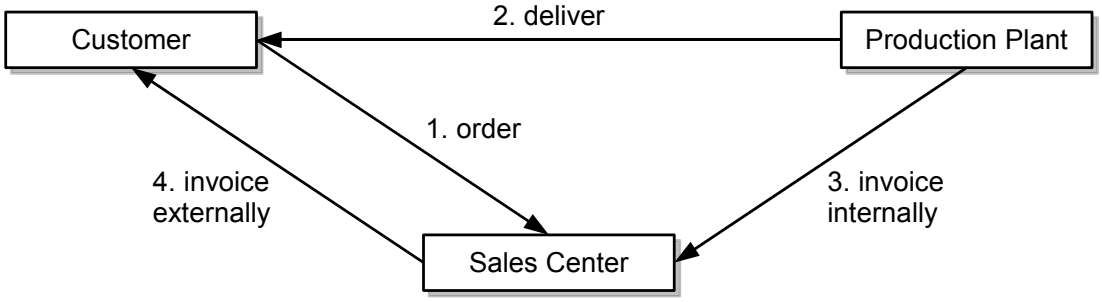


Figure 10.1: Cross-company-code sales order processing.

The business process of cross-company-code sales order processing works as follows: A customer orders goods from the sales organization of their vendor. The vendor has a distribution plant that belongs to a different company code. The goods are delivered from the distribution plant directly to the customer. The customer receives their invoice from the sales organization. Intercompany billing takes place between the two company codes.

Following the premise of this dissertation, we assume that customer and seller are companies willing to engage in e-business. It is common for a company purchasing goods to execute an internal purchasing process. Therefore, in addition to the cross-company-code sales order processing best practice business process, we assume that the customer follows a complex purchasing business process internally. The business process of the customer involves the creation of a purchase order and its approval by an appropriate person.

We conclude that the scenario consists of two participants willing to perform e-business integration, i. e., to integrate their EAI perspective behavioral models to an orchestration in the e-business perspective. As a prerequisite, each of the partners has to perform EAI, i. e., to define their EAI perspective behavioral models as an integration of their component perspective behavioral models.

10.2 Integrating the EAI perspective

We assume that each component of the participants is described via a behavioral model. For simplicity of the behavioral models presented in this chapter, we further assume that no activities fail during cross-company-code sales order processing at the seller. However, as we want to demonstrate the ability of our approach to deal with failing activities, we keep failing activities in the purchasing process at the customer.

10.2.1 Customer

The customer's purchasing business process internally communicates with two components depicted on the left-hand side of Figure 10.2 on the next page:

1. **Order creation (OCR).** That component consists of a user interface that allows entering the goods a human needs to order. When the user is done, the system outputs a purchase order (PO). In order to update the user about the status of the order, the component may either receive a failure notification (FAIL) or a positive notification (DONE).

2. **Purchase order management (POM).** That component is capable of performing two functions:

   (a) A given purchase order is approved or declined by a representative of the finance department. Therefore, the component takes a purchase order (PO) as an input and may communicate an approval (APR) or rejection message (REJ) in response.

   (b) In the case a purchase order was approved, the purchase order management component handles the payment of the ordered goods once an appropriate invoice was received. Therefore, the final operation of that component is to accept an invoice (INV).

We now follow the steps of our solution for deriving the EAI perspective of the customer from the component perspective as introduced in Section 5.2.2 on page 59.

Extract consumed and define provided behavioral models

The first step consists of three sub-activities:

1. **Define consumer view of provided behavioral model.** The externally observable interface (CUST)—or the provided behavioral models—of the customer's purchasing business process consists of

   (a) sending a purchase order (PO) and

   (b) expecting an invoice (INV) or a failure notification (FAIL) in response.

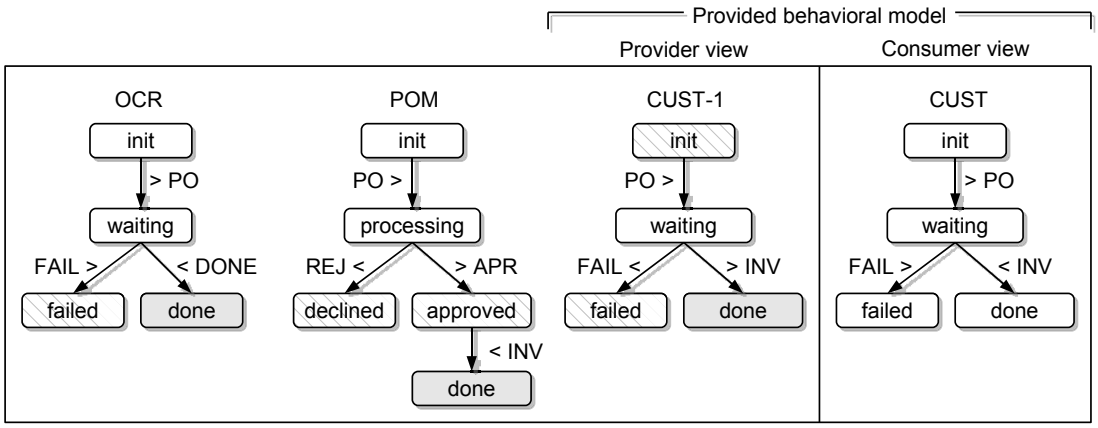


Figure 10.2: Customer components and provided interface.

That is depicted on the right-hand side of Figure 10.2.

2. **Convert provided behavioral model to provider view.** We construct the opposite behavioral model (CUST^{-1}) from the provided behavioral model in order to yield the following consumer view which is the remaining behavioral model of Figure 10.2.

   (a) A purchase order is received.

   (b) An invoice or a failure notification are given in response.

3. **Excerpt consumed behavioral model fragment.** That step is trivial as the components correspond to the behavioral fragments we need for integrating the component behavioral models with the provided behavioral model.

Assign communication

An integration expert defines the following assignments of variables:

$$((\mathsf{OCR}, \mathsf{PO}), (\mathsf{POM}, \mathsf{PO})), \qquad ((\mathsf{OCR}, \mathsf{PO}), (\mathsf{CUST}^{-1}, \mathsf{PO})),$$
$$((\mathsf{POM}, \mathsf{REJ}), (\mathsf{OCR}, \mathsf{FAIL})), \qquad ((\mathsf{POM}, \mathsf{APR}), (\mathsf{OCR}, \mathsf{DONE})),$$
$$((\mathsf{CUST}^{-1}, \mathsf{FAIL}), (\mathsf{OCR}, \mathsf{FAIL})), \qquad ((\mathsf{CUST}^{-1}, \mathsf{INV}), (\mathsf{POM}, \mathsf{INV}))$$

Build orchestration

In order to build the orchestration, the integration expert has to define desired and acceptable outcomes of the orchestration, namely, the composition goal consisting of primary and recovery goals:

- **Primary goal.** A desired final state is reached when OCR has received DONE, POM has received INV, and CUST^{-1} sent INV.

- **Recovery goals.** It is acceptable that
  - OCR receives FAIL when POM sent REJ and CUST^{-1} is still in its initial state and
  - OCR receives FAIL when POM sent APR, but did not receive INV and CUST^{-1} sent FAIL.

We only give the resulting copy rules of the subsequent complex-goal-based WS composition because the composition procedure explained in Chapter 8 on page 113 is not the main focus of this chapter.

$$
\begin{aligned}
c_1 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{approved}), (\mathsf{CUST}^{-1}, \mathsf{done})\ , \\
& ((\mathsf{CUST}^{-1}, \mathsf{INV}), (\mathsf{POM}, \mathsf{INV})), ((\mathsf{POM}, \mathsf{APR}), (\mathsf{OCR}, \mathsf{DONE}))\ ) \\
c_2 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{approved}), (\mathsf{CUST}^{-1}, \mathsf{init})\ , \\
& ((\mathsf{OCR}, \mathsf{PO}), (\mathsf{CUST}^{-1}, \mathsf{PO}))\ ) \\
c_3 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{init}), (\mathsf{CUST}^{-1}, \mathsf{init})\ , \\
& ((\mathsf{OCR}, \mathsf{PO}), (\mathsf{POM}, \mathsf{PO}))\ ) \\
c_4 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{declined}), (\mathsf{CUST}^{-1}, \mathsf{init})\ , \\
& ((\mathsf{POM}, \mathsf{REJ}), (\mathsf{OCR}, \mathsf{FAIL}))\ ) \\
c_5 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{approved}), (\mathsf{CUST}^{-1}, \mathsf{failed})\ , \\
& ((\mathsf{CUST}^{-1}, \mathsf{FAIL}), (\mathsf{OCR}, \mathsf{FAIL}))\ )
\end{aligned}
$$

After performing the above steps, we have succeeded in linking the customer's provided EAI perspective behavioral model to its component perspective behavioral models. We continue with the same procedure for the seller.

10.2.2 Seller

The business process the seller exposes as its part of the cross-company-code sales order processing scenario internally communicates with three components as depicted in Figure 10.3 on the following page:

1. **Sales order management (SOM).** The sales order is the seller's counterpart of the buyer's purchase order. As both purchase order and sales order share a lot of similar fields, a sales order is normally constructed from a purchase order (PO). The first component of the seller is thus capable to receive a purchase order and output the newly derived sales order (SO) for internal use.

   The sales order is used to keep track of the selling process. When the selling process is done, a sales order becomes closed. In the seller's component of our example, that is expressed by a second operation taking a delivery (DEL) and an invoice (INV) as inputs. The sales order management component is located at the sales center department of the seller.

2. **Invoicing (FIN).** Similar to the sales order, a "delivery" (DEL) is used to keep track of the activities to prepare the final shipment of goods. These activities normally include the picking of the goods from a storage, packing the goods, labeling the package, and posting the goods issue. After posting the goods issue, a delivery can be used to generate an invoice from its data. Therefore, the invoicing component in our example takes as input a delivery and outputs an invoice (INV).

   Additionally, an incoming invoice can be paid by the invoicing component. Therefore, another one-way operation takes an invoice as an input.

   No ordering of the two operations is prescribed. That means that the component's implementation can cope with any sequential invocation of both operations. Determining the operations' sequence in that case means adding further business logic and depends on their use in an integration scenario.

   The invoicing component is as well part of the seller's sales center.

3. **Production (PROD).** The production component is part of the seller's production plant department. For simplicity, we assume that the component has a single operation that receives a sales order (SO) as an input and returns a delivery (DEL) and an invoice (INV) upon completion. We thus abstract from the complex internal communication that manages the production procedure.

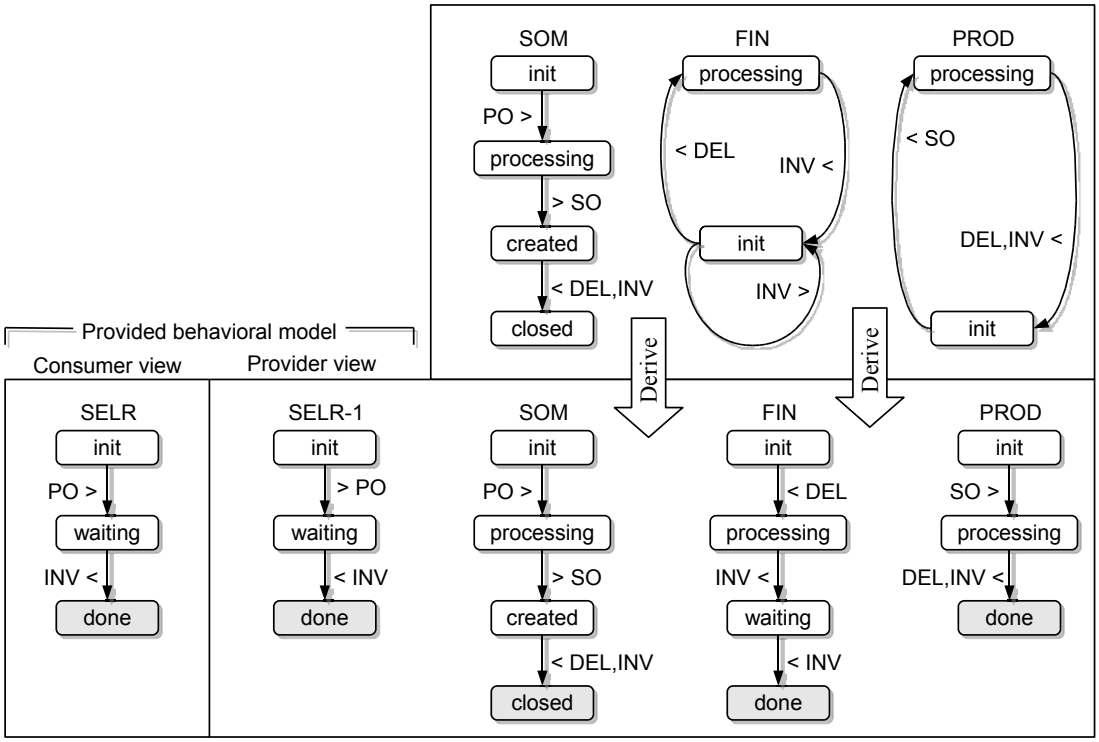


Figure 10.3: Seller components, derived behavioral models, and provided interface.

We now again follow the steps of our solution for deriving the EAI perspective of the seller from the component perspective as introduced in Section 5.2.2 on page 59.

