

Managing Dependency Relations in Inter-Organizational Models

Lianne Bodenstaff

Ph.D. dissertation committee

Chairman and secretary

Prof. dr. ir. A.J. Moushaan University of Twente, the Netherlands

Promotor

Prof. dr. R.J. Wieringa University of Twente, the Netherlands

Co-promotor

Prof. dr. M.U. Reichert University of Ulm, Germany

Members

Prof. dr. ir. M. Aksit University of Twente, the Netherlands

Prof. dr. P.M.G. Apers University of Twente, the Netherlands

Prof. dr. E. Damiani University of Milan, Italy

Prof. dr. J. Gordijn VU University Amsterdam, the Netherlands

Dr. H. Ludwig IBM TJ Watson Research Center, USA

Prof. dr. S. Rinderle-Ma Universität Wien, Austria



CTIT Ph.D. Thesis Series No. 10-167
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, The Netherlands



SIKS Dissertation Series No. 2010-15
The research reported in this thesis has been carried out
under the auspices of SIKS, the Dutch Research School
for Information and Knowledge Systems.



This research was financially supported by the Netherlands Organisation for Scientific Research (NWO) under contract number 612.063.409.

Printed and bound by Ipskamp Drukkers B.V.

Cover image from <http://dreamstime.com> (photographer: Nadya Pyastolova)

ISBN: 978-90-365-2996-9

ISSN: 1381-3617 (CTIT Ph.D. thesis Series No. 10-167)

<http://dx.doi.org/10.3990/1.9789036529969>

Copyright © 2010, Lianne Bodenstaff

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photography, recording, or any information storage and retrieval system, without prior written permission of the author.

**MANAGING DEPENDENCY RELATIONS
IN INTER-ORGANIZATIONAL MODELS**

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, 17 June 2010, at 16.45

by

Lianne Bodenstaff

born on June 4, 1977
in Assen, The Netherlands

This dissertation has been approved by:

Prof. dr. R.J. Wieringa (promotor)

Prof. dr. M.U. Reichert (co-promotor)

ABSTRACT

In various fields like software development, information systems development, and e-business development, model-based approaches allow specifying different models of which each emphasizes one specific aspect or part of the software system. In this thesis we consider particularly model-based approaches for defining inter-organizational cooperations. These cooperations are usually complex in terms of coordination, agreements, and value creation for involved partners.

At design time one should ensure that the different models are consistent with each other, i.e., that they describe the same system. At runtime we additionally have to deal with the fact that behavior of the software system might be different from that agreed upon. Such deviant behavior can, for example, be caused by partners in the cooperation that do not behave according to the agreement. Therefore, the challenges are to ensure consistency at design time as well as to monitor the system at runtime in order to detect inconsistencies with the models it relies on.

When managing complex cooperations, it is also vital to maintain the models describing them to keep an overview on the successfulness of the cooperation. Changing one model to regain consistency with the running system might result in new inconsistencies between the different models. As a consequence, this maintenance phase of the models is time consuming and grows in complexity with increasing number of models describing the system.

This thesis proposes a method that supports ensuring and maintaining consistency between running system and underlying models for inter-organizational cooperations. We provide a structured and model-independent approach to check and maintain consistency. Thereby, we focus on identifying and maintaining these inter-model relations.

We validate our method by conducting two case studies in two different fields of research. The first scenario deals with business and coordination models, while the second one addresses Web service compositions. Furthermore, we provide a prototypical implementation as proof-of-concept evaluation of both scenarios. We conclude with an empirical validation of the Web service composition scenario by an extensive and interactive survey conducted among 34 participants. This survey confirms the suitability of our proposed management solution provided for real life use.

ACKNOWLEDGEMENTS

CONTENTS

List of figures	xi
List of tables	xv
I Introduction & Motivation	1
1 Introduction	3
1.1 Motivation	3
1.2 Problem statement	4
1.3 Research design	7
1.4 Contribution	7
1.5 Outline	9
II Solution	11
2 Conceptual frame	13
2.1 Scope	13
2.2 Consistency	14
2.3 Categorization for models and consistency	16
2.3.1 Type of models	16
2.3.2 Type of consistency	16
2.3.3 Ensuring consistency	17
2.4 Orthogonal categorization for models and consistency	17
2.5 Summary	18
3 Problem investigation & related work	19
3.1 Model heterogeneity	19
3.1.1 Syntactic heterogeneity	20
3.1.2 Semantic heterogeneity	21

3.1.3	Pragmatic heterogeneity	21
3.2	Alignment with the running system	25
3.3	Maintaining models	26
3.4	Solution requirements	26
3.5	State-of-the-art research	28
3.5.1	Discussion on related work and solution requirements	34
3.6	Summary	36
4	MaDe4IC: An abstract method for managing model dependencies in inter-organizational cooperations	37
4.1	General approach	37
4.1.1	Model characteristics	37
4.1.2	The method	38
4.2	Running example	40
4.3	Phase I: Model analysis	41
4.3.1	Step 1: Model analysis	41
4.3.2	Step 2: Homogenization	43
4.4	Phase II: Inter-model analysis	46
4.4.1	Step 3: Inter-model relation detection	46
4.4.2	Step 4: Inter-model consistency constraints	50
4.5	Phase III: Intra-model analysis	55
4.5.1	Step 5: Intra-model relation detection	55
4.5.2	Step 6: Intra-model consistency constraints	55
4.6	Phase IV: Combined analysis	58
4.6.1	Step 7: Dependency analysis	58
4.7	Phase V: Management phase	61
4.7.1	Step 8: Log analysis	61
4.7.2	Step 9: Causal analysis	63
4.8	Summary	66
III	Scenario 1: Business & Coordination Models	69
5	Managing dependency relations: Business and coordination models	71
5.1	Basics	72
5.1.1	Business perspective	72
5.1.2	Process perspective	72
5.1.3	Event logs	73
5.2	Step 1: Model analysis	75
5.3	Step 2: Homogenization	75
5.4	Step 3: Inter-model relation detection	77
5.5	Step 4: Inter-model consistency constraints	79
5.5.1	Transfer level	79
5.5.2	Business transaction	80

5.6	Step 5: Intra-model relation detection	81
5.6.1	Value model	82
5.6.2	Coordination model	82
5.7	Step 6: Intra-model consistency constraints	82
5.7.1	Value model	83
5.7.2	Coordination model	85
5.8	Step 7: Dependency analysis	87
5.8.1	Value model abstraction	87
5.8.2	Coordination model abstraction	88
5.8.3	Formalization of constraints	89
5.9	Step 8: Log analysis	90
5.10	Step 9: Causal analysis	92
5.10.1	Causes	92
5.10.2	Changes	93
5.10.3	Causal analysis	96
5.10.4	Minimize the number of changes: Theorem and proof	97
5.11	Related work on value and coordination models	99
5.12	Summary	101
6	Proof-of-concept implementation: Business and coordination model	103
6.1	The business case	103
6.1.1	Value model	104
6.1.2	Coordination model	105
6.2	Consistency constraints	106
6.3	Implementation	107
6.3.1	Average value of transfers	108
6.3.2	Average number of transfers	108
6.3.3	The equation system	108
6.4	Managing the value model	111
6.5	Visualization	112
6.6	Evaluation of our developed management approach	114
6.7	Summary	114
IV	Scenario 2: Service Level Agreements for Composite Services	117
7	Managing dependency relations: Service compositions	119
7.1	Basics	120
7.1.1	Service Level Agreements	120
7.1.2	Business case	121
7.2	Step 1: Model analysis	123
7.3	Step 2: Homogenization	124
7.4	Step 3: Inter-model relation detection	125
7.5	Step 4: Inter-model consistency constraints	129

7.5.1	Generalization: Impact factors	130
7.6	Step 5: Intra-model relation detection	133
7.7	Step 6: Intra-model consistency constraints	134
7.8	Step 7: Dependency analysis	136
7.8.1	Composition tree	138
7.8.2	Expected impact tree	139
7.9	Step 8: Log analysis	146
7.10	Step 9: Causal analysis	148
7.10.1	Realized impact tree	148
7.10.2	Feedback tree	150
7.11	Related work on SLAs	154
7.11.1	SLA models	154
7.11.2	Managing Web services	155
7.11.3	Root Cause analysis	156
7.11.4	Service monitoring approaches	156
7.11.5	QoS-based service composition	157
7.12	Summary	157
8	Proof-of-concept implementation: Service compositions	159
8.1	Example scenario	159
8.2	Implementation	160
8.2.1	Contribution factors	162
8.2.2	Service contribution	163
8.2.3	Impact factors	164
8.3	Generation and execution	164
8.4	Visualization	165
8.5	Summary	169
9	Evaluation of MoDe4SLA: Service compositions	171
9.1	Evaluation plan for effectiveness	171
9.2	Evaluating usefulness: Setup	174
9.3	Course of evaluating usefulness	176
9.4	Conclusions from evaluating usefulness	177
9.4.1	Demographics	177
9.4.2	Statistics	178
9.4.3	Conclusions	185
9.5	Summary	186
V	Discussion	187
10	Discussion & lessons learnt	189
10.1	Method requirements	189
10.2	Cross-scenario discussion	192

10.3	Applying our MaDe4IC method	193
10.3.1	Step 1: Model analysis	193
10.3.2	Step 2: Homogenization	194
10.3.3	Step 3: Inter-model relation detection	194
10.3.4	Step 4: Inter-model consistency constraints	195
10.3.5	Step 5: Intra-model relation detection	195
10.3.6	Step 6: Intra-model consistency constraints	196
10.3.7	Step 7: Dependency analysis	196
10.3.8	Step 8: Log analysis	196
10.3.9	Step 9: Causal analysis	197
10.3.10	Lessons learnt	197
10.4	Validating MoDe4SLA	198
10.5	Answering research questions	199
10.6	Future research	201
10.6.1	Our MaDe4IC method for managing dependencies	201
10.6.2	Our approach for managing SLAs of composite services: MoDe4SLA202	
Appendix A	Evaluation	205
A.1	Transcript	205
A.2	Hand-out	207
A.3	Survey results	240
Appendix B	Related publications by the author	259
Bibliography		261

LIST OF FIGURES

1.1	Research design	8
2.1	Intra-model, inter-model, and runtime consistency relations	15
2.2	Multi-model approaches for maintaining consistency	18
3.1	Consistency relations between models, event logs, and information systems	20
3.2	Perspective and focus	22
3.3	Coarsening models	23
4.1	MaDe4IC: Method for managing dependency relations in inter-organizational models	39
4.2	Running example: Selling bikes	41
4.3	Dependency relations between viewpoints and partial models	42
4.4	Example: Asymmetric and symmetric dependency relations	48
4.5	Possible generalization over symmetric consistency constraints	53
4.6	Possible generalization over asymmetric consistency constraints	54
4.7	Possible generalization over intra-model consistency constraints	57
4.8	Consequence analysis for changing models	65
5.1	Example case: Business model (e ³ -value notation)	73
5.2	Example case: Coordination model (BPMN notation)	74
5.3	Example case: Event log (XML notation)	74
5.4	Relations between models and real-life entities	78
5.5	Inter-model dependency relations	78
5.6	Constraints between models and event log	81
5.7	Example case: Intra-model dependencies	82
5.8	Example case: Intra-model consistency constraints	85
5.9	Same class models	94
5.10	Observable changes	94
5.11	Example case: Causal analysis of coordination model	97

LIST OF FIGURES

5.12	Relating constraints and changes	98
6.1	Example case: Business model (e^3 -value notation)	105
6.2	Example case: Coordination model (Petri Net notation)	106
6.3	Example case: Realized and estimated average value of transfers	108
6.4	Example case: Realized and estimated average number of transfers	109
6.5	Example case: Customer ratios and introduced variables (e^3 -value notation)	111
6.6	Proof-of-concept implementation: Graphical user interface	112
6.7	Proof-of-concept implementation: Coloring the value model	114
7.1	Overview: MoDe4SLA approach	120
7.2	SubscribedNews and NewsRequest: Dependency cost model	127
7.3	SubscribedNews: Response time dependency model	128
7.4	NewsRequest: Response time dependency model	128
7.5	SubscribedNews and NewsRequest: Impact trees	132
7.6	Illustrative service composition	137
7.7	Small example: Service composition	141
7.8	Main figure	145
7.9	Illustrative example: Cost feedback tree	154
8.1	Example scenario: Service composition	160
8.2	Example scenario: Agreed upon QoS	161
8.3	Example scenario: Estimated impact tree for cost	162
8.4	Example scenario: Realized QoS	166
8.5	Monitoring example scenario: Cost feedback tree	167
8.6	Monitoring example scenario: Response time feedback tree	168
9.1	Evaluating effectiveness	172
9.2	Evaluating usefulness	175
9.3	The offered composition appears to be complex.	178
9.4	Concerning costs: It is easier to determine the impact each service has on the composition with the analysis than without it.	179
9.5	MoDe4SLA approach is helpful when managing this composition with regard to accurately depicting malfunctioning services.	179
9.6	Concerning costs: It takes less time to see relations between different services and the composition.	180
9.7	MoDe4SLA approach is helpful when managing this composition with regard to faster selecting services to renegotiate.	181
9.8	After seeing the MoDe4SLA analysis, how is your confidence about the service selection for renegotiation you made before?	181
9.9	Assume only a subset of services can be renegotiated regarding their SLAs. I would feel confident in selecting services for renegotiation.	182

9.10 Assume only a subset of services can be renegotiated regarding their SLAs. I would feel more confident in selecting services for renegotia- tion with MoDe4SLA than without.	182
9.11 MoDe4SLA is helpful for managing service compositions.	183
9.12 The presentation before the evaluation is sufficient to properly understand MoDe4SLA approach.	184

LIST OF TABLES

1.1	Research questions related to the chapters	9
3.1	Approaches for checking consistency	29
3.2	Solution requirement compatibility of related work	34
5.1	Estimations value model	73
5.2	Causes of intra-model constraint violation	93
5.3	Categorization of model changes	95
6.1	Implementation: Equation system	110
6.2	Implementation: Realized and estimated customer values	111
7.1	SLAs for the SubcribedNews service	122
7.2	SLAs NewsRequest	123
7.3	Cost dependency model constructs	126
7.4	Response time dependency constructs	127
7.5	Impact tree vertices	131
7.6	Supporting services of SubscribedNews and NewsRequest: Impact factors	133
7.7	Matching composition vertices and vertices for dependency trees	137
9.1	Effectiveness indicators	173
A.1	Time Transcript	206

Part I

INTRODUCTION & MOTIVATION

INTRODUCTION

1.1 Motivation

Model-based implementation approaches are used in various fields like software development, information systems development, and e-business development [53, 71, 112]. Many of these approaches allow specifying *several* models, each emphasizing one specific aspect or part of the software system. In this thesis we consider such model-based approaches for modelling *inter-organizational cooperations*. For example, we consider Web service compositions and e-business cooperations. These cooperations are often complex in terms of coordination, agreements, and value creation for the involved partners.

Due to the complex nature of inter-organizational cooperations, usually, a variety of models is used to specify the information system to be developed. For example, financial benefits are captured in a business model [92], while coordination details are captured in a process model [40]. Using several models to represent one complex information system has many advantages. Especially, understandability of the models is enhanced since each model only represents some of the information about the system to be developed. As a result, complexity is reduced for managers reading the models as well as for engineers developing and maintaining them; especially in cooperations where different partners with different business goals need to come to an agreement, such a multi-model approach is beneficial. As typical example of inter-organizational cooperation, consider product and change management where different partners need to agree on a particular product change. For example, an automotive vendor and its suppliers need to agree on changes in the design of a car [89].

Although using several models to represent one complex system enhances usability when developing a specific model, new challenges arise. Modelling complexity is reduced by developing several models at design time, but these models together do form the basis for the running cooperation and in the end the running information system; i.e., the implementation must represent the combination of these different models. Therefore, it is of importance that the different models describe the same system, i.e., that they are *consistent* with each other, which constitutes our first major challenge. For example, if a high

level business model states that partner A receives money for delivering several products or services to partner B, the coordination model describing the system should contain this exchange, and the information systems model should enable it. The challenge is to *ensure consistency* between the different models describing one system before implementation. We refer to this as the problem of ensuring *design time consistency*.

If the different models describing a particular cooperation are consistent with each other, there is a proper basis for implementing the information system. However, at runtime the behavior of the system might be different from that agreed upon. Such deviant behavior can be caused by implementation errors. Another major cause for these deviations are partners in the cooperation that do not behave according to the agreement. For example, business partners might not do their payments in time, or agreed upon response times are violated. Furthermore, deviant behavior is caused by events that cannot be controlled by the business partners, but this behavior is merely estimated when the models are developed. For example, customer behavior described in models is typically estimated. However, in real life there can be deviations from these expectancies. This may be bad (e.g., business partners not satisfying SLAs) or good (e.g., customers buying more goods) but in any case this is something to be observed. In all these cases the running system behaves differently from the agreed upon models, i.e., the running system and the models describing it are inconsistent. We refer to this as *runtime consistency*. Runtime inconsistency is not always problematic, but in any case typically some action is taken when inconsistencies occur. The challenge is to *monitor* the system such that inconsistencies with the models it relies on can be detected. This thesis introduces techniques to make these things observable.

Furthermore, when managing complex cooperations, it is vital to maintain the models describing them in order to keep an overview on the behavior and successfulness of the cooperation. If one model is changed, consistency with other models might be broken. Therefore, it is a big challenge to *regain consistency* between the running system and its models. Especially this maintenance phase is challenging since the different models are tightly connected and describe different perspectives of the same system. Changing one model to regain consistency with the running system might result in new inconsistencies between the different models. As a consequence, this maintenance phase of the models is time consuming and grows in complexity with increasing number of models describing the system.

1.2 Problem statement

The problem of checking consistency between related models, of checking consistency between a running system and its underlying models, and of managing the running system by maintaining consistency between models and running system is not new. Concerning design time consistency, there exist multi-model approaches, for example, Unified Modelling Language (UML) [79] and Open Distributed Processing (ODP) [29]. Both aim at consistent model development. However, these approaches to consistency are *model-*

specific and not applicable to other modelling languages. Especially when using different modelling languages that are not directly related, developers do not have such support. Furthermore, there exist some approaches that support multi-model development, but they stay on a very high level explaining rather *what* should be done than *how* this should be accomplished [88, 103].

Concerning runtime consistency, there exist monitoring approaches that support consistency checking of the running system and the models describing it [99, 101]. However, these approaches mainly focus on monitoring the running system against *one* model, neglecting dependencies between this model and others. Furthermore, the most challenging and dynamic part of the problem, i.e., maintaining consistency between models and the running system, is even less supported in such consistency approaches.

The main problem in ensuring and maintaining consistency between a set of models is the following: these models are *interrelated* and, therefore, changing one model might affect several other models. Therefore, the main challenge is to first identify the exact nature of the relation between the models, and, secondly, to identify the effects changes in one model have on the other models.

In this context, this thesis proposes a method that supports ensuring and maintaining consistency between models and running system for inter-organizational cooperations. Our goal is to provide a *structured* and *model-independent approach* on how to check and maintain consistency. Thereby, we focus on identifying and maintaining these inter-model relations.

There are several research questions to be answered in this thesis, and driving the development of our approach. Basically, we consider four main research questions. The first one aims at defining criteria against which we evaluate our solution. These criteria are determined based on a problem analysis through literature research to figure out characteristics of models and running system for inter-organizational cooperations. Furthermore, we keep the intended users of our method (i.e., researchers challenged with checking and ensuring consistency in models for inter-organizational cooperations) in mind when determining the criteria. Based on these characteristics we formulate criteria for our solution method. In addition, we determine whether current research approaches suit our requirements.

Research Question 1: What are solution criteria that a method for checking and ensuring consistency in inter-organizational cooperations should satisfy?

Q1a: What are characteristics of models and running system in the context of inter-organizational cooperations?

Q1b: What are criteria for a method that checks and ensures consistency in such models?

The second research question aims at determining current state of the art on the above sketched topic. In addition, we check whether this research suits the criteria defined by answering Research Question 1.

Research Question 2: What is state of the art on maintaining consistency relations in inter-organizational models?

Q2a: How is consistency checked at design time in existing solutions?

Q2b: How is consistency ensured during runtime in existing solutions?

Q2c: How suitable are current solutions with regard to the criteria defined in question Q1b?

The third research question addresses the design of the method for managing consistency relations. Based on the identified criteria and research gap, we build the method. Firstly, we need to define guidelines on how to ensure inter-model consistency, i.e., we need to define when we consider two models to be consistent with each other. Secondly, we need to define guidelines on how to ensure intra-model consistency, i.e., when do we consider a model to be consistent with regard to the cooperation. Thirdly, we need to define guidelines on how to ensure consistency during runtime and, finally, we need to define guidelines on how to maintain consistency for a running system in an efficient manner.

Research Question 3: How can a solution method for checking and ensuring inter-model consistency be built?

Q3a: How can inter-model consistency be ensured?

- How can inter-model dependencies be detected?
- How can consistency constraints be defined using these inter-model dependencies?

Q3b: How can intra-model consistency be ensured?

- What intra-model dependencies exist?
- How can consistency constraints be defined using these dependencies?

Q3c: How can consistency between a running system and its underlying models be checked?

Q3d: How can consistency between a running system and its underlying models be efficiently maintained?

Our fourth and last research question addresses the validation of our method. Since this method aims at being suitable for a variety of models, the scenarios should be sufficiently different. We chose as a first scenario *business and coordination models* and as a second one *Service Level Agreements* of composite services. Furthermore, the results of applying the method to both scenarios are evaluated through implementation.

Research Question 4: How can the solution method be validated?

Q4a: How well applicable is the solution method in different scenarios according to the criteria identified by answering Research Question 1?

Q4b: How good are the developed solutions when applying the method?

- Validate the solution of both scenarios with a proof-of-concept implementation.
- Validate the implementation of one of the scenarios through a usability survey.

1.3 Research design

The research design for this thesis is depicted in Figure 1.1. We start with a *literature study* to facilitate a thorough problem investigation. Based on this literature study, we formulate solution criteria our method should comply with. Furthermore, we evaluate current *state of the art* to confirm the necessity of our method for checking and ensuring consistency.

Secondly, we *design* the method based on the problem investigation. It is developed through *literature study* and *design research*. The method consists of a stepwise approach assisting the developer in setting up a proper management environment for the different models.

Thirdly, we validate the method by conducting two *case studies* in two different fields of research. One scenario comprises business and coordination models, while the other one addresses Web service compositions. Applicability to different scenarios in different research fields supports the claim that our method is applicable to a variety of conceptual models.

Fourthly, we provide a *prototypical implementation* as evaluation of both scenarios.

We conclude with an *empirical validation* of the Web service composition scenario by an *interactive survey* among 34 participants.

1.4 Contribution

The primary contribution of this thesis is the development of an abstract method for managing model dependencies in inter-organizational cooperations. This method is described in Chapter 4. In addition to this primary contribution, we provide several secondary contributions:

- We provide an extensive categorization of characteristics of different models used for modelling inter-organizational cooperations. This contribution is provided in Chapters 2 and 3, and it answers Research Questions 1 and 2.

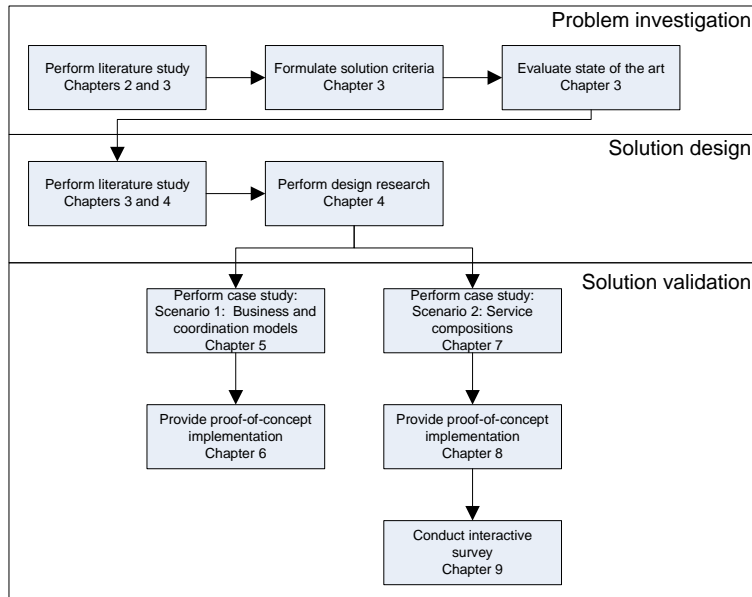


Figure 1.1: Research design

- We introduce a comprehensive method that allows ensuring consistency for models at design time, checking consistency between models and running system during runtime, and maintaining consistency between models and running system during runtime of the system. This contribution is provided in Chapter 4 and it answers Research Question 3.
- We present evaluation of our method application to two different scenarios:
- Scenario 1: Business and coordination models:
 - We introduce formal inter-model and intra-model dependency relations, and dependency relations between running system and underlying models. This contribution is provided in Chapter 5, and it answers Research Question 4a.
 - We provide a proof-of-concept implementation of these dependency relations that allow users to manage their business models. This contribution is provided in Chapter 6, and it answers Research Question 4b.
- Scenario 2: Service compositions:
 - We introduce formal inter-model and intra-model dependency relations, and dependency relations between running system and underlying models. Chapter 7, Research Question 4a.

	Part II			Part III		Part IV		
	Ch. 2	Ch. 3	Ch. 4	Ch. 5	Ch. 6	Ch. 7	Ch. 8	Ch. 9
Q1	x	x						
Q2		x						
Q3			x					
Q4				x	x	x	x	x

Table 1.1: Research questions related to the chapters

- We provide a proof-of-concept implementation of these dependency relations that allow users to manage their composite services. This contribution is provided in Chapter 8, and it answers Research Question 4b.
- We demonstrate usefulness of the implementation for service managers through an interactive survey. This interactive survey confirms importance of our research. This contribution is provided in Chapter 9, and it answers Research Question 4b.

1.5 Outline

Table 1.1 shows in which chapters which research questions are addressed. *Part I* shows the motivation for this thesis. More specifically, Chapter 1 gives a motivation and problem statement.

Part II of this thesis introduces our method for managing models of inter-organizational cooperations. Chapter 2 provides the conceptual frame of our research with basic definitions used throughout this thesis. In Chapter 3 we conduct a problem investigation by considering related work. This investigation leads to identification of solution criteria. The state of the art is reviewed in the light of these criteria. Chapter 4 introduces our comprehensive method for managing models of inter-organizational cooperations.

In *Part III* we show applicability of our method to the first scenario where different models of inter-organizational cooperations (i.e., business and coordination models) need to be managed. Chapter 5 discusses the application of the method, while Chapter 6 provides a proof-of-concept implementation of these results to support application of the method.

Part IV discusses the second scenario, which is inherently different from the first one. Here, we discuss applicability of our method to service compositions. Again, Chapter 7 describes the application of the method. Then, we discuss our proof-of-concept implementation in Chapter 8. In addition to this validation, in Chapter 9 we discuss an extensive interactive survey conducted among experts. This survey supports the suitability of the proposed management solution provided by the method for real-life use.

Part V concludes this thesis with a discussion on the results and some suggestions for future research in Chapter 10.

Part II

SOLUTION

CONCEPTUAL FRAME

In this chapter we discuss the conceptual frame for our research. The goal is to clarify the setting and scope of this thesis. In Section 2.1 we explicate our research scope. Furthermore, we discuss the term ‘consistency’ as used in this thesis in Section 2.2. We conclude with a *categorization* of types of models and types of consistency in Section 2.3, and an orthogonal categorization in Section 2.4.

2.1 Scope

The aim of this thesis is to provide a method to manage models of inter-organizational cooperations. Our goal is to check consistency relations between different models, and to ensure consistency of models during runtime. Our method is applicable to models describing *inter-organizational interactions* that are enabled by information technology. It is important to clarify the scope of our method, and to be clear on the terminology used in this thesis.

Inter-organizational relations can be categorized based on the *type* of relationship partners have. Traditionally, a distinction is made between *markets* and *nonmarkets* [60]. Markets are characterized by discrete interactions [73] with limited personal involvement, no future commitments, and noncooperative and self-interested interactions between the partners [97]. Nonmarket interactions are characterized by the existence of some sort of *relationship* between the partners. Often there exist longer running contracts where the same service or product is repeatedly exchanged [60]. Such nonmarkets are either *hierarchically* organized with *unilateral interactions* as, for example, in franchise contracts, or they are *collaborations* which are based on *bilateral interactions* [73, 97].

In collaborations, partners tighten their relationships to gain market strength, to achieve higher profits, or to exploit new opportunities [98]. A collaboration is often defined as inter-organizational cooperation which is negotiated in an ongoing communicative process [94]. Typically, it is emphasized that a collaboration does neither rely on market nor on hierarchical mechanisms of control [70].

With the emergence of electronic commerce (e-commerce), the ways of doing business have changed dramatically [35, 116]. Inter-organizational systems that do not fit

classical categorization of either market, hierarchical or collaborative interaction arise. Furthermore, different authors use different definitions, and many authors do not explain which definition they use.

Especially the terms *cooperation* and *collaboration* are often used interchangeable in e-commerce research. However, in cognitive psychology a clear difference between these terms is made [75]. In a cooperation the problem is split into several tasks, and each task is executed by one participant. In a collaboration a mutual engagement of participants exists to solve the problem together [39]. Furthermore, in a cooperation the participants work together to achieve a clearly defined goal, while in a collaboration often open-ended tasks without clear goals are present. The main focus is on sharing information in order to evolve ideas and opinions rather than to achieve a certain goal [16]. From this perspective, current e-commerce systems resemble more a cooperative than a collaborative form since there is some mutually compatible interest, each participant has its own contribution to the overall product, and the relations between the participants are often not personal. The main difference between cooperations and collaborations on the one hand and markets on the other hand, is the element of *competition* [36]. Competitiveness, and as a result opportunistic behavior, make marketplaces a non-cooperative form of inter-organizational interaction. The control factor in markets is price. In collaborations or cooperations the important factor is some relation you have with other parties [97].

In this thesis we refer to inter-organizational *cooperations* where a cooperation is some voluntary interaction between two or more partners. Such a cooperation can be short term and market based, but also long term and of collaborative nature. Models describing these cooperations are typically *conceptual models*, but can be described in a variety of languages like UML Activity Diagrams [90], Petri Nets [64], and BPMN [113]. An inter-organizational model is suitable to be assessed by our method when it is a conceptual model that focuses on the *exchanges*, i.e., interactions, between different partners. We do not support management of any *internal* behavior of the partners involved. Furthermore, we assume communication and exchange of information between partners is (partly) dependent on *information technology*. Typically, such information technology enabled business models are referred to as e-commerce business models [108].

2.2 Consistency

In this thesis we support consistency *checking* at design time as well as *maintaining* consistency during runtime of the system. Consistency is checked and maintained *between* different models and *within* models.

Consistency can be defined in many ways, however, in this thesis we use the classical definitions for consistency. Classical Aristotelian logic provides us with a semantic notion of consistency [102]:

Two or more statements are considered to be *consistent* if they are simultaneously true under some interpretation.

In modern logic the syntactic notion for consistency is defined as follows [68]:

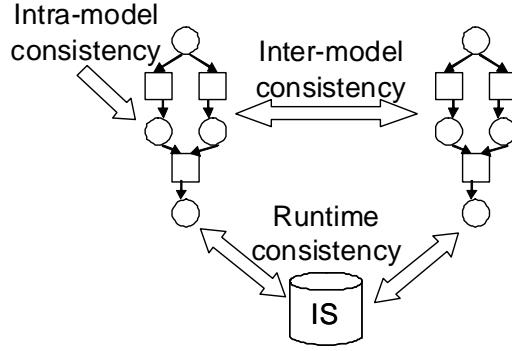


Figure 2.1: Intra-model, inter-model, and runtime consistency relations

A set of statements is considered to be consistent to a certain logical calculus if no formula $P \wedge \neg P$ can be derived from those statements by the rules of the calculus, i.e., the statements are free from contradictions.

Therefore, when we use the term ‘consistency’ we mean the *absence of contradictions*. At design time of the models, we distinguish between consistency within the models (i.e., intra-model consistency), and consistency between two or more models (i.e., inter-model consistency). Consistency is always determined under some interpretation. A schematic view on the different consistency checks in this thesis is given in Figure 2.1.

For *intra-model consistency* we assume the interpretation under which consistency is determined is given by the definition of the model-specification language. The produced model needs to be syntactically well-formed and meaningful, i.e., consistent with the specification. Aside from the official specification, additional constraints on the models can be formulated in a specific context. For example, one might want to reduce expressiveness to avoid complex models. Furthermore, one can extend a language for a specific goal if it is not sufficiently rich. In this thesis we assume that the models provided are intra-model consistent (i.e., *well-formed*) with respect to their specification. Our interest lays in the relations *between* the different models.

Refinement of problem statement. We refine the problem statement as defined in Section 1.2 for design time and runtime. The challenge in checking *inter-model consistency* at *design time* is defining the proper interpretation under which these models are considered being consistent. Especially for models defined on different levels of abstraction, or defined in different modelling languages, this is not a straightforward exercise.

During *runtime* we first *check* consistency between the running system and each model against some interpretation. Again, defining this interpretation is a challenge. As a second step, we *maintain* consistency by adapting models or implementations when contradictions are detected. As a consequence of a change in one model, the relations it has with the other models need to be checked for consistency again.

2.3 Categorization for models and consistency

Both in Software and Systems Engineering, developing several models that describe together one system is a commonly used approach to reduce complexity of the models. To discuss research done in these areas, we define a categorization of the different approaches. We distinguish between the *type of models* which is considered, the *type of consistency* which is ensured, and how consistency is *checked* through the different approaches.

2.3.1 Type of models

We distinguish approaches which handle consistency between different *viewpoints* on a system and approaches which handle consistency between different *partial models* of a system. A *viewpoint* on a system describes the entire system under development and focuses on a specific characteristic (e.g., messages exchanged between partners). Reduction of complexity in modelling the system is accomplished by leaving out those aspects of the system that do not belong to the viewpoint characteristic. For example, one viewpoint might be the cost perspective of the cooperation, while another one describes the order in which messages are exchanged. As opposed to viewpoint models, *partial models* describe different parts of the system in separate models. To reduce complexity, the system under development is divided in parts that are separately modelled. For example, a company develops separate models for each partner it is interacting with.

The distinction between these two approaches is important since it influences the consistency relation between the models. Different viewpoints have a complete overlap in the modelling *domain* while their *focus* is disjoint. The challenge is to find the exact relation between the different foci. Partial models might have an overlap in the domain, but this is never a complete overlap. The focus of the models, however, might be equal. For example, two Entity-Relationship diagrams where each describes part of the system, have the same focus. In this case the challenge is to find the relation between the different partial models, rather than to find the relation between the foci.

Since conceptual models used for modelling inter-organizational cooperations can be both viewpoints and partial models, we look for an approach that checks consistency for both types.

2.3.2 Type of consistency

We distinguish different types of *consistency*. *Intra-model consistency* considers the well-formedness of a model. The interpretation used for determining consistency is according to the requirements set for the specification language. *Inter-model consistency* checks consistency *between* two or more models. The interpretation used for determining consistency depends on the type of model used and on restrictions set by the engineer. However, in this thesis two models are considered to be consistent with each other if a specification can be found which represents both models. *Homogeneous* (also referred to as *intra-language*) approaches consider models of the same type, while *heterogeneous* (i.e.,

inter-language) approaches are able to handle consistency checks between two models expressed in different languages.

For modelling inter-organizational cooperations typically heterogeneous models are used. Therefore, we look for an approach that handles such heterogeneity.

2.3.3 Ensuring consistency

Further, we distinguish between different ways of *ensuring* consistency. Two main options are to *check* consistency after models are developed or to *ensure* consistency by construction during the development process. Checking consistency can be done by *testing* the models with some model checker, or by finding a *translation*. Usually models are translated into a semantically well-defined formalism which allows for formal consistency checking. When translating models, either they are completely translated or only the overlapping parts between them are translated. A complete translation is time consuming, while in a partial translation first the overlap between models is determined. Especially when dealing with heterogeneous models this is not straightforward. When consistency is ensured during *construction* of the model, either additional development requirements for the models are set, or consistency is defined by relating their meta-models.

Aside from consistency checking, we aim at maintaining consistency during runtime of the system through some adaptations. Maintaining consistency cannot be done through construction since consistency through construction is done at design time. Maintaining consistency is done at runtime. Therefore, our approach should allow consistency ensuring through checking rather than through construction.

2.4 Orthogonal categorization for models and consistency

An orthogonal distinction between types of consistency is on the *type of relation* these models have. These relations are depicted in Figure 2.2.

In viewpoint approaches the relation is between *foci* of the models (cf. Figure 2.2a). For example, how does the order of messages in a coordination model relate to costs of collaborating with a partner. Usually, this is a heterogeneous consistency relation defined on the entire model.

The relation between parts of partial model approaches is different in nature. Here, different types are distinguished. Firstly, there are models that describe a different part of the system, but are on the *same level of abstraction*. These might be homogenous models, for example two Entity-Relationship diagrams, or heterogeneous models, for example an Entity-Relationship Diagram and an Activity Diagram. This type of consistency relation is often referred to as *horizontal consistency* [78] (cf. Figure 2.2b).

Another relation is between models on *different levels of abstraction* covering the same part of the system. One model is a *refinement* of the other one. These models are said to have a *vertical consistency relation* [45] (cf. Figure 2.2b). In partial model approaches often high-level models are defined which are iteratively refined until the level of abstraction is such that the model can be translated into executable code. With this type

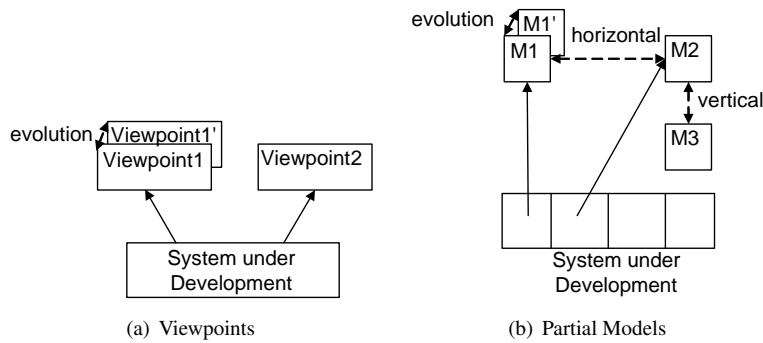


Figure 2.2: Multi-model approaches for maintaining consistency

of development, refinement consistency is constructed rather than checked. It should be noted that during the refinement process often information is added to models that is not present in the higher levels. Therefore, these relations are usually not pure refinements.

A third dimension is *evolution consistency* [78] which is present in both viewpoint and partial model approaches (cf. Figure 2.2). When a model evolves during development process, the relation between old and new model is referred to as evolution consistency. This relation is always between homogeneous models.

2.5 Summary

In this chapter we discuss techniques for checking and maintaining consistency across models describing a single system. For our purpose, we seek an approach that is *language-independent*, is able to handle *heterogeneous models*, is suitable for both *viewpoints* and *partial models*, and ensures consistency through *checking* rather than through construction.

PROBLEM INVESTIGATION & RELATED WORK

Inter-organizational cooperations are getting ever more complex due to the increasing use of IT. In particular this holds for models describing these systems. To reduce complexity, inter-organizational cooperations are often described by more than one model where each model focuses on a specific aspect of the cooperation. The models together form the basis for implementing the cooperation. As a consequence, consistency between models needs to be ensured when designing the cooperation and be maintained while running it.

In this chapter we assess the problem of ensuring and maintaining consistency between models and running system by considering difficulties in identifying relations between models and running system. When comparing different models their *heterogeneity* gives rise to several challenges that are discussed in Section 3.1 (cf. Figure 3.1). Details on checking *alignment* of the running system with the models describing it are discussed in Section 3.2 (cf. Figure 3.1). Besides compatibility issues between models and between models and running system, we analyze specific issues that occur when *maintaining* such systems in Section 3.3. Based on this analysis, we discuss *solution requirements* for a method designed to ensure and maintain consistency between models and running system at hand in Section 3.4. These requirements solve a subset of the identified problems in this chapter. Finally, we discuss state of the art concerning existing methods for managing consistency between models in Section 3.5. Several methods exist but, as we show, none of them complies with all solution requirements identified in Section 3.4.

3.1 Model heterogeneity

Different conceptual models jointly describe an inter-organizational cooperation. Each model has a different purpose and, therefore, the models are often denoted in different modelling languages. Checking consistency between such *heterogenous models* while designing the system is a difficult process. Challenges arise because different models have different focus or view, but often have an *overlap* in parts of the system they describe. Consistency problems occur at overlapping parts if models contradict in their description. For example, most models describe the actors involved in the cooperation. It is therefore important to ensure that all models describe the different actors in a consistent manner.

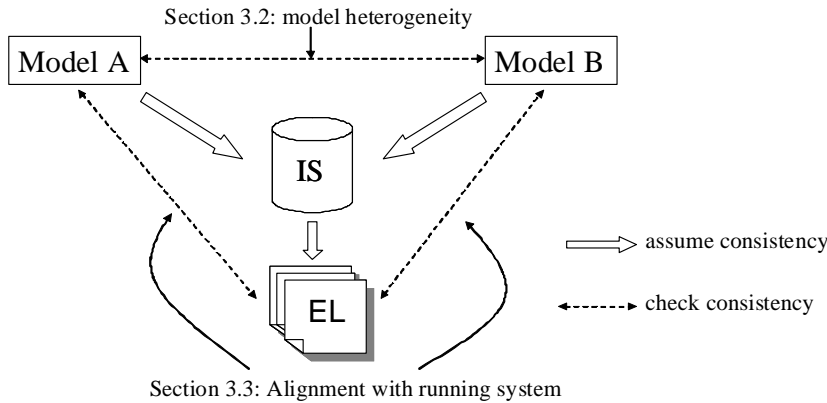


Figure 3.1: Consistency relations between models, event logs, and information systems

Furthermore, models may *depend* on each other. For example, a business model might describe what information and products are exchanged in a cooperation, while an implementation model describes how these exchanges are realized. Consequently, whether the goals in the business model are achieved depends on a proper implementation of the implementation model. As a result, the challenge at hand is to compare overlapping parts in models possibly described in different modelling languages and with different purposes. Furthermore, *dependencies* between models should be identified, analyzed, and made explicit. Next, we identify heterogeneity problems that arise when comparing heterogeneous models. These problems are in the way of an easy identification of overlap and dependencies. We distinguish between *syntactic*, *semantic*, and *pragmatic* heterogeneity.

3.1.1 Syntactic heterogeneity

When comparing two models described in different languages the first challenge is to compare the relation between constructs in different languages. For example, an arrow in one language might be used to describe data flow, while in another language it denotes event flow. By comparing the syntax (i.e., the structure) of the different languages, relations and dependencies between them can be identified. For example, an activity in one language might be related to data flow in another language. These syntactic characteristics are language dependent and need to be identified by hand.

When dealing with syntactic heterogeneity, the challenge is to identify both matching concepts and non-matching concepts. Typically, conceptual modelling languages use concepts and relations between concepts to structure the world. Often, these concepts and relations appear to match concepts and relations in another language. However, usually there exist subtle differences between them and we need to identify exactly these differences.

3.1.2 Semantic heterogeneity

The second kind of heterogeneity between models concerns semantic heterogeneity. This is a broad research area closely related to ontology matching [55, 87, 93], where the challenge is to find a match between an ontology used in one model and an ontology used in another model. A common problem is to identify differently named concepts in two models referring to the same entity in the cooperation; i.e., to find coreferences in the different models. For example, one model might use the term “seller” where another one uses the term “provider”. Another common problem is to identify homographs where one semantic concept is used in different models to refer to different entities in the cooperation. For example, in one model the term “seller” might refer to a wholesaler in the cooperation, while in another model it refers to the retailer that buys from the wholesaler. When modelling inter-organizational cooperations, semantic heterogeneity is often the result of different model developers, different actors discussing the models, and different purpose of the models. Although ontology matching is a well established research area, automatic ontology matching constitutes a challenge where many matchings are still done by hand which is a tedious process.

3.1.3 Pragmatic heterogeneity

We refer to heterogeneity between two conceptual models describing an inter-organizational cooperation that is not caused by semantical or syntactical differences as *pragmatic heterogeneity*. Pragmatic heterogeneity is the result of differences in *purpose and focus* of the models, which leads to a variety of representations [95]. However, both overlap and dependency between models need to be identified so that consistency can be checked. For this, heterogeneity is identified and handled. Here, we summarize common pragmatic heterogeneity in inter-organizational models.

Perspective & focus. As discussed, it is important to provide several models on the inter-organizational cooperation that together capture the full complexity of the cooperation. Every model *focuses* on a specific aspect of the system ensuring all details of that aspect are captured in the model. As a consequence, other parts of the cooperation that are not part of the focus, are suppressed or filtered out in the model at hand. Secondly, while focusing on a specific aspect, the cooperation is described from different *perspectives*. Next, we describe both focus and perspective of a model.

First, there is a choice how to *focus* the model. Typically, either *partial models* or *viewpoints* are used, as discussed in Chapter 2.3 (cf. Figure 3.2). In a partial model the focus is on a subset of the cooperation that is described. For example, one model might describe interactions between all suppliers in a cooperation, while another one describes the interactions between one specific supplier and its customers. Here, the distinction is made by splitting the physical world to be described into parts. In a viewpoint, a particular aspect of the cooperation is modelled, ignoring all other aspects. For example, one model might focus on money being exchanged between actors, while another describes network

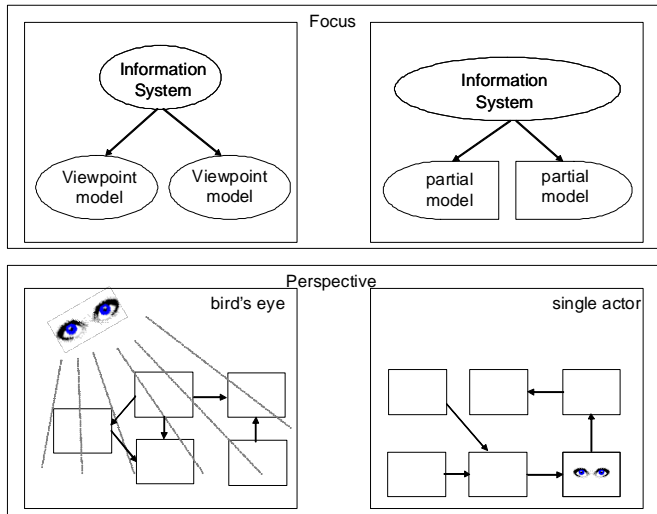


Figure 3.2: Perspective and focus

messages exchanged between the actors. In both cases the complete physical world is modelled, but details about this world are left out.

Second, there is a choice from what *perspective* to describe the model. Typically, either a *single actor perspective* or a *bird's eye perspective* is chosen (cf. Figure 3.2). On the one hand, a model with single actor perspective ignores any information that is not related to this specific actor concerning the cooperation. For example, a cooperation where a group of wholesalers sells goods to a group of retailers, might be described by a model depicting the relation between one specific retailer with wholesalers, ignoring retailers and wholesalers it has no relation with. On the other hand, a model from a bird's eye perspective describes the cooperation with all actors involved.

Both foci (i.e., partial models and viewpoints) can be described from both perspectives (i.e., single actor perspective and bird's eye perspective). For example, a model might describe network connections in a cooperation (i.e., viewpoint) from the perspective of a specific seller (i.e., single actor perspective). Another example is a model describing just the suppliers (i.e., partial model) and all their relations (i.e., bird's eye view).

Although the models together describe the complete cooperation, typically, several parts of the cooperation are described in more than one model, i.e., typically, there exists an overlap between the different models. The challenge is to check consistency, especially concerning overlapping parts, between models with different focus. For example, comparing a model describing a specific supplier with one describing average behavior of a group of suppliers, raises the challenge to compare one specific supplier with average values of a group. Secondly, there often exist dependencies between the different models where some aspects in one model influence the performance of another model. Consider, for example, one partial model describing a company's relation with its customers and

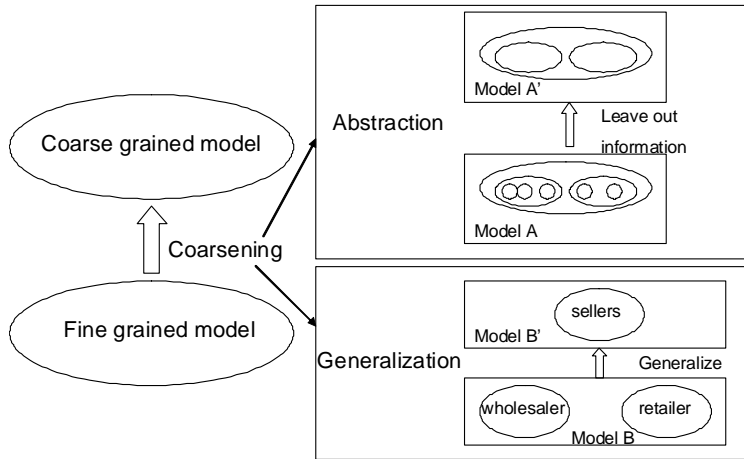


Figure 3.3: Coarsening models

another one describing its relation with its suppliers. Now, performance of the model between company and suppliers influences performance of the model depicting the relation with its customers. In more detail, when the supplier model describes delivery times of two weeks after a request, the customer model needs to comply with these two weeks. Identifying focus and perspective of a model provides a starting point to find overlap and dependencies between models as well as to ensure consistency between them.

Granularity. Typically, a cooperation is described through models of various *granularity*. Granularity is the level of detail with which the cooperation is described in a model. More granular models (i.e., more specific models) are referred to as *fine-grained* models as opposed to *coarse-grained* ones. Coarsening a model, i.e., making a model less detailed, is another way of filtering out details on the cooperation that are not necessary for the purpose of the model [117]. We distinguish two different types of coarsening: *abstraction* and *generalization* [117] (cf. Figure 3.3).

Coarsening through abstraction is the process of *leaving out* details on the cooperation (cf. Figure 3.3). In conceptual models, when describing inter-organizational cooperation, *transitive abstraction* is often used to reduce complexity. For example, a company might deliver its products using a transporting company, while a model describing this process might mention the transfer of products from company to customer, leaving out the transporter. A requirement for transitive abstraction is that relationships are of the same type and in the same direction. Another commonly used technique is *substitutional abstraction* where a set of related concepts is substituted by one concept describing the set. Other than in transitive abstraction, relations might be of different types. Moreover, besides removing concepts and relations, a new concept is added. For example, the description of the internal business processes for handling an incoming order, might be represented as one process named “handle order”.

Coarsening through generalization is the process where *commonalities* between concepts or their relations are identified and the result is used to describe a *set of* concepts or relations (cf. Figure 3.2). In this case, no information is left out, but rather described on a higher level of detail. Common ways of generalizing in conceptual models for co-operations are (1) to identify *patterns* [80] and (2) to identify *hierarchies* [107]. When generalizing through patterns, several sets of concepts and their relations are described as one general set. For example, a company allows both payment by bank transfer and by credit card, while a model describes the pattern of money payment in general. When generalizing through hierarchies, a set of concepts is described through one general concept. For example, ‘wholesalers’ and ‘retailers’ are grouped into ‘sellers’.

The challenge at hand is to find relations and dependencies between concepts and relations in models with different granularity levels. For example, relating high level concepts on sales targets in one model with low level concepts on network exchanges in another model is not straightforward. Another solution is to bring models to the same granular level by coarsening through abstraction and generalization. An obvious consequence of coarsening models is loss of information.

Time frames. A third pragmatic heterogeneity factor is difference in *time frames* of models. Each conceptual model of a cooperation is meant for a specific period of time. The smallest possible time frame is captured in *instance-based* models, while other models describe a *period of time*. We distinguish between models that have different *length* in modelled period of time, and models that describe an equal length, but have a *shifted* time frame. For example, a model describing average costs of a commodity for the coming year and another model describing expected profits for selling the product containing that commodity for the coming month have different time frame length. A model describing the cooperation for the coming week and another one for the following week have a shifted time frame.

The problem at hand is to check models for consistency since their time frames do not match. Consider the example where average commodity costs are determined per year and expected profit per month. It is difficult to state something about their consistency since the current expected profit might not fit average commodity costs, while the remaining eleven months of profit might make up for this. Therefore, a choice in handling these time frame differences needs to be made, and a first step is to recognize these differences.

Estimation and prescription. Since models considered in this thesis describe inter-organizational cooperation as it should be, they are referred to as *prescriptive* models. Typically, these models describe *agreements* between different actors in the cooperation. Their implementation *enforces* actors involved to act according to the agreement. For example, a model might describe delivery of goods is done only *after* receiving a payment. This behavior might be enforced in the implementation. However, besides agreements such models might also contain *estimations*. Typically, these estimations are done for this part of the cooperation that cannot be controlled through implementation like customer behavior. Typically, estimations are made on the number of expected customers, expected

revenue, payment choices, etc. Implementation of these estimations should *allow* the estimated behavior as well as deviations from it. For example, when it is estimated that fifty customers will purchase a product this month, the system should allow this as well as deviations from this behavior, i.e., more or less customers.

Often, it is not immediately clear whether certain behavior is an *estimated* average or an *agreed* average. For example, a business model might describe an average of fifty customers per month (i.e., an estimated average) that should receive their ordered products on average in three days (i.e., an agreed average with the suppliers). Both averages are depicted as transfers between actors, leaving the difference between estimated and agreed average implicit. However, this difference should be implemented and when comparing high level models (like business models) with more detailed, low level models that are directly implemented (like workflow models) this difference should become apparent.

In this thesis we refer to *prescriptions* when parts of a model *enforce* behavior, and we refer to *estimations* when parts of a model are *expected* to behave in a certain way.

3.2 Alignment with the running system

Aside from checking consistency between the different models at design time, their consistency with the running system should be checked (cf. Figure 3.1). Checking a model against the running system is usually done based on *event logs* and is typically referred to as *conformance checking* [99] or *consistency checking* [8, 49]. In particular, it is crucial to check whether the models are implemented accurately, whether all actors behave according to the made agreements, and whether estimated behavior is indeed realized. An event log is consistent with a model if the essential parts of the model do not contradict reality, i.e., it does not contradict the content of the event log, or vice versa.

In this thesis, we assume the event log is consistent with the running system (cf. Figure 3.1). Consequently, the event log is used as correct representation of the running system. For consistency checking between a running system and its models, we distinguish two main challenges. The first one is to identify which essential parts in the model actually *appear* in the event logs. For example, when estimations are done on the number of customers that register the coming month on a Web site, this data is detected as events in event logs. However, estimations on the male-female ratio of these registrations might not be visible in such log. The second challenge is to *abstract* essential information from event logs, i.e., to abstract information that enables consistency checking between running system and model. Typically, either the system is adapted in such a way that events entering the event log have the proper format or the necessary format is reconstructed from raw event logs after they are created. Although the first option, i.e., adaptation during runtime, is preferred since it is a one time effort, often this is not possible because of used software. As a consequence, event logs are often analyzed after runtime, i.e., necessary information is reconstructed. In this thesis, we explain both techniques in Chapter 4.

3.3 Maintaining models

When checking consistency, contradictions between model and running system or between two models might be detected. Ideally, these inconsistencies are resolved by adapting either model or implementation. Resolving inconsistencies is part of *model maintenance*. Although we do not propose any maintenance solutions, we do provide sufficient feedback for the developer to make proper maintenance decisions. Furthermore, the method in this thesis should allow for extensions and further development of automatic model adaptation.

Maintaining a set of models describing the same system is particularly challenging since the different models overlap and contain dependencies. As a consequence, changing one model to regain consistency with another one could introduce inconsistencies with other models. Therefore, not only the inconsistency itself should be identified, but also its causes and, if possible, information on the effects of changes in the model to resolve inconsistency. As discussed in Chapter 2, this is typically not provided by multi-model consistency approaches, where inconsistencies are identified, but no solutions for model maintenance are provided.

3.4 Solution requirements

Based on the *conceptual frame* of our research discussed in Chapter 2 and the *problem analysis* done in this chapter, we formulate solution requirements for our method. The aim of the method is to provide an approach to check and maintain consistency between models for inter-organizational cooperations.

The *scope* of our research determines the scope of our method. The first requirements concern this scope. It is important that our method enables handling models specified in different languages since we aim supporting management of conceptual models for inter-organizational cooperations in general. Therefore, the approach should be *language-independent* (Requirement 1). Furthermore, the approach should support management of *heterogeneous models* (cf. Section 3.1) since a cooperation is typically described by a set of models using different specification languages (Requirement 2). A third design time requirement is that the method supports *checking* consistency (cf. Chapter 2.3.3). This is necessary since ensuring consistency through construction does not provide support for checking consistency at runtime where models are already constructed. In summary, we provide the following requirements:

- **R1** The method should be language-independent.
- **R2** The method should be able to handle heterogeneous models, i.e., models described in different languages.
- **R3** The method should ensure consistency through checking rather than through construction.

After defining the scope of the method, we define *guidelines* the approach should provide to manage consistency of models at *design time*. One goal of the method is to define *consistency constraints* for inter-organizational models (Requirement 7) (cf. Chapter 2.2). These guidelines for defining consistency constraints are only possible after we *compare* heterogeneous models. Therefore, the method should provide guidelines to *overcome heterogeneity problems* discussed in Section 3.1 (Requirement 4). To compare models we should provide guidelines to *identify overlap* (Requirement 5) and to *identify dependencies* (Requirement 6) between different models. In summary, the following requirements should be fulfilled:

- **R4** The method should provide guidelines for overcoming model heterogeneity.
- **R5** The method should provide guidelines for identifying overlap between models.
- **R6** The method should provide guidelines for identifying dependencies between models.
- **R7** The method should provide guidelines for defining consistency constraints for inter-organizational cooperation models.

In Section 3.2 we discuss alignment of models with the running system. The method should provide guidelines to identify *overlap* between event logs and models (Requirement 8). If there are inconsistencies between running system and models, these are detectable in overlapping parts. Furthermore, it is important that necessary information to check consistency between running system and models is logged in a usable format (Requirement 9). Aside from gathering necessary information, it is important to provide an approach that allows *abstracting* this information from event logs (Requirement 10). Using overlap between models and running system, and abstracted information from event logs, it is possible to define *consistency constraints*. These consistency constraints depict the relation between running system and model (Requirement 11). In summary, we formulate the following requirements to enable consistency checking between event logs and models:

- **R8** The method should provide guidelines to identify the overlap between event logs and model.
- **R9** The method should provide an approach to log information necessary for consistency checking.
- **R10** The method should provide an approach to analyze an event log to abstract necessary information for consistency checking.
- **R11** The method should provide guidelines for defining consistency constraints between running system and its underlying models.

To complete lifecycle management of our models, we discuss model maintenance in Section 3.3. We identify the necessity to show *consequences* of model adaptations to regain consistency (Requirement 12). Furthermore, it is important to provide *feedback* to users of our management approach. These users can be, for example, managers that maintain models. Such feedback should contain information on constraint violations, and on consequences of solving such violations (Requirement 13). We formulate the following method requirements for maintaining consistency at runtime:

- **R12** The method should provide an approach to show consequences of model adaptations on consistency relations.
- **R13** The method should provide an approach to provide monitoring results as feedback.

3.5 State-of-the-art research

Terminology differs greatly among researchers. However, related work described in this section is done in terminology used in this thesis. Table 3.1 provides an overview of the different approaches discussed in this section. The table arranges approaches according to the *categorization* discussed in Chapter 2.3. The *type of models* that are handled by the approaches is specified in the first part of the table. Secondly, we depict whether approaches provide mechanisms to cope with *inter-model* and *intra-model* consistency constraints. Furthermore, some approaches are limited to *homogeneous* models, i.e., different models described in one language, while others are able to handle *heterogeneous models*. The last part of the table specifies how consistency is *ensured*. Some approaches provide mechanisms to *check* consistency through *testing*, *translation of overlapping parts*, or *complete translation* of models. Three approaches ensure consistency through *construction*. At the end of this section, we discuss whether or not any approach fits the solution requirements for the method (cf. Section 3.4).

	Type of Models		Type of Consistency			Ensuring Consistency		
	Viewpoints	Partial Models	Inter-model Consistency		Intra-model Consistency	Checking		Construction
			Homo-geneous	Hetero-geneous		Testing	Translation OverlapComplete	
Mens et al. [79]		x		x	x	x		
Astesiano et al. [10]		x		x	x			x
Engels et al. [47]		x		x	x		x	
xlinkit [85]		x		x				x
Egyed et al. [44]		x		x				x
Varró et al. [111]		x		x	x			x
χbel [41]	x		x					merge
Uchitel et al. [110]	x		x					merge
Fradet et al. [52]	x			x				x
Bowman et al. [30, 29, 37]	x			x				x
Hunter et al. [61]	x			x				x
Viewpoints [51]	x			x				x

Table 3.1: Approaches for checking consistency

Partial models. Mens et al. [79] target at supporting consistent evolution of UML models. However, their approach also allows checking intra-model consistency. Their model checker is description logic based and implements the different UML metamodels. The metamodels ensure intra-model consistency, and relations between metamodels as defined by UML ensure inter-model consistency. Since their approach relies on implementing existing metamodels (that specify consistency constraints) of UML, this approach is not usable in a non-UML context.

Astesiano et al. [10] investigate existence of ambiguities and inconsistencies in the language definition of UML. The authors do not aim at solving inconsistencies between UML diagrams for a particular specification, but aim at finding these ambiguities and inconsistencies in the language itself. Their aim is to develop a consistent UML language using a logic-algebraic method. This approach focuses on reducing inconsistencies in UML by improving the metamodels. Therefore, they are classified as achieving consistency through construction by improving the metamodel (cf. Table 3.1). However, this approach relies on translating models.

Engels et al. [47] develop a method to check consistency of UML models to decide at which point in time of the development process, UML partial models should be consistent with each other. They distinguish between consistency of *dynamic* and of *static models* where dynamic models specify *behavior* as opposed to static models. In UML consistency requirements exist that specify consistency relations between different model types. An example of such a consistency rule is that a statechart has to accept each stimulus a sequence diagram specifies. Their implementation (the Consistency Workbench) tests whether two models are consistent against these consistency rules. They partially formalize the models (i.e., only overlapping parts) into a *common semantic domain*. For each consistency rule the suitable semantic domain for consistency checking is determined. A discovered inconsistency is either tolerated or resolved. Their approach is originally developed for horizontal consistency, but is later extended to evolution consistency [46]. With evolution consistency the emphasis lies in preserving certain aspects of the model while it is evolving, e.g., preserving the absence of deadlocks. This is achieved by adding rules to the implementation. A drawback of this approach is the use of several semantic domains for one set of models. Relations between constraints across domains are then not expressed. A specific problem is the question what the effect of solving an inconsistency between two models is in respect to other constraints these models might have to satisfy. In our work we avoid the use of different semantic domains to avoid losing these relations.

Xlinkit [85] is a method for expressing constraints between heterogeneous models. It offers a semantics that shows links between two mutually inconsistent elements of different models. Their focus is on identifying inconsistencies rather than solving them although their diagnostic method has later been extended with a repair actions method [86]. Although xlinkit is mainly used for UML models the authors argue their method is language-independent. The consistency rules are expressed in the tool using a restricted form of first order logic, which restricts expressiveness of the rules. Furthermore, the models to be checked for consistency have to be transformed completely into XML format. Both restriction on the rule definition and constraint that models must be expressed

in XML makes this approach unsuitable for our problem definition.

Egyed and Medvidovic [44] provide an approach for heterogeneous software development. It enables mapping of an architectural design model into UML models. This is a heterogeneous refinement approach since the architecture is refined into UML models. It ensures initial consistency only since it is a unidirectional approach. This means that consistency is ensured between architectural model and UML model, but any updates to UML models, or refinement of UML models into other UML models might interfere with the original architectural model. To overcome the problem of further refinement of UML models and their possible inconsistency with the overall architectural model, abstractions from concrete models to abstract ones (vertical) and from specific ones to generic ones (e.g., instance to class) are supported. Finally rules for transforming one language into the other in a consistent manner are defined. Although the authors state their approach is language-independent, it has only been illustrated by transforming C2 models [44] into UML models.

Many approaches rely on model transformations. The goal is to transform models from one tool to another by using an intermediate universal language (e.g. XML). However, a correctly defined metamodel is crucial to handle these transformations. Varró and Pataricza [111] tackle several of the metamodeling problems (causing inconsistencies) by defining different rules in their construction. They offer a metamodeling method which is applicable to UML as well. Their method proposes a multilevel metamodeling approach to overcome the well-known problem of replication of concepts. Heterogeneous refinement is supported in a consistent way. Although focus is on metamodeling and not on consistency checking, it does facilitate the development of consistent models. However, their approach does not extend beyond the redefinition of metamodels.

Viewpoints. Easterbrook and Chechik [41] develop the χ bel framework for merging inconsistent state machine models, i.e., state machine models that describe different behavior. χ check is a model checker which checks properties of merged models. Multi-valued logics are used which allow statements like “the majority says” instead of only “true” and “false” (i.e., “everyone says” and “no one says”). This enables reasoning over inconsistent models. The merging and reasoning process over different models depends on the relation models have with each other. Different models can be (partially) overlapping, can be different versions, or can be just interacting. The goal is to reason with stakeholders about different options they have to merge models. The models can be analyzed in different merge scenarios as well as before the merge. Since there is inter-model inconsistency, several ways of regaining consistency through merging techniques are possible. However, some stakeholders have to give up certain requirements and make concessions. The approach is only suitable for homogeneous models and is, therefore, not suitable for our problem.

Uchitel and Chechik [110] develop an approach to merge partial behavioral models in the same language. Here, *partial* refers to partial behavior where models provide concepts for not yet known behavior. Their approach is suitable for viewpoint models. It assumes models being intra-model consistent and argues that the result of the merger should be the

minimal common refinement of considered models. The result is said to be consistent if it is a common refinement of both models. This approach is suitable for any state-based behavioral system with formal semantics. It focuses on observable behavior of models rather than on structural aspects. The restriction made in respect to homogeneous state-based models is not sufficient in the context of our problem statement.

Fradet et al. [52] develop a method for defining multiple view architectures. Their approach is formal, suitable for heterogeneous models, and not language-specific. The structural part of each view is represented as uninterpreted graph. Consistency between different views is checked on graphs by means of an algorithm. The interpretation of graphs is done through formulating a set of constraints that specifies both intra-model and inter-model requirements. Checking consistency of diagrams with respect to requirements is done through some decision procedure. Although it is recognized that formalizing all constraints in graphs might be cumbersome for large models, the authors state that this is not a problem since their graphs can be expressed in constraints. Their approach is specifically designed for software architectures where the focus is on different components and their connections. More specifically, they focus on the structural part of the architecture, while our method aims at a more complete approach where also communication between different actors is modelled.

Consistency checking is a big concern in Open Distributed Processing (ODP). ODP provides a method for distributed development where five viewpoints are proposed for the modelling phase: enterprise, information, computational, engineering, and technology. Several viewpoints on the same system are developed, each having its own focus. The requirements modelled in each of these viewpoints should be reflected in the overall description of the system. Each viewpoint is described in a separate specification. All specifications together should be reflected in the overall description of the system. An approach to ensure or check the existence of such a description in ODP is proposed by Bowman et al. [30, 29, 37]. They aim at checking and ensuring existence of an overall system description capturing each viewpoint. Consistency between specifications is defined as the existence of an internally valid description which represents all requirements of the specifications. Consistency is checked between a viewpoint specification and the overall description, but not between two viewpoints. As a result, consistency requirements between one viewpoint and the overall description, might differ from requirements between another viewpoint and the overall description. This difference occurs when viewpoints are described in different languages and have different levels of abstractions. The authors refer to this as unbalanced consistency. By doing bilateral consistency checks (i.e., between each viewpoint and the descriptions separately) it is difficult to create global consistency. Each viewpoint might be consistent with one or more descriptions, but finding a description which is consistent with all viewpoints is hard to find. Therefore, they propose to ensure global consistency by unification of viewpoints. Two viewpoints are unified after one of them is translated into the other, or after both of them are represented in a common semantic domain. By showing intra-model consistency of the unified model, two viewpoints are proven to be consistent with each other. Now, a third viewpoint can be unified in the same manner with the result of the previous unification. For our purpose this ap-

proach is less suitable since it relies on one universal language in which all viewpoints are translated. This approach might work for ODP, but not for our purposes.