Extract consumed and define provided behavioral models

The first step consists of three sub-activities:

1. **Define consumer view of provided behavioral model.** The externally observable interface (SELR)—or the provided behavioral models—of the seller's business process first expects a purchase order (PO) and answers with an invoice (INV).

2. **Convert provided behavioral model to provider view.** The opposite behavioral model (SELR^{-1}) of the provided behavioral model first sends a purchase order and afterwards expects an invoice as input.

3. **Excerpt consumed behavioral model fragment.** That step is trivial for the sales order management component because SOM's behavioral model already is a tree.

   For the invoicing component, we need to sequence and unfold the two components. We place the operation taking a delivery and returning an invoice in the sequence before the one-way operation that takes an invoice.

   Through the sequencing, we introduce business logic that will be completed by the subsequent variable assignments. In particular, we use the first operation to create the invoice for the customer. The second operation pays the intercompany bill from the production plant.

   The production component's behavioral model just needs to be unfolded to yield the fragment to consume.

Assign communication

An integration expert defines the following assignments of variables according to the desired business logic described above:

$$
\begin{aligned}
&((\mathsf{SELR}^{-1}, \mathsf{PO}), (\mathsf{SOM}, \mathsf{PO})), && ((\mathsf{SOM}, \mathsf{SO}), (\mathsf{PROD}, \mathsf{SO})),\\
&((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{FIN}, \mathsf{DEL})), && ((\mathsf{PROD}, \mathsf{INV}), (\mathsf{FIN}, \mathsf{INV})),\\
&((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{SOM}, \mathsf{DEL})), && ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SELR}^{-1}, \mathsf{INV})),\\
&((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV}))
\end{aligned}
$$

Build orchestration

As we consider a simplified process for the seller whose execution may not fail, we only need to define a primary goal and no recovery goals. The desired final state is reached when

- SELR^{-1} has received INV,
- SOM received DEL and INV,
- FIN received INV, and
- PROD sent DEL and INV.

We give the resulting copy rules of the complex-goal-based WS composition in the following:

$$
\begin{aligned}
c_6 = (\ & (\mathsf{SELR}^{-1}, \text{waiting}), (\mathsf{SOM}, \text{waiting}), (\mathsf{FIN}, \text{waiting}), (\mathsf{PROD}, \text{done}) \ , \\
& ((\mathsf{PROD}, \mathsf{INV}), (\mathsf{FIN}, \mathsf{INV})), ((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{SOM}, \mathsf{DEL})), \\
& ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV})), ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SELR}^{-1}, \mathsf{INV})) \ ) \\
c_7 = (\ & (\mathsf{SELR}^{-1}, \text{waiting}), (\mathsf{SOM}, \text{waiting}), (\mathsf{FIN}, \text{init}), (\mathsf{PROD}, \text{done}) \ , \\
& ((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{FIN}, \mathsf{DEL})) \ ) \\
c_8 = (\ & (\mathsf{SELR}^{-1}, \text{waiting}), (\mathsf{SOM}, \text{waiting}), (\mathsf{FIN}, \text{init}), (\mathsf{PROD}, \text{init}) \ , \\
& ((\mathsf{SOM}, \mathsf{SO}), (\mathsf{PROD}, \mathsf{SO})) \ ) \\
c_9 = (\ & (\mathsf{SELR}^{-1}, \text{waiting}), (\mathsf{SOM}, \text{init}), (\mathsf{FIN}, \text{init}), (\mathsf{PROD}, \text{init}) \ , \\
& ((\mathsf{SELR}^{-1}, \mathsf{PO}), (\mathsf{SOM}, \mathsf{PO})) \ )
\end{aligned}
$$

10.3 Integrating the e-business perspective

In the e-business perspective, the behavioral models to integrate are the consumer views of the provided behavioral models of the customer (CUST) and the seller (SELR).

We follow again the respective steps of our solution—this time for deriving the e-business perspective from the community perspective as introduced in Section 5.2.2 on page 59. For deriving the e-business perspective from the community perspective, only two of the three sub-activities of the derive activity are relevant.

10.3.1 Assign communication

In our cross-company-code sales order processing scenario, the customer possesses non-deterministic behavior: The first time, non-determinism occurs when the purchase order management component may approve or decline the purchase order. The second time, the provided behavioral model may not be able to deliver the requested purchase order and answer with a failure notification. Whereas that appears as non-deterministic behavior to the customer, the opposite behavioral model in the e-business perspective is deterministic. Namely, the orchestration in the e-business perspective can determine whether to send an invoice or a failure notification to the customer. The customer can cope with both answers.

However, in our example, the seller always succeeds and thus never sends a failure notification. Consequently, only one branch of the customer's provided behavioral model will be used. As the branching decision is deterministic, we need not connect variable assignments to the unused branch. The complex-goal-based WS composition would never visit that branch.

Finally, the integration expert defines the following variable assignments:

$$(\,(\mathsf{CUST},\,\mathsf{PO}\,),\,(\,\mathsf{SELR},\,\mathsf{PO}\,)\,),\qquad(\,(\mathsf{SELR},\,\mathsf{INV}\,),\,(\,\mathsf{CUST},\,\mathsf{INV}\,)\,)$$

10.3.2 Build orchestration

The orchestration requirement consists of only one primary goal: The desired final state is reached when CUST has received INV and SELR sent INV.

The following copy rules result from the subsequent complex-goal-based WS composition:

$$\begin{aligned} c_{10} = (\;\; &(\,\mathsf{CUST},\,\mathsf{waiting}\,),\,(\,\mathsf{SELR},\,\mathsf{done}\,)\;\;, \\ &(\,(\,\mathsf{SELR},\,\mathsf{INV}\,),\,(\,\mathsf{CUST},\,\mathsf{INV}\,)\,)\;\;) \\ c_{11} = (\;\; &(\,\mathsf{CUST},\,\mathsf{waiting}\,),\,(\,\mathsf{SELR},\,\mathsf{init}\,)\;\;, \\ &(\,(\,\mathsf{CUST},\,\mathsf{PO}\,),\,(\,\mathsf{SELR},\,\mathsf{PO}\,)\,)\;\;) \end{aligned}$$

We have now generated orchestrations for each of the participants in the EAI perspective and for the e-business perspective in a model-driven manner. In the subsequent section, we jointly execute the generated orchestrations without having to write any implementation for the EAI perspective.

10.4 Executing the joined orchestrations

As described in Section 7.6 on page 108, we can jointly execute the EAI perspective orchestration and the e-business perspective orchestration by merging their copy rules. In this section, we walk through the copy rules in the natural order of their execution. But before we start, let's have a closer investigation of the MEDIATOR* ASM that defines the execution semantics of a set of joined copy rules.

The definition of the MEDIATOR* ASM contains two sets of identifiers:

1. ID**.** That set contains the identifiers of behavioral models in the component perspective. That is, OCR and POM from the customer and SOM, FIN, and PROD from the seller.

2. PID**.** That set contains the identifiers of the e-business perspective behavioral model. That is, CUST and SELR.

The execution of MEDIATOR* triggers in alternation

- ADVANCEBM through ADVANCEORCHESTRATION for the members of PID ID and
- SEND and RECEIVE for the members of ID.

As $\delta^{\mathsf{CUST}} = \delta^{\mathsf{CUST}^{-1}}$ and $\delta^{\mathsf{SELR}} = \delta^{\mathsf{SELR}^{-1}}$ (see Section 7.4 on page 101), we can treat CUST and CUST^{-1} as well as SELR and SELR^{-1} the same in the joined orchestration and when given as an argument to an ASM. These behavioral models act as the synchronizing interfaces between the EAI perspective and the e-business perspective. To illustrate that, we walk through the execution of the joined copy rules in the following.

Table 10.2 on the next page shows the states of all participating behavioral models during the execution of the joined orchestration.

Initially, all behavioral models are in their initial states. OCR initiates the collaboration by sending a PO. That happens non-deterministically from the perspective of the orchestration, totally depending on the underlying Web service performing the operation. The receipt of the PO by the RECEIVE(OCR) ASM allows ADVANCEBM(OCR) to set OCR's state to waiting. That triggers the first copy rule:

$$c_3 = (\ \ (\mathsf{OCR},\ \mathsf{waiting}),\ (\mathsf{POM},\ \mathsf{init}),\ (\mathsf{CUST}^{-1},\ \mathsf{init})\ \ ,\\ ((\mathsf{OCR},\ \mathsf{PO}),\ (\mathsf{POM},\ \mathsf{PO}))\ \ )$$

Forwarding the PO to POM triggers SEND(POM) to invoke the underlying implementation of POM. As the implementation is known to be non-deterministic, RECEIVE(POM) may trigger ADVANCEBM(POM) to set POM's state to either declined or approved.

In the first case, the following copy rule is triggered which generates a failure notification from REJ and forwards it to OCR.

$$c_4 = (\ \ (\mathsf{OCR},\ \mathsf{waiting}),\ (\mathsf{POM},\ \mathsf{declined}),\ (\mathsf{CUST}^{-1},\ \mathsf{init})\ \ ,\\ ((\mathsf{POM},\ \mathsf{REJ}),\ (\mathsf{OCR},\ \mathsf{FAIL}))\ \ )$$

The receipt of FAIL by OCR triggers OCR's state to be changed to failed. That is an acceptable recovery goal. Thus, the execution came to a consistent end.

In the case that POM's state was set to approved by the non-deterministic implementation, the following copy rule is triggered:

$$c_2 = (\ \ (\mathsf{OCR},\ \mathsf{waiting}),\ (\mathsf{POM},\ \mathsf{approved}),\ (\mathsf{CUST}^{-1},\ \mathsf{init})\ \ ,\\ ((\mathsf{OCR},\ \mathsf{PO}),\ (\mathsf{CUST}^{-1},\ \mathsf{PO}))\ \ )$$

Thus, the approved PO is now forwarded to the external interface of CUST.

The receipt of PO triggers ADVANCEBM(CUST) to set CUST's state to waiting. We switch now from the customer's EAI perspective to the e-business perspective as the state change is recognized by the following copy rule:

$$c_{11} = (\ \ (\mathsf{CUST},\ \mathsf{waiting}),\ (\mathsf{SELR},\ \mathsf{init})\ \ ,\\ ((\mathsf{CUST},\ \mathsf{PO}),\ (\mathsf{SELR},\ \mathsf{PO}))\ \ )$$

Table 10.2: State transitions for joined orchestration

Prev. No	No	Cp. rule	OCR	POM	CUST	SELR	SOM	FIN	PROD
	0		waiting	init	init	init	init	init	init
0	1	c_3	waiting	declined	init	init	init	init	init
1	2	c_4	failed	declined	init	init	init	init	init
0	3	c_3	waiting	approved	init	init	init	init	init
3	4	c_2	waiting	approved	waiting	init	init	init	init
4	5	c_{11}	waiting	approved	waiting	waiting	init	init	init
5	6	c_9	waiting	approved	waiting	waiting	created	init	init
6	7	c_8	waiting	approved	waiting	waiting	created	init	done
7	8	c_7	waiting	approved	waiting	waiting	created	waiting	done
8	9	c_6	waiting	approved	waiting	done	closed	done	done
9	10	c_{10}	waiting	approved	done	done	closed	done	done
10	11	c_1	done	done	done	done	closed	done	done

Executing the copy rule triggers SELR's state to change from init to waiting which is recognized by the following rule from the seller's EAI perspective:

$$c_9 = (\ \ (\mathsf{SELR}^{-1},\ \mathsf{waiting}),\ (\mathsf{SOM},\ \mathsf{init}),\ (\mathsf{FIN},\ \mathsf{init}),\ (\mathsf{PROD},\ \mathsf{init})\ \ ,\\ ((\mathsf{SELR}^{-1},\ \mathsf{PO}),\ (\mathsf{SOM},\ \mathsf{PO}))\ \ )$$

The seller's components' states are subsequently advanced by the following copy rules:

$$\begin{aligned}
c_8 = (\ & (\mathsf{SELR}^{-1}, \mathsf{waiting}), (\mathsf{SOM}, \mathsf{waiting}), (\mathsf{FIN}, \mathsf{init}), (\mathsf{PROD}, \mathsf{init})\ , \\
& ((\mathsf{SOM}, \mathsf{SO}), (\mathsf{PROD}, \mathsf{SO}))\ ) \\
c_7 = (\ & (\mathsf{SELR}^{-1}, \mathsf{waiting}), (\mathsf{SOM}, \mathsf{waiting}), (\mathsf{FIN}, \mathsf{init}), (\mathsf{PROD}, \mathsf{done})\ , \\
& ((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{FIN}, \mathsf{DEL}))\ ) \\
c_6 = (\ & (\mathsf{SELR}^{-1}, \mathsf{waiting}), (\mathsf{SOM}, \mathsf{waiting}), (\mathsf{FIN}, \mathsf{waiting}), (\mathsf{PROD}, \mathsf{done})\ , \\
& ((\mathsf{PROD}, \mathsf{INV}), (\mathsf{FIN}, \mathsf{INV})), ((\mathsf{PROD}, \mathsf{DEL}), (\mathsf{SOM}, \mathsf{DEL})), \\
& ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SOM}, \mathsf{INV})), ((\mathsf{FIN}, \mathsf{INV}), (\mathsf{SELR}^{-1}, \mathsf{INV}))\ )
\end{aligned}$$

The latter copy rule triggers SELR's state to be set to done. That triggers the following copy rule from the e-business perspective:

$$\begin{aligned}
c_{10} = (\ & (\mathsf{CUST}, \mathsf{waiting}), (\mathsf{SELR}, \mathsf{done})\ , \\
& ((\mathsf{SELR}, \mathsf{INV}), (\mathsf{CUST}, \mathsf{INV}))\ )
\end{aligned}$$

After performing the previous copy rule, ADVANCEBM(CUST) sets CUST's state to done. That triggers the final copy rule:

$$\begin{aligned}
c_1 = (\ & (\mathsf{OCR}, \mathsf{waiting}), (\mathsf{POM}, \mathsf{approved}), (\mathsf{CUST}^{-1}, \mathsf{done})\ , \\
& ((\mathsf{CUST}^{-1}, \mathsf{INV}), (\mathsf{POM}, \mathsf{INV})), ((\mathsf{POM}, \mathsf{APR}), (\mathsf{OCR}, \mathsf{DONE}))\ )
\end{aligned}$$

Finally, ADVANCEBM(OCR) sets OCR's state to done. Now, the primary goal is achieved. Our exemplary execution ended successfully and consistently.

Please note that the copy rule c_5 was never triggered. The reason is that the seller acts deterministically in our example and never responds to a purchase order with a failure notification.

10.5 Summary

We have shown in this chapter how our solution to transform behavioral models for EAI and e-business is capable of reusing behavioral knowledge at run time and integration design time which was gathered at component design time.

Current approaches to store behavioral information such as UML activity diagrams, OWL-S, BPEL, and BPMN lack a formal semantics as part of their specification. That implies that the models are not exchangeable across multiple tools making use of them. In particular, model-to-model and model-to-code transformations are essential in a model-driven approach. Without proper semantics, there is a risk to loose or falsify behavioral information throughout the transformations.

With our approach, behavioral knowledge is uniformly represented throughout the software lifecycle. Due to the uniform representation and the execution semantics given via ASM, our behavioral models can interact with orchestrations generated by our approach at run time. When our complex-goal-based WS composition is used to generate the orchestrations, a transactionally correct execution of an e-business scenario can be guaranteed by construction based on our solution to transform behavioral models for EAI and e-business.

Today, all steps presented in this chapter are manual and involve the interpretation of natural language. That is one of the main reasons for integration projects today lasting between 6 months and a year. Especially, smaller customers of enterprise software cannot afford that much expenses. Our approach supports the storage and transformation of behavioral knowledge which is essential to integration in the area of Web services architecture.

Chapter 11

Evaluating the Composer Against Existing Work

This chapter demonstrates the scalability of our complex-goal-based WS composition. For that purpose, this chapter is structured as follows: In Section 11.1, we present an implementation of our complex-goal-based WS composition. In Section 11.2, we introduce a problem case generator that is capable to generate arbitrarily complex input for the composition component. In Section 11.3, the generator is used to execute performance tests with the composer implementation. Section 11.4 discusses the results of the test runs and Section 11.5 compares our implementation with existing work.

11.1 An implementation of complex-goal-based WS composition

We have implemented a proof-of-concept prototype of our complex-goal-based WS composition. For the explanation, we present screenshots that were taken when composing the behavioral models of the transfer student example depicted in Figure 8.3 on page 115. When executed, the program first asks the user to determine a "locations file" via a file chooser dialog which must contain URL addresses of SA-WSDL files to load. Afterwards, the program loads the SA-WSDL files denoted in the locations file.

The implementation of our complex-goal-based WS composition supports that the generated orchestration is later used in a collaborative business process in the e-business perspective. Therefore, on the subsequent screen, our implementation asks the user to specify which—if any—of the loaded SA-WSDL files acts as the behavioral model that should be provided by the orchestration (Figure 11.1 on the next page). Following the procedure defined in the introduction to Chapter 7 on page 85, the provided behavioral model is given in the consumer view. Therefore, the implementation of our complex-goal-based WS composition converts the provided behavioral model to the provider view by computing the opposite behavioral model as defined in Section 7.4 on page 101.

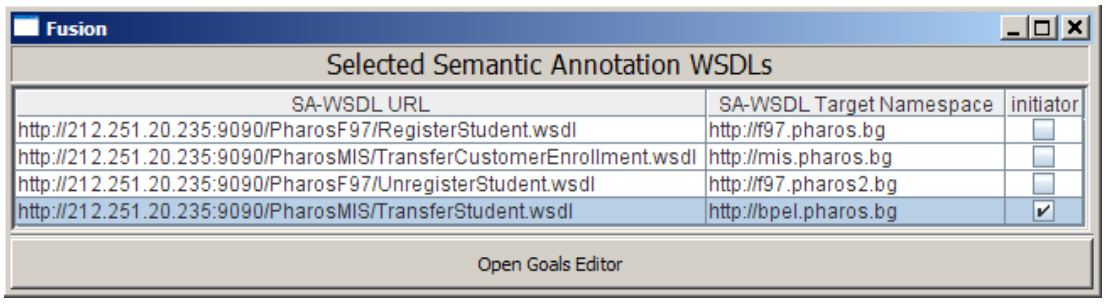


Figure 11.1: Determining the provided behavioral model.

The behavioral models loaded from the SA-WSDLs are displayed on the right-hand side of the subsequent screen (displayed on the following page). The tool uses color coding to differentiate between input and output which cannot be reproduced in a grayscale print. However, as the same example was used as displayed in Figure 8.3 on page 115, that figure may be used as a reference for the direction of communication. Each gray ellipse denotes a state. Each pink word denotes an output variable. Each blue word denotes an input variable. That screen is used to annotate the goals and the variable assignments.

A goal is annotated by selecting one state of each behavioral model and pushing either the “Add New Primary Goal” or the “Add New Recovery Goal” button. After that is done for all goals, the variable assignments can be annotated by drag and drop. A variable assignment is displayed by a dashed arrow starting from an output ending in an input variable. The completely annotated example is depicted on the subsequent page.

When the annotation is done, the composition can be started. When composition is possible given the orchestration requirements, a set of copy rules is generated. In the case that composition was not possible given the orchestration requirements, the failure is reported.

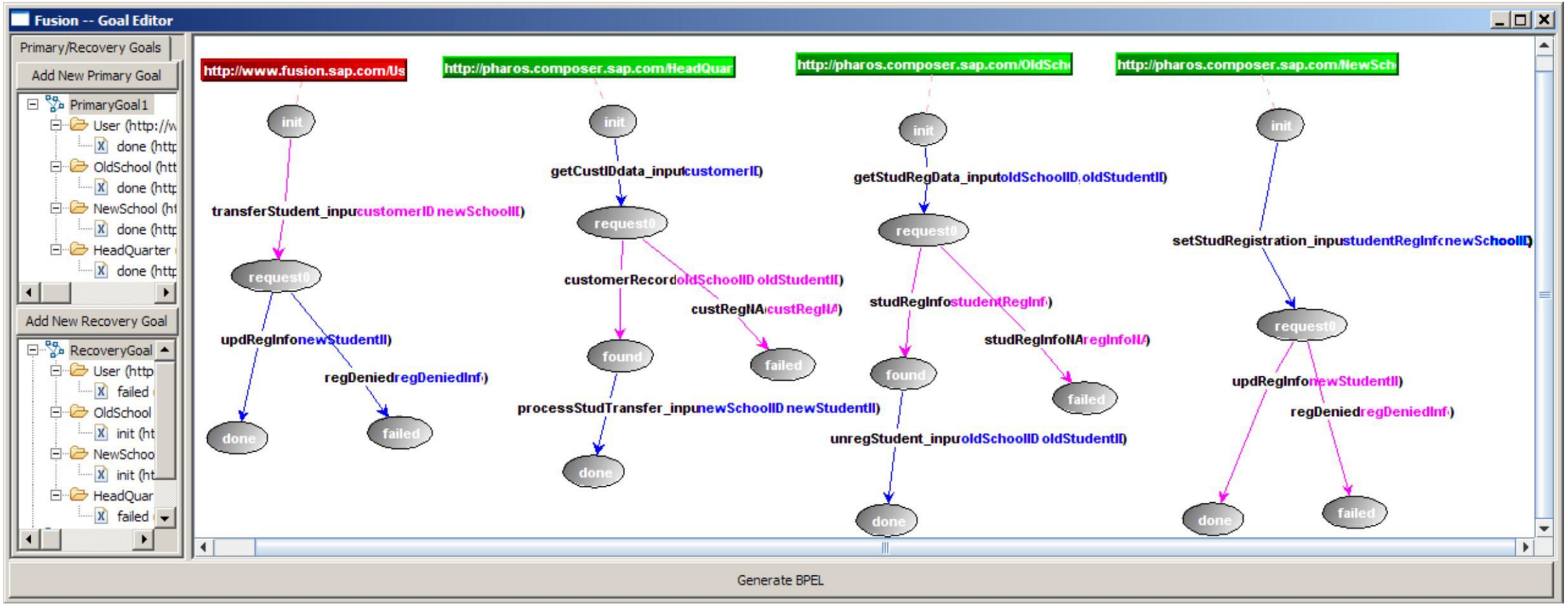
11.2 Problem case generator

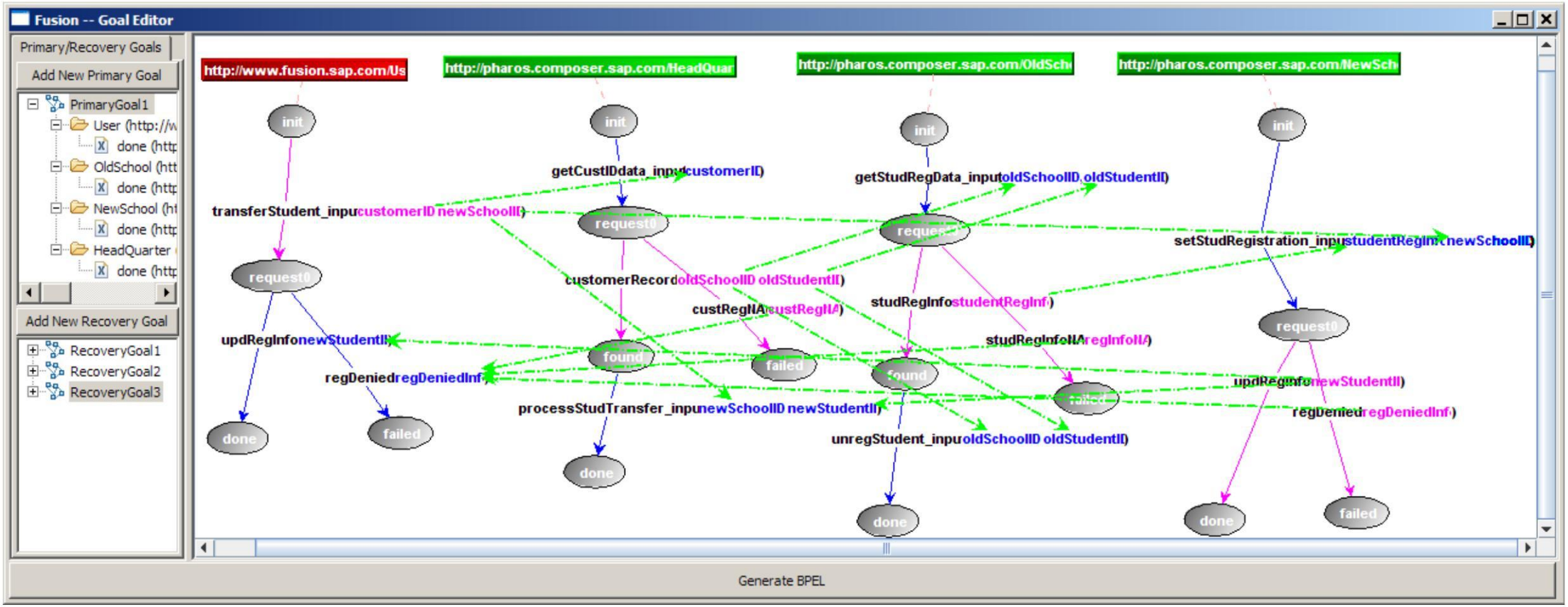
To execute the performance test of the implementation of our complex-goal-based WS composition, we introduce a problem case generator. We have developed the problem case generator to construct arbitrarily complex composition problem cases.

In order to define the complexity of a problem, we introduce three key figures of a composition problem:

1. **Participants (P).** The number of participants states how many behavioral models are being combined to an orchestration.

2. **Branches (B).** The number of branches determines how many non-deterministic decision points each behavioral model has on average.