Hunter and Nuseibeh [61] propose a logic-based approach for reasoning in the presence of inconsistencies. They propose quasi-classical logic, which is a weakened version of classical logic. It allows reasoning with inconsistencies by deriving all resolvents of assumptions without allowing trivial derivations. This approach first labels information and then these labels are propagated through the reasoning process. This allows tracking inconsistent information and provides a better problem analysis. Furthermore, this approach computes maximally consistent subsets of the inconsistent information. Obviously, this means that viewpoints are translated into logic.

The Viewpoints framework [51] and extensions to it [42, 50] allow inconsistencies when developing the different viewpoints in order not to kill creativity. Their method is logic-based and allows checking well-formedness (i.e, intra-model consistency) as well as inter-viewpoint consistency (i.e., inter-model consistency). The viewpoints are translated into logic and tested against predefined consistency and well-formedness rules. Consistency rules are created per viewpoint and are locally stored. So, there is no overall consistency checking mechanism. The authors assume that each developer is concerned with consistency of the viewpoint he is working on. For a relation between two models the Viewpoints framework defines two rules: one belonging to the first model and one to the other model. As a result, engineers of the models both have one consistency rule to manage. If a rule is violated, its “owner” determines whether or not it should be resolved. For each kind of rule violation there exists a solution approach. Since the goal is not to maintain global consistency, this approach is not suitable for our problem. The focus is rather on bilateral consistency, resulting in a local view. As a result, it is not calculated what the effects are on relations other than the bilateral relation that is dealt with.

Overview approaches. Spanoudakis and Zisman [103] introduce a method with a six step approach to maintain consistency in model development. Their approach is the unification of several other approaches and tries to cover all concerns. The six steps are as follows: detecting overlap, detecting inconsistencies, diagnosis of inconsistencies, handling inconsistencies, tracking, and specification and application of an inconsistency management policy. An interesting detail in this method is the distinction between different overlap types. Many approaches assume that there either exists a complete overlap or no overlap at all, which is not realistic. For example, small differences in arity are often not allowed in such approaches when comparing two Entity Relationship Diagrams, while this approach does. As solution most approaches use a shared ontology, or require some human inspection, or there is some automated similarity analysis in which syntactics and semantics are checked between models. Detection of inconsistencies is done with logic-based approaches, model checking approaches, specialized model analysis, or with human centered exploration. The authors state that the main challenges in inconsistency detection are efficiency and scalability, especially when dealing with evolving models. Although this approach does not provide a tool or specific approach for maintaining consistency, the identification of concerns in maintaining consistency relations is impressive.

Nuseibeh et al. [88] describe a high-level method for managing consistency in software development. In their method consistency rules are applied to detect inconsistencies during the monitoring process. Then inconsistency is analyzed and it is determined how to deal with the inconsistency. Consequences of handling the inconsistency are, in turn, monitored. Conclusions of some practical experience are observations on how difficult it is to decide *when* to do a consistency check. Especially since partial models are developed in parallel and updates might look incomparable to the original model checked for consistency. It might be too costly to do the check. As a last point they state that during model development, inconsistencies might be tolerated. Many of these inconsistencies sort out themselves. Obviously, hoping that an inconsistency resolves itself or that the inconsistency does not cause any problems, causes significant risks in information systems engineering.

Although these overview approaches provide some starting points for maintaining consistency of inter-organizational models, they do not provide sufficient detail to overcome model heterogeneity and to define consistency constraints. Therefore, they provide a good starting point, but are not sufficient for our purpose.

3.5.1 Discussion on related work and solution requirements

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
Mens et al. [79]		v	v			v	v						
Astesiano et al. [10]		v				v	v						
Engels et al. [47]		v	v			v	v						
xlinkit [85]	v	v	v	v		v	v						
Egyed et al. [44]	v	v		v		v	v						
Varró et al. [111]	v	v		v		v	v						
χ bel [41]			v		v	v	v						
Uchitel et al. [110]	v		v		v		v						
Fradet et al. [52]	v	v	v	v	v	v	v						
Bowman et al. [30, 29, 37]	v	v	v	v	v	v	v						
Hunter et al. [61]	v	v	v	v	v	v	v						
Viewpoints [51]	v	v	v	v	v	v	v						

Table 3.2: Solution requirement compatibility of related work

Table 3.2 depicts how well discussed related work fits solution requirements for the method defined in Section 3.4, where a “v” indicates the approach satisfies the requirement. None of the discussed approaches satisfies all requirements. However, many of them could be extended so that they satisfy most. Requirements R1 - R3 should be satisfied from the start: R1 states the approach should be *language-independent*. Approaches that are language-dependent are too specific in their solution to be applicable in general. R2 states the approach should be able to handle heterogeneous models. Approaches that

focus on homogeneous models cannot be extended so that they are suitable for heterogeneous models since their solution is too specific. R3 states that ensuring consistency should be done through construction. The reason is that our aim is to maintain the models during runtime. Therefore, it should be possible to check and adapt models after finishing the development phase. Consistency ensuring through construction makes it impossible to check consistency afterwards.

Requirements R4 - R7 concern the definition of *guidelines*. If approaches do not provide guidelines, it is possible to extend it. Requirements R8 - R13 concern *runtime* requirements. None of the discussed approaches is developed for runtime use, but most of them could be extended for this purpose.

Considering the necessity for satisfying R1 - R3, we conclude that xlinkit [85], Fradet et al. [52], Bowman et al. [30], Hunter et al. [61], and Viewpoints [51] provide an approach that might be extendible for our purposes (shaded grey in Table 3.2). However, each of these approaches lacks some functionality as we will show in the following discussion.

Xlinkit [85] has several restrictions which makes its use in an inter-organizational cooperation setting less suitable. Particularly, xlinkit requires a complete translation of the considered models into XML format. Especially since we aspire a language-independent approach, it is not feasible to provide a general approach for this translation. Furthermore, their method requires specification of consistency rules in a restricted form of first order logic. This limits the expressibility of consistency rules too much for our purpose.

Fradet et al. [52] rely on a complete translation of each viewpoint model into an uninterpreted graph. Also consistency constraints are represented as graphs. Consistency checking is done on graphs. As the authors recognize themselves, such an approach is too cumbersome when considering large, complex, and semantically rich models. Therefore, this approach is not suitable for our needs.

Bowman et al. [30] develop an approach for ODP consistency checking, which is - strictly speaking - not language-independent. However, this approach appears to be applicable to other languages as well. Consistency requirements are modelled for each relation between viewpoints separately, which is, in our opinion, a smart approach. However, it relies on a universal language into which all viewpoints are completely translated. This translation is done through unification of viewpoints: one viewpoint is unified with another one, then a third one is added to this unification, etc. The authors provide a universal language for all ODP specifications. However, for our language-independent approach, it is not possible to provide one language that allows complete translation of every possible inter-organizational model. Therefore, this approach is not suitable for our purpose.

Hunter et al. [61] provide an approach that is suitable for multi-model development where not all inconsistencies need to be resolved immediately. This approach relies on a full translation of each viewpoint in quasi-logic which is not possible for inter-organizational cooperations where we aim to preserve necessary details. A quasi-logic translation does not allow for these details.

The Viewpoints [51] approach supports multi-model development from the developers perspective where each developer is responsible for his own model. As a result this approach provides techniques to preserve local consistency. However, we aim at global

consistency through a consistency management approach rather than leaving consistency checks at the developers table. Especially since it is difficult for a single developer to oversee the results of his local choices on the overall system.

In conclusion, there exist many approaches for checking and/or maintaining consistency over many different disciplines. However, none of these approaches is suitable for our problem. Many solutions have a high level description of the problem and lack suggestions how to solve it. Other solutions are often language specific and are not usable in other domains. The few approaches which are on the right level of abstraction, lack either the option to apply them in a heterogeneous context or rely on a common semantic domain (i.e., an universal language is assumed).

3.6 Summary

In this chapter we discuss specific complexities that arise when ensuring and maintaining consistency for conceptual models on inter-organizational cooperations. We discuss design time issues (Section 3.1), runtime issues (Section 3.2), and maintenance issues (Section 3.3). Together with results from Chapter 2 we formulate requirements for our method (Section 3.4). We conclude our chapter with a review of related work against solution requirements (Section 3.5). We conclude that there is currently no method available that satisfies our complete set of requirements, or one that can be extended for this purpose. Therefore, we define our method in the next chapter.

MADE4IC: AN ABSTRACT METHOD FOR MANAGING MODEL DEPENDENCIES IN INTER-ORGANIZATIONAL COOPERATIONS

In this chapter we describe our MaDe4IC method (MANaging DEpendencies for Inter-organizational Cooperations) for managing models of inter-organizational cooperations. When companies cooperate, different conceptual models need to be related to each other. In this thesis we describe a stepwise method for relating such models. Our goal is to provide a method that can be used to *manage* rather than solely monitor models. We develop our method using results from our problem investigation (cf. Chapter 3). Typically, monitoring and managing models is done by gathering information about behavior of different entities in the cooperation. Conclusions on *why* deviant behavior appears are drawn by trying to reconstruct relations between the different entities. In this method, we treat dependencies and relations between entities as the most important part for monitoring and managing such models. We use dependencies between and within models to analyze *causes* for deviant behavior. By considering these dependencies from the beginning, as opposed to reconstructing them in retrospect, we enable a detailed analysis of problems and provide useful information on consequences of model changes when trying to regain consistency. In Section 4.1 we sketch specific characteristics of such models and discuss what our MaDe4IC method entails. We introduce a running example in Section 4.2. In the remainder of the chapter we describe each step of the method in detail.

4.1 General approach

4.1.1 Model characteristics

We consider conceptual models that depict cooperation between different companies or systems. A typical conceptual model consists of *concepts* representing *entities* from the real world. These entities are characterized by some *properties* which are also represented by the concepts. Furthermore, *relations* are used to depict interrelated concepts. Typically, models of cooperative systems that abstract from any internal behavior contain

the following types of concepts and relations:

1. Actors in the cooperation, i.e., parties cooperating together,
2. Exchanged objects (such as information or money) that are used to establish the relation between actors,
3. Relations between concepts, i.e., between actors, and between exchanged objects in different models.

We characterize these concepts by properties. For example, a concept representing a bike might have the property “price”, indicating the specified bike has a certain price. Furthermore, the *nature* of the relation might differ. For example, there are causal and temporal relations indicating some concepts have a causal relationship, while others have a temporal one.

Which real-life entities and relations in the cooperation are captured by a specific model depends on the type of model used. Our MaDe4IC method is applicable to cooperative systems showing these characteristics. It focuses on identifying dependencies between concepts within and between models. Furthermore, at runtime we construct an event log model that depicts relations between the different entities. These event logs register events occurring between the different actors. Our event log model is related to and compared with models developed at design time. These dependencies within and between models form the basis for managing the models by analyzing causes for inconsistencies.

4.1.2 The method

Figure 4.1 gives an overview of our approach. It is divided in five phases. Input is provided by different conceptual models developed for the inter-organizational cooperation. As example consider two models with one model describing how each actor benefits from the cooperation, and another model describing details which messages are exchanged between the actors to accomplish this.

In the first phase, **model analysis**, preparations are done for identifying dependencies within and between models. This phase comprises two steps. In **Step 1**, each model is analyzed resulting in model characteristics. The latter are used to homogenize the models in **Step 2** in order to make these (potentially heterogeneous) models comparable. Although this constitutes an important step for enabling model comparison [15, 105], this part does not constitute the focus of this thesis. We give some guidelines to tackle heterogeneity problems in a structured way, but without giving a detailed description on how to do this.

In the second phase, **inter-model analysis**, we identify different model relations and use them to define consistency constraints. This second phase also comprises two steps. In **Step 3** we compare each pair of models, using knowledge about their characteristics (cf. Step 2) to identify inter-model relations. For example, the existence of a concept in one model might depend on the existence of a concept in another model. Based on these relations, for each model pair we define a set of consistency constraints in **Step 4**. These

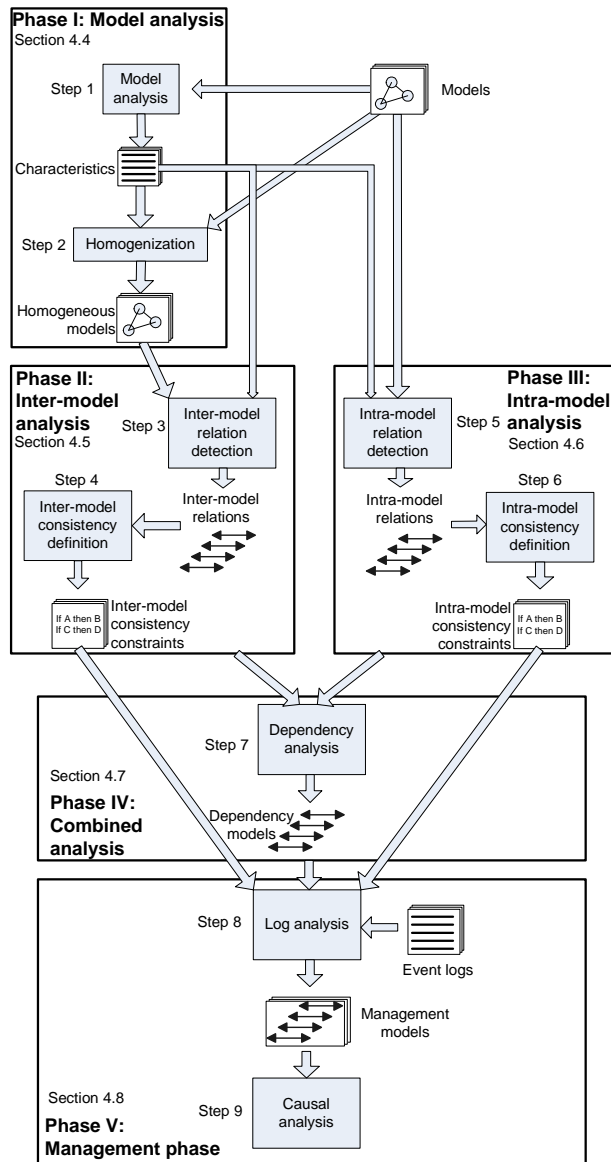


Figure 4.1: MaDe4IC: Method for managing dependency relations in inter-organizational models

constraints explicate when we consider two models to be consistent with each other. For example, if the existence of a concept from one model depends on the existence of a concept from another model, the constraint explicates this dependency relation. Now, the models are considered inconsistent if the concept exists in one model, but not in the other.

In the third phase, **intra-model analysis**, we identify in **Step 5** for each model the type of relations used to connect the different concepts. This analysis helps us to explicate intra-model consistency constraints in **Step 6**. These constraints describe when a model is considered to be consistent with the running system.

In the fourth phase, **combined analysis**, the identified dependency relations and the consistency constraints are used to perform a combined dependency analysis in **Step 7** that creates the combined dependency models. Together, these models describe the different dependencies and consistency constraints in a structured way so that they form a proper base for managing the models. These dependency models are formal models for easy implementation. Furthermore, they only depict those parts of the original models that influence the consistency constraints.

The fifth phase, **management phase**, occurs at runtime of the system and comprises two steps. **Step 8** checks consistency constraints in the *log analysis* against the event logs where the dependency models are used to construct feedback for managing the models. This feedback enables causal analysis for constraint violation as well as prediction of consequences for solving inconsistencies in **Step 9**.

The first seven steps are done manually and (possibly) before the system is running. Steps 8 and 9, the actual management steps, can be automated through implementation, using the formalization created in Step 7. In the following sections we describe each phase in detail.

4.2 Running example

We introduce a running example to illustrate the different steps of our MaDe4IC method (cf. Figure 4.2). In this example, we are interested in the cooperation between a company selling bikes online, and its customers and providers. The company offers both *mountain bikes* and *city bikes* for online purchase. The company buys the bikes in parts from a provider, then assembles them and finally resells them. Each product is sent to the customer for a fixed delivery price. In addition, a customer can choose fast delivery, or delivery on a specific moment like evenings or weekends. For such service an additional fee is calculated.

We use an easy modelling notation to model our running example in different ways. For example, we show how to model viewpoints (cf. left part of Figure 4.3) and partial models (cf. right part of Figure 4.3). With this modelling notation, *concepts* that contain one or more *properties* can be modelled. These concepts and properties can have *symmetric* or *asymmetric* relations.

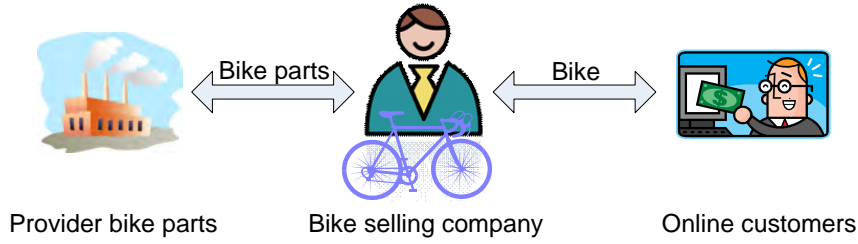


Figure 4.2: Running example: Selling bikes

4.3 Phase I: Model analysis

In the model analysis part of our MaDe4IC method, first of all, each model is analyzed to identify its specific characteristics (Step 1). Following this, related model pairs are homogenized (Step 2). As discussed, homogenization is not the core part of this thesis. However, in consistency checking between different models of one system, it is inevitable to address this problem. Therefore, we provide a set of guidelines in Step 2 to stepwise tackle the problem without discussing solutions in detail. In the remainder of this thesis we assume heterogeneity problems are overcome.

4.3.1 Step 1: Model analysis

Goal: Identify model characteristics of each conceptual model of the cooperation to enable inter-model and intra-model analysis.

Our method starts with a *model analysis* phase in which each model is analyzed for its characteristics (cf. Step 1 in Figure 4.1). This analysis aims at identifying different conceptual characteristics as discussed in Section 3.1.3. These characteristics are later on used to enable inter-model as well as intra-model analysis. We analyze each model for the following characteristics:

- | | | | |
|----------------------|----------------|---|-----------------|
| (i) Focus: | Viewpoint | ↔ | Partial model |
| (ii) Perspective: | Single actor | ↔ | Bird's eye view |
| (iii) Property type: | Estimation | ↔ | Prescription |
| (iv) Time frame: | Instance-based | ↔ | Period of time |

To illustrate Step 1, we consider two examples:

Example 1. When considering the example described in Section 4.2 a possible choice for modelling this scenario is to develop two viewpoint models. Assume model A describes exchanges that have some value between customer and company, while model B describes messages exchanged between actors. The left part of Figure 4.3 depicts these two models.

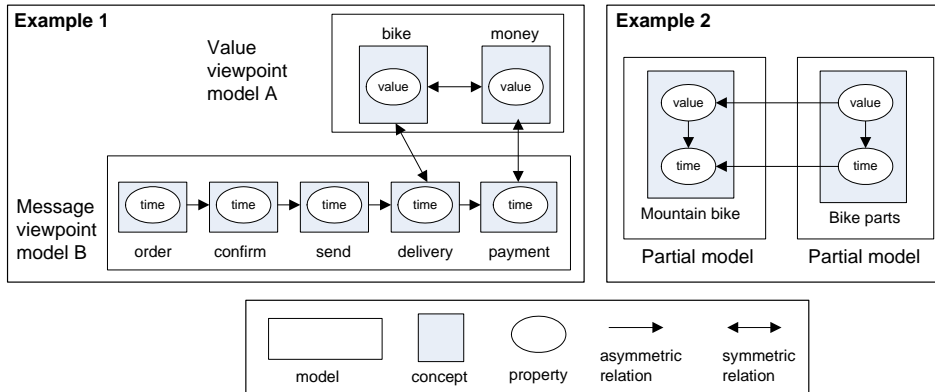


Figure 4.3: Dependency relations between viewpoints and partial models

Example 2. Another possible choice for modelling the scenario from Section 4.2 is to develop two partial models. The right side of Figure 4.3 depicts two such models where each one describes one entity with different properties. The mountain bike is the product purchased by the customer, while the bike parts are represented by the box containing the unassembled mountain bike as purchased by the company.

(i) **Focus:** A model focusses on a subset of entities as well as relations of the system (i.e., a partial model), or on the complete system with focus on a specific characteristic (i.e., a viewpoint model). Typically, models related to the same system have the same focus type, i.e., all of them either constitute partial models or viewpoints. Example 1 in the left part of Figure 4.3 depicts our scenario with two viewpoint models, while Example 2 in the right part of the figure depicts two partial models. Identifying the focus type is an important prerequisite for both intra-model and inter-model analysis (cf. Figure 4.1).

(ii) **Perspective:** A model can be described from a local perspective of a specific actor in the cooperation (i.e., single actor perspective) or from a global perspective where the system is considered as a whole (i.e., a bird's eye perspective). The models in Figure 4.3 all reflect a single actor perspective, i.e., they are described from the perspective of the company. Other than with focus, perspectives often differ between models describing one system. This heterogeneity difference is dealt with rather than homogenized since the choice in perspective is model-specific and should not be changed. A difference in perspective might lead to a difference in granularity where a single actor perspective might describe the situation for one specific actor, while a bird's eye perspective describes the situation for a set of actors. In such situation it is important to identify how the part description from the single actor perspective is related to the whole description of the bird's eye perspective. Identifying perspective type is necessary for both inter-model and intra-model analysis.

(iii) **Property type:** The type of a property describes whether a property value is an estimation or prescription. This type can differ *within* a model where some properties are

estimations, while others are prescriptions. Consider, for example, the left part of Figure 4.3 where the company models with the value viewpoint the value of a bike, and what a customer pays for the bike. These are prescribed values of the properties which *prescribe* behavior. The message viewpoint, however, describes an expected time frame in which messages are exchanged. There are estimations on exchange times described with the property ‘time’. The estimations are *predictions* of reality, rather than prescriptions. It is important to distinguish the two types for both intra-model and inter-model analysis where properties of different types are related.

(iv) **Time frame:** The time frame used in the model should be identified. Especially whether a model is valid for a specific period of time or whether it is an instance-based model. For example, a model might describe behavior for the coming half year (period of time) or describe behavior for each invocation separately (instance-based). This information is needed to homogenize the models for the inter-model analysis (cf. Figure 4.1).

The result of Step 1 is the following:

- Each model is characterized by its *focus*, *perspective*, and *time frame*.
- Each property in the model is characterized by its *property type*.

4.3.2 Step 2: Homogenization

Goal: Identify heterogeneity between model pairs using model characteristics identified in Step 1. Homogenize differences between model pairs if possible.

We *homogenize* the different models to enable their comparison (cf. Step 2 in Figure 4.1). For homogenization we need to overcome differences between the models. For this, we address *syntactic*, *semantic* and *pragmatic* heterogeneity (cf. Chapter 3.1) where certain heterogeneity problems are dealt with in later step, while others are homogenized in Step 2:

- syntactic: deal with in later steps
- semantic
 - coreferences: homogenize in the current step
 - homographs: homogenize in the current step
- pragmatic
 - perspective: deal with in later steps
 - focus: deal with in later steps
 - granularity: homogenize in current step
 - time frame: homogenize in current step

- estimation and prescription: deal with in later steps

It is not necessary to solve all heterogeneity problems through homogenization since this might cause loss of information. For example, *syntactic heterogeneity* is inherently present since models are simply described using different languages. The key here is to identify syntactic commonalities and differences between the modelling languages. Explicating the differences and commonalities between *concepts*, *relations*, and *properties* of two models suffices.

Example. *The models from Figure 4.3 are described in the same language, i.e., they use the same constructs for concepts, relations, and properties. There are no syntactic differences.*

Furthermore, *semantic heterogeneity* must be addressed, especially through identifying and solving coreferences and homographs. Recall that coreferences are semantically different concepts in two models referring to the same entity in the system, and homographs are semantically equal concepts in two models referring to different entities in the system.

Example. *The value viewpoint model A in Figure 4.3 describes a concept bike, while the partial model in the same figure models a concept mountain bike. Although the concepts differ semantically, they refer to the same real-life entity, i.e., they are coreferences.*

This needs to be identified and resolved by choosing one semantics to describe the entity. Also when two semantically equal concepts in different models refer to different entities (i.e., homographs) one concept should be renamed. As discussed in Section 3.1.2, this is a broad research area and a detailed description on how to handle semantic heterogeneity is out of scope for this thesis.

In addition, some *pragmatic heterogeneities* can be homogenized in the current step, while others are simply recognized here and dealt with when comparing the models in later steps. Dealing with heterogeneity between models later means it becomes then more difficult to find overlap or relations between them since the models are still heterogeneous. However, by identifying where the models are different in the current step, rather than homogenizing them, relating the models in later steps becomes easier. As discussed before, foci, perspectives, and property type differences are used to ease identifying relations in later steps, rather than being homogenized in Step 2. Concerning pragmatic homogenization, only *time frame* differences and *granularity* differences are homogenized.

A *time frame* difference between two models needs to be resolved before the models can be compared in the following steps. Changing the time frame of a model such that it fits the time frame of another model is not a straightforward process. It involves discussions with all involved actors to come to an agreement on how to handle the changes. Generally speaking, there are two approaches that can be followed to homogenize time frames: either a time frame is shortened or it is lengthened. Both approaches have their own hurdles that need to be overcome.

Example. The Value viewpoint model A from Figure 4.3 describes the monetary relation between bikes the company sells and money the company receives for this. The model might be valid for a period of one year, i.e., there is a time frame of one year. The related Message viewpoint model B, however, describes necessary message exchanges between partners per bike sale, i.e., the time frame is instance-based. This time frame difference is resolved by shortening the time frame of the Value viewpoint model A from a period of one year to an instance-based model.

When *shortening the time frame*, challenges arise with agreed upon *average* values and with *indivisible* exchanges. For example, if a contract specifies an average value to accomplish over twelve months, this average value cannot be assumed to hold in the first six months only. This especially cannot be assumed if values differ, for example, due to seasonal behavior. Furthermore, if there are exchanges between actors that only occur once during a period, shortening this period implies dividing a single exchange which is not always possible. For example, if the company expects to sell one bike per year, reducing a model describing this year to a model with a time frame of half a year will not be possible since splitting the sale of one bike into two parts is not possible.

When *extending a time frame*, in turn, the biggest challenge is to estimate content of future contracts and models. For example, if a contract is specified for the coming year and we need to extend the model for the coming two years, either all contracts need to be extended, or estimations on future contracts become necessary. Changing a time frame is done *per model pair* since it is desirable to keep the comparison between models as close to their original time frame as possible. Therefore, the changes typically come with estimations of behavior.

The second pragmatic difference between models to be homogenized in this step constitutes the *granularity* difference between the models. In order to check consistency between two models they need to describe the system on the same level of granularity. Coarsening is typically used to overcome granularity differences as described in Chapter 3.1.3. Whether abstraction or generalization techniques are used needs to be decided by the developer and will differ per model. Coarsening is done *per model pair* since it is important to consider models on the most fine-grained level as possible because coarsening also leads to possible inconsistencies in the coarsened parts of the models.

The result of Step 2 is the following:

- All models are semantically homogeneous,
- Each model pair is time frame homogeneous, and
- Each model pair has the same level of granularity.

4.4 Phase II: Inter-model analysis

The goal of inter-model analysis is to identify dependencies *between* different models describing the same system, and to use these dependencies for explicating inter-model consistency constraints. These constraints are then checked and ensured during the management phase (cf. Figure 4.1). We first identify inter-model relations between the homogenized models (Step 3). Then we define inter-model consistency constraints based on identified relations (Step 4).

4.4.1 Step 3: Inter-model relation detection

Goal: Identify inter-model relations for each model pair. More specifically, we identify *dependencies* between models, concepts, and properties.

In Step 2 models are prepared per pair for comparison. The goal of Step 3 is to explicate inter-model relations. When considering partial models, models describing different system parts are interconnected. This interconnection needs to be identified. For example, if one partial model describes interactions on the customer side and another one describes the interactions on the provider side, these two models need to be interconnected before implementation. When considering viewpoint models *overlapping* parts between models (i.e., where two models describe the same system) need to be identified. For example, if one viewpoint model describes which monetary actions are expected in a cooperation and another one describes the order in which messages are exchanged between the actors in the cooperation, the monetary actions and messages that refer to the same real-life entity need to be related. These inter-model relations are used in Step 4 to formulate consistency constraints.

The term *relationship* has a broad meaning. Therefore, we start with explicating which types of relations are distinguished in this step. The goal is to ensure consistency between models which is accomplished by explicitly identifying those parts of the models that influence each other. We consider any type of *dependency* relation between concepts. This dependency can be strong as, for example, in a causal relation where one concept causes another one to occur. However, also less strong dependencies are considered. For example, in a temporal dependency one concept always occurs before another one without being its cause for occurrence.

Further, relations can be *symmetric*:

Example. In Figure 4.3, payment of a bike (Value viewpoint model) and its delivery (Message viewpoint model) have a symmetric relation since there is no payment without delivery and no delivery without payment.

Relations can be *asymmetric* as well. For example, regarding *payment* of a bike and its *bill*, the existence of the payment depends on the existence of a bill, whereas the bill can also exist without the existence of a payment.

We distinguish relations between *concepts* and those between *properties*. A dependency relation between concepts is referred to as *existence dependency*, while a dependency relation between properties is referred to as *property dependency*.

Existence and property dependencies. An *existence dependency* between concepts indicates that the *existence* of these concepts depends on each other.

Example. *Considering our running example (cf. Section 4.2), the existence of a concept referring to the payment of a mountain bike by a customer depends on the existence of a concept referring to the actual bill created by the company stating the total cost of the purchase. In other words, without a bill of the company the customer will not pay.*

A *property dependency* indicates the *value* of one property depends on the value of the other one:

Example. *The amount of money (represented as property value of concept mountain bike, cf. Figure 4.3) paid by the customer for a bike depends on the amount of money (represented as property value of concept bike parts) paid by the company to its suppliers for the bike parts. Here, the value property of one concept depends on the one of another concept.*

If properties of two concepts are dependent on each other, the concepts themselves have an existence dependency as well:

Example. *Since the properties of concepts mountain bike and bike parts depend on each other (cf. Figure 4.3), the two concepts are existence dependent on each other. The existence of concept mountain bike depends on the existence of concept bike parts since there exists no bike if there are no bike parts.*

In other words, if two values are dependent on each other the existence of concepts containing these values depends on each other.

Symmetric and asymmetric dependencies. A second characteristic of dependency is its *direction*. A dependency can be *asymmetric* where one concept depends on one or more other concepts (i.e., 1-to-n relation). To illustrate asymmetric dependencies, we use an additional model constructed from our running example:

Example. *To model how the costs of purchasing a bike from the company are divided, a cost dependency model is developed. The left part of Figure 4.4 depicts that purchase cost depends on the product that is purchased, the delivery cost, and additional service costs for fast or weekend delivery. The total cost to be paid by the customer for a bike depends on product cost (i.e., the bike), delivery cost, and possibly service cost for fast or special delivery. This is an asymmetric dependency relation.*

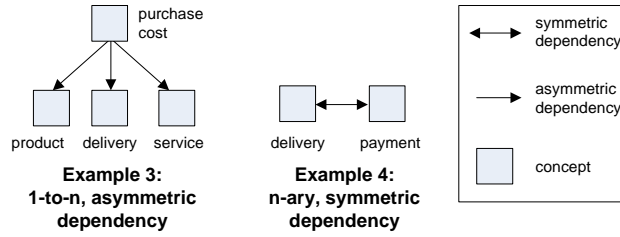


Figure 4.4: Example: Asymmetric and symmetric dependency relations

Alternatively, a dependency relation may be *symmetric* if concepts depend on each other (i.e., n-ary relation). To illustrate this dependency, we use an additional model constructed from our running example (cf. Section 4.2):

Example. To model a dependency between delivering the bike by the company to the customer and payment of the costs by the customer, another dependency model is developed in the right part of Figure 4.4. Delivery of the bike depends on payment of the cost, and vice versa. However, if it is not specified whether payment occurs before or after delivery, the payment depends on the delivery as well.

Both existence and property dependencies can be symmetric as well as asymmetric. In summary, we distinguish the following four types of dependency relations:

- Asymmetric existence dependency, where the existence of a concept depends on the existence of one or several other concepts,
- Asymmetric property dependency, where the property value depends on one or many other property values,
- Symmetric existence dependency, where the existence of concepts depend on each other,
- Symmetric property dependency, where property values depend on each other.

How to identify inter-model relations is explained next for viewpoint and partial models. Corresponding inter-model relations depict for viewpoint models *overlapping* parts, while for partial models they depict *connecting* parts.

Viewpoints. When identifying relations between two viewpoint models, the goal is to identify their *overlap*. We adopt the definition of overlap as given by Engels et al. [47]:

“Overlap is defined as two specifications which are not independent since they describe common aspects.”

Our aim is to find these common aspects and to identify the type of relation they have. In conceptual modelling the common aspects are concepts with potentially different properties referring to the same real-life entity. Preprocessing models in the homogenization phase (Step 2) eases identification of concepts referring to the same entity. Identifying the type of relation between these commonalities is done by hand:

Example. *The example in the left part of Figure 4.3 depicts a Value viewpoint model as well as a Message viewpoint model. Furthermore, it depicts the dependencies between the two models. In our example, we assume the model engineers refer to the bike sold to the customer with both concept bike in model A and concept delivery in model B. Furthermore, they refer to the money paid by the customer to the company with both concepts money in model A and payment in model B. In both cases the concepts refer to the same entity in the real world, and in both cases existence of one concept assumes existence of the other. For example, when a bike is delivered (Message viewpoint model), it is also paid for (Value viewpoint model). Therefore, the concepts have a symmetric existence dependency.*

Partial models. When identifying relations between partial models, the goal is to identify relations between concepts, referring to different real-life entities, and between the same properties in different concepts. This can be a more challenging task than identifying overlap in viewpoints since there are no commonalities to observe because each entity is modelled in only one model:

Example. *In the right part of Figure 4.3 one partial model depicts the mountain bike and the other one the bike parts. The two models refer to different entities, but there exist dependencies between the models. For example, the value of the bike depends on the value of its parts. Identifying these dependencies constitutes a challenge since there is no overlap between the models.*

Usually, there exists some (implicit) model that links the partial models. For example, when a complex cooperation between different systems is developed, there exists a common model that explains which partial model is connected to which other one and what this connection looks like. However, the challenge is that these are high level connections, describing the relation on model level rather than on concept level:

Example. *In the right part of Figure 4.3 the two partial models are related. There exists a common model that the mountain bike depends on its parts (i.e., a common model describing high level connections). The challenge, however, is to find the exact relations between these two models on a concept level. For example, what is the exact relation between delivery time of the bike parts and possible delivery time of the bike itself (i.e., identify the relations on a concept level).*

The approach taken in our MaDe4IC method is to identify different properties that are modelled for each real-life entity in a concept. This way, property dependencies are identified between the same properties in different, but related concepts. In partial models each

real-life entity is represented in one model as one concept with a set of properties. For example, a concept in a partial model might contain information on its monetary value, its size, and its validity. The concepts in related models are related through identifying equal properties. For example, the monetary value of one concept might depend on the monetary value of other concepts, while their physical size is unrelated:

Example. *In the right part of Figure 4.3 a schematic representation of identifying these relations is represented. The two partial models each describe one entity with different properties. The value of the mountain bike depends on the value of its parts. Also the possible delivery time of the bike depends on the delivery time of the parts. This asymmetric property dependency relation is depicted.*

The result of this step is the following:

- Identification of inter-model dependency relations between concepts and properties,
- Inter-model relations are categorized as follows (i) symmetric and asymmetric, and (ii) property and existence dependencies.

4.4.2 Step 4: Inter-model consistency constraints

Goal: Identify inter-model consistency constraints for each model pair. More specifically, we use the identified dependency relations from Step 3 to formulate constraints for each relation.

As discussed in Section 2.2, we consider two models being consistent if they have no *contradictions*. Since contradictions only occur in *related* models, they only emerge in *inter-dependent* model parts. In Step 3 we identify these inter-model dependencies. In Step 4 we now use them to formulate *consistency constraints*. These constraints are divided into different categories, in analogy to the different dependency relations introduced in Step 3. For each identified dependency, we formulate a consistency constraint. Each dependency constraint is formulated according to the related dependency type:

- If concept x is **asymmetric existence dependent** on set Y of one or more concepts, the constraint states: If x exists then Y exists.
- If concepts in set X of two or more concepts are **symmetric existence dependent** on each other, the corresponding constraint states: If $x \in X$ exists, all concepts in X exist.

We illustrate formulating consistency constraints for symmetric existence dependencies by means of an example:

Example. In the left part of Figure 4.3 the two viewpoint models have a symmetric existence dependency relation as discussed in Step 3. These relations are translated into consistency constraints. For example, concepts *bike* of Model A and *delivery* of Model B are symmetric existence dependent. The corresponding constraint states: If concept *bike* exists, concept *delivery* exists, and vice versa. This constraint depicts that if at runtime the bike gets delivered it also is paid for, and vice versa. If this is not the case, the constraint is violated.

In case of asymmetric and symmetric property dependencies there exists a relation between property *values*. The exact relation between the values is model-dependent. For example, property value of x might be *twice as big* as related property value of y . The general format for defining consistency constraints for asymmetric and symmetric property dependency is as follows:

- If concept x is **asymmetric property dependent** on set Y of one or more concepts, and z is the predicate describing this relation, the corresponding constraint states: Property value of x relates to property values of Y according to predicate z .
- If concepts in set X of two or more concepts are **symmetric property dependent** on each other, and z is the predicate describing this relation, the corresponding constraint states: For each $x \in X$ it holds that property value of x relates to all other concepts in X according to predicate z .

We illustrate formulating consistency constraints for asymmetric property dependencies by means of an example:

Example. In the right part of Figure 4.3 the two partial models have asymmetric property dependency relations as discussed in Step 3. Assume that the relation between value property of *Mountain bike* and value property of *Bike parts* states the value property of *Mountain bike* is always larger than the value property of *Bike parts*. Then, this relation is translated into the following consistency constraint: Concept *Mountain bike* is asymmetric property dependent on concept *Bike parts*, where the property value of *Mountain bike* is larger than the property value of *Bike parts* (i.e., this is the predicate). The corresponding constraint states: The property value of *Mountain bike* is larger than the property value of *Bike parts*. This constraint depicts at runtime the value of the bike is larger than the value of its parts. If this is not the case then the constraint is violated.

Often it is possible to formulate *generalized* consistency constraints where the constraints are defined over a *set* of inter-model dependencies. Since every identified dependency relation results in a consistency constraint, the number of constraints becomes very large. Therefore, it is more convenient to explicate one general consistency constraint that captures a set of similar single consistency constraints.

Example. For example, if hundreds of payments in one model are existence dependent on a bill in another model, each of these dependency relations results in a separate constraint stating “If payment A exists, also bill B exists”. However, a single constraint stating: “If a payment with characteristic Y exists, also a bill with characteristic Z exists”, would generalize over this large space of constraints.

This generalization process can be compared to the one used to solve granularity differences between models as discussed in Step 2. Generalization is possible under specific circumstances. Furthermore, it is applicable in both viewpoint models and partial models. For both types of models it holds that generalization is only possible over related combinations *with the same type of dependency relation* and, therefore, over consistency constraints of the same type. For example, it is not possible to define a general consistency constraint for two sets of related concepts where one set has an asymmetric existence relation and the other one a symmetric existence relation. Generalization constraints differ for viewpoint models and partial models. Therefore, we discuss them separately in the following paragraphs.

Viewpoint models. In viewpoint models it often occurs that a set of concepts in one model is related to concepts in another model. Currently, each relation between the two models has a separate constraint. However, it is often possible to create one *general* constraint that describes the complete set of constraints:

Example. Consider the left part of Figure 4.3 where two symmetric existence dependency relations are depicted. Figure 4.5 shows exactly these two relations. The value viewpoint model describes which products are transferred to the actors. The message viewpoint model, in turn, specifies which messages are exchanged between the actors. If there is a message indicating the bike is delivered or a payment is done (message viewpoint), it is concluded the bike or payment is received (value viewpoint), respectively. If the bike or money has exchanged hands (value viewpoint), it further is concluded there must be a message stating the bike is delivered or the money is paid (message viewpoint). Therefore, existence of a good transfer (e.g. a bike or money) and existence of a message exchange (e.g. a delivery message or payment message) depend on each other, i.e., they have a symmetric dependency. Without any generalization over constraints, there are now two consistency constraints describing the relation between the value and message viewpoint model. One states that if the bike is transferred also a message confirming the bike delivery exists, and vice versa. The second constraint states that if money for the bike is transferred, a message confirming the bike payment exists, and vice versa.

We construct a general consistency constraint that covers both single constraints by stating something in general about the relation between the value viewpoint model and the message viewpoint model. The general consistency constraint states, for example: “If a valuable product is handed over, a message confirming this exists, and vice versa”.

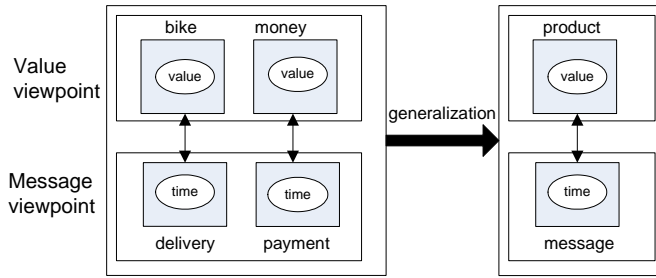


Figure 4.5: Possible generalization over symmetric consistency constraints

Generalization in viewpoint models is possible if there are several consistency constraints between two models that are based on the same type of relation (e.g. symmetric existence dependency). If generalization of property dependency constraints occurs, the properties in the different constraints is the same. For example, if one constraint describes a dependency between two time frames, while another one describes a dependency between monetary values, it is not possible to generalize over these two constraints.

Main advantage of a general consistency constraint between two models is the possibility to check whether they describe the same cooperation. The constraint typically describes the *nature* of the models. In this case, we state that every concept that represents some valuable transfer in the value viewpoint has a related concept in the message viewpoint describing the message transfer:

Example. In a value and message viewpoint, several entities in the cooperation need to be modelled in both viewpoints. For example, in the left part of Figure 4.3, money paid for a bike is modelled as valuable transfer between actors in the first viewpoint and as message transfer between two actors in the second viewpoint. The other way around does not hold since not every concept describing messages transferred between actors has a monetary value.

Therefore, we state that for each message transfer in the message viewpoint containing some value there is a concept representing this value in the value viewpoint model. With this consistency constraint we check whether all necessary concepts are modelled or whether a model is incomplete.

Aside from checking whether all concepts are present, it is important for each business transaction to check whether both models describe the same *set* of transfers. For example, if both viewpoints model entities “bike” and “money” for the bike, it is important that no viewpoint model assumes *two* bike transfers occur for one money transfer, while the other viewpoint assumes *one* bike transfer occurs for a money transfer. Therefore, an additional general consistency constraint is formulated describing that for each business transaction modelled in the viewpoint models, the concepts should occur in the same setting.

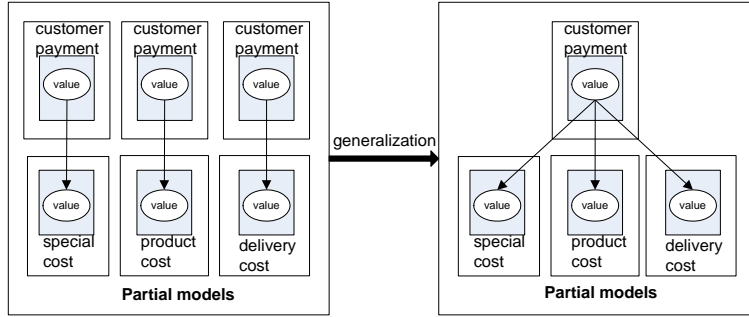


Figure 4.6: Possible generalization over asymmetric consistency constraints

Partial models. In partial models we often observe the property values of a set of concepts in different models are dependent on each other:

Example. In the left part of Figure 4.6 the concept customer payment has three dependencies: money paid by the customer depends on special costs, but also on product costs and delivery costs. Each property dependency relation has a separate constraint clarifying how customer payment depends on the different costs. For example, one constraint might state: “The amount of money paid by the customer contains special costs”.

However, it is possible to construct one general constraint that describes the three separate ones:

Example. The right part of Figure 4.6 depicts that customer payment and its three relations are interconnected. Now, the general constraint states: “The amount of money paid by the customer is at least the sum of special costs, product costs, and delivery costs.”

In general, we state that constructing a generalization of constraints in partial models is possible if (i) the dependency relations described by the constraints are interconnected, and (ii) they connect the same type of property. For asymmetric relations it should be possible to form a tree out of the dependency relations and for symmetric relations it should be possible to create an undirected graph from the dependency relations.

In both scenarios analyzed in this thesis (i.e., in the viewpoint model scenario in Chapter 5 and the partial model scenario in Chapter 7) we show in detail how to generalize sets of consistency constraints.

Result of Step 4 are as follows:

- Identification of consistency constraints for both symmetric and asymmetric dependencies, and property and existence dependencies, and
- Illustration of generalization of consistency constraints.

4.5 Phase III: Intra-model analysis

Intra-model analysis identifies dependencies between different concepts *within* one model (Step 5). Furthermore, we define intra-model consistency constraints based on these relations (Step 6).

4.5.1 Step 5: Intra-model relation detection

Goal: Identify intra-model relations for each model. More specifically, we identify *dependencies* between concepts and properties.

For detecting intra-model relations we use the original models and not the homogenized ones since we want to preserve as much original data as possible. Although conceptual models have in common that they describe *relations* between concepts and properties, the *type* of relation used differs over different models. In order to get a complete view on dependencies between concepts and properties over different models, it is important to analyze the type of relation used *within* each model. Therefore, we explicate the relation type used in the language for depicting relations between concepts and properties within a single model. We use the same distinction as discussed in Step 3. However, here dependencies between concepts and properties are detected within *one* model as opposed to inter-model relations in Step 3. More than one type of relationship can be used in a model.

In viewpoint models these dependency relations typically exist between concepts and properties referring to *different* entities, but describing the *same* property. For example, in the left part of Figure 4.3, concepts *bike* and *money* both have a value, while they describe different entities that depend on each other.

In partial models these dependency relations typically exist between *different* properties referring to the *same* entity. For example, in the right part of Figure 4.3 the concept *mountain bike* contains a property *value* that depends on the property value of *delivery time*, both properties refer to the same real-life mountain bike entity.

The results of this step are the following:

- Identification of intra-model dependency relations between concepts and properties,
- Dependency relations are categorized in (i) symmetric and asymmetric, and (ii) in property and existence dependencies.

4.5.2 Step 6: Intra-model consistency constraints

Goal: Identify intra-model consistency constraints for each model. More specifically, we use the intra-model dependency relations identified in Step 5 to formulate constraints for each relation.

Analogously to defining inter-model consistency constraints in Step 4, we define intra-model consistency constraints in Step 6. Note that these constraints still assume the models are built properly, i.e., that the models are well-formed. For example, if a modelling language prescribes every activity is followed by an arc, we assume this is the case. Intra-model constraints depict constraints on concepts, properties, and their relations within the model, not on the language used to describe the model. For example, there is no constraint stating that concepts can only be connected through a specific relation since this is a language constraint. However, there exist intra-model constraints stating that a specific concept shall be present if another one occurs. For example, if a payment is done, a bike needs to be shipped because this is a model-specific constraint (i.e., a specific constraint for our example).

Dependency relations identified between the different concepts in Step 5 are used to define the constraints. The same structure for building the constraints as in Step 4 is applied. The difference is that constraints are now defined *within* a model, and not *between* models.

In addition to consistency constraints, which are defined based on dependencies *between* concepts, the *existence* of the concepts itself, with or without a specific property value, is defined here. For example, if a concept represents money transfer from customer to company, this concept should also *occur* in the implementation, regardless of dependencies with other concepts. The general form of such a constraint is as follows:

- Let x be a concept with property y in a model. The corresponding consistency constraint states: x exists with property y .

For example, if concept *money transfer* with property value *100 euro* exists in a model, the constraint states this concept with property value needs to exist in the implementation.

Often it is possible to generalize over different consistency constraints as it is done for inter-model consistency constraints in Step 4. However, the approach for generalizing intra-model consistency constraints differs from the one used in Step 4. The same advantage for generalization of inter-model consistency constraints holds for intra-model consistency constraints. Namely, by generalizing over a set of constraints, the number of constraints to be checked reduces significantly, increasing applicability of our method. Next, we discuss for both viewpoint models and partial models which consistency constraints are generalized and how this is accomplished.

Viewpoint models. In Step 5 intra-model dependencies are identified. In this step, each of them is used to formulate a consistency constraint. As a result there are many similar consistency constraints. To illustrate generalization in viewpoint models, we consider an example:

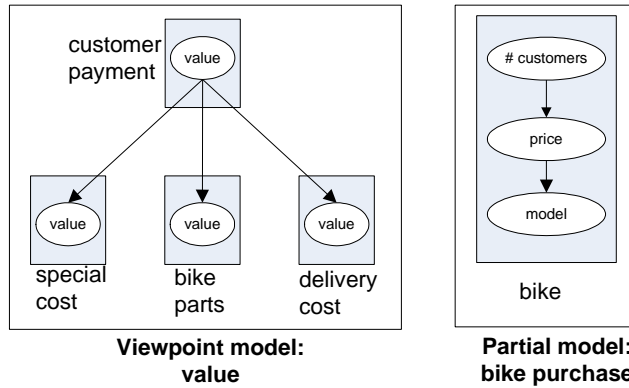


Figure 4.7: Possible generalization over intra-model consistency constraints

Example. The left part of Figure 4.7 depicts a viewpoint model that describes monetary values of different entities of our cooperation. For example, the bike parts have a certain monetary value and delivery costs are paid by the company to the delivery service. The properties of the concepts (i.e., monetary values) are dependent on each other. For example, the total price (i.e., property value) for the customer depends on how much the bike parts cost.

Each intra-model dependency relation results in a separate constraint. However, it is possible to generalize over these constraints under certain circumstances.

Generalization is done for constraints that depict the *same type of dependency*, and then only over constraints being *inter-connected* through the different concepts and properties. This is the same approach as taken for generalization of partial models for inter-model consistency constraints.

Example. There are three constraints formulated for the value Viewpoint model (cf. left part of Figure 4.7): 1) costs for the customer depend on special costs, 2) costs for the customer depend on bike part costs, and 3) costs for the customer depend on delivery costs. These constraints are generalized into one general constraint that states: Total costs for the customer depend on the sum of special costs, bike part costs, and delivery costs.

In general, we state that generalization in viewpoint models is possible if the dependency relations are *interconnected* and if they connect the same type of *property*. For asymmetric relations it should be possible to form a *tree* out of the dependency relations (as done in the example) and for symmetric relations it should be possible to create an *undirected graph* from the dependency relations.

Partial models. The dependencies in partial models identified in Step 5 typically exist between properties within one concept that refer to the same entity. To illustrate generalization in partial models, we consider the following example:

Example. The right part of Figure 4.7 depicts a partial model for the concept bike from our running example. The bike has three properties: expected number of customers, price, and model type (e.g. mountain bike). The number of customers depends on the price that needs to be paid for the bike and the price, in turn, depends on the model type. Each of these dependencies results in a consistency constraint. Generalization over these constraints is possible. In this case we develop one consistency constraint that captures both dependency relations, i.e., the relations between number of customers and price, and between price and model type. The separate constraints state: 1) For a mountain bike (i.e., a specific model) x euro are paid, and 2) If a customer pays x euro for a bike, y customers will buy one. The general constraint now states: For a mountain bike the price is x euro, and y customers buy the bike.

Generalization within partial models is possible over those constraints that describe relations between different properties of one concept (i.e., one entity) if these properties are inter-connected. For asymmetric relations like in our example, it should be possible to build a tree of the different properties and their relations. For symmetric relations, in turn, it should be possible to build an undirected graph of the different properties and their relations.

Joint constraints. Each model is built with a specific *purpose*. It prescribes (i) which entities should occur, (ii) which properties they should have, and (iii) in which way they should manifest in the cooperation. Typically these are *model generic* constraints where a model depicts, for example, (i) which transfers occur between actors in one business transaction, (ii) what the value of certain exchanges between actors is, or (iii) the order in which entities should occur in a business transaction. Model-specific consistency constraints are formulated taking the specific model purpose into account. It is not possible to formulate a general consistency constraint for this purpose since this is model-dependent. Generalization is often not necessary for these constraints since they are typically model-specific and therefore general.

The results of this step are the following:

- Identification of intra-model consistency constraints, and
- Illustration of generalization of consistency constraints.

4.6 Phase IV: Combined analysis

4.6.1 Step 7: Dependency analysis

Goal: Formalization of model parts that are checked for consistency, and formalization of the defined intra-model and inter-model consistency constraints. This formalization allows easy implementation for automatic consistency checking.

As described in Section 2.3.3, we *check* consistency between models after their development. Checking consistency is done by *testing* the models with some model checker, or by finding a *translation*. Since we not only check consistency between models, but also between models and running system, we choose to translate the models. Typically, models are translated into a semantically well-defined formalism, which allows for formal consistency checking. Either models are completely translated, or only their overlapping parts are translated. In the inter-model and intra-model analysis phase (Phase II and Phase III) we identified crucial parts for consistency checking, i.e., we identified their overlapping parts. Therefore, we choose to *partially translate* the models, i.e., only these parts that are crucial for consistency checking.

During runtime we check whether both intra-model and inter-model consistency constraints (Step 4 and Step 6) hold with regard to the running system. In this method we check these constraints by *translating* those model parts that influence consistency into a *language-independent*, formal notation. Consistency influencing parts of the models are identified in the inter-model and intra-model analysis phase where both model relations and consistency constraints are analyzed. In Step 7 we combine these results and represent both constraints and essential model elements in a formal model. Formalization of model parts and consistency constraints allow easy implementation. As explained at the beginning of this chapter, implementation facilitates automatic checking of the constraints at runtime in the Management Phase (cf. Steps 8 and 9).

To manage models of inter-organizational cooperations, we monitor several parts of the models. We monitor:

- concepts with their properties,
- relations between properties, and
- relations between concepts.

We monitor these items by checking consistency constraints as defined in Steps 4 and 6. These constraints depict dependencies between concepts and between properties. We check whether the constraints hold by comparing running system and models.

To allow *automatic monitoring* in the following steps, we formalize those parts of the models that are part of the consistency constraints, and the consistency constraints themselves so that they can be easily implemented. This formalization of model parts results in a *formal model*. We illustrate what is present in the formal models with the following examples:

Example. When considering the left part of Figure 4.3, the value viewpoint model consists of two concepts with a value property that have a symmetric dependency relation. Both concepts and their constraint (based on the dependency relation) are included in the formal model. Furthermore, from the message viewpoint model all concepts have some dependency relation and are, therefore, used in one or more constraints. As a result, all concepts in this model are part of the formal model. Also constraints describing dependency relations between concepts in the message viewpoint model are included in the formal model. This also holds for constraints describing symmetric relations between bike and delivery, and between money and payment, respectively.

Example. When considering the right part of Figure 4.3, both concepts mountain bike and bike parts are included in the formal model since both their properties have dependency relations. Furthermore, constraints describing the asymmetric property dependency relations between the two prices and the two delivery dates, respectively, are also included in the formal model. These are inter-model consistency constraints for the models. Also the intra-model consistency constraints (i.e., property dependency between price and delivery date of the bike and the bike parts, respectively) are part of the formal model.

Although many formalizations are possible, in this thesis we describe the use of *sets* and *graphs* to create the formal models. In Scenario 1 (cf. Chapter 5) we show how to apply sets, while in Scenario 2 (cf. Chapter 7) we show how to apply graphs for this purpose. However, many other formalizations are possible. The key is to *group* representations of concepts that belong to the same business activity. Using sets one instance is represented as a set, while using graphs one instance is represented as a graph. Consider the following example:

Example. In the left part of Figure 4.3 concepts bike and money are grouped since one purchase of a bike entails both the bike and money transfer. Furthermore, also the concepts order, confirm, send, deliver, and pay are grouped since these messages together make up one purchase of a bike. Regarding the right part of Figure 4.3, the value properties and the delivery time properties are grouped.

The results of Step 7 are the following:

- Formalization of model parts that are used in the consistency constraints, and
- Formalization of consistency constraints.

This formalization allows easy implementation as preparation for automatic monitoring of the models at runtime.

4.7 Phase V: Management phase

In the management phase the aim is to check whether the running system performs as prescribed in the models. In other words, we check consistency between event logs and formal models, i.e., the dependency models. To allow for such a comparison between models and event logs, we analyze event logs and create an *event log model* that allows easy comparison in Step 8. The result of this comparison is reflected in the *management models* where inconsistencies are depicted. Furthermore, in Step 9 the management models allow for *causal analysis* to find out why certain inconsistencies occur. Using this analysis, different solutions for handling inconsistencies are identified. However, as a consequence of applying these solutions new inconsistencies might be introduced. It is important to identify these consequences because we aim at *minimizing* the number of model changes when regaining consistency between models and running system.

4.7.1 Step 8: Log analysis

Goal: Abstract necessary information from the event logs to monitor the models and their constraints.

Although mining data from event logs does not constitute the focus of this thesis, we first explain how to abstract necessary information from them. Secondly, we explain how to compare this information with the dependency models constructed in Step 7.

In inter-organizational models conceptual structures are present that are not visible in the event logs. To check them for evidence of consistency between models and system behavior, we need to add this structural information to the event logs. Therefore, we suggest to reconstruct relations between the different event log entries. For example, for each entry in the event logs we need to recognize to which instance it belongs, i.e., to which other entries it is related and it depends on. For example, if a payment is done and afterwards stored as entry in the event log, it becomes necessary to identify for which service or product this payment is done.

Identifying these relations is a widely known problem for which different solutions exist: Many approaches aim at deriving this structural information from the event logs through mining techniques [3], while other approaches add structural information to the event logs when they are created [67, 96]. In this thesis, we do not describe data mining or event log structuring in detail. We discuss which information is necessary for our approach, rather than how to acquire it.

We illustrate this with a technique where *identifiers* are used to reconstruct model structures captured by the event logs. Note, we assume that the structure as it is described in the models is reflected in both implementation and event logs. Without this assumption a first step is to mine structure from the event logs [3]. Furthermore, we assume identifiers are added to an event log entry to provide information on its nature. For example, a *contract number* can be used to identify to which transaction an entry belongs, *log on* and *log off* messages can be used to identify separate transactions, and *timestamps* (e.g. “Thu, 17 July 2009 09:23:12”) can be used to identify in which order certain events occur.

For each inter-organizational cooperation it needs to be decided which identifiers can and need to be used.

Necessary information abstracted from the event logs is structured using *sets*. Event log entries belonging to one transaction are grouped into one set. Each entry is represented as a *tuple*. Typically, each tuple contains a timestamp, an issuer, a recipient, a unique name, and some property information. The resulting model contains all property information necessary for checking the constraints of the formal models. In other words, it contains all properties as they are present in the formal models (cf. Section 4.6, Step 7). In general, the event log model is constructed as follows:

1. Identify which property values should be present in the event log model using the dependency models.
2. Represent each entry in an event log as tuple containing a timestamp, issuer, recipient, unique name, and all property values.
3. Group tuples belonging to one transaction into sets using identifiers like log-on and log-off messages, and contract numbers.

After creating the event log model by abstracting necessary information from the event logs, we compare the runtime results with the inter-organizational cooperation models describing the system. The result from this comparison is represented in *management models*. When using sets to represent the formal models, consistency constraints are checked by *matching* tuples and sets between event log model and formal model. When using graphs, tuples and sets of the event log model are matched with vertices and edges of the graphs. Furthermore, additional consistency constraints are matched by checking occurrences of tuples, properties of tuples, and occurrences of sets in the event log model. For example, if a consistency constraint states that the presence of a concept (i.e., tuple) indicates the presence of another one (i.e., tuple), this relation is checked in the different sets of the event log model.

Example. *In our running example (cf. Section 4.2) the company offers two business transactions: purchasing mountain bikes and purchasing city bikes. At runtime, we collect information from the event logs to support the business model of the company. For example, we expect evidence of selling mountain bikes to customers, represented as sets of events representing the sale. In addition, we expect evidence of selling city bikes in a similar manner. Furthermore, in order to sell the bikes, the company needs to purchase its parts before. This constraint is checked during runtime. We expect evidence to support the constraint that bike parts are purchased before bikes are sold, represented as events with different timestamps. More specifically, we expect events, represented as tuples in the event log model, that represent purchasing bike parts with earlier timestamps compared to events that represent selling bikes.*

Violation of consistency constraints is different for existence dependencies and for property dependencies. When a constraint describing an *existence dependency* is violated, this is a violation where the concept does not exist. For example, a bike needs to be paid

by the customer if he receives the bike. If there is no evidence of payment in the event log, this is a complete violation of the constraint. If a constraint describing *property dependency* is violated, this violation can come in gradations, depending on the scale of the value of the concept. For example, if the selling price of a bike is twice the amount paid for the bike parts, the runtime result in the event log model deviates (but does not completely violate) if the selling price is only one and a half times the amount paid for the bike parts.

The results of this step are the following:

- A description of information that needs to be abstracted from the event logs for monitoring the models and their constraints.

4.7.2 Step 9: Causal analysis

Goal: Identification of causes for constraint violations, and identification of consequences of restoring consistency between models and running system.

The monitoring results containing constraint violations need to be presented in an intuitive way. We suggest to represent monitoring results by showing deviations in *color codings*. For example, *red* indicates violation of an existence dependency constraint or violation of a property dependency constraint, *green* indicates compliance of the constraint, while *orange* indicates a deviation of a property dependency constraint with not more than 10%. The coloring can be done in the original models as well as in the dependency models. In this way, both arrows (indicating dependencies) and concepts (containing property values) are colored. The results are management models that show relations between different concepts, whether these relations are violated or not during runtime, and whether their property values are accomplished.

When the management models are created, it becomes clear which parts of the running system comply with the models and which parts do not. Ideally, models and running system are consistent with each other. If there is a violation the analyst can either evolve the *models* so that they properly represent the running system, or adapt the *running system* so that it properly reflects the agreed upon models. In this thesis we focus on changing the models so that they reflect the running system, rather than adapting the system itself.

Violations are often related. A constraint violation in one part of the model often results in violations of other parts of the model. These causal relations are important to identify for efficient model management since solving a constraint violation of the source might solve numerous other violations at the same time. The dependency models created in Step 7 show all dependencies between concepts. Here, dependencies are used to go through the management model and to identify sources for violations. For this causal analysis *only asymmetric dependencies* are used because in those relations it is clear which concept influenced the other. In symmetric dependencies you might travel to the end leafs of the problem instead of identifying the source of the violations.

Example. *If there is an existence constraint violation of concept money in the Value viewpoint model A (cf. left part Figure 4.3) (i.e., there is no evidence that money has been paid), the symmetric dependency relations the concept has make it difficult to determine the cause for this violation. For example, the symmetric relation between concepts bike and money does not indicate whether the money is not paid because the bike is not delivered, or the other way around. In other words, determining causes in symmetric relations is difficult.*

By identifying causes, it becomes possible to decide which parts of the models to *change* to regain consistency. After choosing these parts, it is important to identify what the *consequences* of these changes are. For example, when making a product more expensive to resolve a lack of income, the number of expected customers might decrease, leading to even less income. Therefore, besides identifying *what the causes* for violation are, it is also important to decide *which parts* to change.

Every element present in the consistency constraints is able to cause an inconsistency and is, therefore, also able to regain consistency. Often, there are several different ways to make a change in one of the models to regain consistency. For example, if the *bike parts* are at runtime more expensive from that agreed upon in the models, one solution is to negotiate a lower price for the parts, while another possibility is to change provider and purchase the bike parts elsewhere. Although both solutions solve the constraint violation, the first one is less intrusive for the model than the second one, since the second solution results in deletion of an offered service.

To distinguish intrusive from less intrusive changes, we suggest to divide possible model changes into different categories. Each constraint violation is now solved by applying a subset of these categories. Each category has its own consequences for the models.

- *Non-observable changes* in a model do not influence the formal model, i.e., the change does not influence the dependency models. As a consequence, these changes are outside the parts that influence consistency. For example, in the right part of Figure 4.3 the partial models describe the mountain bike and its parts. If the payment method changes, this does not affect these models since payments are not captured. Therefore, this is a non-observable change in that model.
- *Observable structural changes* in a model are changes where concepts are removed or added to the model while preserving a well-formed model. These changes influence *existence dependencies* since these dependencies rely on the existence of certain concepts. By removing or adding a concept, its existence changes. However, *property dependencies* are also influenced since the non-existence of a concept will also imply its property value does not exist. For example, if we remove the concept *bike* in the left part of Figure 4.3, this is an observable structural change. It affects the existence dependencies with concept *money* of the same viewpoint model, and with concept *delivery* of the message viewpoint model B.
- *Observable non-structural changes* are changes where the property value of a concept is affected, or where the way two concepts are related is affected. For example,

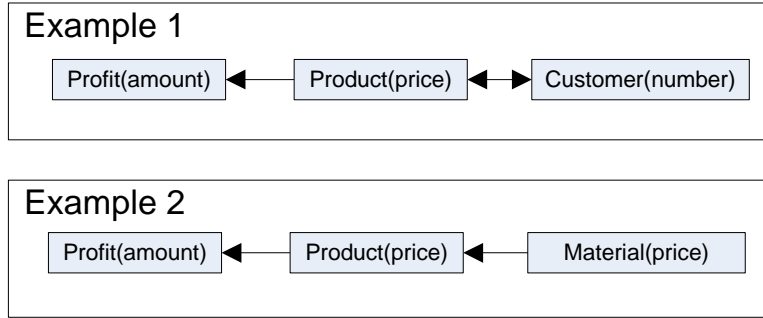


Figure 4.8: Consequence analysis for changing models

when changing the order of two concepts, their relation is affected. These changes do not affect constraints on existence dependencies since such it does not change the existence of concepts. However, it does affect constraints on property dependencies since it changes property values. For example, if the *value* property of concept *bike parts* in the right part of Figure 4.3 becomes larger, i.e., the bike parts become more expensive, then this is an observable non-structural change. It affects also the *value* property of concept *mountain bike* since the price of the mountain bike depends on its parts.

Explicating for each constraint which changes can be used to regain consistency if necessary is a step towards more efficient and precise model management. By relating changes to constraints, it is possible to predict the impact of a change, not only on the violated constraint, but also on other constraints that are related to this particular change.

With this last step we conclude the analysis of the models and event logs. Now, it is possible to *monitor* consistency constraints between models and between models and the running system, it is possible to identify *causes* for violations with a causal analysis through the different dependency relations. Based on the causal analysis, violated constraints are selected for inconsistency resolution. Now, we describe the last step where we use the identified possible changes and the dependency relations to predict the *consequences* of changes made in the models to regain consistency.

Each suggested change in a concept to regain consistency might affect more than the violated constraint. By considering the dependency models we identify which constraints are affected when changing a particular concept. An observable structural change affects both the existence and property dependencies, while an observable non-structural change only affects related property dependencies. When the affected dependency is symmetric, the change also affects the related concept. When the affected dependency is asymmetric, the change only affects the related concept if the related concept depends on the original concept, not if the original concept depends on the related concept. Consider Figure 4.8 where both situations are depicted:

Example. Both Example 1 and Example 2 depict (part of) a property dependency model where the dependency constraint between product price and amount of profit is violated. In other words, the event logs show that the depicted property relation between the product and profit does not hold. The model can be changed in several ways to solve this inconsistency. The amount of profit can be adjusted, the product price can be adapted, or the ratio between profit and price can be adjusted. Assume the developer considers changing the product price. Now, the consequences for the other dependencies are analyzed. In Example 1, product price and number of customers are dependent on each other. Therefore, if the product price changes, the number of customers is also affected, just as their dependency relation. In Example 2, product price depends on the material price it is made of. Now, changing the product price does affect their relation, but not the material price. The developer now concludes changing the product price in the second example has limited consequences, while changing the product price in the first example leads to more adjustments.

Especially with large models and many dependencies it is very useful to enable such an analysis where changes are related to types of dependencies, and analysis is done for the consequences of such changes in the rest of the models, but also for constraints it has with other models. The approach aims at using the constraints, dependencies, and types of changes to show for a chosen adaptation of the model, which parts are affected by the specific change. As a result the developer is better able to estimate the amount of effort needed to adapt the models as well as better able to make a choice between the different change possibilities. In general, the most suitable change is the one having the least impact on other constraints.

The results of this step are the following:

- We provide a method for causal analysis of the dependency models created in Step 7. This analysis identifies possible causes for constraint violations.
- Furthermore, we provide a method to predict consequences model changes have on consistency constraints. This supports the analyst in identifying the least intrusive adaptation of the model to regain consistency.