Fusion -- Goal Editor
Primary/Recovery Goals
Add New Primary Goal
PrimaryGoal1
User (http://w
done (http
OldSchool (htt
done (http
NewSchool (ht
done (http
HeadQuarter
done (http
Add New Recovery Goal
RecoveryGoal
User (http
failed
OldSchool
init (ht
NewSchoo
init (ht
HeadQuar
failed
http://www.fusion.sap.com/Us
init
transferStudent_inputcustomerID newSchoolID)
request0
updRegInfonewStudentID)
regDeniedregDeniedInf)
done
failed
http://pharos.composer.sap.com/HeadQuar
init
getCustIDdata_inputcustomerID)
request0
customerRecordoldSchoolID oldStudentID)
custRegNAcustRegNA)
found
failed
processStudTransfer_inputnewSchoolID newStudentID)
done
http://pharos.composer.sap.com/OldSch
init
getStudRegData_inputoldSchoolID, oldStudentID)
request0
studRegInfostudentRegInf)
studRegInfoNAregInfoNA)
found
failed
unregStudent_inputoldSchoolID oldStudentID)
done
http://pharos.composer.sap.com/NewSch
init
setStudRegistration_inputstudentRegInfnewSchoolID)
request0
updRegInfonewStudentID)
regDeniedregDeniedInf)
done
failed
Generate BPEL


Fusion -- Goal Editor
Primary/Recovery Goals
Add New Primary Goal
PrimaryGoal1
User (http://w
done (http
OldSchool (htt
done (http
NewSchool (ht
done (http
HeadQuarter
done (http
Add New Recovery Goal
RecoveryGoal1
RecoveryGoal2
RecoveryGoal3
http://www.fusion.sap.com/Us
http://pharos.composer.sap.com/HeadQuar
http://pharos.composer.sap.com/OldSch
http://pharos.composer.sap.com/NewSch
init
getCustIDdata_inputcustomerID)
getStudRegData_inputoldSchoolID oldStudentID)
transferStudent_inpucustomerID newSchoolID)
request0
setStudRegistration_inputstudentRegInfo newSchoolID)
customerRecordoldSchoolID oldStudentID)
custRegNAcustRegNA)
studRegInfostudentRegInfo)
updRegInfonewStudentID)
studRegInfoNAregInfoNA)
found
failed
regDeniedregDeniedInfo)
updRegInfonewStudentID)
processStudTransfer_inpunewSchoolID newStudentID)
unregStudent_inputoldSchoolID oldStudentID)
done
Generate BPEL

3. **Length (L).** The mean length denotes the average number of transitions of one path in a behavioral model's state transition system.

The problem case generator performs the steps enumerated below. Step 1 initializes the P behavioral models constructed by the algorithm.

Steps 2 and 3 generate a path that leads to a primary goal consisting of L state transitions in each behavioral model. In one construction step, we always generate a pair of input and output state transitions that exchange the same variable. Therefore, we need $\frac{P \cdot L}{2}$ construction steps in total.

Step 4 is concerned with generating B branches non-deterministically deviating from the path to the primary goal. The construction procedure for each branch is explained in the steps 4a to 4c. As a path to a recovery goal deviates from the path to the primary goal, we pick an arbitrary planning state that would be reachable by the composer and extend each behavioral model by half as many transitions as we did for the primary goal. Concretely, we extend each model by $\frac{L}{2}$ transitions which corresponds to a total of $\frac{P \cdot L}{4}$ construction steps for adding one branch to P participants.

Step 5 returns the generated behavioral models.

1. Create P initial behavioral models, each consisting of an initial state.

2. (a) Pick a behavioral model p_1 whose leaf state is after an input transition or after the initial state.

   (b) Pick a behavioral model $p_2 = p_1$ whose leaf state is after an output transition or after the initial state.

   (c) Introduce a new variable v. Add an output transition sending v to p_1's leaf state. Add an input transition receiving v to p_2's leaf state.

   Perform that step $\frac{P \cdot L}{2}$ times in total.

3. Store all leaf states as the primary goal.

4. (a) Simulate the execution of the path to the primary goal for the so far generated behavioral models. That is, start from the initial states and advance those pairs of states that communicate the same variable. Stop at an arbitrary point and note the current state s_x for each behavioral model p_x.

   (b) i. Pick a behavioral model p_1 where s_1 is after an input transition, before an output transition, or, when p_1 only consists of the initial state, s_1 is the initial state.

   ii. Pick a behavioral model $p_2 = p_1$ where s_2 is after an output transition, before an input transition, or, when p_2 only consists of the initial state, s_2 is the initial state.

   iii. Introduce a new variable v. Add an output transition t_1 sending v to s_1. Add an input transition t_2 receiving v to s_2.

iv. Let s_1 be the sink state of t_1 and s_2 be the sink of t_2.

Perform that step $\frac{PL}{4}$ times in total.

(c) Store $(p_{\ 1}, s_1), \ldots, (p_P, s_P)$ as new recovery goal.

Perform that step B times in total.

5. Output the P behavioral models, the primary goal and the B recovery goals.

We give an example of a generated composition problem in Figure 11.2. States are represented via boxes. Output and input is denoted by text next to edges with a "–>" or a "<–" sign to denote the direction. An arrow pointing toward the transition arc means input, the opposite means output. Goals are marked by the smaller, labeled boxes within the state boxes.

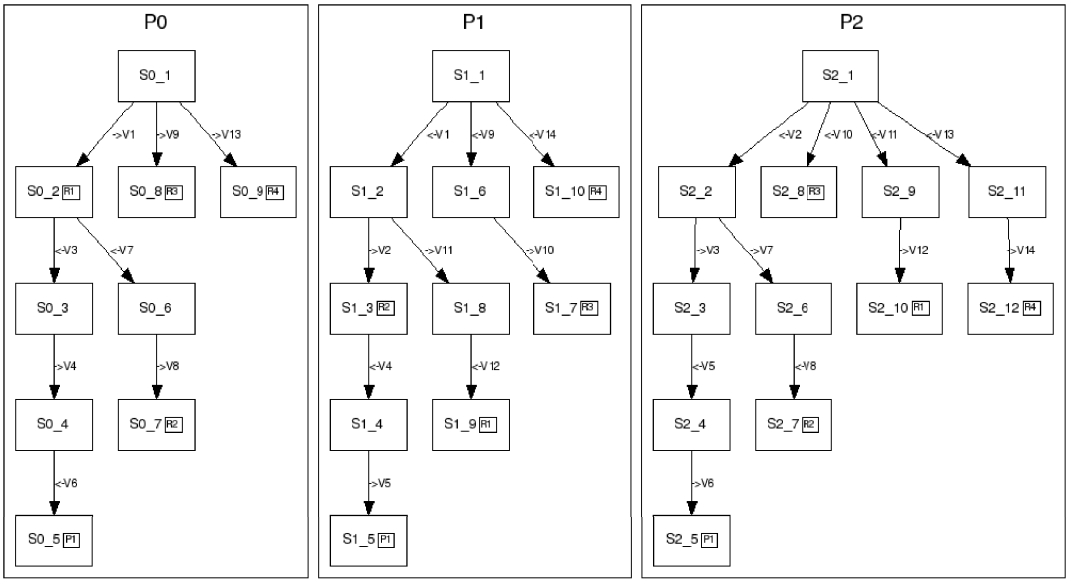


Figure 11.2: Generated composition problem (P:3, B:4, L:4).

11.3 Evaluation runs

In order to assess the quality of our complex-goal-based WS composition implementation, we perform test runs with the following configurations.

No	Participants (P)	Branches (B)	Length (L)
1	$5, \ldots, 80$	0	4
2	10	0	$5, \ldots, 80$
3	10	$1, \ldots, 20$	10
4	10	$1, \ldots, 20$	20

We separately assess the performance of our implementation with respect to the key figures. Therefore, the first set of test runs is fed with an increasing number of behavioral models. In order to reduce the impact of the other key figures, the test behavioral models have no branches and are of a short length. The results are displayed in Figure 11.3.

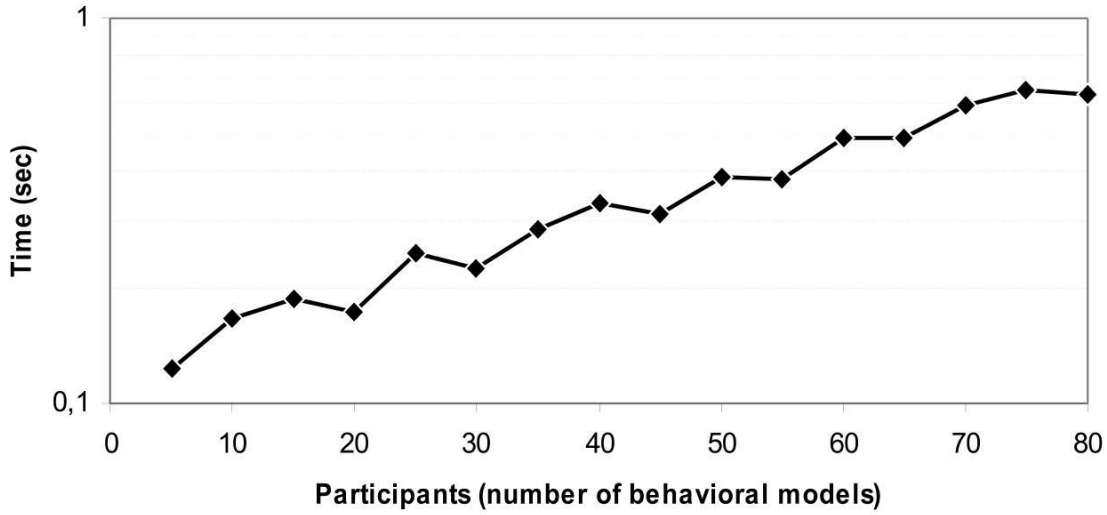


Figure 11.3: Composition time with increasing participants (B:0, L:4).

The second set of tests runs on a fix number of behavioral models without branches. This time, the mean length of the paths is increased for each test run. The results are shown in Figure 11.4 on the next page.

The third and fourth set of test runs are two variations of the same setting. The number of behavioral models and the length are kept fix. This time, the number of branches is increased for each test run. The two variations introduced for the test setting differ on the fix, mean path length of each behavioral model. The run times of the two sets of evaluation runs are displayed in Figure 11.5 on the following page.

11.4 Discussion

For determining the complexity class, we use the data regression analysis technique. During data regression analysis, the data is approximated by a function. Different

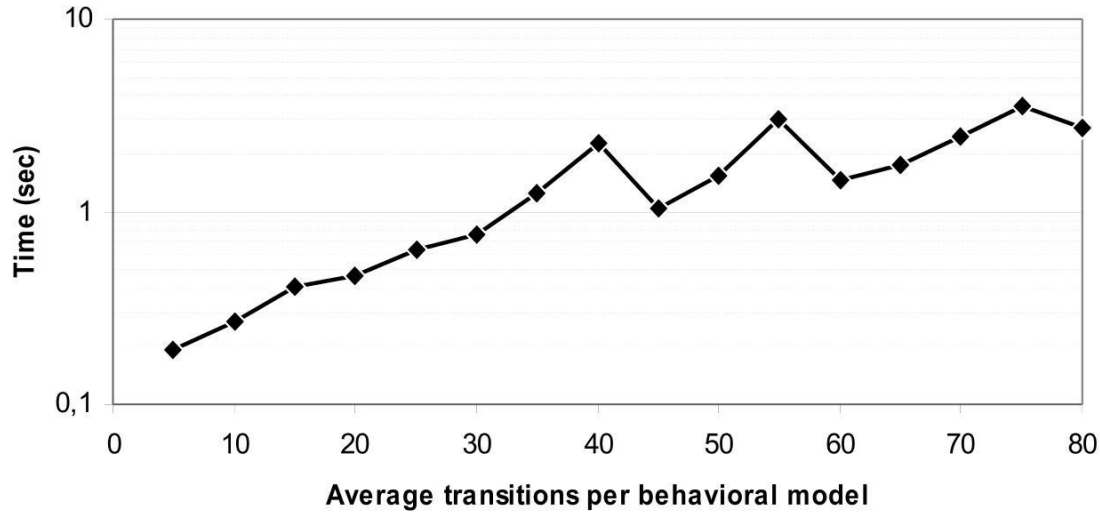


Figure 11.4: Composition time with increasing length (P:10, B:0).

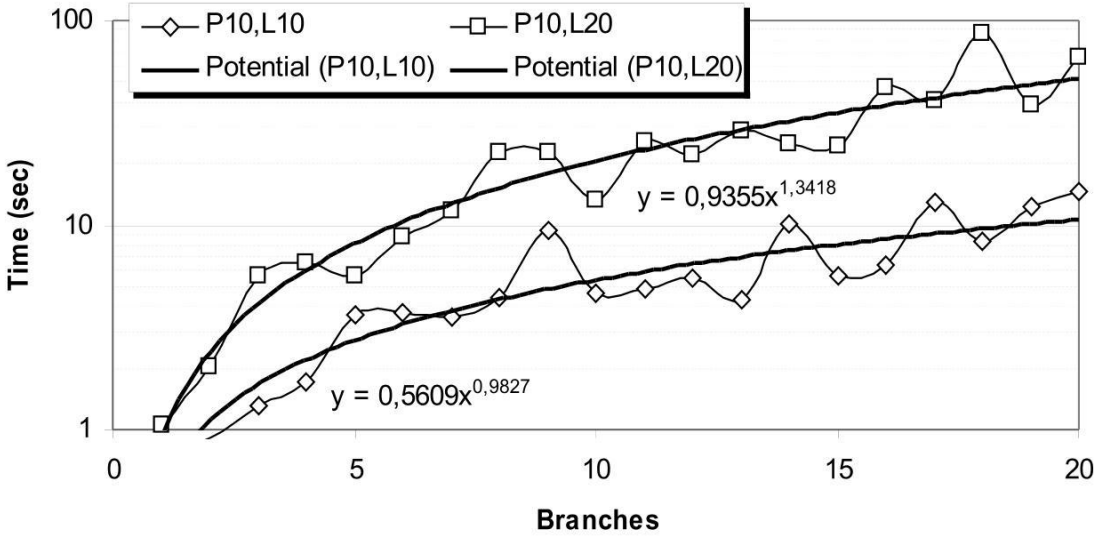


Figure 11.5: Composition time with increasing branches.

types of functions may be compared during regression analysis based on the quality of their approximation. The quality of approximation can be determined using the "coefficient of determination." The coefficient of determination ranges between 0 and 1. The closer the coefficient to 1, the better the approximation. The following table presents the coefficient of determination for different function types for the test runs explained before. For each test run, the best fitting functional type was underlined.