4.8 Summary

In this chapter we introduce a stepwise method towards efficient model management of inter-organizational cooperations. The approach relies on the identification of different types of dependencies between the different concepts within and between models. Furthermore, we suggest translating relevant model parts into formal models for easy consistency checking. The same approach as used for these formalizations is used for abstracting useful information from event logs into an event log model such that runtime behavior of the system can be compared with the models describing the system. Furthermore, an

additional causal analysis, using identified dependencies, allows identifying causes for inconsistencies between models and running system. As a last option our MaDe4IC method provides an approach to check the consequences for consistency constraints when changes are made to a model. In other words, we analyze what the consequences are for other consistency constraints when trying to resolve an inconsistency. As a whole, this method presents a stepwise, structured approach into managing these complex constellations in an effective and efficient way. In the following chapters two scenarios are discussed where this method is applied.

Part III

SCENARIO 1: BUSINESS & COORDINATION MODELS

MANAGING DEPENDENCY RELATIONS: BUSINESS AND COORDINATION MODELS

To illustrate the importance of inter-model consistency, we consider two fundamental perspectives which are of high relevance for modelling inter-organizational cooperations: the *business* and the *process perspective* [56, 2]. At business level expectations (e.g., agreements on the number of transferred products) between partners are modelled. At process level coordination of inter-organizational processes is modelled. Both perspectives describe necessary transfers between partners although focusing on different aspects (financial versus coordination). Assume, for example, that a payment is captured in the business model, while it is omitted in the process model. Then the two models are considered to be *inconsistent* with each other. If a system implementation is based on these models, payment will be expected to occur (due to the business model), while the occurrence of this payment is not prescribed in the implementation (due to the incomplete process model). In other words, business and process model do not describe the same system. Since the two models have a different level of abstraction, use different modelling notations, and have a different purpose, determining consistency constitutes a big challenge. This chapter illustrates how to apply our MaDe4IC method for identifying dependencies between business and process models [17, 26].

In complex, inter-organizational cooperations where repetitions within business transactions occur, *relations* between sets as well as between elements in sets need to be checked. In this case it should be checked whether certain transfers are executed an equal number of times (e.g., are as many payments done by the customer as services offered by the company?). Since the goal of this chapter is to illustrate usability of our method to business and process models we exclude repetitions for the sake of readability. However, a formalization of consistency constraints including such repetitions can be found in a technical report extending this chapter [20].

The remainder of this chapter first discusses *basics* of the modelling techniques, while introducing our running example in Section 5.1. In the following sections we discuss the stepwise application of our MaDe4IC *method* to this running example (cf. Sections 5.2 - 5.10). We conclude this chapter with a discussion of *related work* on business and process modelling in Section 5.11.

5.1 Basics

We first introduce an illustrating business case. It consists of a *copier company* which *sells* and *leases* copiers to customer companies. When leasing a copier, it is mandatory to purchase *maintenance* on a yearly basis. Before implementing the business case, the copier company wants to evaluate financial consequences (business perspective) as well as coordination requirements (process perspective). For this purpose, it develops a *value model* denoting value exchanges as well as a *coordination model* describing how the interaction between the two partners is arranged. To validate its models, information on interactions with partners is gathered from the *event log*.

5.1.1 Business perspective

The copier company reasons about *value transfers* to and from companies to *estimate* financial benefits. The *value model* (cf. Figure 5.1) depicts estimations on who gets what from whom and for how much in e^3 -value notation [57].¹ The e^3 -value methodology is developed to support exploration of innovative e-business ideas using a graphical modelling tool [38].

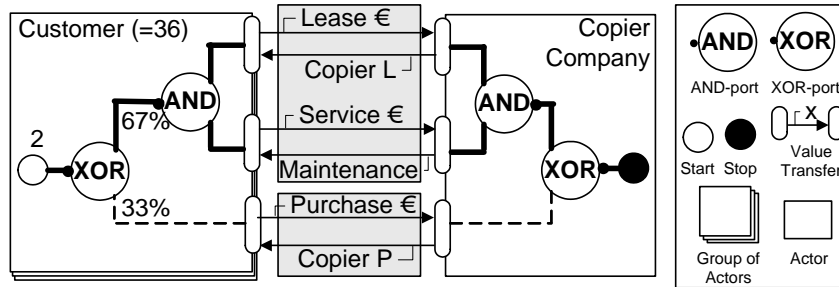
In our running example, actor *copier company* has a group of *customer* companies, and three kinds of *value exchanges* take place between them (cf. Figure 5.1): money for leasing a copier (*Lease, Copier L*), money for maintaining the copier (*Service, Maintenance*), and money for purchasing a copier (*Purchase, Copier P*). Interdependent value transfers (i.e., transfers exchanged in one business transaction) are connected through dotted and solid lines in Figure 5.1, representing two possible *business transactions*. One is highlighted through a thick line representing the *customer need* for having a copier, starting at the customer side. Furthermore, value exchanges that belong to one business transaction are grouped in a grey box. The *XOR-split* models that customers have a *choice* to either lease or purchase the product. The *AND-split*, in turn, indicates that for every lease a maintenance contract is purchased.

To obtain financial estimations, several quantifications are done for a specified *time frame*. In Figure 5.1 estimations on the number of customers (=36), the number of customer needs (=2), and the purchase-lease ratio (33%-67%) are made. These quantifications result in an estimation on the *number* of leases and purchases. Together with an *average value* of each transfer, this gives an indication on the income for this business activity in the specified time frame (one year in our case). Although these quantifications are part of the value model, for the sake of clarification, we represent them in Table 5.1.

5.1.2 Process perspective

Besides financial validation the copier company needs to agree on *how* to implement the business. For example, should the customer pay before receiving the copier, or the other

¹Other value-based modelling techniques (e.g. REA [76] and Business Modelling Ontology [92]) can be used as well. We select e^3 -value in this thesis due to its graphical notation.

Figure 5.1: Example case: Business model (e³-value notation)

Value Transfer	Average Value	Occurrences
Lease /Copier L	1200 €	48
Service /Maintenance	700 €	48
Purchase /Copier P	7500 €	24

Table 5.1: Estimations value model

way around? The coordination model describes which messages are to be exchanged between partners and in which order this message exchange shall take place. Such an *inter-organizational* process model provides the basis for any implementation. We use the Business Process Modeling Notation (BPMN) [113] to represent the coordination model (cf. Figure 5.2).² The customer *chooses* to purchase or lease a copier (cf. the *decision* in the figure). As a consequence of this choice, either the set of message exchanges associated with purchasing or leasing a copier is initiated. Both business transactions are indicated with a grey box, grouping messages for purchasing or leasing a copier, respectively (cf. Figure 5.2). The first message is sent by the customer as a *request message* to the copier company. The company, in turn, sends back an *offer message with the contract*. Finally, the customer *pays* and afterwards *receives the copier*. After this, the process *ends*.

5.1.3 Event logs

To evaluate the operational system, data on its execution is gathered. The event log of the Information System (IS) contains such data. Here, we focus on data exchanged *between* actors, and disregard any internal data. Furthermore, *timestamps* show the order in which data are exchanged. The event log enables traceability of executed processes during co-operation. Furthermore, it enables checking whether profitability estimates made in the

²Other modelling techniques like Activity Diagrams and Petri Nets are applicable as well. Here we select BPMN due to its graphical notation.

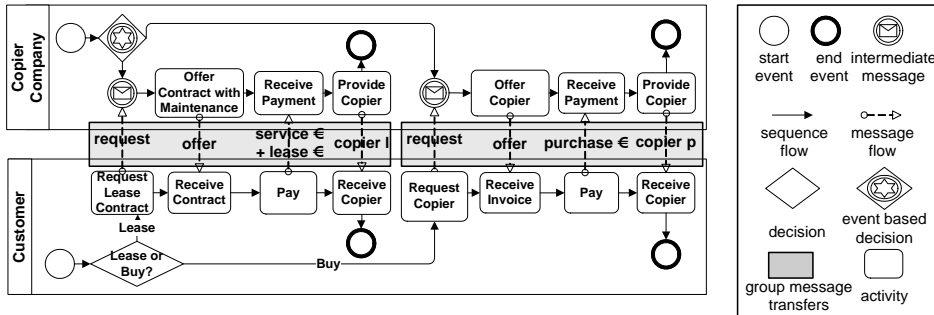


Figure 5.2: Example case: Coordination model (BPMN notation)

State	Time	Sender	Receiver	Message
Done	2007-08-17 09:13:33	customer_a	copier_company	request
Done	2007-08-17 09:15:30	copier_company	customer_a	offer
Done	2007-08-17 09:23:12	customer_a	copier_company	copier_payment
Done	2007-08-17 09:25:14	copier_company	customer_a	copier

```

Date: Fri, 17 Aug 2007 09:23:12
Sender: customer_a
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header />
  <soapenv:Body>
    <copier_payment xmlns="http://www.utwente.nl/consistency">
      <Process_payment>
        <Good>Service</Good>
        <Amount>700</Amount>
      </Process_payment>
      <Process_payment>
        <Good>Lease</Good>
        <Amount>1200</Amount>
      </Process_payment>
      <contract-number>NL-TWENTE-98267854</contract-number>
    </copier_payment>
  </soapenv:Body>
</soapenv:Envelope>

```

Figure 5.3: Example case: Event log (XML notation)

value model are realized. As example take Figure 5.3 in which parts of an XML-based event log (i.e., one business transaction) are shown. It depicts data exchanged between customer and copier company of payment for leasing a copier. Each message is annotated with a *timestamp*, *issuer*, *recipient*, and *name*. A message contains information on the value of a transfer (cf. *Amount* in Figure 5.3), the type of the transfer (cf. *Good* in Figure 5.3), and a *contract number*. Messages with the same contract number belong to one business transaction, while one specific customer might be involved in multiple business transactions.

Next, we discuss how to apply our MaDe4IC method for managing model dependencies in inter-organizational cooperations (cf. Chapter 4) to this running example.

	Value model	Coordination model
Focus	viewpoint	viewpoint
Perspective	bird's eye	single actor
Property type	estimations & prescriptions	prescriptions
Time frame	time period	instance

5.2 Step 1: Model analysis

In Step 1 of the method we analyze the models for their characteristics. Please, recall the different characteristics from Section 4.3:

- | | | | | |
|-------|----------------|----------------|---|-----------------|
| (i) | Focus: | Viewpoint | ↔ | Partial model |
| (ii) | Perspective: | Single actor | ↔ | Bird's eye view |
| (iii) | Property type: | Estimation | ↔ | Prescription |
| (iv) | Time frame: | Instance-based | ↔ | Period of time |

Value model. The value model from Figure 5.1 is a *viewpoint model*. It models the complete cooperation and focusses on one specific aspect, namely, value exchanges between actors. The value model is developed with a *bird's eye perspective*, i.e., it is developed for all actors, and captures their value exchanges for a specified period of time. The value model contains both *estimations* and *prescriptions*. For example, the ratio between purchased and leased copiers constitutes an estimated value, while the price of a purchased copier is fixed. The value model is developed for a *period of time*, in this case for one year.

Coordination model. The coordination model constitutes a *viewpoint model* as well. It models the complete cooperation while focussing on one aspect, namely, message exchanges between actors. The coordination model is developed from a *single actor perspective* (the copier company). This means the coordination model depicts the interaction of the copier company with its customers. However, the coordination model only depicts one customer that represents the average behavior of a typical customer. All exchanges between the actors are *prescriptions*; messages need to be exchanged in the specified order. The coordination model is *instance-based*. In the year described in the value model, the coordination model is expected to execute several times.

5.3 Step 2: Homogenization

In Step 2 we homogenize the models on syntactic, semantic, and pragmatic level so they can be compared.

Syntactic homogenization. As discussed in Section 4.3, it is not possible to homogenize syntactic differences. However, respective differences and correspondences must be identified to enable comparison.

- *Actor versus Swim lane.* Our value model uses actors and groups of actors to depict partners in the cooperation. This corresponds to swim lanes in the coordination model.
- *XOR versus Decision.* The value model uses XOR-splits to indicate choices in business transactions. In other words, either the one or the other business transaction is chosen. This corresponds to the decision (i.e., lease or buy a copier) represented in the coordination model, where also a choice is made between two business transactions.
- *AND.* The AND-split in the value model does not have a matching concept in the coordination model. It indicates grouping of value transfers that are exchanged in one business transaction between actors. In the given coordination model this relation is implicitly modelled in the sequence of messages that connects different messages being exchanged in one business transaction.
- *Value transfer versus Message transfer.* A value transfer in the value model represents the transfer of a valuable object from one partner to the other. Each value transfer corresponds to one or more message transfers in the coordination model since each value transfer also results in a message transfer in the given scenario. For example, the physical transfer of money for leasing a copier in the value model (cf. *lease* € in Figure 5.1), results in a message transfer confirming this payment (cf. *lease* € in Figure 5.2). However, there might be message transfers that do not have any economical value and, therefore, do not have a correspondence in the value model.

Semantic homogenization. The models in this chapter are built in such a way that they are semantically homogeneous by construction. For the interested reader, there exists an approach, designed by Zlatev et al. [120], to manage semantical differences between value and coordination models explicitly; Zlatev et al. also discusses granularity differences.

Pragmatic homogenization. Concerning pragmatic homogenization we consider five different aspects. Some of them are homogenized, while others are identified in the models (cf. Section 4.3, Step 2).

- *Focus.* The focus of both value and coordination model is a viewpoint one. Therefore, the models are homogeneous concerning their focus.
- *Perspective.* The perspectives of the models are different in the given case. The value model has a bird's eye perspective, while the coordination model has a single actor perspective. This is a difference which we cannot homogenize since it is a

model characteristic. Therefore, we identify the difference in this step and deal with the heterogeneity in the management phase (cf. Section 5.9). Here, heterogeneity does not influence consistency in the following steps. For the interested reader, Bodenstaff et al. [21] discusses several examples where these differences influence consistency maintenance in value and coordination modelling.

- *Granularity.* The models are defined at the same level of granularity.
- *Time frame.* The time frame of the models is different. The value model considers a period of one year, while the coordination model is instance-based. As discussed in Section 4.3, this cannot be homogenized. Therefore, we handle this heterogeneity in later steps by comparing coordination and value models on a business transaction level (i.e., at an instance level). At runtime, the value model is considered for a period of time decided upon by the manager when comparing it with the event logs.
- *Property type.* The estimations modelled in the value model do not have a counterpart that is prescribed in the coordination model. For example, the estimated ratio between purchase and lease copiers is not present in the coordination model since the latter is an instance-based model. Therefore, these differences do not cause any problems in maintaining consistency, and are simply checked for consistency with the event log.

The main results of Step 2 are as follows:

- Non-valuable coordination model messages do not have a counterpart in the value model.
- Differences in perspective are identified.
- To overcome time frame differences, consistency checks between value and coordination model need to be performed at an instance level.
- Estimations in the value model are checked during runtime with the event log.

5.4 Step 3: Inter-model relation detection

In Steps 1 and 2 we analyze the value and coordination models to prepare them for comparison. In Step 3 we start with detecting relations between the given value and coordination model. These relations enable us to define consistency constraints between the models in Step 4.

To define these constraints for viewpoint models, first, their overlap needs to be identified. As discussed in Chapter 4 this means we first identify those parts value and coordination models have in common. We focus on the communication between actors and do not consider internal behavior. Therefore, we identify entities in the real world that

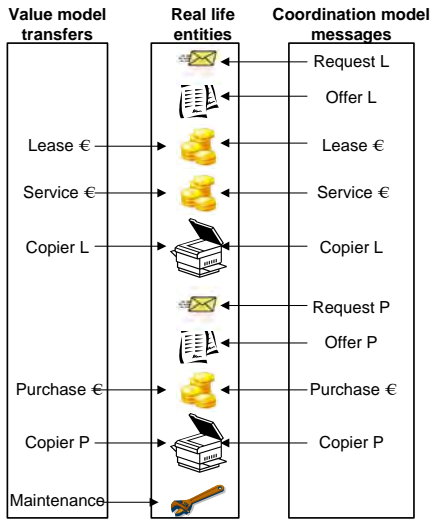


Figure 5.4: Relations between models and real-life entities

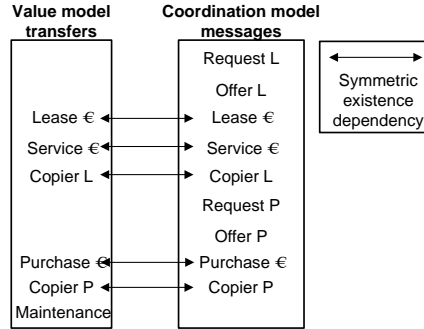


Figure 5.5: Inter-model dependency relations

are exchanged between the actors and that are modelled in both value and coordination model. Figure 5.4 depicts these entities and shows which value and message transfers refer to them. For example, the value model depicts the transfer of money paid for leasing a copier, and the coordination model depicts a message transfer representing leasing a copier as well, i.e., both concepts refer to *money paid for a lease*. Further, *money for service on the copier*, *money for purchasing a copier*, *lease copiers*, and *purchase copiers* are modelled as concepts in both value and coordination model. *Requests* and *contracts* are only modelled in the coordination model, while *maintenance* is only modelled in the value model. Concepts referring to the same entity constitute overlapping parts between value and coordination models. Consequently, they are not independent.

After identifying the overlap between the two models with respect to the concepts, we analyze the *type* of relation they have. In this case, if a dependent concept (e.g. a lease copier) in the value model exists, the related concept (e.g. a lease copier) in the coordination model exists as well, and vice versa. There is no property dependency since concepts in the value model describe the property *value*, and concepts in the coordination model do not contain properties. Therefore, the value of a concept in the value model is not dependent on a message in the coordination model. As a result, overlapping parts have a *symmetric existence* dependency as depicted in Figure 5.5.

The main result of Step 3 is as follows:

- Identification of symmetric existence dependencies between transfers in value and coordination model.

5.5 Step 4: Inter-model consistency constraints

In Step 3 we identify inter-model relations. In Step 4, we now use these symmetric existence dependency relations to define inter-model consistency constraints.

In general, we consider two models as being consistent if they are contradiction-free. We consider value and coordination models to be consistent if they facilitate the same business transactions (e.g., one option for purchase and one for leasing a product). Furthermore, each business transaction needs to be facilitated in the same way (e.g., purchase over the internet versus purchase in a store). Therefore, the consistency constraint defined between value and coordination models should describe their relation on business transaction level and on transfer level.

5.5.1 Transfer level

The result of Step 3 is a set of interrelated concepts having a symmetric existence dependency (cf. Figure 5.5 where these relations are depicted). According to Step 4 of our method (see Section 4.4.2) every relation is translated into a corresponding constraint:

If concepts in set X of two or more concepts are **symmetric existence dependent** on each other, the corresponding constraint states: If $x \in X$ exists, all concepts in X exist.

Here, each dependency is bilateral, i.e., set X contains two concepts. For example, for the symmetric existence dependency $Lease \in$ in the value model and $Lease \in$ in the coordination model (cf. Figure 5.5), the consistency constraint is formulated according to the above definition:

If and only if a concept for a lease payment (i.e., $Lease \in$) exists in the value model, also a concept for a lease payment (i.e., $Lease \in$) exists in the coordination model.

Since each of these consistency constraints describes a symmetric existence dependency relation between a concept with a value property in the value model and a concept describing a message transfer in the coordination model, it becomes possible to *generalize* over these constraints (cf. Step 4, Section 4.4.2). Here, the common denominator is that for each concept having a value property in the value model there should be some message transfer in the coordination model. However, not every message transfer in the coordination model results in a value transfer in the value model. For example, making a purchase or lease *request* is not associated with any value. In other words, the real-life entity to which the message refers (i.e., the email or web form request sent by the customer) does not have any monetary value. Therefore, we also state that for every message transfer in the coordination model that refers to an entity *with a value property* (i.e., a valuable message) there should be a value transfer in the value model as well.

5.5.2 Business transaction

There are two possible *business transactions* in the models, i.e., a copier is *purchased* or *leased*. Both transactions are associated with a set of value and message transfers that realize the business transaction.

In addition, we check whether the *direction* in which transfers occur is the same in both models. For example, if the value model describes money is paid by actor A to actor B, while the coordination model describes the matching message transfer indicating money is paid by actor B to actor A instead, these two models are not consistent with each other. Therefore, an additional constraint on each transfer is that *issuer* and *recipient* of the transfers should match. First, we state when we consider two sets to match:

Definition 1 (Match between value and coordination model). *Sets match if*

- *for each value transfer in the value model there exists a message transfer with matching issuer and recipient in the coordination model, and*
- *for each valuable message in the coordination model there exists a value transfer with matching issuer and recipient in the value model.*

We use the definition for *matching sets* to formulate our *general consistency constraint*:

Constraint 1 (Value and coordination model). *(1) Each set of value transfers describing a business transaction in the value model has exactly one matching set (cf. Definition 1) of message transfers representing a business transaction in the coordination model, and (2) vice versa.*

The set representing the business transaction of *purchasing a copier* in the value model, and the set representing this transaction in the coordination model of our running example, match. Also the elements in these sets match. For example, value transfer *Purchase* matches message transfer *Purchase* since both have the same issuer, recipient and name.

Also the set representing the business transaction of *leasing a copier* in the value model matches the set representing this transaction in the coordination model. However, the *elements* in these two sets do not match. More specifically, value transfer *Maintenance* should have a matching message transfer in the coordination model set.

This matching problem occurs due to the different *purpose* of value and coordination model. A value model captures everything that is of value for the actors. In this case, *Maintenance* (cf. Figure 5.1) is something of value for the customer. However, he only receives maintenance during *execution* of the contract, not when the contract is established. In the coordination model only the establishment of the lease contract is depicted, not how the contract is executed. Therefore, the coordination model does not show the actual maintenance of the copier. As a result, we *detect* this difference, but do not adapt the models. Therefore, we consider the value model from Figure 5.1 to be consistent with the coordination model from Figure 5.2 (i.e., Constraint 1 is met).

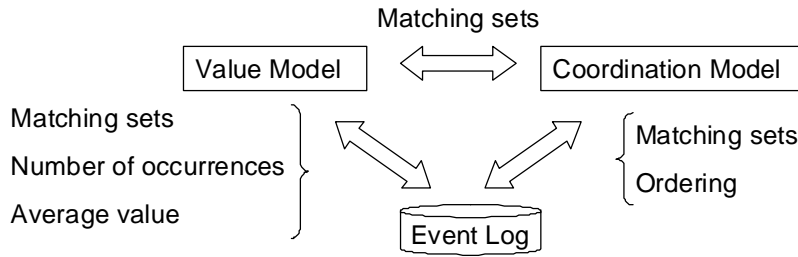


Figure 5.6: Constraints between models and event log

As discussed at the beginning of this chapter, we do not show how to check consistency for models that contain *loops* or *back edges*. Rather, we give a short impression on how to handle such cyclic structures. An overview is given in Figure 5.6 where consistency relations between value and coordination model are depicted. Aside from checking *matching sets*, *average values* in value models, and *order* of messages in coordination models, additional consistency constraints are required. In particular, constraints that check for *matching number of occurrences* of transfers, and for *co-occurrence* of business transactions are required. These constraints are checked by comparing models and event logs (cf. Figure 5.6). For example, when both payment of a product and its delivery are influenced by the same loop, it can be expected that both transfers occur the same number of times and this should be facilitated by both value and coordination model. Furthermore, if it is possible to order and purchase more than one copier in a single order, this results in a co-occurrence of business transactions. This co-occurrence should be facilitated in both value and coordination model (cf. Figure 5.6, *set matching* between value and coordination model). For the interested reader, a complete discussion on how to formulate such consistency constraints can be found in Bodestaff et al. [20].

The main results of Step 4 are as follows:

- Definition of a consistency constraint between value and coordination model (cf. Constraint 1).
- Consistency checking between value and coordination model of our running example.

5.6 Step 5: Intra-model relation detection

In Step 3 we analyze inter-model relations to create a basis for defining consistency constraints between value and coordination model. In Step 5 we now analyze existing intra-model dependencies between concepts. This enables us to define in Step 6 intra-model consistency constraints.

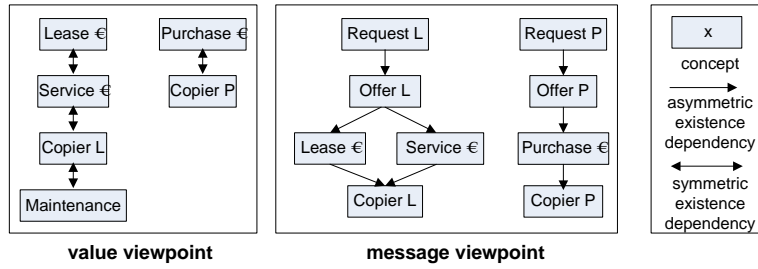


Figure 5.7: Example case: Intra-model dependencies

5.6.1 Value model

For the value model it holds that the different concepts are connected through dotted and solid lines (cf. Figure 5.1). Both dotted and solid line depict one business transaction. Each business transaction involves *all* connected transfers. The latter means that if one transfer occurs, all connected transfers should occur as well. This indicates a *symmetric existence* dependency between the concepts of each business transaction (cf. left part of Figure 5.7).

5.6.2 Coordination model

For the coordination model it holds that the different concepts are connected through sequence and message flows (cf. Figure 5.2). Again, there are two possible business transactions, depending on whether a customer wants to *purchase or lease* a copier. Both business transactions involve interconnected transfers that occur one after the other, i.e., there exists an order between them. This indicates an *asymmetric existence* dependency between the concepts of each business transaction (cf. right part of Figure 5.7).

The main results of Step 5 are the following:

- Identification of the symmetric existence dependency between transfers in the value model.
- Identification of the asymmetric existence dependency between transfers in the coordination model.

5.7 Step 6: Intra-model consistency constraints

In Step 6 we explicate intra-model consistency constraints, using the identified dependency relations from Step 5. These constraints enable consistency checking of the models with the running system. As discussed in Step 5 of Section 4.5, intra-model consistency

constraints are (1) based on identified *intra-model dependency relations*, and (2) on *model characteristics*. Here, we first discuss consistency constraints on the value model after which we discuss consistency constraints on the coordination model.

The intra-model consistency constraints ensure the model describes the inter-organizational cooperation properly. These constraints hold if at runtime the behavior of the system is the same as the behavior captured in the model. We check this by gathering information from event logs that describe the behavior of the actors, and compare this with the models. Therefore, intra-model consistency constraints are defined using event logs. Again, we consider models on both *business transaction* and on *transfer* level.

5.7.1 Value model

We consider a value model to be consistent with the inter-organizational cooperation if the business transactions described by the value model (e.g. leasing and purchasing a copier) are used in the cooperation (i.e., customers indeed purchase and lease copiers), and if each business transaction executed in the cooperation is also described by the value model (e.g. there are no second hand copiers sold since this is not described in the value model). Furthermore, each business transaction is facilitated in the same way in the value model and in real life (e.g. purchase over the internet versus purchase in a store). Therefore, the consistency constraint defined between model and event log should describe their relation on both business transaction and transfer level.

Transfer level. In Step 5 we identify that all intra-model dependency relations are symmetric existence dependency relations (cf. value viewpoint in Figure 5.7). According to our method, each of these relations is translated into a consistency constraint (cf. Section 4.4.2):

If concepts in set X of two or more concepts are **symmetric existence dependent** on each other, the corresponding constraint states: If $x \in X$ exists, all concepts in X exist.

Here, set X consists of the set of concepts for *purchasing* a copier or of the set of concepts for *leasing* one (cf. value viewpoint in Figure 5.7). For example, for purchasing a copier concepts *Purchase* \in and *Copier* P have a symmetric existence dependency and their consistency constraint is formulated according to the above definition:

If and only if a concept for *Purchase* \in exists in the value model, also a concept for *Copier* P exists.

Each of these intra-model consistency constraints describes a symmetric existence dependency. Therefore, it is possible to *generalize* over these constraints and to define one general intra-model consistency constraint describing this relation. Through transitivity we conclude that all value transfers in a business transaction depend on each other for existence. In other words, if one value transfer of a business transaction occurs, also the

other transfers occur. For example, if the event log shows evidence that lease money is paid by the customer (i.e., *Lease* € in Figure 5.7), there should be evidence of service money paid by the customer (*Service* €), delivery of a copier by the company (*Copier* *L*), and maintenance performed on the leased copier (*Maintenance*). As a result of this transitivity relation between concepts in one business transaction, we generalize these constraints into one constraint for each *business transaction* (i.e., one for purchasing and one for leasing a copier).

Business transaction level. Furthermore, each business transaction in the event log should match one of these business transactions in the value model (e.g., each business transaction in the event log needs to be a purchase or lease transaction). In addition, each business transaction in the value model has to occur at least once in the event log. If, for example, no purchases are made by the customers, but only lease transactions occur, we do not consider this to be consistent with the value model.

In addition to the existence of value transfers and business transactions, transfers are also required to be exchanged between the same actors. For example, when a copier is purchased, it needs to be purchased *by a customer from the copier company*. In other words, *issuer* and *recipient* of transfers captured in value model and event log need to match. First we state when two sets match:

Definition 2 (Match: value model - event log). *Sets match if:*

- *for each value transfer in the value model there exists an entry with matching issuer and recipient in the event log, and*
- *for each valuable entry in the event log there exists a value transfer with matching issuer and recipient in the value model.*

Now, we define the *general intra-model consistency constraint* for the value model:

Constraint 2 (Business transaction, value model). *The value model is intra-model consistent if:*

1. *Each set (representing one business transaction) in the event log matches a set of value transfers representing a business transaction in the value model for the specified time frame.*
2. *Each business transaction in the value model occurs as business transaction in the event log for the specified time frame.*

This constraint is graphically represented in Figure 5.8 where event log entries for one business transaction of *purchasing a copier* are depicted. These entries are checked for consistency with the value model by matching *value transfers* in the event log with those of the value model. Here, the two required value transfers (*Purchase* € and *Copier* *P*) are present in the event log.

In addition to the general consistency constraint, we define *model-specific* constraints. Since the value model denotes financial benefits of the cooperation over a specified period

5.7. STEP 6: INTRA-MODEL CONSISTENCY CONSTRAINTS

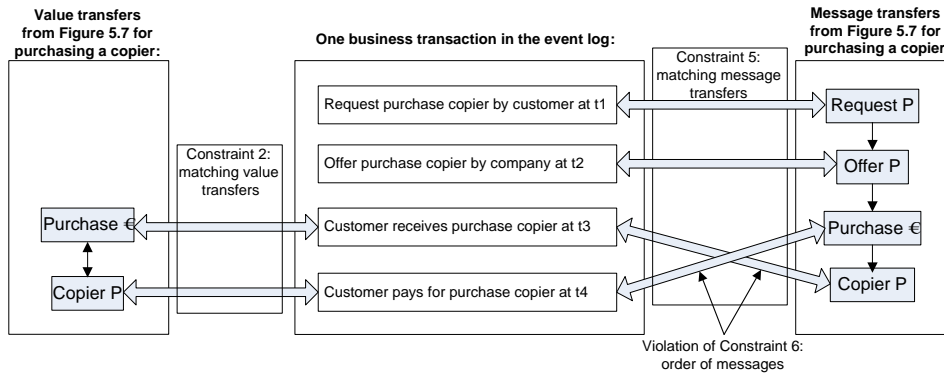


Figure 5.8: Example case: Intra-model consistency constraints

of time, we also check whether estimated profits are achieved. This is done by checking whether the *number* of transfers and their *values* are equal to the estimations (cf. Table 5.1).

Constraint 3 (Number of occurrences). *For each business transaction in the value model the estimated number of occurrences is the same as the realized number of business transactions in the event log during the specified time frame.*

Constraint 4 (Average value). *The estimated average value of the transfers in each business transaction of the value model is the same as the realized average value of this transfer in the event log for the specified time frame.*

5.7.2 Coordination model

In analogy to the value model, we consider a coordination model to be consistent with the inter-organizational cooperation if the business transactions described by the coordination model are used in the cooperation, and if each business transaction executed in the cooperation is also described in the coordination model. Furthermore, each business transaction is facilitated in the same way in the coordination model and in real life. Therefore, the consistency constraint defined between model and event log should describe their relation on both business transaction and transfer level.

Transfer level. In Step 5 we conclude all intra-model dependency relations are asymmetric existence dependency relations. Each of these relations is translated into a consistency constraint (cf. Section 4.4.2):

If concept x is **asymmetric existence dependent** on set Y of one or more concepts, the constraint states: If x exists, Y exists.

In this coordination model all dependency relations are bilateral, i.e., they exist between two concepts. One such constraint now states:

If a concept for *purchasing a copier* exists in the coordination model, a concept for *receiving a copier* exists as well.

Each intra-model consistency constraint describes an asymmetric existence dependency. Therefore, it is possible to *generalize* over constraints and to define one general intra-model constraint. In this case, connected transfers are acyclic, directed graphs (cf. Figure 5.7 where both business transactions form a tree).

Business transaction level. Furthermore, each business transaction in the event log should match one of the business transactions in the coordination model and each business transaction in the coordination model has to occur at least once in the event log.

In addition to the existence of message transfers and business transactions, the transfers are required to be exchanged between the same actors as well. For example, when a copier is purchased, it needs to be purchased *by a customer from the copier company*. In other words, *issuer* and *recipient* of transfers need to match. First, we state when two sets match:

Definition 3 (Match: coordination model - event log). *Sets match if*

- *for each message transfer in the coordination model there exists an entry with matching issuer and recipient in the event log, and*
- *for each message in the event log there exists a value transfer with matching issuer and recipient in the coordination model.*

Now, we define the *general intra-model consistency constraint* using Definition 3 for the coordination model:

Constraint 5 (Business transaction, coordination model). *The coordination model is intra-model consistent if:*

1. *Each set (representing one business transaction) in the event log matches a set of messages representing a business transaction in the coordination model for the specified time frame.*
2. *Each business transaction in the coordination model occurs as business transaction in the event log for the specified time frame.*

We illustrate this constraint in Figure 5.8 where event log entries for one business transaction of *purchasing a copier* are represented. These entries are checked for consistency with the coordination model by matching *message transfers* in the event log with those in the coordination model. Here, the required message transfers are present in the event log.

In addition to this general consistency constraint, a *model-specific* constraint is formulated as well. Since the coordination model explicates in which order messages are to be exchanged, it is checked whether this prescribed order matches the order of entries in the event log. For example, when purchasing a copier, money is paid *before* the copier is

delivered. This strict partial order in the abstraction of the coordination model is checked in a straightforward manner with the timestamps occurring in event log entries.