Figure	Linear	Logarithmic	Polynomial	Potential	Exponential
11.3	0.9512	0.7532	0.9765	0.895	0.9729
11.4	0.7701	0.6559	0.7707	0.896	0.8477
11.5 (L:10)	0.7443	0.614	0.759	0.8809	0.7965
11.5 (L:20)	0.7328	0.5499	0.773	0.9359	0.8488

The above table indicates that the development of the run time of our complex-goal-based WS composition with respect to the three key figures can be best approximated by polynomial or potential (power) functions. A power function has the form

$$f : x \quad ax^n \quad a, n \quad \mathbb{R}.$$

A polynomial is built as a sum of power functions with natural number exponents. Both polynomial and power functions grow slower than the exponential function. Thus, we can say that the computation time consumed by our complex-goal-based WS composition grows less than exponentially with an increasing number of participants, length, and branches.

11.5 Comparison with existing approach

In Pistore et al. (2005c), the authors use an increasing number of simple Web services to determine the time needed for composition with their approach. The services to compose follow the request-response pattern. In the request-response pattern, a Web service receives a message and either answers normally or terminates with a failure message. The primary goal of the orchestration is that the normal answers of all participating Web services are communicated to an invoker. In the case that at least one Web service fails, a failure notification is forwarded to the invoker.

We now express the evaluation setting of Pistore et al. (2005c) using our key figures: The number of participants increases from 1 to 24. The resulting orchestration consists of as many non-deterministic decision points as Web services participate besides the invoker: $B = P - 1$. Each Web service consists of two transitions, thus $L = 2$.

We perform one further set of evaluation runs in order to compare our complex-goal-based WS composition to the existing approach of Pistore et al. (2005c).

No	Participants (P)	Branches (B)	Length (L)
5	$2, \ldots, 24$	$1, \ldots, 23$	2

We use the data of the evaluation graph given in Pistore et al. (2005c) to compare our complex-goal-based WS composition against the existing approach. Figure 11.6 displays the run times of our composer plotted above the ones of Pistore et al. (2005c).

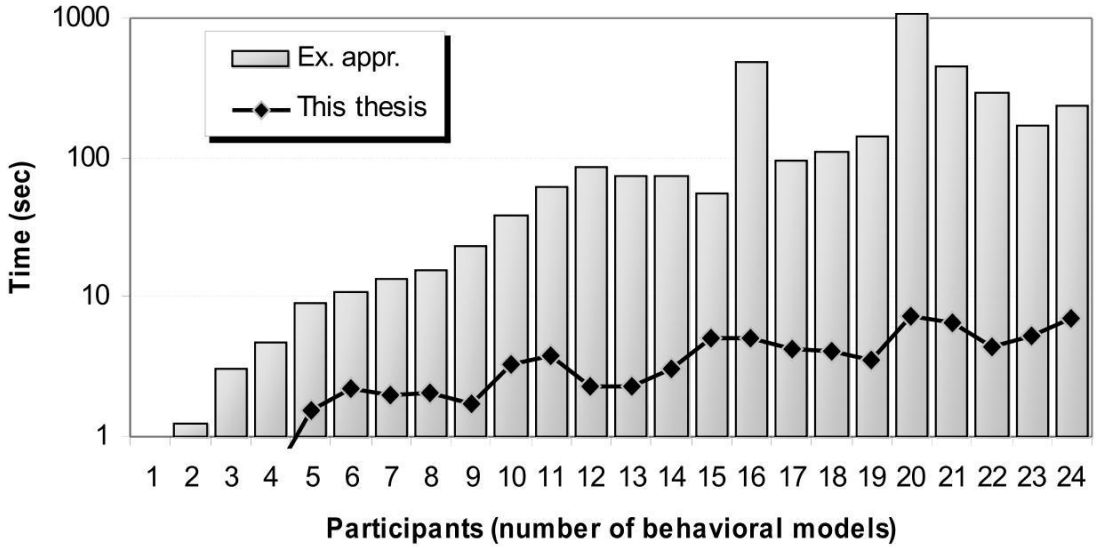


Figure 11.6: Comparison of our complex-goal-based WS composition with existing approach.

We can conclude that our complex-goal-based WS composition runs at least one magnitude of scale faster than the composer presented by Pistore et al. (2005c). The main reasons for that result are that our composer omits the construction of a belief-level model as done by Pistore et al. (2005c) and the more restricted goal definition.

In the approach of Pistore et al. (2005c), a belief-level model is constructed from the joined, original behavioral models by removing all non-observable states. A non-observable state is a state before an output transition or after an input transition if it is not a leaf or initial state. In contrast to the work of Pistore et al. (2005c), our complex-goal-based WS composition operates on the whole behavioral models including non-observable states. Therefore, the expensive computation of the belief-level model can be omitted.

As explained in Section 4.3.3 on page 50, the composition goal is stated as a complex logic expression by Pistore et al. (2005c). Our restriction of the composition goal to nominate a primary and recovery goal instead of a complex logic expression allows for a more efficient implementation of the composition algorithm. However, as we could show in Chapter 10 on page 155, even with the more restrictive goal definition, real-life business processes with non-determinism spanning multiple participants can be successfully composed using our complex-goal-based WS composition.

11.6 Summary

As discussed in the literature review in Section 4.3.3 on page 50, the two closest approaches to our complex-goal-based WS composition are the work performed by Berardi et al. (2005) and by Pistore et al. (2005c). The approach of Berardi et al.

(2005) considers very complex descriptions of the participating components. Due to the complexity of the problem description, the approach of Berardi et al. (2005) has exponential complexity. In contrast, we have shown in this chapter, that our approach has polynomial complexity.

The approach presented by Pistore et al. (2005c) utilizes abstract BPEL files to describe the participating components. Comparing such an abstract BPEL description to the behavioral models we describe using SA-WSDL reveals that the abstract BPEL description contains for the same business process about 3 times as many nodes and transitions. The reason is that internal operations, such as assign, prepare for output, and process input are contained in the abstract BPEL description. That extra burden of information multiplies in the construction of the belief model and in the computation of the joined state space of all participants. In contrast, our work uses a more targeted participant description without internal operations. Additionally, there is no necessity to generate a belief model in our approach, the goal definition is not as rich, and the type of participant descriptions is restricted to trees. These restrictions cause a performance gain of at least one magnitude of scale compared to the closest related work performed by Pistore et al. (2005c).

Part IV

Finale

Chapter 12

Conclusions and Future Work

This chapter serves many purposes: First, we sum our contributions in Section 12.1. Second, we present our conclusions in Section 12.2 by answering the research questions we posed in the introduction. Third, we name relevant activities within SAP in Section 12.3 that benefit from the works presented in this thesis. And finally, we present the further potential of our work in Section 12.4.

12.1 Summary

In this dissertation, we have presented a method and a system for EAI and e-business integration that considers besides technical also ontological aspects in a scalable way. In particular, we identified three current integration challenges that we address with the parts of our solution:

1. **Free definition of interface objects.** The mostly unrestricted definition of interface objects in the past—also supported by technologies such as XML that tremendously simplify the definition of new data types—has led to high maintenance costs. The reason is that redundantly introduced interface object types tend to be inconsistent from the time of their definition or become inconsistent upon change. To tackle our requirement to detect redundant interface objects, we have introduced our CFIM of hierarchical types. We have also evaluated its scalability to the large data sets of the public interfaces of SAP's business software.

2. **Behavioral information only in experts heads.** The knowledge about the intended behavior of software modules created at component design time is today only—if at all—kept in natural language documentation. That has led to expensive reengineering of the behavioral information at integration design time because knowledge stored in natural language leads to different, mostly incompatible interpretations. As a requirement, we formulated to track behavioral information through software design phases. We have proven in this thesis that

our solution transforming behavioral models for EAI and e-business is able to track behavioral information through software design phases.

3. **Manual exception handling.** Identifying the relevant exceptional situations and consistently managing them has been identified as the most expensive part of building an orchestration consuming 80% of the development cost. In addition, if wrongly composed, an orchestration fails only at run time as opposed to already finding errors at integration design time with automated support. Therefore, we require our solution to mechanically support exception handling. Our complex-goal-based WS composition is able to mechanically support exception handling and outperforms related approaches with respect to computation speed as evaluated in this thesis.

12.2 Conclusion

In this thesis, we have answered the three research questions posed in the introduction.

1. *Can ontologically redundant, but structurally different artifacts involved in business collaboration be ef ciently detected?*

   We examined the possible relations between structural and ontological similarity based on the theory of the semiotic triangle. It turned out that we can use a structural analysis to detect potential ontological redundancy. We have adapted an efficient CFIM method to be used with hierarchical types. Our CFIM of hierarchical types scales up to inspecting the large data type repository of SAP's public interface objects.

2. *What is needed to store knowledge of the generally intended and supported behavior of a software component such that it can later be used for creating a partner s role model and a collaborative business process?*

   We defined a solution to transform behavioral models for EAI and e-business as a means to store intended and supported software component behavior. In our solution, behavioral models are considered in EAI when connecting the components of a company to be available for e-business. In e-business, our behavioral models are considered when a collaborative business process is built. As opposed to current practice, no explicit implementation of a company's interface is needed because on orchestration utilizes the components of a company's software directly at run time in our solution.

3. *Can the design of a complex collaborative business procedure with real-world features be ef ciently supported by an automated approach?*

   We provide a complex-goal-based WS composition in this thesis that is capable of handling real-world features of collaborative business procedures such as nondeterministic Web service responses and consistent transactions. Our evaluation

shows that our complex-goal-based WS composition outperforms the fastest existing approach with the named features. We have also shown that our complex-goal-based WS composition can be used to build a realistic best practice business process supported by SAP.

12.3 Impact

The work presented in this dissertation contributes to two activities in SAP:

1. **CCTS Modeler Warp 10.** The core component type specification (CCTS)[1] provides a method, a naming convention, and a schema to define types of data and to cover their intended meaning. The aim is to improve the alignment between technical data types of different collaborators. The CCTS Modeler Warp 10 is a software tool developed at SAP that allows the CCTS-based definition of data types in conjunction with existing types defined by others before (Stuhec, 2007). Our CFIM of hierarchical types successfully contributed to a prototype of the CCTS Modeler. Our CFIM of hierarchical types is used in an offline extension to the CCTS Modeler prototype that detects redundant data type definitions and proposes a restructuring of the CCTS Modeler's data type repository in order to establish a normal form of the overall schema.

2. **SAP NetWeaver CE.** The composition environment (CE) of SAP NetWeaver is a software tool to develop, run, and manage composite applications using SAP's enterprise SOA. A composite application, or composite, accesses existing software via Web service technology to achieve new functionality. It was announced that a future version of SAP NetWeaver CE will provide a BPMN-based modeling environment to integrate software based on Web service technology purely by modeling without the necessity to write code. We could show how a proof-of-concept implementation of our complex-goal-based WS composition can semi-automatically support the manual modeling in SAP NetWeaver CE.

12.4 Further development

In this section, we present how our work could be extended in the future. We advocate the vision that our work could be extended to bring forth a new kind of middleware for improved collaboration. To serve that purpose, the three main topics of this thesis could be further developed in the ways described in this section.

[1] `http://www.untmg.org/specifications/`

12.4.1 CFIM of hierarchical types

That solution detects redundant interface objects. In addition to the detection, an automated support for the alignment of the two technically different, but ontologically redundant interface objects could be given. Alignment could be reached by the redefinition of some of the redundant interface objects. That involves reprogramming of software. Alternatively, alignment could be reached by introducing adapters and, thus, leaving existing software untouched. The creation of adapters could further be supported by the result of the CFIM of hierarchical types because a redundancy group contains information about already matching elements.

In the case that a way can be found to create adapters that work in both directions "from the old to the new" and "from the new to the old" interface object, a middleware can be defined with the following characteristics:

- **Publish interface objects.** Potential collaborators can connect to the middleware by publishing their interface objects.
- **Identify redundancy.** Our CFIM of hierarchical types can be used to identify ontologically redundant interface objects. That is a major prerequisite for collaboration as discussed in this thesis.
- **Wrap collaborators.** Collaborators establish a connection with other collaborators exclusively through the middleware. That means in particular that a collaborator perceives the other collaborators' interfaces through the view provided by the middleware.
- **Manage versions.** A new, aligned version of an interface object can be defined from redundant interface objects. All newly established connections only use the most recent version of an interface object. Collaborators already connected to elderly versions of interface objects keep working because the middleware cares for proper conversion between the different versions of interface objects the collaborators rely on.
- **Remove obsolete versions.** When a collaborator permanently disconnects from the community of potential collaborators, the middleware checks whether the interfaces used by the leaving collaborator have been replaced by newer versions in the meantime. If that is the case and no other collaborator relies on the old interface object, the object definition plus all respective adapters become removed from the middleware.

12.4.2 Transform behavioral models for EAI and e-business

When we introduced behavioral models, we provided a formal model of behavioral knowledge. However, a business user with little technical expertise could be challenged to use the state transition system notation for defining collaborative business process

requirements including goal definition and variable assignments in an e-business setting. Therefore, further development should provide a more usable notation of behavioral models, variable assignments, and composition goals.