Constraint 6 (Ordering). *In each business transaction of the event log messages are ordered as prescribed in the coordination model for the specified time frame.*

This constraint is illustrated in Figure 5.8 where event log entries for one business transaction of *purchasing a copier* are represented. The order of these entries is checked for consistency with the predefined order in the coordination model by matching *timestamps*. Here, the event log shows the copier is delivered *prior* to payment. This is inconsistent with the predefined order in the coordination model. In other words, Constraint 6 is violated. Main results of Step 6 are the following:

- Defining intra-model consistency constraints for the value model, and
- Defining intra-model consistency constraints for the coordination model.

5.8 Step 7: Dependency analysis

In the previous steps we identify consistency constraints between models and event log, i.e., we define inter- and intra-model consistency constraints (cf. Figure 5.6). In this step we translate all necessary model parts into a *language-independent* notation to enable easy checking of consistency constraints.

By representing dependencies in and between the models independent from any formalism, we are able to handle different notations used in different models. Here, we use *sets* and *tuples* for representation and refer to these representations as *abstractions* of the models. Each business transaction is represented as a set containing tuples, and each tuple represents a value transfer, message transfer, or event log entry. In addition, the use of sets and tuples enables us to define more *formal* constraints for checking consistency. More specifically, it enables us to redefine constraints from Steps 4 and 6 in matching sets and tuples. Here, we start with abstracting necessary information in the models into sets and tuples after which we discuss formalization of different consistency constraints.

5.8.1 Value model abstraction

In the value model we abstract from information that has no overlap with the coordination model and that is no part of any intra-model consistency constraint. In other words, we only capture these parts of the value model needed to check intra- and inter-model consistency constraints. Essential in these consistency constraints are transfers between actors, properties of these transfers, and inter-transfer relations.

In Figure 5.1 the highlighted grey areas depict the two possible business transactions. The value transfers are important in the overlap between value model and coordination

model. Furthermore, intra-model consistency constraints check the *number* of value transfers and their *value*.

A value transfer in a value model has an *issuer*, *recipient*, unique *name*, estimated average *value*, and estimation on the number of *occurrences*, which we represent as quintuple $x=(a,b,c,d,e)$ where $\text{issuer}(x)=a$, $\text{recipient}(x)=b$, $\text{name}(x)=c$, $\text{value}(x)=d$ and $\text{occurrences}(x)=e$. For example, in Table 5.1 value transfer *Copier P* is expected to be issued by the copier company (represented as: *cc*), to be received by the customer (represented as: *c*), to have an average value of 7500 €, and to occur 24 times. This is represented as: *Copier P*=(*cc*,*c*,*copierp*,7500,24).

These quintuples capture the information for the inter-model consistency constraints, i.e., they contain the different value transfers. Furthermore, they capture the information for the intra-model consistency constraints, i.e., they contain both expected number of occurrences and expected value. As a last part, we *group* different tuples into *sets*, where each set describes one business transaction.

As a result, the abstraction of the value model from Figure 5.1 contains two sets (the two grey areas) as well as the values and number of occurrences in Table 5.1:

$$\mathcal{V}=\{(c,cc,lease,1200,48),(cc,c,copier1,1200,48), \\ (c,cc,service,700,48),(cc,c,maintenance,700,48), \\ \{(c,cc,purchase,7500,24),(cc,c,copierp,7500,24)\}\}$$

5.8.2 Coordination model abstraction

The abstraction of the coordination model captures the overlap with the value model (i.e., those model parts used for inter-model consistency constraints), and those model parts that are used for the intra-model consistency constraint. Therefore, the abstraction captures message transfers between actors, the order in which these transfers occur, and inter-transfer relations.

We use sets of message transfers performed in a single business transaction to represent the coordination model (see highlighted grey areas in Figure 5.2). A message transfer is represented by *issuer*, *recipient*, and unique *name* as triplet $x=(\text{issuer},\text{recipient},\text{name})$. For example, message transfer *copier p* in Figure 5.2 is represented as (*cc*,*c*,*copierp*) with $\text{issuer}(\text{Copier P})=cc$, $\text{recipient}(\text{Copier P})=c$, and $\text{name}(\text{Copier P})=copierp$. Furthermore, a *strict partial order* ‘<’ is defined between messages based on the order in which they occur in the coordination model.

The triplets capture the information for the inter-model consistency constraints, i.e., they contain the different message transfers. Furthermore, they capture information for the intra-model consistency constraint, i.e., they contain the order in which messages should occur. As a last part, we *group* the different tuples into *sets*, where each set describes one business transaction. As a result, the abstraction of the coordination model from Figure 5.2 contains two sets (the two grey areas) as well as a strict partial order:

$$\mathcal{W} = \{ \{ (c, cc, request), (cc, c, offer), (c, cc, lease), (cc, c, copier1), (c, cc, service) \}, \\ (c, cc, request) < (cc, c, offer), (cc, c, offer) < (c, cc, lease), \\ (c, cc, lease) < (cc, c, copier1), (c, cc, service) < (cc, c, copier1) \}, \\ \{ (c, cc, request), (cc, c, offer), (c, cc, purchase), (cc, c, copierp) \}, \\ (c, cc, request) < (cc, c, offer), (cc, c, offer) < (c, cc, purchase), \\ (c, cc, purchase) < (cc, c, copierp) \} \}$$

5.8.3 Formalization of constraints

In Steps 4 and 6 we define intra- and inter-model consistency constraints. These constraints are expressed in natural language. In Step 7, so far, we define abstractions of models and event log based on these constraints. Based on formalization of the models, it is possible to formalize constraints. We demonstrate this for the *inter-model* consistency constraint between value and coordination model. We use the definition of *matching sets* (cf. Definition 1 page 80) to formulate this consistency constraint (cf. Constraint 1 page 80). To check whether abstractions of models are semantically equal (i.e., represent the same sets of transfers) we need to match:

1. *business transactions*: check whether value and coordination model offer the same business transactions, i.e., whether the sets in their formal models match, and
2. *concepts*: check whether the concepts in the business transactions match, i.e., whether the elements in the sets representing value and coordination model match with each other.

For example, we check whether value and coordination model both describe purchasing and leasing a copier, i.e., we check whether the sets representing the models match. Furthermore, we check whether the transfers that are prescribed by the value model and coordination model for these business transactions match with each other, i.e., we check whether elements in the sets of the different models match with each other.

Definition 1 that matches business transactions as well as transfers between the actors, is formalized in two different definitions. First, a formal definition for matching the transfers is given. Two concepts (e.g., value and message transfer) *match* if they have the same *issuer*, *recipient*, and *name*. In the abstraction, concepts are represented as elements in sets:

Definition 4 (Matching elements). *Let x and y be elements of a set. Then x and y are matching elements ($match(x,y)$) iff:*

1. $issuer(x)=issuer(y)$,
2. $recipient(x)=recipient(y)$, and
3. $name(x)=name(y)$.

Second, a formal definition for matching business transactions is formulated. Two business transactions *match* if they facilitate the same exchanges between actors (i.e., the elements in the sets match). A business transaction is represented as set in the abstraction:

Definition 5 (Matching sets). *Sets M and N match ($M \cong N$) iff*

1. $\forall x \exists! y (x \in M \wedge y \in N \wedge \text{match}(x, y))$, and
2. $\forall y \exists! x (y \in N \wedge x \in M \wedge \text{match}(x, y))$.

In both formal definitions, sets and elements are used to capture the notions of business transactions and concepts, respectively. Constraint 1 defined in Step 3 states that each business transaction in the value model should have a matching business transaction in the coordination model, and vice versa. To formalize this part of the constraint, we define *mapping* business transactions in the value model to business transactions in the coordination model, and vice versa:

Definition 6 (Mapping). *Let set \mathcal{V} and \mathcal{W} be abstractions of value and coordination model. Then there exists a mapping $v: \mathcal{V} \rightarrow \mathcal{W}$ if:*

1. $\forall M \exists! N (M \in \mathcal{V} \wedge N \in \mathcal{W} \wedge N \cong M)$, and
2. $\forall N \exists! M (N \in \mathcal{W} \wedge M \in \mathcal{V} \wedge M \cong N)$.

Based on these definitions, we formalize Constraint 1. If there exists a mapping from one model to the other, the inter-model consistency constraint is satisfied:

Constraint 7 (Value and coordination model - formal -). *Let set \mathcal{V} and \mathcal{W} be abstractions of value and coordination model. Then these models are inter-model consistent iff there exists a mapping $v: \mathcal{V} \rightarrow \mathcal{W}$.*

The main results of Step 7 are the following:

- Formalization of value model parts for checking consistency,
- Formalization of coordination model parts for checking consistency, and
- Formalization of the inter-model consistency constraint.

5.9 Step 8: Log analysis

In the previous steps we analyze the models, define consistency constraints, and formalize these parts of the models necessary to check consistency. In Step 8 we demonstrate how we analyze event logs to check consistency constraints at runtime as well.

The event log contains information exchanged between actors in the cooperation. The abstraction of the event log captures these messages necessary to check intra-model consistency constraints of both value and coordination model. Therefore, the abstraction

contains all message transfers needed for coordinating the cooperation and all message transfers containing some value. In addition, we capture the order of these transfers, their value, their number of occurrences, and the inter-transfer relations.

Each set in the abstraction contains entries performed in a single business transaction. Each entry in an event log is represented as *timestamp, issuer, recipient, unique name*, and specific *value*. For example, entry *Service* in transfer *copier.payment* (cf. Figure 5.3) is represented as: (3,c,cc,service,700). We use a simplified notation for timestamps; a higher integer indicates a later point in time. The following example shows the set abstraction of an event log for one month.

$$\mathcal{E} = \{ \{ (1, c, cc, request, 0), (2, cc, c, offer, 0), (3, c, cc, lease, 1200), (4, cc, c, copier1, 1200), (3, c, cc, service, 700) \}, \{ (5, c, cc, request, 0), (6, cc, c, offer, 0), (7, c, cc, lease, 1400), (8, cc, c, copier1, 1400), (7, c, cc, service, 700) \}, \{ (9, c, cc, request, 0), (10, cc, c, offer, 0), (11, c, cc, purchase, 6000), (14, cc, c, copierp, 6000) \}, \{ (15, c, cc, request, 0), (16, cc, c, offer, 0), (17, c, cc, lease, 2500), (20, cc, c, copier1, 2500), (17, c, cc, service, 1000) \}, \{ (21, c, cc, request, 0), (22, cc, c, offer, 0), (23, c, cc, purchase, 10000), (24, cc, c, copierp, 10000) \}, \{ (25, c, cc, request, 0), (26, cc, c, offer, 0), (27, c, cc, lease, 1000), (28, cc, c, copier1, 1000), (27, c, cc, service, 700) \} \}$$

A formalization of the intra-model consistency constraints (as discussed in Step 7 for the inter-model consistency constraint) is described in Bodestaff et al. [20]. Since the aim of this thesis is to show applicability of the method for managing inter-organizational cooperations, describing the formalization of the intra-model consistency constraints is out of scope.

After formalizing the event log, it is possible to check intra-model consistency constraints for both value and coordination model:

Value model. Each business transaction in event log \mathcal{E} is a realization of a business transaction in value model \mathcal{V} (cf. Constraint 2). Each valuable entry in the event log matches a value transfer in the value model. Again, we disregard value transfer *maintenance* since this is not part of the contract establishment, but of the execution phase. Furthermore, both business transactions modelled in the value model are present in the event log. Also, the number of occurrences should be equal to estimations in the value model (cf. Constraint 3). Estimations in the value model are for one year, while event log \mathcal{E} represents activities over one month. Therefore, we divide the estimated number of occurrences by twelve. For example, the realized number of *Purchase* transfers in the event log is 2 and the estimated number of occurrences *Purchase* in the value model is $24 \frac{1}{12} = 2$. Furthermore, realized average value should be equal to the estimated average value (cf. Constraint 4). For example, realized average value of *Purchase* is $\frac{6000}{2} + \frac{10000}{2} = 8000$ euro and therefore, *not* equal to estimated average value in the

value model of 7500 euro. Therefore, value model and event log are *not* consistent at September 15, 2007.

Coordination model. To check consistency between event log \mathcal{E} and coordination model \mathcal{M} , we check both constraints. Each business transaction in the event log matches a set in the abstraction of the coordination model (cf. Constraint 5). Each entry in the event log matches a message transfer in the coordination model. For example, event log entry $\{(1, c, cc, request, 0), (2, cc, c, offer, 0), (3, c, cc, lease, 1200), (4, cc, c, copier1, 1200), (3, c, cc, service, 700)\}$ matches $\{(c, cc, request), (cc, c, offer), (c, cc, lease), (cc, c, copier1), (c, cc, service)\}$. Furthermore, the *order* of messages in the coordination model should be equal to the order in the event log (cf. Constraint 6). The coordination model prescribes payments have to occur before delivery. This is the case regarding this example. Event log and coordination model are consistent since both constraints are met.

5.10 Step 9: Causal analysis

In the previous steps we define consistency constraints between models and event log which we check as well. After checking the consistency constraints, it is now possible to do causal analysis of reasons for inconsistencies. In Step 9, we prepare actual management of models through causal analysis. The aim is (1) to find causes for violations of the consistency constraints, and (2) to predict consequences when adapting models to regain consistency. By identifying causes we enable adaptation of the models so that inconsistencies are solved, preferably without introducing new ones.

For this, we first discuss what *causes* exist for violated constraints. Second, we discuss possible *changes* to restore consistency between the models. Third, we link causes for violations to changes in the models. We indicate which causes can be solved by which changes in the models. We conclude with some *theorems and proofs* that demonstrate the relation between coordination and value model using the causal analysis.

5.10.1 Causes

An intra-model constraint violation of a value or coordination model is an inconsistency between running system and model (cf. Constraints 2, 3, 4, 5, and 6). A corresponding categorization of these causes is depicted in Table 5.2. Of course, causes can appear simultaneously.

Cause 1. *There is a set in the event log abstraction that does not match any set in value or coordination model abstraction.* There is a business transaction executed at runtime and not modelled (cf. Constraint 2, (1) and Constraint 5, (1)). This situation occurs if:

(a) A combination of transfers is missing. For example, consider an event log containing entries of leasing two copiers with only one maintenance contract. These entries will not match with a set in the value model abstraction where each copier has its own maintenance contract.

Name	Cause	VM	CM
Cause 1	Missing set in model	x	x
Cause 2	Missing set in event log	x	x
Cause 3	Mismatch number of occurrences	x	
Cause 4	Mismatch average value	x	
Cause 5	Mismatch message order		x

Table 5.2: Causes of intra-model constraint violation

(b) There is a transfer missing in the model, for example, if the event log contains entries of selling printers while no printer purchase is modelled in the value model.

Cause 2. *A set in the model abstraction has no occurrence in the event log.* This means that one business transaction never occurred (cf. Constraint 2, (2) and Constraint 5, (2)). For example, if the event log does not show any entries of selling copiers, this is inconsistent with Figure 5.1.

Cause 3. *The number of estimated occurrences of a value transfer in a specific set is not the same as the realized number of occurrences of that value transfer in sets of the event log* (cf. Constraint 3). For example, assume the value model estimates 24 copiers will be purchased. If the event log shows only 20 sold copiers, an inconsistency occurs.

Cause 4. *The estimated average value of a value transfer in a specific set is not the same as the realized average value of transfers in sets of the event log* (cf. Constraint 4). For example, in the value model it is estimated that a purchase copier costs on average 7500 €. If the event log shows an average of only 6000 €, an inconsistency occurs.

Cause 5. *The message order in a set in the coordination model abstraction does not match the message order in a set in the event log* (cf. Constraint 6). For example, in the coordination model a copier is first paid and then delivered. If the event log shows a delivery before payment, an inconsistency occurs.

5.10.2 Changes

In the previous subsection we identify *causes* for inconsistencies. In this section we review which *changes* in the models restore consistency. To maintain consistency between running system and models, changes might become necessary. For example, if the event log shows a lower number of sales than expected it makes sense to adapt estimations in the value model, or to change implementation to enforce more sales. Handling *impact* of changes within one model on inter-model relations for complex cooperations is tedious work. Each consistency relation a changed model has with other models, must be reevaluated and, if necessary, be updated.

To enable more efficient and structured checking and maintenance of consistency, we propose determining *upfront* effects certain changes have. In this way, a well-informed decision on the type of change can be made. Furthermore, it is possible to oversee which other relations are affected. We demonstrate our approach by illustrating how to categorize

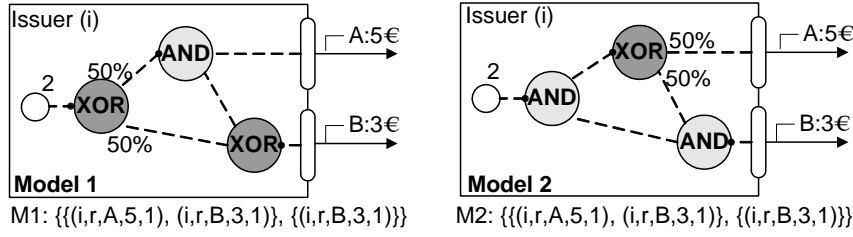


Figure 5.9: Same class models

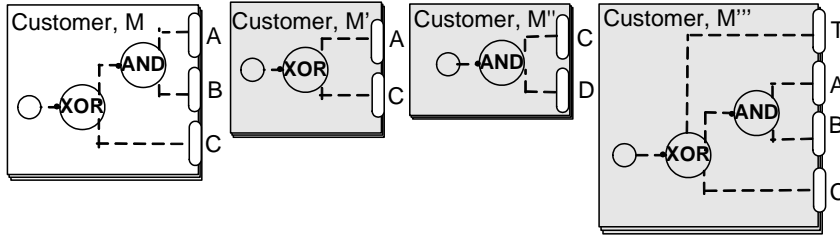


Figure 5.10: Observable changes

size changes in value and coordination models (cf. Table 5.3).

Change 1. *Non-observable changes* in a model do not influence the abstraction of the model, i.e., the change does not influence the possible business transactions. There exist two ways for applying non-observable changes:

(i) Typically, there is more than one way to structure a model while preserving the same possible business transactions. For example, though *Model 1* and *Model 2* in Figure 5.9 are different, they facilitate the same transactions since abstractions of M1 and M2 are equal. Since a change from Model 1 to Model 2 does not change the possible business transactions, we refer to this as non-observable change.

(ii) The other way is to change part of the model with no connection to the abstraction. For example, changing choice ‘*lease or buy?*’ in Figure 5.2 into ‘*sufficient money or not?*’ does not influence the abstraction of the model since it is not a message transfer.

Depending on the type of model, also several categories of *observable* changes are identified (cf. Table 5.3):

Change 2. *Observable structural changes* add or remove (part of) a business transaction by adding or removing *constructs* (e.g. XOR-splits and transfers) while preserving a valid model. Figure 5.10 depicts three observable structural changes represented as e^3 -value model. Model M with set $\{\{A, B\}, \{C\}\}$ is the original model of our running example. However, we rename transfers for the sake of simplicity. Models M' , M'' , and M''' are adapted models of M where (sets of) transfers are removed or added.

Other observable changes are *model-specific*. Changes in the estimated number of occurrences (Change 3 in Table 5.3) and in the estimated average value of a transfer (Change

Type	Subtype	Change	VM	CM
Non-observable		Change 1	x	x
Observable	Structural	Change 2	x	x
	# Occurrences	Change 3	x	
	Average value	Change 4	x	
	Message order	Change 5		x

Table 5.3: Categorization of model changes

4) are value model-specific. Changes in the order of messages (Change 5) are coordination model-specific. In general, model-specific changes are related to these parts of the abstraction which are not captured by observable structural change. By going through the abstraction and addressing each part not influenced by an observable structural change, model-specific changes are identified.

Change 3. An *observable change in the number of occurrences* is achieved by adapting the last element of a quintuple (i.e., number of occurrences). For example, Model 1 in Figure 5.9 represents a single actor with 2 customer needs. Adapting this from 2 to 4 results in doubling the last element in the quintuple from 1 to 2.

Change 4. An *observable change in average value* indicates the fourth element of a quintuple (i.e., the average value) is changed. As opposed to Change 3 the average value is not the result of other estimations, but of combining information outside the model. For example, information on production costs determine, partially, the average value of a product.

Change 5. An *observable change in the order* reorganizes exchanges when they occur. Here, the coordination model depicts payment before delivery. The copier company might decide to deliver prior to payment as service to its customers. Changing this order does not affect other parts of the abstraction, it only affects the strict partial order.

Based on this categorization of changes, we show that a non-observable change has different properties compared to an observable change:

We refer to models with the same functionality, i.e., facilitating the same business transactions, as models that belong to the same *class* (like Model 1 and Model 2):

Definition 7 (Class). Let sets \mathcal{V} and \mathcal{W} be value model abstractions. Then these models belong to the same class if there exists a mapping from \mathcal{V} to \mathcal{W} according to Constraint 7 page 90.

As opposed to non-observable changes, *observable changes* change the possible business transactions. As a consequence, we state that observable changes change the class of the model:

Lemma 1 (Observable Structural Change). An *observable structural change in a model changes its class*.

Proof: According to Definition 6, there exists a mapping between two abstractions if their sets *match* (cf. Definition 5). If there exists a mapping, they belong to the same

class (cf. Definition 7). By adding or deleting a set or part of a set, the new set and the original one are not consistent with each other (cf. Definition 5). As a consequence, the new abstraction of the model belongs to a different class than the original one. Also changes in *issuer*, *recipient* or *name* result in a class change since such change influences matching tuples (cf. Definition 4). \square

5.10.3 Causal analysis

The causal analysis consists of two parts. (1) We analyze possible *causes* for violations. For this, we apply the cause categorization from Section 5.10.1 and use the asymmetric intra-model dependencies to find the cause of an inconsistency. (2) Further, we determine which *changes* solve the inconsistency and what the consequences are for other constraints when applying this change.

Causes. As discussed in Step 9 of Chapter 4, a causal analysis can be done for asymmetric dependency relations. The intra-model dependencies in the value model are symmetric and those of the coordination model are asymmetric (cf. Figure 5.7).

As a consequence, if there is a transfer modelled in the value model that is inconsistent with the coordination model or event log, it is not possible to analyze whether another transfer caused this problem. The reason for this is that it is not modelled whether one value transfer causes the existence of another, but merely that several transfers occur in one transaction (hence the symmetric dependency relation). Furthermore, if a message transfer in the coordination model is inconsistent with value model or event log, it is possible to analyze whether there are problems earlier in the chain that causes this inconsistency. The reason for this is that the order in which messages are expected to occur is modelled in the coordination model (hence the asymmetric dependency relation).

Consider the example from Figure 5.11 where one business transaction of the coordination model is represented (cf. Figure 5.7), and an event log entry that contains messages *purchase* and *copier P*, but apparently misses the original *request P* and *offer P* messages. Using the *causal analysis* of the asymmetric dependency relation we determine that the cause of the nonexistence of message *offer P* is most probably the nonexistence of message *request P*. In other words, a customer bought a copier without a written request and offer. Solving this inconsistency might be done by either allowing purchase and delivery of copiers without a written request and offer (i.e., add a business transaction to the coordination model), or by changing the implementation in such a way that sending products without an official offer is not possible anymore.

Changes. In Section 5.9 an inconsistency between value model and event log is detected. Estimated average value of a copier is 8000 € while data in the event log show 7500 €, i.e., a violation of Constraint 4. As discussed, it is not possible to analyze what caused this violation. However, we provide possible *changes* to the value model and predict *effects* resulting from these changes.

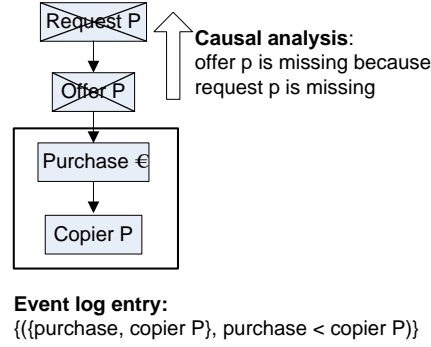


Figure 5.11: Example case: Causal analysis of coordination model

We identify which changes affect which constraints (cf. Figure 5.12). In general, we determine for each type of model change (cf. Table 5.3) which constraints are affected. Each change affects a specific part of the abstraction of a model (e.g., changing the estimated average value of a transfer affects the fourth element of the quintuple representing that transfer in the value model abstraction (cf. Change 4)). This specific part of the abstraction appears in one or more consistency constraints. For example, the fourth element of a value model tuple (i.e., estimated average value) is part of Constraint 4. Figure 5.12 depicts these relations graphically by pointing an arrow from each type of change to the constraints it affects. Here, we see that in the example case there is one possible type of model adaptation to solve the inconsistency in Constraint 4, namely type Change 4.

In complex models, constraint violations might be solved by applying multiple types of changes (e.g. a problem with Constraint 1 can be solved by changing the value model using type Change 2a or 2b, or by changing the coordination model using type Change 2a or 2c). However, each type of change might affect more than one constraint (e.g. value model Change 2a and 2b affect Constraints 1 and 2). Visualizing these complex relations enables the user to determine the most suitable type of change for a specific constraint violation. The most suitable one is the change having the least negative effects on other not-violated constraints.

5.10.4 Minimize the number of changes: Theorem and proof

We aim at maintaining consistency while *minimizing changes* in the models, and *without introducing new inconsistencies*. We assume the implementation is based on inter-model consistent value and coordination models (cf. Constraint 7). We then show the following: If the coordination model is intra-model consistent (cf. Constraint 5 and 6), maintaining intra-model consistency for the value model (cf. Constraint 2, 3, and 4) can be achieved *without structural changes* (Change 2) and without introducing new inconsistencies. The advantage is that Changes 3, 4, and 5 each affect only one constraint, and therefore minimize the number of required changes for introducing new inconsistencies.

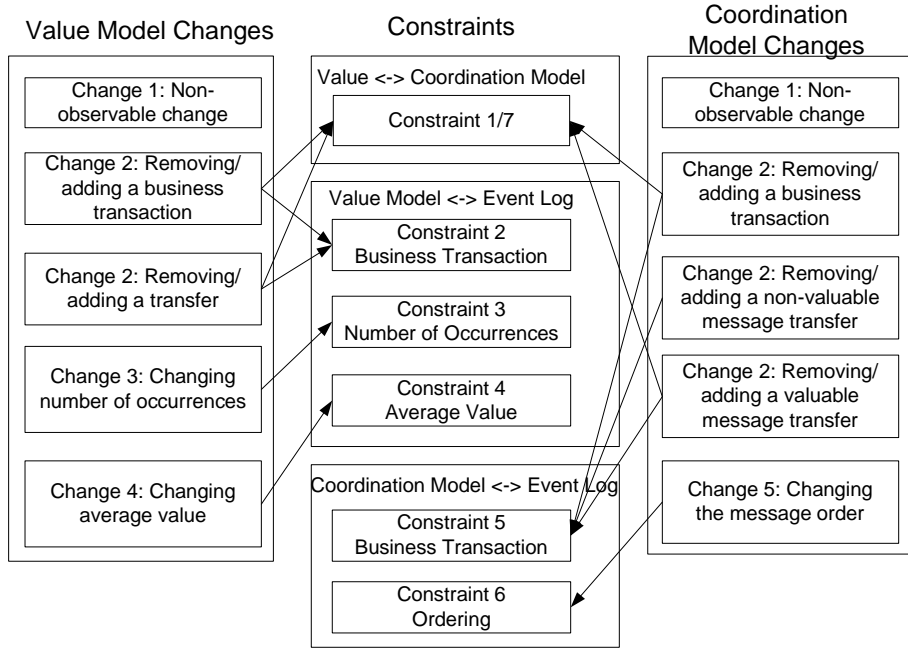


Figure 5.12: Relating constraints and changes

We use definitions and categorizations from previous sections.

Lemma 2. *Assume the coordination model is intra-model consistent. Assume further inter-model consistency. Then there is always a mapping between the abstraction of the event log and the abstraction of the value model.*

Proof: If there is no mapping there must either be a set in the event log not present in the value model (Constraint 2 (1)), or there is a set in the value model not present in the event log (Constraint 2 (2)).

Violation of (1) means the set in the event log has to be an occurrence of a set in the coordination model due to intra-model consistency of the coordination model (Constraint 5). This set in the coordination model must have a matching set in the value model because of inter-model consistency (Constraint 7). However, the set in the coordination model is an occurrence of this set in the value model \perp .

Violation of (2) means this set in the value model has to match a set in the coordination model due to inter-model consistency (cf. Constraint 7). Due to intra-model consistency of the coordination model this set must have an occurrence in the event log (cf. Constraint 5). If this set has an occurrence in the event log it will be an occurrence of the value model. However, this contradicts our initial assumption \perp . \square

Lemma 3. *Assume the coordination model is intra-model consistent. Assume further inter-model consistency. Then it is always possible to resolve intra-model inconsistency*

of the value model by adapting the value model with an observable non-structural change (Change 3 or 4).

Proof: If the coordination model is intra-model consistent, there is a mapping between event log and value model (cf. Lemma 2). If there is a mapping and an intra-model inconsistency of the value model, this is due to Cause 3 (i.e., a mismatch in the estimated number of occurrences) or due to Cause 4 (i.e., a mismatch in the average value of a transfer) (Table 5.2). By transitivity, if the coordination model is intra-model consistent, intra-model inconsistency of the value model can be solved through an observable non-structural change. \square

Lemma 4. *An observable non-structural change in the value model does not influence inter-model consistency between value and coordination model.*

Proof: If an observable non-structural change had influenced inter-model consistency, it would influence the mapping between the abstractions of value model and coordination model (Constraint 7). To influence this mapping, a non-structural change would have to add or remove a set from the abstraction of value or coordination model (Constraint 6). A non-structural change (cf. Change 3 or Change 4 in Table 5.3) only adapts values of elements in a quintuple and never influences sets. Therefore, a non-structural change never influences inter-model consistency. \square

Using the results from Lemmas 2 - 4 we can now prove Theorem 1.

Theorem 1. *Assume that the coordination model is intra-model consistent and that value and coordination model are inter-model consistent. Then it is possible to solve intra-model inconsistencies of the value model while preserving inter-model consistency between value and coordination model.*

Proof: If assuming intra-model consistency for the coordination model, and inter-model consistency, there is always a mapping from the value model to the event log (see Lemma 2). Intra-model inconsistency of the value model can then be solved through an observable, non-structural change (e.g., Change 3 or 4) (see Lemma 3). An observable non-structural change does not influence inter-model consistency (see Lemma 4). By transitivity: If coordination model is intra-model consistent, intra-model consistency of the value model can be solved without influencing inter-model consistency. \square

By constructing such proofs, some possible changes are not considered when attempting to regain consistency. By ruling out necessity of certain changes, the process of adapting models to regain consistency becomes more efficient. Furthermore, investigating formal properties improves our understanding of relations between value model, coordination model and event log.

5.11 Related work on value and coordination models

Checking consistency. Several approaches for ensuring consistency between different models at operational level exist. For example, *Business Process Intelligence* (BPI) aims

at supporting business and its users in managing process execution quality. Grigori et al. [59] acknowledge the importance of inter-model alignment. Recently efforts are made to focus on the analysis of costs related to the use of BPI [84]. Here, Mutschler et al. introduce two cost models. One model analyzes the Total Cost of Ownership, while the other one analyzes the impact of BPI on Software Development Efforts. Although in BPI quantifications are made and data is related to process models, BPI focuses on execution quality and not on the overall performance of the cooperation. Another example is the *Astro-project* [65] where business requirements and business processes are integrated into one method to enable flexibility. Formal verification of, for example, consistency within the method can be checked.

A well known approach for assessing business models is using *Key Performance Indicators (KPIs)*. In respective approaches, KPIs are chosen as evaluation criteria for business models. In Giaglis et al. [54] KPIs are used to overcome the problem of measuring a priori the benefits of E-Commerce investments. The e-business is assessed by business process simulation where users can experiment with different configurations. The resulting simulated values of the KPIs are compared with the estimated values in the process models. A business decision is made based on this comparison. In our mechanism, the profitability evaluation criterium can be considered a KPI.

Another approach is *forecast modelling* where a prediction on future behavior is made based on current available data. These models are used for decision making. In Zhao et al. [119] decisions on cooperative investments are supported using forecasting models. Here, also simulation models are used for selecting proper forecasting models. However, these approaches do not provide a business view on the dependencies of measured values. Furthermore, these approaches do estimations on the future rather than representing observed behavior.

Another mechanism for adapting models during runtime is the use of *reflection*. Greenwood et al. [58], for example, separate representation and enactment domains which relates to our separation of design time and runtime environments. Their approach supports ongoing transformations between both domains. The use of reflection allows generation of new programs by another running program. This allows users to add new processes to a running program. In Edmond et al. [43] reflection is applied for reusing, extending and customizing current processes. While these reflection-based approaches focus on evolution of processes, they disregard monitoring the business from a value viewpoint.

Consistency through construction. Besides checking consistency between different models, there exist constructive approaches guaranteeing consistency of the model derived from another model. For example, in Andersson et al. [6] an approach is proposed to use an intermediate model as a bridge between a business model and a process model. The approach is based on identifying tasks needed to accomplish the consumer need and to derive the interdependencies of these tasks. Andersson et al. [7] propose a *chaining method* to derive from a business model a corresponding process model.

Another approach is suggested by Koehler et al. [69] who proposes a *pattern based* approach to come from a business process model to a consistent implementation. Model

checking techniques are used to automatically verify consistency. However, these constructive approaches focus mainly on inter-model consistency.

It is a big challenge in process management to support the modelling, monitoring and maintenance of the relations between the different sub-processes [82, 83]. In this context Müller et al. consider consistency between data and process structures. Their COREPRO framework provides mechanisms for maintaining data-driven process structures

In this thesis we discuss checking consistency between value and coordination models. However, there also exists work on ensuring consistency between value and coordination models. For example, Wieringa et al. [114] describe a method for constructing physical delivery models that describe the exchanges between partners in the real world. These delivery models close the gap between value and coordination model and show in this way how they are related.

Value modelling. Although value modelling for business models is getting more popular, modelling with a focus on value creation is used in other disciplines as well. For example, in value based software engineering the focus is on value creation through software development. Here, the importance of linking value models and software design is recognized. Sullivan et al. [106] propose to use real options to link value and software design. Boehm et al. [28, 27] develop a roadmap to add the concept of value creation to modelling techniques and decision making related to software development. Furthermore, they also acknowledge the challenge of relating technical models with value creation.

5.12 Summary

In this chapter we apply our method for managing dependency relations between models for inter-organizational cooperations (cf. Chapter 4) to value and coordination modelling. We show how to apply the different steps. Furthermore, we demonstrate that when using this method it is possible to formalize different consistency constraints so that formal properties of the models can be identified (cf. Section 5.10.4) and their implementation is more straightforward. This case is illustrated with the use of a running example. Later, in Section 10.3 we discuss the lessons learnt from applying our MaDe4IC method for managing dependencies between inter-organizational models to the given case.

PROOF-OF-CONCEPT IMPLEMENTATION: BUSINESS AND COORDINATION MODEL

In Chapter 5 we illustrate the use of our MaDe4IC method for managing dependency relations for inter-organizational models. In this chapter, we show how the results of this analysis are implemented [25]. Here, we provide a mechanism to monitor and adapt the *value model* using intra-model consistency constraints and dependencies identified in Chapter 5.

We demonstrate the implementation by means of a running example which is discussed in Section 6.1. In Section 6.2 we discuss the formal consistency constraints that are implemented. In Section 6.3 we discuss how the different parts of consistency constraints are implemented. Furthermore, we show in Section 6.4 how the implementation is used to analyze consistency constraints and how to identify causes for inconsistencies. We give a description on how monitoring and adaptation results are visualized in the value model in Section 6.5. We conclude this chapter with an evaluation of the developed method for managing business and coordination models in Section 6.6.

6.1 The business case

In this chapter we use a running example based on a real case in the health insurance sector in the Netherlands for illustrating the implementation. We use a different example than in the previous chapter since the structure of this case is more complex, and, therefore, better suitable for validation purposes.

An insurance company provides insurance on an annual basis to its customers. Customers pay premium on a monthly basis and claim refunds for received treatments. Every paid refund by the insurance company is compensated by CVZ, a Dutch organization distributing tax money. CVZ gets its funding from the government. Next, we model this scenario as value model and part of the scenario as coordination model.

6.1.1 Value model

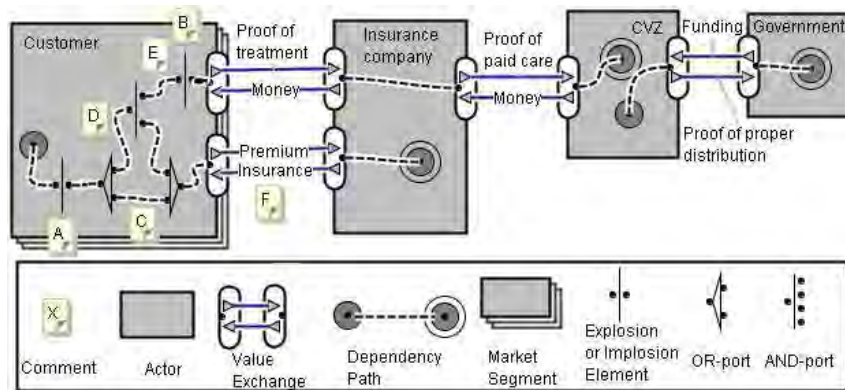
For evaluating economic profitability of a cooperation, a value model is created. Our business case, represented as e^3 -value model [57] (cf. Figure 6.1), is described in detail in a technical report [22]. In this chapter we restrict to those constructs necessary for illustrating the implementation.

The example from Figure 6.1 comprises four *actors* and eight *value transfers*. For example, the value object *premium* is transferred from the customer to the insurance company. Another value object, the *insurance* itself, is transferred from the insurance company to the customer. In Figure 6.1 these two transfers are annotated with an ‘F’. A combination of value transfers in one transaction is referred to as a *value exchange*. In e^3 -value a distinction is made between different kinds of value objects. A value object is either a *product*, *service*, *money*, or a *consumer experience*. In this example the *premium* is a value object of type *money* and the *insurance* provided by the insurance company is considered a *service*.

The consumer need is “having a health insurance for one year”. This is represented by placing the *start stimulus* at the customer. The set of value objects that is transferred to fulfill the consumer need, consists of all value transfers connected through the *dependency path* in the model. Every month there are two possible sets of value transfers that fulfill the consumer need. Either the customer claims restitution for *treatments* he paid for himself, and he pays the monthly *premium*, or he only pays the monthly *premium*. When the customer claims a restitution, the insurance company claims compensation from CVZ. CVZ, in turn, gets its *funding* from the government. Also note that the health insurance company has multiple customers, represented as *market segment* in Figure 6.1.

In Figure 6.1, the twelve monthly payments for fulfilling one consumer need are realized by adding an *explosion element*, annotated with ‘A’ in the figure, associated with ratio 1 : 12. The choice between the two options for fulfilling the consumer need is represented as OR-port. In e^3 -value an OR-port is an exclusive OR. After such an OR-port only one of the dependency paths is selected. If the customer has not received treatments that month, the path annotated with ‘C’ is chosen. The two resulting value transfers constitute the first set of transfers that fulfill the consumer need. Otherwise, if the customer receives treatment that month, the path annotated with ‘D’ is chosen. This path further splits through an AND-split, representing a parallel occurrence of two or more dependency paths. In an AND-join, in turn, all incoming dependency paths share the continuation of the dependency path. The two value exchanges between customer and insurance company take place. To enable more than one restitution claim per month another explosion element, annotated with ‘B’, is added. The insurance company claims restitution from CVZ. These value transfers constitute the second set of value transfers. Finally, the dependency path starting within CVZ represents the third set of value transfers.

The quantifications (not shown in Figure 6.1) that are associated with the graphical representation of the value model, provide estimations. The market segment, for example, is quantified by estimating the number of customers. Also ratios on the explosion elements and the OR-split are set. For every monetary value transfer a quantification is

Figure 6.1: Example case: Business model (e^3 -value notation)

given in the profitability sheets. The expected revenue for every actor in the model is calculated.

6.1.2 Coordination model

A coordination model depicts *how* the transfer of value objects between the parties is realized. In particular, it describes the *order* in which messages between parties are exchanged. An ordered set of messages is referred to as *execution sequence*. This ordering information is omitted in the value model.

Since we do not implement the coordination model, we only depict a small part of the business case as Petri Net [64] in Figure 6.2. However, [21] provides a detailed description of the business case as well as the model represented as Petri Net. Figure 6.2 represents coordination of the payment process of a customer to the insurance company for having insurance for one year. This part of the coordination process is related to the value exchange of *premium* and *insurance* in the value model, annotated with 'F' in Figure 6.1. The money value transfer *premium* is represented as message exchange in the coordination model. This is not the case for the service value transfer *insurance*, since services do not instantiate explicit message exchanges.

Places (indicated as circles) can hold any number of *tokens* (represented as black dots), and *transitions* (indicated as squares) act on input tokens by *firing*. Message exchanges are represented as places, and tasks as transitions in the coordination model. Places and transitions are connected through *arcs*. These arcs indicate the ordering of tasks and message exchanges. In Figure 6.2 the customer first does a payment through executing task *Pay*, after which message *Premium*, place *p3* in Figure 6.2, is transferred from customer to insurance company.

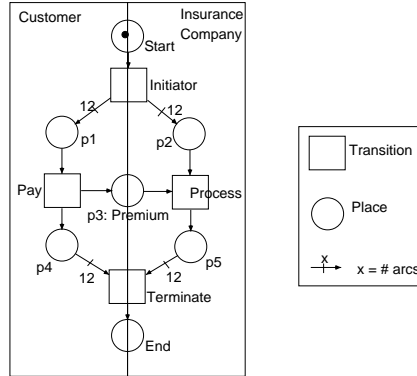


Figure 6.2: Example case: Coordination model (Petri Net notation)

6.2 Consistency constraints

In the proof-of-concept implementation we consider *intra-model* consistency constraints for the value model. In Section 5.7.1 we define intra-model consistency constraints for the value model. These constraints specify when the value model is considered to be consistent with the running system. We check this by analyzing the event log that captures essential events of the cooperation. The first constraint is that business transactions in the model have to be reflected in the event log, and vice versa (cf. Section 5.7.1, Constraint 2 page 84):

Constraint (Business transaction, value model). *The value model is intra-model consistent if:*

1. *Each set (representing one business transaction) in the event log matches a set of value transfers representing a business transaction in the value model for the specified time frame.*
2. *Each business transaction in the value model occurs as a business transaction in the event log for the specified time frame.*

Secondly, the number of occurred value transfers in the event log has to be equal to the estimated number of occurrences in the value model. If, for example, the estimated number of treatments is twice as high as the realized number of treatments, model and event log are not consistent (cf. Section 5.7.1, Constraint 3 page 85):

Constraint (Number of occurrences). *For each business transaction in the value model the estimated number of occurrences is the same as the realized number of business transactions in the event log during the specified time frame.*

Furthermore, the average value of transferred messages has to be equal to estimations in the value model. If, for example, the average value of paid premium is lower than the

estimated average premium, model and event log are not consistent with each other (cf. Section 5.7.1, Constraint 4 page 85):

Constraint (Average value). *The estimated average value of the transfers in each business transaction of the value model is the same as the realized average value of this transfer in the event log for the specified time frame.*

Constraints for matching *number of occurrences* and *average value* of the transfers are in this chapter more precisely defined for implementation:

Constraint 8. *A value model is δ consistent at time t if for all exchanges x in the event log representing value transfers y in the value model it holds that:*

- (i) *the average number of realized x between time $t - \delta$ and t ($\delta > 0$) is equal to the number of estimated occurrences of y , and*
- (ii) *the average value of x between time $t - \delta$ and t ($\delta > 0$) is equal to the estimated value of y .*

Now, consistency checking is based on calculating averages over period of time δ . Model and event log are consistent with each other if the average value and the average number of occurrences are consistent during that period of time.

6.3 Implementation

δ -consistency forms the basis of our approach on relating value model and event log. We use data from the event log as feedback information for the value model. The value model is actively adapted during runtime using data perceived from the monitoring process. This approach provides a mechanism for monitoring the business from a value perspective and adapting the model accordingly.

Item (ii) of Constraint 8 addresses the average value of each transfer. The *average value* of each transfer is represented in the event log. The monitored value of transfers is compared with estimations made in the value model. This is described in Section 6.3.1.

Sections 6.3.2 and 6.3.3 concern item (i) of Constraint 8. The *event log* contains information about messages exchanged between actors. These messages, in turn, hold information on the number of value transfers that occur between actors. For using this information in evaluating the value model, a correlation between model and event log and their constructs is established. When constructing the value model, several estimations are made regarding behavior of the system like consumer needs and ratios on explosion elements. The estimated number of value transfer occurrences is based on these estimations. An *equation system* is derived which comprises formulas for calculating the number of occurrences based on used constructs in the value model. Data from the event log is entered into the equation system. When solving the equation system with these values there may be free variables representing the ratios. These are shown in a *Graphical User Interface*.

		2005				2006		Estimated Value Per transfer
		Q1	Q2	Q3	Q4	Q1	Q2	
X		45	50	47	52	60	58	47
Y		130	127	128	132	138	139	128

T1								
	T2							
		T3						

Figure 6.3: Example case: Realized and estimated average value of transfers

After adaptation of the *ratios* in the e^3 -value model, these results are *graphically* represented together with the result of monitoring *value* of the transfers. In the following sections, the different parts of the approach are explained in detail, and illustrated on behalf of our example business case.

6.3.1 Average value of transfers

The *value* of transfers is monitored in the event log. Figure 6.3 represents estimations made on the value of each transfer as well as the monitored average of each value transfer. We monitor the average value of each transfer by calculating the *moving average*. Since we assume for the given example the customer has constant behavior over time, we are able to use time series. Every quarter the average value of a transfer over the preceding year is calculated. Using a moving average results in a smoothly changing average over the time series. Figure 6.3 depicts measurements in quarter Q4 of 2005, quarter Q1 of 2006 and quarter Q2 of 2006, annotated with T1, T2 and T3, respectively. For calculating the realized average value of a transfer over a year, each quarter average is multiplied by the number of realized transfers, and the total is divided by the total number of realized transfers over that year.

6.3.2 Average number of transfers

During runtime, we monitor the process by calculating the *moving average* as depicted in Figure 6.4. In the figure, the *number* of message transfers as observed at different times during execution is depicted. In our example every quarter the averages over the preceding year are calculated. Figure 6.4 depicts measurements in quarter Q4 of 2005, quarter Q1 of 2006 and quarter Q2 of 2006, annotated with T1, T2 and T3, respectively. These message exchanges are correlated to value transfers in the value model.

6.3.3 The equation system

The equation system enables calculating the number of occurrences through formulas based on constructs in the value model. The equation system of a value model is de-

	2005				2006		Estimated Number Per year
	Q1	Q2	Q3	Q4	Q1	Q2	
X	240	170	210	230	190	270	900
Y	180	110	150	160	130	210	600

T1							
	T2						
		T3					

Figure 6.4: Example case: Realized and estimated average number of transfers

rived with an algorithm. The value model is represented as a graph where vertices represent constructs and edges represent parts of dependency paths. The equation system is constructed by assigning variables to each part of the dependency path between the different constructs of the value model. The equations in the system are related to each other through these variables. When two constructs are directly connected through a dependency path, the variable used in the equations of both constructs is the same. The algorithm is as follows:

Assign to each edge an unique variable. For each vertex determine the edges and associated variables. Represent each vertex as equation based on the equation system in Table 6.1. Instantiate the unique variables with the assigned values.

For each construct in the value model a formula exists in Table 6.1. For example, in Figure 6.1 there is one dependency path which enters an *OR-split* whereas two paths leave this element. For the fulfillment of a consumer need, only one of the outgoing paths is chosen. In the profitability sheets an estimation is made on the relation between the number of times each path is chosen. In the table this is indicated by r and s , stating path y is chosen r times, and path z is chosen s times. The resulting formula states the estimated number of times path y is chosen is equal to the number of times dependency path x occurs times the ratio on the OR-split, namely $\frac{r}{r+s}$. In e³-value an OR-port is an exclusive OR.

For the *OR-join* the number of occurrences of both incoming paths is added up.

For the *AND-join* as well as for the *AND-split*, the number of occurrences on each part of the dependency path is the same. An implosion or explosion element multiplies the number of occurrences with ratio $\frac{s}{r}$ associated with the port.

The number of occurrences on outgoing path x of a *start-stimulus* equals the number of occurrences of consumer need y times the number of actors z in the market segment.

The number of occurrences on *stop-stimulus* y equals the number of occurrences of incoming dependency path x times the number of actors z in the market segment. When the market segment consists of a single actor the value of z equals 1.


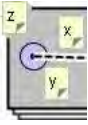
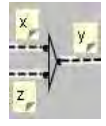
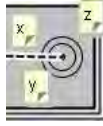
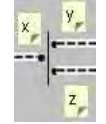
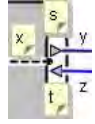

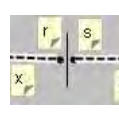
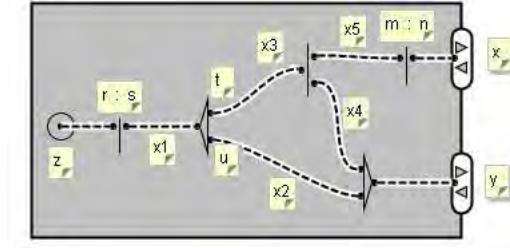
OR-split: $y = \frac{r}{r+s}x$ $z = \frac{s}{r+s}x$ 	Start: $x = yz$ 
OR-join: $y = x + z$ 	Stop: $y = \frac{x}{z}$ 
AND-split: $y = x$ $z = x$ 	Transfer: $y = sx$ $z = tx$ 
AND-join: $y = x$ $y = z$ 	Expl_impl: $y = \frac{s}{r}x$ 

Table 6.1: Implementation: Equation system

Formula *Transfer* in Table 6.1 states the outgoing number of value transfers y is equal to the number of occurrences x times ratio s , and the incoming number of value transfers z is equal to the number of occurrences x times ratio t . This ratio represents, for example, several payments for receiving one value object.

We illustrate our implementation by deriving the equation system for the customer. Figure 6.5 depicts the customer with ratios and introduced variables as used for the equation system. Since this example is for demonstration purposes only, we consider the customer to be a single actor and assume there is no ratio on the value interfaces. Next, the resulting equation system, without pure renaming, is represented. For a synoptic representation of the graphical user interface, the ratio on the first explosion element, $\frac{s}{r}$, is referred to as *fraction* f_1 and the ratios on the OR-split, $\frac{u}{t+u}$ and $\frac{t}{t+u}$, are referred to as *fractions* $1 - f_2$ and f_2 , respectively. The ratio on the second explosion element, $\frac{n}{m}$ is referred to as *fraction* f_3 .

- $x_1 = f_1 z$ with $f_1 = \frac{s}{r}$
- $x_2 = (1 - f_2)x_1$ with $1 - f_2 = \frac{u}{t+u}$
- $x_3 = f_2 x_1$ with $f_2 = \frac{t}{t+u}$
- $x_5 = x_3$
- $x_4 = x_3$
- $y = x_4 + x_2$

Figure 6.5: Example case: Customer ratios and introduced variables (e³-value notation)

	Estimated Numbers	T1	T2	T3
x	900	850	800	900
y	600	600	550	650
z	50	z	z	z
f_1	12	$\frac{600}{17}$	$\frac{550}{16}$	$\frac{650}{18}$
f_2	$\frac{3}{4}$	$\frac{17}{12f_3}$	$\frac{16}{11f_3}$	$\frac{18}{13f_3}$
f_3	2	f_3	f_3	f_3

Table 6.2: Implementation: Realized and estimated customer values

- $x = f_3 x_5$ with $f_3 = \frac{n}{m}$

6.4 Managing the value model

In the value model estimations are made for the number of consumer needs and different ratios of the constructs. Based on these estimations, the estimated number of value transfers is calculated. In Table 6.2 entries in: ‘*Estimated Numbers*’-column depict these estimations and calculations. Information from the event log is correlated with value transfers, and results in three different times, T1, T2, and T3, depicted in Table 6.2. The observed average of message exchanges during monitoring might deviate from estimations made in the value model. For example, measurements on the number of value transfers x at time T1, namely 850, deviate from the estimated number of value transfers x in the value model, namely 900. This result makes value model and event log δ inconsistent according to Constraint 8.

When the value model is not δ -consistent with the event log, one or more of the estimated ratios in the value model might be adapted to make regain consistency. Entering results of the event log into the equation system as derived from the value model while keeping ratios as free variables, and solving the equation system have two types of solutions.

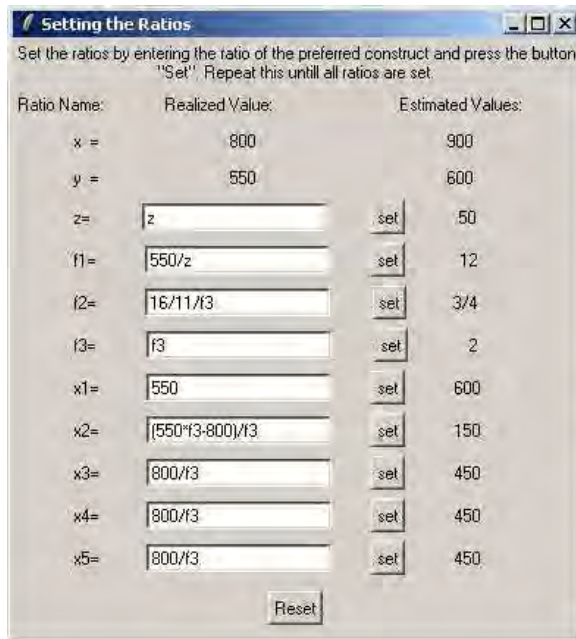


Figure 6.6: Proof-of-concept implementation: Graphical user interface

In the first case the equation system is *over-specified* and there is no or just one solution to the equation system, in which case there is no necessity for a graphical user interface. In the second case the equation system is *under-specified* leaving a possible infinite number of solutions. In the latter case, representing the options in a graphical user interface allows the user to dynamically set the different ratios and visualize the effects on the value model. The visualization is important for evaluating the economical value of the business during runtime.

An example of this graphical user interface is depicted in Figure 6.6. Here, results of the event log at time T2 are depicted. If the user chooses a free variable to set, and enters this value into the user interface, the equation system is reevaluated. The results, possibly still with free variables, are again represented in the graphical user interface. After setting all free variables, the ratios are adapted in such a way that value model and event log are one year consistent.

6.5 Visualization

In this chapter only one actor in a simplified business case is evaluated. Real-life business constellations, however, are more complex, and potentially consist of many constructs. Adapting one of the ratios in the value model might influence many other ratios and estimations in the value model. These effects are visualized by automatically adapting

colors of the constructs according to adaptation of the ratios. The user can directly see what the effects are of adapting one specific ratio on the entire business constellation.

Here, effects of entered values in the value model, compared to the original estimated values of the value model, are calculated, classified and represented by an appropriate color. In the example a construct is colored *green* if the entered or calculated ratio matches estimations in the value model. A construct is colored *dark green* if the ratio deviates less than 8.5% from the estimated value, and it is colored *red* if the ratio deviates over 8.5% from estimations in the value model.

In Figure 6.7 examples of value-model coloring for time series T2 from Figure 6.4 and Figure 6.3 are given. Figure 6.7 (a) shows the value model after entering results of T2. The average value of transfer x , a restitution, is in timeseries T2 52.16 (cf. Figure 6.3). This is more than 8.5% deviation from estimated value 47. Therefore, the lines of value transfer x are colored red. The average value of transfer y in timeseries T2, premium, is 131.33. This is less than 8.5% deviation, and, therefore, the lines of value transfer y are colored dark green. The average number of occurrences of x in T2 is 800, while the estimation in the value model for x is 900 (cf. Figure 6.4). Deviation between realized number of value transfers x and the estimated amount is greater than 8.5%, and, therefore, the interface is colored red. The realized number of y is 550, while the estimated number is 600. Deviation is less than 8.5%, therefore, the interface is colored dark green. The remaining variables are still open and, therefore, grey.

Figure 6.7 (b) depicts the situation after setting ratios for $f1$. $f1$ denotes the number of payments each customer makes for having insurance for one year. Since each customer pays every month its premium, this is a fixed ratio of 12. When choosing which ratios to adapt in the value model, $f1$ is not chosen since it is fixed. Therefore, this ratio is first set in the graphical user interface. After entering value 12 for variable $f1$, the explosion element colors green. This indicates the estimated ratio of 1 : 12 is equal to the realized ratio. As a consequence, the realized number of consumer needs, z , is 45.8. The estimated number of consumer needs is 50. The deviation is within 8.5%. Therefore, the start stimulus is colored dark green.

In Figure 6.7 (b) a possible final marking is depicted. If we assume there is additional information available on the percentage of customers that use the possibility of asking restitution in a month, ratio $f2$ can be set. We assume this ratio is 70%. The deviation between the estimated ratio of 75% and the realized ratio of 70% is lower than 8.5%. Therefore, the OR-port associated with $f2$ is colored dark green. As a result, $f3$ is assigned a value of approximately 2.1 which is also within the 8.5% deviation. The explosion element associated with ratio $f3$ is colored dark green.

The conclusion from these results is that less people ask for a restitution on a monthly basis, but if a customer asks for a restitution, he asks more restitutions than estimated. This is indicated by the slightly increased ratio on $f3$.

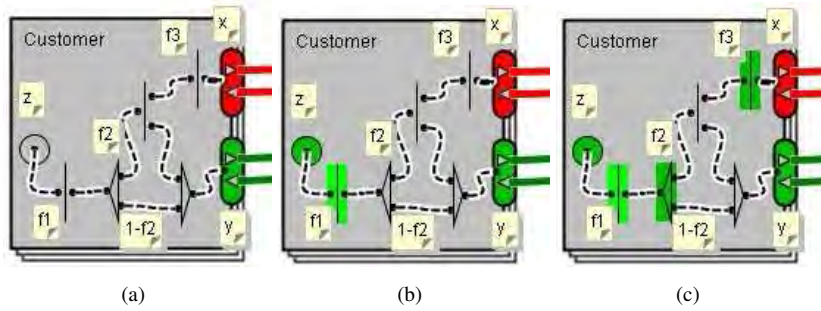


Figure 6.7: Proof-of-concept implementation: Coloring the value model

6.6 Evaluation of our developed management approach

For our consistency definitions we make some assumptions that influence the way consistency between models is maintained. These assumptions are fairly strict. For example, each business transaction in the business model must have a matching business transaction in the coordination model, and vice versa (cf. Constraint 1). This is a strict constraint because it only considers models to be consistent with each other if they describe the exact same situation, and nothing in addition. In other words, if model A describes all business transactions present in model B and an additional business constraint not present in model B, then these models are not considered to be consistent with each other according to Constraint 1. In practice, developers might not consider this an inconsistency since the models do not actually contradict. Model A simply contains more information than model B. Another example of strict consistency as defined in this chapter is consistency of models with the running system (i.e., intra-model consistency, cf. Constraints 3, 4, and 6). Here, models and running system are considered consistent with each other if and only if the running system behaves exactly as modelled. Slight deviations in, for example, selling costs or the order of messages, are considered an inconsistency. In real life, developers might choose to weaken these strict consistency rules by, for example, allowing slight deviations in behavior, or by allowing additional business transactions.

In Sections 6.4 and 6.5 we demonstrate the applicability of the developed approach using our method for managing inter-organizational models. To our knowledge, there do not exist any comparable approaches for managing business and coordination models. More particularly, there do not exist approaches that support runtime management of these models. Therefore, we conclude that our approach to managing business and coordination models is an improvement to existing related work.

6.7 Summary

In this chapter we show how consistency constraints between a model and running system are implemented and used to manage the model. More precisely, we show how to imple-

ment the intra-model consistency constraint of the value model, and how to monitor this constraint at runtime using the event log. In addition to implementation of the constraints and monitoring of the model, we also depict how to use this information for managing the model at runtime. More particularly, we show how to visualize deviations between estimations in the value model and runtime results in the event log. By adding to this the intra-model dependencies of the value model, we enable dynamic adaptation of the model while directly showing consequences of these adaptations for other parts in the model.

Part IV

SCENARIO 2: SERVICE LEVEL AGREEMENTS FOR COMPOSITE SERVICES

MANAGING DEPENDENCY RELATIONS: SERVICE COMPOSITIONS

For a business operating in a networked environment it is vital to accurately manage services it provides to its customers. This is particularly challenging if a company offers *composite* services where interactions with services offered by other providers influence its performance. The quality of service (QoS) that can be offered to customers is calculated taking all these dependencies into account [63, 33]. Consider a composite service which returns combined information from several search engines. In this case, the quality of service (e.g., response time) that can be offered depends on the quality of service delivered by the search engines. Together with constraints the customer has regarding the service, these calculations form the basis for a Service Level Agreement (SLA) between customer and service provider [66, 100].

Several approaches exist for *monitoring* the service level during runtime (e.g., Tomic et al. [109]). Monitoring results are compared with constraints specified in the SLA for possible violation detection. Since SLAs are typically bilateral agreements, current monitoring approaches (e.g., Sahai et al. [100], Keller et al. [66], Tomic et al. [109]) focus on identifying violations in bilateral communication. Important research questions in this area are, for example, how to gather reliable data, how to structure these data, and how to combine data from different sources. Such monitors are often process-specific, activated or created if a service is invoked, and terminated if the service is completed.

Most approaches for managing composite services combine the level of quality a company provides, with monitoring bilateral communication. However, to properly manage its composite service a company has to reason about *causes* of SLA violations. For example, if the offered response time for a composite service provided by a company depends on response times of other services this company uses, it is vital to identify and monitor these *dependencies*. Exactly these dependencies are ignored in bilateral monitoring approaches. However, combining bilateral monitoring results based on their dependencies is, as we discuss in this chapter, highly challenging. With our MaDe4IC method for managing dependency relations in inter-organizational models (cf. Chapter 4) we develop the *MoDe4SLA approach* (MONitoring DEpendency relations for SLAs) [23]. With this approach, we analyze during development phase different types of *dependencies* between

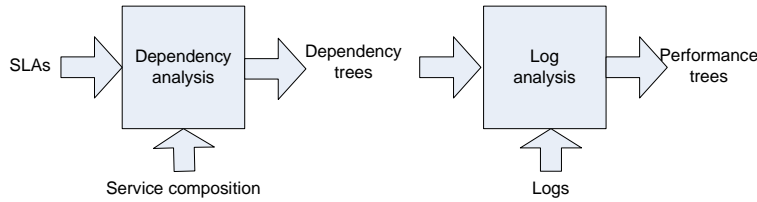


Figure 7.1: Overview: MoDe4SLA approach

services, and the *impact* these services have on each other. Further, it allows to combine bilateral monitoring results with analyzed dependencies and impact services have on the composition.

With MoDe4SLA we allow monitoring dependencies between SLAs, enabling decision support when managing composite services. Figure 7.1 gives an overview of MoDe4SLA. We first analyze *dependency relations* (cf. Step 3 in Section 7.4). The relations are used to perform an *event log analysis* (cf. Step 8 in Section 7.9) to determine *causes* for SLA violations in composite services.

The remainder of this chapter first discusses *basics* of SLAs and introduces our running example in Section 7.1. In the following sections we discuss every step in applying our MaDe4IC *method* to our running example (cf. Sections 7.2-7.10). We conclude this chapter with a discussion of *related work* on managing services and SLAs in Section 7.11.

7.1 Basics

In this section we discuss basics on SLAs and on the business case we use to illustrate our approach.

7.1.1 Service Level Agreements

Our approach aims at supporting a company in managing its composite services by identifying and monitoring their dependencies to services requested from other providers. These dependencies are explicitly represented in a *dependency model*. SLAs describe constraints on the service. For example, there might be a constraint on response time of a service. For each of these constraints in the SLA of a composite service, services it depends on are identified. An SLA typically consists of a set of *Service Level Objectives* (SLOs) which contain guaranteed quality constraints [66, 100]. A typical example of such an SLO is:

In 90% of all cases, invocation of service X will have a response time within y milliseconds (ms).

Each of these SLOs is *measurable*, and consists out of one or more *metrics*. Such metric may be composite as well, e.g., a composite metrics might be the average response

time over all customers in a month. Furthermore, these SLOs typically hold a validity *time constraint*, for example, “Mo-Fri 9:00-17:00”.

Offering a composite service to customers implies a company relies on content providers to offer necessary data. Both customers and content providers have an SLA with the company. We propose to explicate for each SLO on which other services its overall performance *depends* (cf. Figure 7.1). Different SLOs in one SLA might depend on different services, or depend on them in different ways. For example, if a company offers information with fast response time by querying five providers, and returning information of the fastest responding one, a cost constraint is influenced by all five services (due to invocation they all have to be paid), while a response time constraint is only influenced by the fastest responding service.

Furthermore, we demonstrate how to calculate the *impact* a service has on a depending composite service. Assume, for example, the cost constraint on a composite service depends on service A and service B, where service A costs on average ten times more than service B. Now, the cost impact of service A on the composite service is ten times higher than the one of service B. Based on the dependency model we calculate the impact a service has on the composition. We do this by means of an impact analysis.

Data on messages exchanged between customer and provider are gathered in *event logs*. These event logs enable monitoring of SLOs and their dependencies based on these data. We abstract and structure necessary data from event logs to enable evaluation and monitoring of SLOs. For decision support in managing composite services, we combine dependencies, calculated impact factors and bilateral monitoring results into one model that graphically represents these relations. Next, we show how to accomplish this, using the MaDe4IC method described in Chapter 4.

7.1.2 Business case

We demonstrate our approach by means of an intuitive example in which a company offers two composite services to its customers. *SubscribedNews* is a composite service where customers automatically receive a news report. This report is created by combining news items on personal interest (e.g., stock market information) from two different news providers. Customers pay a flat fee for this service. *NewsRequest*, in turn, is a composite service which allows customers to request up-to-date news information on a specific search query. The *NewsRequest* service is paid per invocation. For every request the company invokes two content providers, and sends information from the fastest responding to the customer.

To offer the *SubscribedNews* service to its customers, the company automatically receives data (i.e., news items) from its content providers (i.e., news providers). This data is stored in a *database* from which the company retrieves data when composing the news report for its customers. The company has an SLA with both content providers and customers. Furthermore, providing the news report for customers does not trigger a service invocation by the company to the content providers for data, since up-to-date data is retrieved from the database. Therefore, obtaining data from content providers and sending

SLA <i>SubscribedNews</i> , Company & Customer. January 1, 2008 - June 30, 2008
SLO-cost: Company will deliver SubscribedNews on a monthly average of less than 1 euro per service.
SLO-time: Company will deliver SubscribedNews once a day before 9:00 am, Mo-Fri.
SLA <i>NewsUpdate</i> , Company & CP1. January 1, 2008 - June 30, 2008
SLO-cost: CP1 will deliver NewsUpdate for 0.50 euro per news item, with a maximum of 50 euro per day.
SLO-time: CP1 will deliver NewsUpdate to Company once a day, 7 days a week before 6:00 am.
SLA <i>NewsUpdate</i> , Company & CP2. January 1, 2008 - June 30, 2008
SLO-cost: CP2 will deliver NewsUpdate for 0.30 euro per news item, with a maximum of 39 euro per day.
SLO-time: CP2 will deliver NewsUpdate to Company once a day, 7 days a week before 6:00 am.

Table 7.1: SLAs for the SubscribedNews service

reports to customers are different processes (i.e., invocations). However, response time performance for offering NewsRequest is highly dependent on response time of the service provided by the content providers. As a result, SLA violations between the company and its providers might result in SLA violations in the service provided by the company to the customer, which cannot be identified using a bilateral monitoring approach. For example, if the content provider never meets the response time constraint, the company most likely will also not be able to meet the response time it agreed upon with the customer. Therefore, it is highly important for a company to not only monitor SLA violations for each metric, but also their dependencies on the same metrics in other SLAs to identify *causes* for SLA violations.

A composite service depends on one or more other services. In our example, SubscribedNews and NewsRequest both depend on two other services offered by different content providers. In our approach, we specify on *which* services the fulfillment of a specific SLO depends and *how* it depends on these services. The SLOs for cost and response time, specified in SLAs of the company with its content providers (CP1 and CP2) and customers are depicted in Tables 7.1 and 7.2. Note that we do not provide a specification language for SLAs (like WSLA Framework [66] or the SLA language by Sahai et al. [100]). Instead our approach focusses on conceptual issues, i.e., it is language-independent.