12.4.3 Complex-goal-based WS composition

The middleware we described above establishes a common language among all potential collaborators. That is the prerequisite for collaboration. We envision our complex-goal-based WS composition to be a further feature of the middleware that allows the potential collaborators—based on the common understanding reached by our CFIM of hierarchical types—to connect their companies' behavioral models to a collaborative business process.

To increase the usability of our complex-goal-based WS composition, we envision that our complex-goal-based WS composition could be extended in the following ways:

- **Iterative definition of recovery goals.** Currently, the orchestration requirements, i. e., variable assignments and goals, need to be given before our composer is started. Especially, the recovery goals may contain many situations that never occur in an orchestration. Therefore, one could redefine the composer to perform an interactive composition:

  The initial input for the composer would be the primary goal. The composer would start to chain an orchestration backward for the primary goal. Whenever the composer reaches a point of non-determinism, the composer would ask the user to specify allowable recovery goals. The advantage of that technique is that at the time the composer asks to define recovery goals, it may present a restricted graph to the user that only contains those branches that could still be traversed from the current state of backchaining.

  In addition, the composer could first check whether reaching the primary goal is possible in principle, i. e. without considering non-determinism. Thus, the composer would not bother the user with defining recovery goals if in the end it turns out that even trying to reach the primary goal fails.

- **Explanation of composition failures.** As described in this thesis, the complex-goal-based WS composition either generates an orchestration that consistently ensures transactionality or it fails. In the case of failure, one could imagine that a detailed failure report could be generated to hint a human user at the reason for the failure of composition. Potential reasons for failure include

  - incompatible sequencing of messages among the participating behavioral models,
  - inputs that cannot be matched with outputs, and
  - that a consistent outcome cannot be guaranteed for each execution.

A failure report may state the category of failure including information

  - about the sequences that did not match,
  - the inputs with cannot be served, and respectively
  - the branches whose composition fails.

- **Generating mediator stub upon composition failure.** In the case that composition was not possible, a sophisticated extension to our complex-goal-based WS composition could try to generate a mediating behavioral model that, for example, resolves improper sequencing of messages. Such a mediating behavioral model would have to be backed with an appropriate implementation to perform the behavior imposed by the mediating behavioral model in order to be executable at run time.

The named extensions of our solutions presented in this thesis and the middleware described in the outlook target at ensuring and improving the quality and maintainability of integration. As especially our experiments on real interface data from industry have shown, scalability is key to be applicable in reality. Although a linear time and space mining algorithm was used, with increasing granularity of the mining, the run time also increased quickly due to the fast increase of the problem size.

The other essential aspect of supporting integration is considering the intentions the creators had when designing their interface objects. It is mandatory to make that knowledge explicit and to preserve it as complete and as durable as possible during the lifecycle of the objects. An ontological approach is key for later integration because the probably largest cost driver in software and integration development is misunderstanding and forgetting relationships between software artifacts.

In this thesis, we proposed a method and tools to acquire knowledge about software components and make it available to people involved in EAI and e-business integration. We turned our attention to provide an ontological approach. In some points, we accepted reduced expressivity for the sake of scalability. That is where future research may contribute.

References

Albert, P., L. Henocque, and M. Kleiner. Configuration-based workflow composition. In *ICWS*, pages 285–292. IEEE Computer Society, 2005a. ISBN 0-7695-2409-5.

Albert, P., L. Henocque, and M. Kleiner. A constrained object model for configuration based workflow composition. In Bussler and Haller (2006), pages 102–115.

Altenhofen, M., E. Börger, and J. Lemcke. An abstract model for process mediation. In K.-K. Lau and R. Banach, editors, *Formal Methods and Software Engineering, 7th International Conference on Formal Engineering Methods, ICFEM*, volume 3785 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2005a. ISBN 3-540-29797-9. URL `http://dx.doi.org/10.1007/11576280_7`.

Altenhofen, M., E. Börger, and J. Lemcke. An execution semantics for mediation patterns. In *Proc. of 2nd WSMO Implementation Workshop (WIW)*. CEUR Workshop Proceedings, Innsbruck, Austria, 2005b. ISSN 1613-0073. URL `CEUR-WS.org/Vol-134/lemcke-wiw05.pdf`.

Altenhofen, M., E. Börger, and J. Lemcke. A high-level specification for mediators (Virtual Providers). In Bussler and Haller (2006), pages 116–129. URL `http://dx.doi.org/10.1007/11678564_11`.

Altenhofen, M., A. Friesen, and J. Lemcke. ASMs in service oriented architectures. *Journal of Universal Computer Science (JUCS)*, 14(12):2034–2058, 2008.

Altenhofen, M., A. Friesen, J. Lemcke, and E. Börger. A high-level specification for Virtual Providers. *International Journal of Business Process Integration and Management (IJBPIM)*, 1(4):267–278, 2006b.

Antoniou, G. and F. v. Harmelen. *A semantic Web primer*. MIT Press, 2004. ISBN 0-262-01210-3.

Baader, F., D. Calvanese, and D. McGuinness. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Press, second edition, 2007. ISBN 978-0-521-87625-4.

Benatallah, B., M. Dumas, M. Fauvet, and F. Rabhi. Towards patterns of Web services composition. 2002. URL citeseer.ist.psu.edu/benatallah02towards.html.

Berardi, D., D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic Web services with messaging. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-A. Larson, and B. C. Ooi, editors, *31st International Conference on Very Large Data Bases (VLDB)*, pages 613–624. ACM, 2005. ISBN 1-59593-154-6, 1-59593-177-5.

Bernstein, A., M. Klein, and T. W. Malone. The process recombinator: a tool for generating new business process ideas. In *International Conference on Information Systems (ICIS)*, pages 178–192. 1999. URL citeseer.ist.psu.edu/bernstein99proces.html.

Berry, M. J. A. and G. S. Linoff. *Data Mining Techniques*. Wiley Publishing, Inc., second edition, 2004.

Bertoli, P., M. Pistore, and P. Traverso. Automated Web service composition by on-the-fly belief space search. In D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, editors, *16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 358–361. AAAI, 2006. ISBN 978-1-57735-270-9.

Booch, G., J. Rumbaugh, and I. Jacobson. *The Uni ed Modeling Language User Guide*. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, second edition, 2005.

Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, et al. Web services architecture – W3C working group note. Technical report, W3C, 2004. URL http://www.w3.org/TR/ws-arch/.

Börger, E. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.

Börger, E. and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

Bray, T., J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fourth edition). Technical report, W3C, 2006. URL http://www.w3.org/TR/xml/.

Bussler, C. and A. Haller, editors. *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, volume 3812. 2006. ISBN 3-540-32595-6.

Cardoso, J. and A. Sheth. Semantic e-workflow composition. *J. Intell. Inf. Syst.*, 21(3):191–225, 2003. ISSN 0925-9902.

Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. Technical report, W3C, 2001. URL `http://www.w3.org/TR/wsdl`.

Chun, S. A., V. Atluri, and N. R. Adam. Domain knowledge-based automatic workflow generation. In *Database and Expert Systems Applications : 13th International Conference, DEXA 2002 Aix-en-Provence, France, September 2-6, 2002. Proceedings*, page 81ff. 2002. `http://www.springerlink.com/content/55ra4dhu04wfxnq1`.

Cimpian, E. Process mediation in semantic Web services. In E. P. B. Simperl, J. Diederich, and G. Schreiber, editors, *Knowledge Web PhD Symposium (KWEPSY)*, volume 275 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Clement, L., A. Hately, C. von Riegen, and T. Rogers. UDDI version 3.0.2, UDDI spec technical committee draft, dated 20041019. Technical report, OASIS, 2004. URL `http://uddi.org/pubs/uddi_v3.htm`.

Dong, A. XML tree finder system: A first step towards XML data mining – final report. Technical report, CS Department, Univ of Calgary, 2005.

Dong, X., A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for Web services. In *30th International Conference on Very Large Data Bases (VLDB)*. 2004.

Doshi, P., R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition using markov decision processes. In *ICWS 04: Proceedings of the IEEE International Conference on Web Services (ICWS 04)*, page 576. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2167-3.

Drumm, C., J. Lemcke, E. Dorner, and L. Heuser. Enterprise services architecture & semantic Web services. In S. Reich, G. Günter, T. Pellegrino, and A. Wahler, editors, *Semantic Content Engineering*, volume 17 of *Schriftenreihe Informatik*, pages 28–37. Trauner, 2006a. ISBN 3854879792.

Drumm, C., J. Lemcke, and K. Namiri. Integrating semantic Web services and business process management: A real use case. In *Semantics for Business Process Management, Workshop at 3rd European Semantic Web Conference (ESWC)*. 2006b. URL `http://jobfunctions.bnet.com/abstract.aspx?promo=50002&docid=276296`.

Drumm, C., J. Lemcke, and D. Oberle. Business process management and semantic technologies. In J. Cardoso, M. Hepp, and M. D. Lytras, editors, *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond*

Computing for Human Experience, pages 209–239. Springer, 2007a. ISBN 978-0-387-48531-7. URL http://dx.doi.org/10.1007/978-0-387-48531-7_10.

Drumm, C., M. Schmitt, H. H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. In M. J. Silva, A. H. F. Laender, R. A. Baeza-Yates, D. L. McGuinness, B. Olstad, Ø. H. Olsen, and A. O. Falcão, editors, *16th ACM Conference on Information and Knowledge Management (CIKM)*, pages 107–116. ACM, 2007b. ISBN 978-1-59593-803-9.

Fallside, D. C. and P. Walmsley. XML schema part 0: Primer second edition – W3C recommendation. Technical report, W3C, 2004.

Farahbod, R., V. Gervasi, and U. Glässer. Coreasm: An extensible asm execution engine. *Fundam. Inform.*, 77(1-2):71–103, 2007.

Fensel, D. and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

Fensel, D., H. Lausen, A. Polleres, M. Stollberg, D. Roman, et al. *Enabling semantic Web services*. Springer, 2007. ISBN 3-540-34519-1, 978-3-540-34519-0.

Friesen, A. and J. Lemcke. Composing Web-service-like abstract state machines (ASM). In J. Koehler, M. Pistore, A. P. Sheth, P. Traverso, and M. Wirsing, editors, *Autonomous and Adaptive Web Services*, volume 07061 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. URL http://drops.dagstuhl.de/opus/volltexte/2007/1034.

Fronk, M. and J. Lemcke. Expressing semantic Web service behavior with description logics. In *Semantics for Business Process Management, Workshop at 3rd European Semantic Web Conference (ESWC)*. 2006. URL http://km.aifb.uni-karlsruhe.de/ws/sbpm2006/papers/sbpm06_Fronk.pdf.

Ganter, B. and R. Godin. *Formal Concept Analysis*. Springer, 2005.

Gerede, C. E. and J. Su. Specification and verification of artifact behaviors in business process models. In B. J. Krämer, K.-J. Lin, and P. Narasimhan, editors, *5th International Conference on Service-Oriented Computing (ICSOC)*, volume 4749 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2007. ISBN 978-3-540-74973-8.

Gmez-Prez, A., M. Fernndez-Lpez, and O. Corcho. *Ontological engineering*. Springer, 2004. ISBN 1-85233-551-3.

Gorton, I. *Essential Software Architecture*. Springer, 2006. ISBN 3-540-28713-2, 978-3-540-28713-1.

Gruber, T. R. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

Han, J. and M. Kamber. *Data Mining. Concepts and Techniques*. Morgan Kaufmann Publishers, 2006. ISBN 1558609016.

Hoffmann, J., P. Bertoli, and M. Pistore. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In *AAAI*, pages 1013–1018. AAAI Press, 2007. ISBN 978-1-57735-323-2.

Kaiser, M. Toward the realization of policy-oriented enterprise management. *IEEE Computer*, 40(11):57–63, 2007.

Kaiser, M. and J. Lemcke. Towards a framework for policy-oriented enterprise management. In K. Hinkelmann, editor, *Papers from the AAAI Spring Symposium*. 2008. ISBN 978-1-57735-357-7. Technical Report SS-08-01.

Kalfoglou, Y. and W. M. Schorlemmer. Ontology mapping: The state of the art. In Y. Kalfoglou, W. M. Schorlemmer, A. P. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, volume 04391 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2005.

Karastoyanova, D. and A. P. Buchmann. Automating the development of Web service compositions using templates. In *GI Jahrestagung (2)*, pages 517–523. 2004.

Kastner, P. S. and R. Saia. The composite applications benchmark report. Technical report, Aberdeen Group, 2006.

Kifer, M. and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In J. Clifford, B. G. Lindsay, and D. Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 134–146. ACM Press, 1989.

Kim, J., M. Spraragen, and Y. Gil. An intelligent assistant for interactive workflow composition. In *IUI 04: Proceedings of the 9th international conference on Intelligent user interface*, pages 125–131. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-815-6.

Krafzig, D., K. Banke, and D. Slama. *Enterprise SOA : Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, 2004. ISBN 0131465759. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&path=ASIN/0131465759`.

Kubczak, C., T. Margaria, M. Kaiser, J. Lemcke, and B. Knuth. Abductive synthesis of the mediator scenario with jABC and GEM. In R. Garcia-Castro, A. Gómez-Pérez,

C. J. Petrie, E. D. Valle, U. Küster, M. Zaremba, and M. O. Shafiq, editors, *6th International Workshop on Evaluation of Ontology-based tools and the Semantic Web Service Challenge (EON-SWSC)*, volume 359 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL http://ceur-ws.org/Vol-359/Paper-5.pdf.

Küster, J. M., K. Ryndina, and H. Gall. Generation of business process models for object life cycle compliance. In G. Alonso, P. Dadam, and M. Rosemann, editors, *5th International Conference on Business Process Management (BPM)*, volume 4714 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2007. ISBN 978-3-540-75182-3.

Lago, U. D., M. Pistore, and P. Traverso. Planning with a language for extended goals. In *AAAI Innovative Applications of Arti cial Intelligence (IAAI)* , pages 447–454. 2002.

Laukkanen, M. and H. Helin. Composing workflows of semantic Web services. In *Proceedings of the Workshop on Web-Services and Agent-based Engineering*. 2003. URL http://jmvidal.cse.sc.edu/library/laukkanen03a.pdf.

Lausen, H., J. de Bruijn, A. Polleres, and D. Fensel. The wsml rule languages for the semantic web. In *W3C Workshop on Rule Languages for Interoperability*. W3C, 2005.

Lemcke, J. Light-weight semantic integration of generic behavioral component descriptions. In G. Mentzas, T. Bouras, P. Gouvas, and A. Friesen, editors, *Semantic Enterprise Application Integration for Business Processes*. IGI Global, Harrisburg, PA, 2009.

Lemcke, J. and C. Drumm. Semantic technologies for enterprise services. In *Semantic Web Days*. Online Proceedings, 2005. ISSN 1613-0073. URL http://www.semantic-web-days.net/proceedings/SAP_SemanticWebDays2005.pdf.

Lemcke, J. and C. Drumm. Semantic business automation. In A. Leger, A. Kulas, L. Nixon, and R. Meersman, editors, *How Business application challenges Semantic Web Research – European Semantic Web Conference (ESWC) Industry Forum*, volume 194. CEUR Workshop Proceedings, Budva, Montenegro, 2006. ISSN 1613-0073. URL CEUR-WS.org/Vol-194/paper7.pdf.

Lemcke, J. and A. Friesen. Composing Web-service-like abstract state machines (ASMs). In *IEEE International Conference on Services Computing - Workshops (SCW)*, pages 262–269. IEEE Computer Society, 2007a. URL http://doi.ieeecomputersociety.org/10.1109/SERVICES.2007.24.

Lemcke, J. and A. Friesen. Considering realistic Web service features for semi-automatic composition. *Annals of Mathematics, Computing and Teleinformatics (AMCT)*, 1(5):26–35, 2007b.

Lemcke, J. and A. Friesen. Considering realistic Web service features for semi-automatic composition. In *3rd South-East European Workshop on Formal Methods (SEEFM)*. Springer, 2007c.

Lemcke, J. and A. Friesen. Semi-automatic design of collaborative business processes in FUSION. In *Demonstrating Research Results on Enterprise Interoperability, Workshop at Int l Conf. Interoperability for Enterprise Software (i-ESA)*. 2008. URL `http://www.genesis-ist.eu/LinkClick.aspx?link=iESA_2008_ProceedingsBerlin.pdf&tabid=79&mid=413`.

Lemcke, J., M. Kaiser, C. Kubczak, T. Margaria, and B. Knuth. Advances in solving the mediator scenario with jABC and jABC/GEM. In *7th Semantic Web Services Challenge Workshop, part of the 7th International Semantic Web Conference ISWC*. 2008.

Mayer, A., S. McGough, N. Furmento, W. Lee, S. Newhouse, et al. Iceni dataflow and workflow: Composition and scheduling in space and time. 2003. URL `citeseer.ist.psu.edu/mayer03iceni.html`.

McCormick, W. T., P. J. Schweitzer, and T. W. White. Problem decomposition and data reorganization by a clustering technique. *Operation Research*, 20(5):993–1009, 1972.

Melzer, I. *Service-orientierte Architekturen mit Web Services*. Elsevier, Spektrum Akad. Verl., second edition, 2007. ISBN 978-3-8274-1885-2, 3-8274-1885-2.

Mens, T. and T. Tourw. A survey of software refactoring. In *IEEE Transactions on Software Engineering*. 2004.

Meyer, H., H. Overdick, and M. Weske. Plngine: A system for automated service composition and process enactment. In *Proceedings of the 14th International Conference on World Wide Web (WWW), Service Composition with Semantic Web Services (wscomps05)*, pages 3–12. University of Technology of Compiegne, Compiegne, France, 2005. ISBN 2-913923-18-6.

Noy, N. F. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

Ogden, C. K. and I. A. Richards. *The Meaning of Meaning: A Study of the In uence of Language Upon Thought and of the Science of Symbolism*. London: Routledge & Kegan Paul, 1923.

Papazoglou, M. P. and P. Ribbers. *e-Business: Organizational and Technical Foundations*. Wiley, 2006. ISBN 0470843764.

Petersohn, H. *Data Mining – Verfahren, Prozesse, Anwendungsarchitektur*. Oldenbourg Wissenschaftsverlag, 2005.

Pistore, M., F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring Web service composition. In C. Bussler and D. Fensel, editors, *11th International Conference on Arti cial Intelligence: Methodology, Systems, and Applications (AIMSA)*, volume 3192 of *Lecture Notes in Computer Science*, pages 106–115. Springer, 2004. ISBN 3-540-22959-0.

Pistore, M., A. Marconi, P. Bertoli, and P. Traverso. Automated composition of Web services by planning at the knowledge level. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 1252–1259. Professional Book Center, 2005a. ISBN 0938075934.

Pistore, M., P. Roberti, and P. Traverso. Process-level composition of executable Web services: "on-the-fly" versus "once-for-all" composition. In A. Gómez-Pérez and J. Euzenat, editors, *2nd European Semantic Web Conference (ESWC)*, volume 3532 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2005b. ISBN 3-540-26124-9.

Pistore, M., P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of executable Web service compositions from BPEL4WS processes. In A. Ellis and T. Hagino, editors, *14th International Conference on World Wide Web (WWW), Special interest tracks and posters*, pages 1186–1187. ACM, 2005c. ISBN 1-59593-051-5.

Rahm, E. and P. A. Bernstein. A survey of approaches to automatic schema matching. *The International Journal on Very Large Data Bases*, 10(4):334–350, 2001.

Rao, J. and X. Su. A survey of automated Web service composition methods. In J. Cardoso and A. P. Sheth, editors, *1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004. ISBN 3-540-24328-3.

Rosenkrantz, G. The science of being. *Erkenntnis*, 48:251–255(5), 1998. URL `http://www.ingentaconnect.com/content/klu/erke/1998/00000048/F0020002/00183913`.

Ruh, W. A., F. X. Maginnis, and W. J. Brown. *Enterprise application integration*. Wiley, 2001. ISBN 0-471-37641-8.

Salaun, G., L. Bordeaux, and M. Schaerf. Describing and reasoning on Web services using process algebra. *International Journal of Business Process Integration and Management (IJBPIM)*, 1(2):116–128, 2006.

Serain, D. *Middleware and enterprise application integration*. Springer, 2002. ISBN 1-85233-570-X.

Sharman, R., R. Kishore, and R. Ramesh, editors. *Ontologies*. Springer, 2007. ISBN 0-387-37019-6, 978-0-387-37019-4.

Shvaiko, P. and J. Euzenat. A survey of schema-based matching approaches. Technical report, Informatica e Telecomunicazioni, University of Trento, 2005.

Snelting, G. and F. Tip. Reengineering class hierarchies using concept analysis. In *Foundations of Software Engineering*, pages 99–110. 1998.

Staab, S. and R. Studer, editors. *Handbook on ontologies*. Springer, 2004. ISBN 3-540-40834-7.

Stachowiak, H. *Allgemeine Modelltheorie*. Springer-Verlag, Wien New York, 1973.

Stahl, T. and M. Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

Stojanovic, L. *Methods and tools for ontology evolution*. Ph.D. thesis, University of Karlsruhe, Germany, 2004.

Studer, R., S. Grimm, and A. Abecker, editors. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007. ISBN 978-3-540-70893-3, 3-540-70893-6.

Stuhec, G. Using CCTS Modeler Warp 10 to customize business information interfaces. Technical report, SAP AG, 2007. URL `https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/70d6c441-507e-2a10-7994-88f6f769d6e8`.

Stumme, G., R. Taouil, Y. Bastide, N. pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data & Knowledge Engineering*, 42:189–222, 2002.

Termier, A. *Extraction of frequent trees in an heterogeneous corpus of semi-structured data: application to XML documents mining*. Ph.D. thesis, Paris-South University, 2004.

Trainotti, M., M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, et al. ASTRO: Supporting composition and execution of Web services. In B. Benatallah, F. Casati, and P. Traverso, editors, *3rd International Conference on Service-Oriented Computing (ICSOC)*, volume 3826 of *Lecture Notes in Computer Science*, pages 495–501. Springer, 2005. ISBN 3-540-30817-2.

Traverso, P. and M. Pistore. Automated composition of semantic Web services into executable processes. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2004. ISBN 3-540-23798-4.

Uno, T., T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pages 16–31. 2004a.

Uno, T., M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE International Conference on Data Mining, Workshop on Frequent Itemset Mining Implementations (FIMI)*. 2004b.

van der Aalst, W. M. P., A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

van Deursen, A. and T. Kuipers. Identifying objects using cluster and concept analysis. In *21st International Conference on Software Engineering*, pages 246–255. 1999.

Verma, K., K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, University of Georgia, Athens, 2005. URL `http://lsdis.cs.uga.edu/projects/meteor-s/techRep6-24-05.pdf`.

Wang, Y. and E. Stroulia. Semantic structure matching for assessing Web service similarity. In *1st International Conference on Service-Oriented Computing (ICSOC)*, volume 2910 of *LNCS*, pages 194–207. Springer, 2003.

Witten, I. H. and E. Frank. *Data Mining*. Elsevier Inc., second edition, 2005.

Zaki, M. J. Efficiently mining frequent trees in a forest. 2002. In KDD'02, URL `citeseer.ist.psu.edu/zaki02efficiently.html`.

Zaki, M. J., N. Parimi, N. De, F. Gao, B. Phoophakdee, et al. Towards generic pattern mining. In *Formal Concept Analysis*. 2005.

List of Figures

Appendix A

Core composition algorithm

In this chapter, we detail the core composition algorithm initially mentioned in Section 8.5 on page 123.

A.1 Formal representation of REACHVARIANT

We formally define the computation of REACHVARIANT below. In the following section, we present the contained high-level steps in more detail.

REACHVARIANT(vnt Variant, g Goal, $A \subsetneq \mathcal{A}$, ss SimState) $\equiv$
return ($fail$, $copyRules$)
step
oldps $:= \varnothing$
$fail :=$ *false*
$copyRules := \varnothing$
ps := (bm, s, m) PlState : (bm, s) g,

$$m = \begin{cases} OUT, \text{hasAdjacentOutTrans}(bm,\ s) \\ IN, \quad \text{hasAdjacentInTrans}(bm,\ s) \\ \text{undef}, s = q_0^{bm} \end{cases}$$

forall bm ids(ps) **do** outPool(bm) := o O :
(q_{pre}, O, q_{post}) $Rules$, (bm, $Rules$) vnt, $O \subseteq \text{OUT}^{bm}$,
((bm_{out}, o), (bm_{in}, i)) A, but only what lies between the initial state and the goal state in this variant
step **while** **not** $fail$ **and** **not** done(ps, ss) **and** **not** ps = oldps **do**
oldps := ps
step **do** **forall** bm ids(ps)
adjInTrans(bm) := (q_{pre}, I, q_{post_1}) $Rules$: I IN^{bm},
(bm, $Rules$) vnt, [(bm, q_{post_2}, IN) ps,
(q_{post_1}, O, q_{post_2}) $Rules$] [(bm, q_{post_1}, IN) ps]
step ($fail$, currAss) :=

```
            CALCINPUTSSERVED(ids(ps), A, adjInTrans, outPool)
        step if not fail then
          copyRules :=
              copyRules      CREATECOPYRULE(ps, currAss, adjInTrans)
          step outPool := UPDATEOUTPOOLS(vnt, ps, currAss, A, outPool)
          step ps :=
              CREATENEWPLANNINGSTATE(vnt, ps, adjInTrans, outPool, ss)
      step if not done(ps, ss) then fail :=true
where
  done(ps  PlState, ss  SimState) ≡  ( bm, q_ps, m )  ps :
      [ m  undef, OUT  ( bm, q_ps )  ss ]
      [ m = IN  ( bm, q_ss )  ss  ( q_ss, V, q_ps )  δ^bm  V ⊆ OUT^bm ]
  hasAdjacentOutTrans(bm, q_post) :=  ( q_pre, O, q_post )  δ^bm_out
  hasAdjacentInTrans(bm, q_post) :=  ( q_pre, I, q_post )  δ^bm_in
```

In the following, we go in detail through the individual steps of the core composition algorithm as presented in Section 8.5 on page 123. Each of these steps is presented as an ASM.

A.2 Input and output assignments

The matching of input variables in adjInTrans and output variables in the different outPools is specified in CALCINPUTSSERVED. First, we build the subset of all possible variable assignments (A) that can be assigned in the current planning state (currAss). Second, we check whether all input transitions (adjInTrans) of all behavioral models can be served by the currAss. If that is not the case, the composition of the variant has failed. That is because the outPools can only shrink during the iterations of REACHVARIANT. Thus, input variables that cannot be served right away cannot be served at any time during composition.

> We assume that for each input variable of each behavioral model in a repository there must be at most one corresponding output variable.

```
CALCINPUTSSERVED(models ⊆ ID, A ⊆A  , adjInTrans, outPool) ≡
  return ( fail, currAss ) in
    step currAss :=  ( ( bm_out, o ), ( bm_in, i ) )  A : i  I,
        ( q_pre_in, I, q_post_in )  adjInTrans(bm_in), o  outPool(bm_out)
    step
      if  ( ( bm_out_1, o_1 ), ( bm_in_1, i_1 ) )  A :
          ( ( bm_out_2, o_2 ), ( bm_in_1, i_1 ) )  A :
```

$bm_{\text{out}_1} = bm_{\text{out}_2} \quad o_1 = o_2$ **then**
do **forall** $bm_{\text{in}} \quad models$
let $unservedVars = \quad i : (\, q_{\text{pre}},\, I,\, q_{\text{post}}\,) \quad \text{adjInTrans}(bm_{\text{in}}),$
$i \quad I, ((\, bm_{\text{out}},\, o\,), (\, bm_{\text{in}},\, i\,))$ / currAss **in**
if $unservedVars = \varnothing$ **then** $fail := true$
else $fail := true$

A.3 Copy rule creation

A copy rule contains information about all variable assignments that are possible in the current planning state (ps). The first component of a copy rule contains the states of all involved behavioral models that are a prerequisite for the copy rule to be applied in an execution. That is calculated as follows. The line numbers refer to CREATECOPYRULE below.

- Line 3 ensures that only states for involved behavioral models are collected.
- If a behavioral model acts as the *source* of a variable assignment, its state must be its current planning state. That is because the behavioral model must be in the state following the output transition in order to have that output available during execution (line 4).
- If a behavioral model acts as the *target* of a variable assignment, its state must be the state preceding the input transition served (line 5).

The second component of a copy rule consists of the currAss themselves.

CREATECOPYRULE($ps \quad \text{PlState}, currAss \subseteq \mathcal{A} \quad , \text{adjInTrans}) \equiv$
return $(\, states,\, currAss\,)$ **in**
let $states = \quad (\, bm,\, s\,) \;\; \mathcal{B} \quad : (\, bm,\, q_{\text{ps}},\, m\,) \quad ps,$
$([\, \text{adjInTrans}(bm) = \varnothing \quad s = q_{\text{ps}} \,]$
$[\, (\, q_{\text{pre}},\, I,\, q_{\text{post}}\,) \quad \text{adjInTrans}(bm) \quad s = q_{\text{pre}} \,])$ **in**
skip

A.4 Adjust output pools

The output pools are used to keep track of available output variables for the variable assignments and to determine whether each output variable has at least been assigned once to another behavioral model during composition. Therefore, we remove an output variable from the output pool only if no behavioral model (bm_{any}) relies on it in any of its input transitions ((q_{pre}, I, q_{post})) with respect to possible variable assignments (A) on the way back from the current planning state (s) to the initial state ($path$). Only if no output variables of an output rule appear in a behavioral model's output pool, the planning state can proceed over such an output transition rule as explained in the following section.

UPDATEOUTPOOLS(vnt Variant, ps PlState, $currAss$ $\subsetneq\mathcal{A}$,
A $\subsetneq\mathcal{A}$, outPool) $\equiv$ **return** outPool **in**
do **forall** (bm_{out}, o) : ((bm_{out}, o), (bm_{in}, i)) $currAss$
let $futureUse =$ i I : ((bm_{out}, o), (bm_{in}, i)) A, (q_{pre}, I, q_{post})
$path$, $path$ CALCPATHS(T, s), (bm_{any}, T) vnt,
(bm_{any}, s, m) ps, $q_{\text{post}} = s$ **in**
if $futureUse = \varnothing$ **then** outPool(bm_{out}) := outPool(bm_{out}) o

A.5 Subsequent planning state

At the end of an iteration of REACHVARIANT, the new planning state is calculated based on the current planning state (ps) as defined in CREATENEWPLANNINGSTATE. We differentiate the following cases which correspond to the alternatives for allocating variable s in CREATENEWPLANNINGSTATE below:

1. The planning state for a behavioral model in *output* mode whose all outputs of its adjacent output transition *are not* members of its own outPool proceeds one step toward the initial state. The rationale for that is that a behavioral model must already have reached the state after an output if the output shall be accessed by other behavioral models.

2. The planning state for a behavioral model in *output* mode where some of the outputs of its adjacent output transition are members of its own outPool remains at the current planning state.

3. The planning state for a behavioral model in *input* mode proceeds one step toward the initial state.

4. If the planning state for a behavioral model represents its *initial* state, it remains as it is.

$\textsc{CreateNewPlanningState}(vnt \quad \text{Variant}, ps \quad \text{PlState}, \text{adjInTrans}, \text{outPool}, startstate) \equiv$ **return** $newPs$ **in**

step do forall $bm \quad \text{ids}(ps)$

$\text{adjOutTrans}(bm) := \quad (q_{\text{pre}}, O, q_{\text{post}}) \quad Rules : (bm, Rules)$
$vnt, (bm, q_{\text{post}}, OUT) \quad ps, O \quad \text{OUT}^{bm}$

step $newPs :=$ $(bm, s, m) : (bm, q_{\text{ps}}, m_{\text{ps}}) \quad ps,$
$(bm, q_{\text{ss}}) \quad startstate,$

$$(s, m) = \begin{cases} (q_{\text{ps}}, IN) & : (q_{\text{pre}}, O, q_{\text{post}}) \quad \text{adjOutTrans}(bm), \\ & \quad O \setminus \text{outPool}(bm) = \varnothing, \\ & \quad \textbf{not} \ \text{stateEquiv}(bm, (q_{\text{ps}}, m_{\text{ps}}), q_{\text{ss}}) \\ (q_{\text{ps}}, OUT) & : (q_{\text{pre}}, O, q_{\text{post}}) \quad \text{adjOutTrans}(bm), \\ & \quad O \setminus \text{outPool}(bm) = \varnothing \\ (q_{\text{pre}}, OUT) & : (q_{\text{pre}}, I, q_{\text{post}}) \quad \text{adjInTrans}(bm), \\ & \quad \textbf{not} \ \text{stateEquiv}(bm, (q_{\text{ps}}, m_{\text{ps}}), q_{\text{ss}}) \\ (q_{\text{ps}}, m_{\text{ps}}) & : [\ \text{adjInTrans}(bm) = \varnothing, \\ & \quad \text{adjOutTrans}(bm) = \varnothing\ ] \\ & \quad\quad \text{stateEquiv}(bm, (q_{\text{ps}}, m_{\text{ps}}), q_{\text{ss}}) \end{cases}$$

where

$\text{stateEquiv}(bm, (q_{\text{ps}}, m_{\text{ps}}), q_{\text{ss}})$
$[\ m_{\text{ps}} \quad \text{undef}, OUT \quad q_{\text{ps}} = q_{\text{ss}}\ ]$
$[\ m_{\text{ps}} = IN \quad (q_{\text{ss}}, V, q_{\text{ps}}) \quad \delta^{bm} \quad V \subseteq \text{OUT}^{bm}\ ]$

Appendix B

Publications

Journal articles

1. Altenhofen, M., A. Friesen, J. Lemcke, and E. Börger. A high-level specification for Virtual Providers. *International Journal of Business Process Integration and Management (IJBPIM)*, 1(4):267–278, 2006b

2. Lemcke, J. and A. Friesen. Considering realistic Web service features for semi-automatic composition. *Annals of Mathematics, Computing and Teleinformatics (AMCT)*, 1(5):26–35, 2007b

3. Altenhofen, M., A. Friesen, and J. Lemcke. ASMs in service oriented architectures. *Journal of Universal Computer Science (JUCS)*, 14(12):2034–2058, 2008

Book chapters

1. Drumm, C., J. Lemcke, and D. Oberle. Business process management and semantic technologies. In J. Cardoso, M. Hepp, and M. D. Lytras, editors, *The Semantic Web: Real-World Applications from Industry*, volume 6 of *Semantic Web And Beyond Computing for Human Experience*, pages 209–239. Springer, 2007a. ISBN 978-0-387-48531-7. URL `http://dx.doi.org/10.1007/978-0-387-48531-7_10`

2. Lemcke, J. Light-weight semantic integration of generic behavioral component descriptions. In G. Mentzas, T. Bouras, P. Gouvas, and A. Friesen, editors, *Semantic Enterprise Application Integration for Business Processes*. IGI Global, Harrisburg, PA, 2009

Conference papers

1. Altenhofen, M., E. Börger, and J. Lemcke. An abstract model for process mediation. In K.-K. Lau and R. Banach, editors, *Formal Methods and Software Engineering, 7th International Conference on Formal Engineering Methods, ICFEM*, volume 3785 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2005a. ISBN 3-540-29797-9. URL http://dx.doi.org/10.1007/11576280_7

Workshop papers

1. Altenhofen, M., E. Börger, and J. Lemcke. An execution semantics for mediation patterns. In *Proc. of 2nd WSMO Implementation Workshop (WIW)*. CEUR Workshop Proceedings, Innsbruck, Austria, 2005b. ISSN 1613-0073. URL CEUR-WS.org/Vol-134/lemcke-wiw05.pdf

2. Altenhofen, M., E. Börger, and J. Lemcke. A high-level specification for mediators (Virtual Providers). In Bussler and Haller (2006), pages 116–129. URL http://dx.doi.org/10.1007/11678564_11

3. Lemcke, J. and C. Drumm. Semantic business automation. In A. Leger, A. Kulas, L. Nixon, and R. Meersman, editors, *How Business application challenges Semantic Web Research – European Semantic Web Conference (ESWC) Industry Forum*, volume 194. CEUR Workshop Proceedings, Budva, Montenegro, 2006. ISSN 1613-0073. URL CEUR-WS.org/Vol-194/paper7.pdf

4. Drumm, C., J. Lemcke, and K. Namiri. Integrating semantic Web services and business process management: A real use case. In *Semantics for Business Process Management, Workshop at 3rd European Semantic Web Conference (ESWC)*. 2006b. URL http://jobfunctions.bnet.com/abstract.aspx?promo=50002&docid=276296

5. Fronk, M. and J. Lemcke. Expressing semantic Web service behavior with description logics. In *Semantics for Business Process Management, Workshop at 3rd European Semantic Web Conference (ESWC)*. 2006. URL http://km.aifb.uni-karlsruhe.de/ws/sbpm2006/papers/sbpm06_Fronk.pdf

6. Lemcke, J. and A. Friesen. Composing Web-service-like abstract state machines (ASMs). In *IEEE International Conference on Services Computing - Workshops (SCW)*, pages 262–269. IEEE Computer Society, 2007a. URL http://doi.ieeecomputersociety.org/10.1109/SERVICES.2007.24

7. Lemcke, J. and A. Friesen. Considering realistic Web service features for semi-automatic composition. In *3rd South-East European Workshop on Formal Methods (SEEFM)*. Springer, 2007c

8. Kaiser, M. and J. Lemcke. Towards a framework for policy-oriented enterprise management. In K. Hinkelmann, editor, *Papers from the AAAI Spring Symposium*. 2008. ISBN 978-1-57735-357-7. Technical Report SS-08-01

9. Kubczak, C., T. Margaria, M. Kaiser, J. Lemcke, and B. Knuth. Abductive synthesis of the mediator scenario with jABC and GEM. In R. Garcia-Castro, A. Gómez-Pérez, C. J. Petrie, E. D. Valle, U. Küster, M. Zaremba, and M. O. Shafiq, editors, *6th International Workshop on Evaluation of Ontology-based tools and the Semantic Web Service Challenge (EON-SWSC)*, volume 359 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL `http://ceur-ws.org/Vol-359/Paper-5.pdf`

10. Lemcke, J., M. Kaiser, C. Kubczak, T. Margaria, and B. Knuth. Advances in solving the mediator scenario with jABC and jABC/GEM. In *7th Semantic Web Services Challenge Workshop, part of the 7th International Semantic Web Conference ISWC*. 2008

Symposium contributions, non-refereed

1. Lemcke, J. and C. Drumm. Semantic technologies for enterprise services. In *Semantic Web Days*. Online Proceedings, 2005. ISSN 1613-0073. URL `http://www.semantic-web-days.net/proceedings/SAP_SemanticWebDays2005.pdf`

2. Drumm, C., J. Lemcke, E. Dorner, and L. Heuser. Enterprise services architecture & semantic Web services. In S. Reich, G. Günter, T. Pellegrino, and A. Wahler, editors, *Semantic Content Engineering*, volume 17 of *Schriftenreihe Informatik*, pages 28–37. Trauner, 2006a. ISBN 3854879792

3. Friesen, A. and J. Lemcke. Composing Web-service-like abstract state machines (ASM). In J. Koehler, M. Pistore, A. P. Sheth, P. Traverso, and M. Wirsing, editors, *Autonomous and Adaptive Web Services*, volume 07061 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. URL `http://drops.dagstuhl.de/opus/volltexte/2007/1034`

4. Lemcke, J. and A. Friesen. Semi-automatic design of collaborative business processes in FUSION. In *Demonstrating Research Results on Enterprise Interoperability, Workshop at Int l Conf. Interoperability for Enterprise Software (i-ESA)*. 2008. URL `http://www.genesis-ist.eu/LinkClick.aspx?link=iESA_2008_ProceedingsBerlin.pdf&tabid=79&mid=413`