From Monday to Friday before 9:00 am ,the company delivers news items the customer is interested in. The price is not higher than 1 euro, and depends mainly on the number of news items that fit the requirements of the customer. With its content providers the company agreed to deliver daily all news items for a fixed price per item with a maximum of, respectively, 50 and 39 euro per day. These news items are delivered before 6:00 am. Regarding the NewsRequest service the customer can request news on a specific topic. These requests costs 0.50 euro. If the customer has more than 100 requests per

SLA <i>NewsRequest</i>, Company & Customer. January 1, 2008 - June 30, 2008
SLO-cost: Invocation <i>NewsRequest</i> by Customer will cost the first 100 times in a month 0.50 euro, thereafter 0.40 euro.
SLO-time: Company will respond to <i>NewsRequest</i> invocation by Customer within 5 ms, 99% of the time.
SLA <i>Request</i>, Company & CP1. January 1, 2008 - June 30, 2008
SLO-cost: Invocation <i>Request</i> by Company will cost 0.20 euro.
SLO-time: CP1 will respond to <i>Request</i> invocation by Company within 3 ms, 99.9% of the time.
SLA <i>Request</i>, Company & CP2. January 1, 2008 - June 30, 2008
SLO-cost: Invocation <i>Request</i> by Company will cost 0.15 euro.
SLO-time: CP2 will respond to <i>Request</i> invocation by Company within 4 ms, 99% of the time.

Table 7.2: SLAs *NewsRequest*

month, price per invocation decreases to 0.40 euro. In 99% of all cases a request is responded to within 5 ms. The company, in turn, invokes services of two content providers for every *NewsRequest*, and sends data from the fastest responding provider to its customer. Content provider CP1 responds within 3 ms for 0.20 euro to an invocation by the company in 99.9% of all cases. Content provider CP2 responds within 4 ms for 0.15 euro to an invocation by the company in 99% of all cases.

We use this business case to illustrate the steps of applying our method to SLAs in the following sections.

7.2 Step 1: Model analysis

In Step 1 of our modelling method we analyze models for their characteristics. First, recall the model characteristics identified in this method (cf. Section 4.3):

(i) Focus:	Viewpoint	↔	Partial model
(ii) Perspective:	Single actor	↔	Bird's eye view
(iii) Property type:	Estimation	↔	Prescription
(iv) Time frame:	Instance-based	↔	Period of time

By applying this analysis to our business case we obtain the following results. Each SLA contract constitutes a *partial model* that has a bird's eye view. Furthermore, each contract specifies an agreement on costs and response time which are both *prescribed* property values. Each contract describes a period of time, namely January 1, 2008 - June 30, 2008:

	SLAs
Focus	partial model
Perspective	bird's eye view
Property type	prescription
Time frame	period of time

7.3 Step 2: Homogenization

In Step 2 we homogenize the models (i.e., the SLAs) on three levels (syntactic, semantic, and pragmatic) in order to make them comparable.

Syntactic homogenization. The SLAs are written in natural language and have a similar structure. Commonly used characteristics are:

- name of the service,
- issuer and recipient of the service,
- period of time, and
- set of SLOs.

Each SLO, in turn, either considers *cost* or *response time*, and typically contains the following characteristics:

- period of time, and
- property value (i.e., costs or response time).

Each SLO connects the considered period of time and the property value using constructs like:

- less than,
- maximum of,
- before time, and
- per instance.

Semantic homogenization. The models in this chapter are built in such a way that they are semantically homogeneous by construction.

Pragmatic homogenization. With pragmatic homogenization we consider five different aspects. Some of them are homogenized, while for others heterogeneity is identified in this step and handled in the following steps (cf. Section 4.3, Step 2).

- *Focus.* The focus of each SLA is a partial model focus. Therefore, the models are homogeneous concerning their focus.

- *Perspective*. Each perspective is a bird's eye perspective. Therefore, the models are homogeneous concerning their perspective.
- *Granularity*. The models are specified on the same level of granularity by construction.
- *Time frame*. Each contract is defined for the same period of time, namely, from January 1, 2008 - June 30, 2008. Furthermore, most time frames within the SLOs are compatible. For example, in SubscribedNews, each cost SLO is defined per service. However, the cost SLOs of NewsRequest do not match. More specifically, the cost SLO for invoking NewsRequest by the customer is defined for the first 100 services, and for every following service separately. Both cost SLOs for invoking service Request by the company are defined per service. As discussed, such heterogeneity cannot be homogenized. Therefore, we handle this by considering *average cost* of the service. For example, if 150 NewsRequests are done by the customer, the average cost of one invocation is as follows: $\frac{100 \times 0.50 + 50 \times 0.40}{150} = 0.47$.
- *Property type*. Both cost and response time are agreed upon and therefore forced values. There is no need for homogenization.

The main results of Step 2 are the following:

- Time frame differences between SLOs are solved by considering average costs of each service invocation,
- All models (i.e., SLAs) are syntactically and semantically homogeneous.

7.4 Step 3: Inter-model relation detection

In Steps 1 and 2 we analyze the models to prepare them for comparison. In Step 3 we detect dependency relations between the different SLAs. These relations allow us to define consistency constraints between the models in Step 4.

As discussed in Step 3 in Section 4.4.2, the first step in identifying inter-model relations between partial models is to identify their interconnections. Each partial model describes a different part of the cooperation. For example, there is a relation between the SLA for SubscribedNews and the two SLAs for NewsUpdate. This common model describing how the different model parts are related, is in this specific context the *service composition*.

The different SLAs are connected through their *properties* (i.e., SLOs). Each SLA describes both *cost* and *response time* properties. Therefore, relations between SLAs (i.e., inter-model relations) are *property dependencies*. Since the news report created when invoking SubscribedNews is dependent on both NewsUpdate services while these services are not dependent on SubscribedNews, there exists an *asymmetric* property dependency between the services. The same holds for the NewsRequest service that depends on both Request services.





#	Construct	Explanation
1		Service A depends on service B
2		AND-split/join
3		XOR-split/join
4		Reuse is X times

Table 7.3: Cost dependency model constructs

Next, for *cost* and *response time* we analyze separately the dependency relations.

Cost dependency relations. Both examples (i.e., *SubscribedNews* and *NewsRequest*) contain one SLO depicting a *cost* constraint (cf. Table 7.1 and 7.2). Here, for both examples we construct the *cost model*. We introduce a domain-specific modelling language for expressing cost dependencies that enables depicting all necessary details while keeping a high level of abstraction. This enhances readability and decreases necessary modelling effort. Our language uses the constructs depicted in Table 7.3. We model that meeting a constraint for one service *depends* on performance of another service (i.e., property dependence). Using the *AND-split* it is possible to model meeting constraints for one service depends on two or more different services. Furthermore, by using *XOR-splits* exclusive choices are modelled (i.e., the service depends on one out of a set of services) with the possibility to add a *ratio* on the likelihood of choosing one of the options. As last modelling construct it is possible to denote *reuse* of data provided by a service. Such a reuse construct depicts how often results of one service invocation are reused.

Figure 7.2 shows on *which* services the cost of *SubscribedNews* and *NewsRequest* depend, and *how* these dependencies look like. The cost of composite service *SubscribedNews* is dependent on the cost of *NewsUpdate* provided by CP1 and CP2, indicated with an AND-split (cf. Figure 7.2). Moreover, *NewsUpdate* data is used more than once (i.e., the company sends more than once the same data to different customers). This *reuse* is estimated to occur 100 times for data from CP1 and 80 times from CP2 (cf. Figure 7.2). Note that data from CP1 are more often reused than data from CP2 because the former are also used in another unrelated service offered by the company.

The cost of composite service *NewsRequest* is dependent on costs of both *Request* services provided by CP1 and CP2. This is again indicated with an AND-split (cf. Figure 7.2). So, even though the customer only receives data from one service provider, he pays for fast delivery, namely the cost of invoking two content providers. Data for a news request are not reused.

Response time dependency relations. *SubscribedNews* and *NewsRequest* both contain an SLO with a *time* constraint (cf. Tables 7.1 and 7.2). Next, we introduce the domain-specific *response time model* for both examples. Apart from the constructs used in modelling cost dependencies, also the *order* in which services are executed is relevant

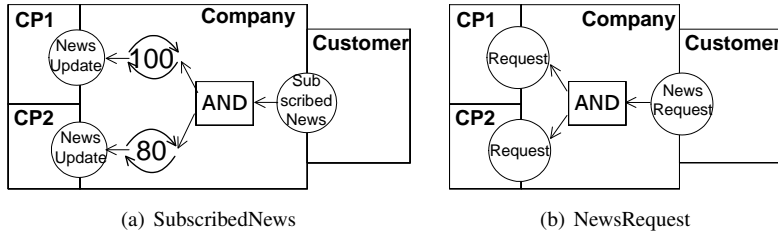


Figure 7.2: SubscribedNews and NewsRequest: Dependency cost model

#	Construct	Explanation
1		Service B depends on service A. Also: serial execution of services.
2		AND-split: parallel execution.
3		XOR-split.
4		Database (DB) with reuse of data.

Table 7.4: Response time dependency constructs

for response time. For example, if services on which the composite service depends are executed in parallel, response time is faster compared to serialized execution. Workflow modelling techniques (e.g., YAWL [4]) are highly suitable for modelling response time dependencies, allowing for the ordering of messages. In this chapter, we use Coloured Petri Nets (CPN) [64] for defining these models. Table 7.4 depicts the requirements for Petri Nets. The first (1) construct in the table depicts how serialization is modelled. By serializing the execution of two services (e.g., service A precedes service B) we express that service B depends on service A. In other words, service B is only executed after service A. The second (2) construct depicts how we model parallel execution of two or more services. The third (3) construct depicts how we model an exclusive choice operator (XOR-split) using Petri Nets. The fourth (4) construct depicts modelling a database. This construct allows to store information created when invoking a service. The information is reused in other services. We illustrate how requirements are modelled. Note that by using Petri Nets these might be modelled in different ways.

Response time for SubscribedNews is dependent on NewsUpdate services of both CP1 and CP2 (cf. Figure 7.3). The CPN depicts how CP1 and CP2 send data (i and j, respectively) that are then stored in the database of the company (k and l, respectively). Both content providers send their identification (CP_id) together with content (i.e., news items, C_id). The company adds this information to the database while inserting a primary

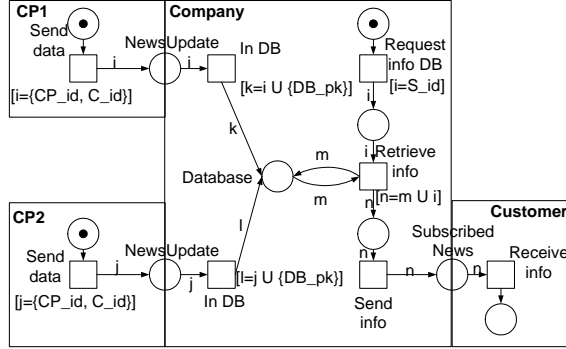


Figure 7.3: SubscribedNews: Response time dependency model

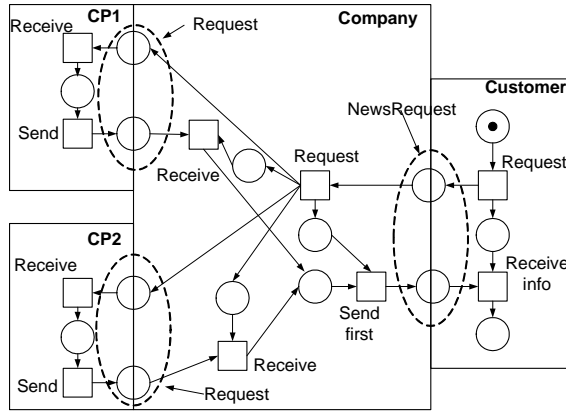


Figure 7.4: NewsRequest: Response time dependency model

key (DB_pk). Independent from filling the database, the company sends daily updates to customers with specific interests by requesting data from the database (S_id). This retrieved data is combined with the specific interests of the customer ($n = m \cup i$). Data used to compose a news report for the customer are reused for other reports. Therefore, information requested from the database is also returned (back edge annotated with m). The resulting report is sent to the customer.

Response time dependencies for NewsRequest are dependent on response times of Request services invoked by the company to both content providers (cf. Figure 7.4). For readability purposes we omit coloring the Petri Net. As soon as the company receives a response from one of the two content providers, it sends out this information to the customer (Send first). This enables faster response times for the customer.

In Step 3 we identify inter-model dependency relations for the different SLAs of the composition for both response time and cost.

The main results of this step are:

- Identification of asymmetric property dependency between cost SLOs of the different SLAs, and
- Identification of asymmetric property dependency between response time SLOs of the different SLAs.

7.5 Step 4: Inter-model consistency constraints

In Step 3 we identify inter-model dependencies of SLAs using the overall composition of the business case. In Step 4, we now use these asymmetric property dependency relations to define inter-model consistency constraints.

We start with formulating a consistency constraint for each identified dependency. According to the description of Step 4 in Section 4.4.2 every relation is translated into a constraint:

If concept x is **asymmetric property dependent** on set Y of one or more concepts, and z is the predicate describing this relation, the corresponding constraint states: Property value of x relates to property values of Y according to predicate z .

Cost dependency. Composite service `SubscribedNews` (x) is cost dependent on set `NewsUpdate` services (Y). The *AND-construct* with *reuse* describes their relation (z). Therefore, we now define the consistency constraint concerning cost for the `SubscribedNews` service as follows:

Constraint 9 (`SubscribedNews` cost). *The cost property value of `SubscribedNews` depends on both (i.e., AND-construct) cost properties of the `NewsUpdate` services, divided by 100 and 80, respectively (i.e., reuse). The cost property value of `SubscribedNews` corresponds at least to the sum of costs for both `NewsUpdate` services divided by 100 and 80, respectively.*

For the `NewsRequest` service, we define a similar constraint:

Constraint 10 (`NewsRequest` cost). *The cost property value of `NewsRequest` depends on both (i.e., AND-construct) cost properties of the `Request` services, where its property value is at least the sum of costs of the `Request` services.*

Response time dependency. Composite service `SubscribedNews` is response time dependent on both `Request` services. The response time property of `SubscribedNews` is at least the response time property of the *slowest* `NewsUpdate` service. Therefore, we define the consistency constraint as follows:

Constraint 11 (SubscribedNews time). *The response time property value of SubscribedNews depends on both response time properties of the NewsUpdate services, where the property value corresponds at least to the value of the slowest NewsUpdate service.*

Since composite service NewsRequest responds as soon as the fastest responding Request service succeeds, its response time dependency is formulated as follows:

Constraint 12 (NewsRequest time). *The response time property value of NewsRequest depends on both response time properties of the Request services, where its property value corresponds at least to the response time value of the fastest responding Request service.*

Next, we generalize over these consistency constraints. This generalization enables easy consistency checking.

7.5.1 Generalization: Impact factors

As discussed in Step 4 of Section 4.4.2 we generalize over consistency constraints of partial models that have an asymmetric dependency relation. We do this by building *trees* that connect the different services through their dependencies. In this chapter, we refer to such trees as *impact trees*. In addition to structural trees, we derive a formula describing the exact relation between composition and its separate services. This formula shows the *impact* each service has on the composition concerning a certain property. We refer to this as *impact analysis*.

The dependency model for an SLO expresses on which services a composite service depends. However, not every service a composite service depends on, has the same *impact* on this composite service. For example, if costs for a composite service depend on services A and B, but service A costs on average only ten percent of service B, impact of service B on the price of the composite service is much higher than impact of service A. Therefore, our approach enables an *impact analysis* for every dependency model.

The impact a service has on the composite service is determined by analyzing the dependency model. Each construct (e.g., AND-split) in the dependency model affects the impact on the composite service in a different way. For example, an AND-split in a cost dependency model denotes that costs for a composite service are influenced by both services it depends on. Opposed to this, an XOR-split indicates the costs for the composite service are influenced by only one of the two services it depends on (i.e., a service only has an impact on part of the composite service instances), which decreases the impact of these services by 50% (i.e., each service only occurs in half of the composite service instances). This impact factor indicates how likely it is the service influences performance of the composite service.

To calculate this impact factor, we first go through the models. We start with the composite service, and then go through all constructs finishing with the services. Second, we consider the consistency constraints. By considering the effects of each construct on the impact while going through the model, we calculate the impact of the service. To analyze the model in a structured way we create an *impact tree*. For each construct in the models a vertex with its impact equation is created. This allows analysis of the impact











Graph	Equation	Cost	Response Time
	$I(A) = xAr$ $y = x$		
	$y = x$ $x = z$		All serial executions
	$y = rx$ $z = sx$		All parallel executions
	$y = 0$		All databases and storages
	$y = \frac{x}{RE}$		

Table 7.5: Impact tree vertices

each construct and service has on the composition. Table 7.5 denotes the *graph notation* and *equation* for all constructs of cost and response time models. The total impact of the services on the composition is always 1.

The impact factor of service A ($I(A)$ for short) on a composite service is equal to factor x on the incoming edge times the value of service A times number of loops r (cf. Table 7.5). Value A is either the estimated cost or the estimated response time. The number of loops r is important if one service gets invoked more than once for one composite service. For example, if for a composite service the company invokes the same service A of its content provider twice ($r = 2$) with a response time of 3 ms before responding to a request of its customer. If service A results in the invocation of another service (serialism) the outgoing edge (y) has the same impact factor as the incoming one (x) (cf. Table 7.5).

The second construct is an *AND-split* where outgoing edges (y and z) have the same impact factor as the incoming edge (x). The third construct is an *XOR-split* where estimated ratio $r:s$ determines the impact of edges y and z . Assume that for fulfilling a composite service, a company chooses between two content providers. The first offers fast, but expensive data, the second offers slow, but cheap data. Depending on the customer, the company chooses either of the two. Estimations by the company are that it will choose 3:1 ($r:s$) for the fast service.

The fourth construct enables modelling a *database* or *storage*. If the results of a service are stored in a database or, with physical goods, in a storage then the dependency on response time for the composite service disappears. Therefore, the impact of services adding information or goods to a database or storage on the composite service regarding response time is zero.

The last construct in Table 7.5 enables modelling reuse of data acquired when invoking a service. Reuse of information from a service decreases the costs for using, and, therefore, it decreases the cost impact factor of that service. If information is reused, the cost for invoking this service is divided among the requesting services. Therefore, the impact factor for costs is divided by the estimated number of reuses.

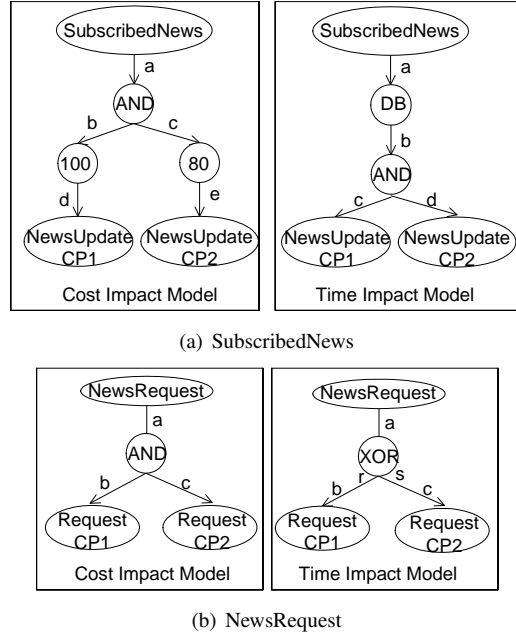


Figure 7.5: SubscribedNews and NewsRequest: Impact trees

Figure 7.5 depicts impact trees for both response time and cost of the SubscribedNews service as derived from the SLA constraints. For *cost* the service depends on both UpdateNews services with a reuse element in between to mitigate the influence. The NewsUpdate service from CP1 is expected to be reused 100 times, while the NewsUpdate service from CP2 is expected to be reused 80 time on average. Figure 7.5 also depicts the impact trees for *response time* and *cost* of the NewsRequest service. The impact factor for response time is now influenced by the estimated ratio $r:s$ on the XOR-split.

Using the equations in Table 7.5 and both graphs in Figure 7.5, cost and time impact factors are calculated and depicted in Table 7.6. Note that average costs of NewsUpdate CP1 cannot directly be derived from the SLO (cf. Table 7.1) since it states 0.50 euro per item with a maximum price of 50 euro per day. The same holds for NewsUpdateCP2. Now, the company makes an estimation on the average value based on previous observations: $\text{NewsUpdateCP1} = 40$ and $\text{NewsUpdateCP2} = 30$.

For the impact on response time for NewsRequest, estimations are made on the XOR-split ratio $r:s$. Since RequestCP1 responds on average faster than RequestCP2, ratio $r:s$ is estimated to be 3:1. Recall that the agreed upon response times by CP1 and CP2 for Request are 3 and 4 ms, respectively. The impact factor is an absolute value, i.e., if an impact value of a service is 5 in Model A and 10 in Model B, the impact of that service on the composite service is twice as big in Model B as in Model A.

The equations of the impact factors must hold for the inter-model consistency constraints to succeed. In other words, the equations are general consistency constraints for

Impact Factor	Equation	Result
$I_{cost}(NewsUpdateCP1)$	$\frac{1}{100}40$	0.4
$I_{cost}(NewsUpdateCP2)$	$\frac{1}{80}30$	0.375
$I_{time}(NewsUpdateCP1)$	0	0
$I_{time}(NewsUpdateCP2)$	0	0
$I_{cost}(RequestCP1)$	0.20	0.20
$I_{cost}(RequestCP2)$	0.15	0.15
$I_{time}(RequestCP1)$	3 times 3	9
$I_{time}(RequestCP2)$	1 times 4	4

Table 7.6: Supporting services of SubscribedNews and NewsRequest: Impact factors

cost and response time properties of NewsRequest and SubscribedNews.

The main results of this step are the following:

- Definition of impact factor equations that constitute the inter-model consistency constraint for cost and response time properties of both NewsRequest and SubscribedNews,
- Graphical notation of the impact tree, visualizing influence of each service on the composition.

7.6 Step 5: Intra-model relation detection

In Step 3 we analyze inter-model relations to create a basis to define consistency constraints between SLAs. In this step we analyze existing intra-model dependencies between the SLOs. This provides the basis for Step 6 in which we define intra-model consistency constraints.

In our business case there exists a clear separation between properties cost and response time. For example, if the costs change, this will not have an effect on the response time. Therefore, in our business case there are no intra-model dependencies between the different SLOs. In other words, a constraint violation for response time does not affect the consistency constraint for cost, and vice versa. However, we analyze possible dependencies between SLOs within an SLA. In the remainder of this step, we discuss intra-model dependencies that are common in real-life SLAs. Consider the following SLA:

Every month, response time will be within 3 ms for at least 99% of the invocations. Costs are 3 euro when the service responds within 2 ms, while a response time of 2 – 3 ms costs 2 euro. If less than 99% of the invocations have response time within 3 ms, a penalty of 1000 euro will be paid.

In this SLA, cost directly depends on response time. The challenge is to represent what this means in respect of the relative contribution to the composition by the cost performance of the service. For example, to diagnose the cause of an increase in cost for the service composition, we show the influence of decreasing response times to the costs (i.e., costs rise to 3 euro).

To solve this, within one SLA dependencies *between* SLOs are represented by links between properties. These links are annotated with some contribution function, similar to the calculations we do for the inter-model dependencies. Our goal is to represent the causal influence between properties as accurately as necessary for meaningful diagnosis. For example, in the above example it may be sufficient to represent that a decrease in response time causes an increase in cost. This leads to a clear step when handling SLAs with intra-model relations:

Step 5a: Extend the computation of the impact tree and impact values with *dependencies between properties* within one SLA.

A second elaboration that occurs in real-life SLAs is the imposition of violation *penalties*. An example is the above SLA, where 1000 euro are paid in case the response time requirement is not met. Such penalty constraints are frequently used in SLAs. We could add this as additional cost, with a dependency on the performance of other attributes. However, this ignores the special status of penalties. As a consequence, separate penalty impact trees with dependencies on the SLOs in the SLA need to be created. Therefore, a second step in handling intra-model dependencies is the following:

Step 5b: Extend the computation of impact tree and impact values with *penalties* for SLA violations.

Based on these two intra-model dependencies, it is possible to construct an intra-model impact tree for each SLA where dependencies between SLOs and penalties are denoted. Typically, these dependencies are *asymmetric property dependencies*.

The main results of Step 5 are the following:

- Conclusion that no intra-model dependencies exist in our business case,
- Discussion on possible intra-model dependency relations within SLAs:
 - Dependencies between SLO properties like response time and cost,
 - Dependency of penalty costs on SLO properties like a penalty for slow response times.

7.7 Step 6: Intra-model consistency constraints

In Step 6 we explicate intra-model consistency constraints, using dependency relations identified in Step 5. These constraints enable consistency checking of models with the

running system. As discussed for Step 6 in Section 4.5, intra-model consistency constraints are (1) based on identified *intra-model dependency relations* and (2) on *model characteristics*.

The intra-model consistency constraints ensure the model (i.e., the SLA) describes the inter-organizational cooperation properly (i.e., that the agreement is respected). These constraints hold if at runtime the behavior of the system is the same as the behavior captured in the model. We check this by gathering information from event logs that describe behavior between actors, and compare this realized behavior with the constraints in the SLAs. Therefore, the intra-model consistency constraints are defined using event logs.

The business case considered in this chapter does not have intra-model dependencies, i.e., there are no dependencies between SLOs within one SLA. However, in Step 5 we discuss that in SLAs from real life there often exist such dependencies. Typically, these are *property dependencies*, i.e., dependencies between properties of concepts. Using the constraint definition from Section 4.4.2 we formulate the following consistency constraints for asymmetric and symmetric property dependency relations, respectively:

If an SLO is **asymmetric property dependent** on another SLO within the same SLA, where z is the predicate describing this relation, the constraint states: Property value of the SLO relates to property value of the other SLO according to predicate z .

Reconsider for example the case from the previous step:

Every month, response time will be within 3 ms for at least 99% of the invocations. Costs are 3 euro if the service responds within 2 ms, while if the service has a response time of 2 – 3 ms, costs are 2 euro. If less than 99% of the invocations have response time within 3 ms, a penalty of 1000 euro is paid.

Here, the property *cost* is asymmetric property dependent on property *response time* according to the predicate that *costs are 3 euro if the service responds within 2 ms, while if the service has a response time of 2 – 3 ms, costs are 2 euro*. As a result, the consistency constraint for this SLA is as follows:

For each service composition invocation the event log information is expected to contain that property *cost* relates to property *response time* according to the following statement: *costs are 3 euro if the service responds within 2 ms, while if the service has a response time of 2 – 3 ms, costs are 2 euro*

The consistency constraint for symmetric property dependency relation is as follows:

If two SLOs are **symmetric property dependent** on each other, and z is the predicate describing this relation, the constraint states: The property values of the SLOs are related according to predicate z .

In addition to the consistency constraints based on intra-model dependencies, there exist *model-specific* constraints. Here, model-specific constraints are formulated according to content of the SLAs. For example, in our business case the SLA for SubscribedNews depicts constraints on both cost and response time:

Constraint 13 (SubscribedNews 1). *Over a one month period, the event log contains entries that the company delivered SubscribedNews on average for less than 1 euro per service.*

Constraint 14 (SubscribedNews 2). *The event log contains entries that the company delivered SubscribedNews once a day before 9:00 am, Mo-Fri.*

The main results of Step 6 are as follows:

- Defining intra-model consistency constraints for SLAs based on intra-model dependencies, and
- Defining intra-model consistency constraints for SLAs based on their general content.

7.8 Step 7: Dependency analysis

In the previous steps we identify consistency constraints between SLAs and event log, i.e., we define inter- and intra-model consistency constraints. Both model analysis and consistency constraints are described in an intuitive manner. In Step 7 we formalize these steps to enable automatic analysis of service compositions. This analysis, in turn, results in a set of *formal models*.

We use *trees* to represent dependencies in and between SLAs, and *algorithms* to describe the analysis. Each service composition is represented as *composition tree*. Next, per SLO we analyze the tree and construct for each SLO (e.g., for cost and response time) an *expected impact tree* (i.e., the impact tree with impact factors from Step 4). We start with constructing the composition tree, after which we derive expected impact trees.

To illustrate the use of these algorithms and construction of trees, we use a more complex and abstract service composition. This example is depicted in Figure 7.6 where the composition consists of an OR-split with discriminative join (ORDISC). Here, 3 out of 4 branches are invoked and the construct succeeds after the fastest two invoked branches succeed. Each branch indicates the invocation chance compared to its siblings. The first branch is an AND-split and AND-join where Web services WS 1 and WS 2 are invoked in parallel. The second branch consists of a single Web service WS 3, and the third branch is an XOR-split where either WS 4 (chance is 0.4) or WS 5 (chance is 0.6) is invoked. The fourth branch consists of WS 6. Each service has an agreed upon average response time and cost attribute described in its SLA.

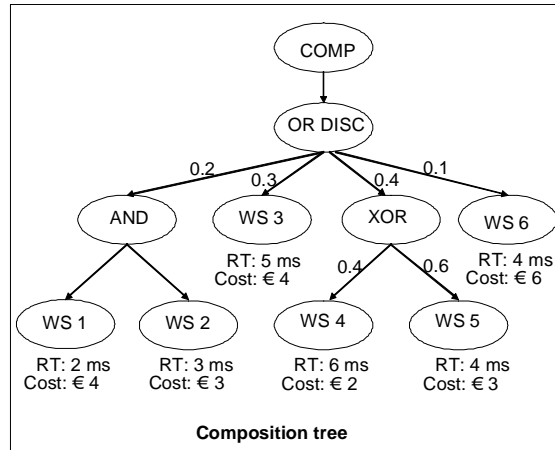


Figure 7.6: Illustrative service composition

Composition	Response Time	Cost
AND	max(total)	sum(total)
ANDDISC	max(subset)	sum(subset)
OR	max(subset)	sum(subset)
ORDISC	max(subset)	sum(subset)
XOR	max(one)	sum(one)
Loop	sum(total*)	sum(total*)
Sequence	sum(total)	sum(total)

Table 7.7: Matching composition vertices and vertices for dependency trees

7.8.1 Composition tree

At design time we analyze dependencies of the composition on its underlying services. For the construction of service compositions, we consider the most commonly used workflow patterns [1] as constructs. Table 7.7 shows the type of relations between services per SLO (response time and cost) and per construct. The constructs are here discussed informally, but are in the algorithms described in more detail. Although there are more SLOs to consider (e.g., availability), Table 7.7 suffices to demonstrate the principles behind our approach. An AND-split (cf. AND in Table 7.7) succeeds after the slowest responding service finishes (i.e., the maximum response time of all branches: $\max(\text{total})$). Its total cost is the sum of all invoked services (i.e., $\sum(\text{total})$). For a parallel AND-split with discriminative join (ANDDISC), a parallel OR-split with a discriminative join (ORDISC), and a parallel OR-split with normal join, the response time is determined by the slowest invoked service (i.e., the maximum response time of a subset of all invoked services: $\max(\text{subset})$). Its costs correspond to the sum of costs of all invoked services (i.e., $\sum(\text{subset})$). For sequences (Sequence) and for sequences executed more than once (Loop), both response time and cost are summed up for all invoked services (i.e., $\sum(\text{total})$ and $\sum(\text{total}^*)$, where $*$ indicates the number of iterations). For an XOR-split and join the invoked service contributes to both response time and cost (i.e., the response time and cost of the started service: $\max(\text{one})$ and $\sum(\text{one})$).

By considering these constructs and their properties we determine the expected impact of each service on the overall composition for a specific SLO (i.e., we calculate the function connecting the SLOs of different services as done informally in Step 4). A service composition is modelled as a tree where the top vertex represents the service composition (COMP) and the leafs represent Web services (WS). Connecting vertices and edges depict the composition structure. This is the same structure as used in Step 4 in the context of impact trees.

Estimations on the number of invocations are captured by annotating vertices and edges with estimated or agreed upon values. Vertices representing Web services are annotated with the agreed upon SLO values (σ). Vertices representing constructs are annotated with estimated behavior, if appropriate (μ). Annotations are (1) the number of started services (for OR-constructs), (2) the number of discriminative success, i.e., after how many successful responses the construct succeeds (for DISC-joins), and (3) the number of iterations (for loop-constructs). Edges are annotated with the probability they are invoked for each service composition invocation (ρ). For example, two edges leaving an XOR vertex are annotated with the probability they are chosen (i.e., this is an estimation). All trees are *acyclic*. *Composition trees* are defined as follows:

Definition 8 (Composition tree). *Let \mathcal{V}_s be the set of types for service vertices $\{WS, COMP\}$ and let \mathcal{V}_c be the set of structural vertices $\{AND, ANDDISC, OR, XOR, ORDISC, LOOP, SEQ\}$. A composition tree is a 6-tuple $CT(\mathcal{V}_c, \mathcal{V}_s) = (V, E, \rho, \mu, \tau, \sigma)$, with*

- V is a set of vertices,
- $E : V \rightarrow V$ is a set of directed edges,

- $\rho : E \rightarrow \mathbb{R}$ the probability of invocation compared to its siblings,
- $\tau : V \rightarrow \mathcal{V}_c \cup \mathcal{V}_s$ specifies the vertex type,
- $\sigma : \{v \in V \mid \tau(v) \in \mathcal{V}_s\} \mapsto \mathbb{R}^n$ specifies the expected SLO values for each SLO, where n indicates the number of SLOs,¹
- $\mu : \{v \in V \mid \tau(v) \in \mathcal{V}_c\} \mapsto (\mathbb{R} \cup \mathbb{R})^3$ vertices are annotated with (1) number of started services, (2) number of discriminative success, and (3) number of iterations.

Now, the composition tree of our illustrative example (cf. Figure 7.6) is denoted as follows:

$$\begin{aligned} \mathcal{CT}_{example} = (\\ & V : \{COMP, ORDISC, AND, XOR, WS1, WS2, WS3, WS4, WS5, WS6\}, \\ & E : \{e1 = \{COMP, ORDISC\}, e2 = \{ORDISC, AND\}, e3 = \{ORDISC, WS3\}, \\ & \quad e4 = \{ORDISC, XOR\}, e5 = \{ORDISC, WS6\}, e6 = \{AND, WS1\}, \\ & \quad e7 = \{AND, WS2\}, e8 = \{XOR, WS4\}, e9 = \{XOR, WS5\}\}, \\ & \rho : \{(e1, 1), (e2, 0.2), (e3, 0.3), (e4, 0.4), (e5, 0.1), (e6, 0.2), (e7, 0.2), (e8, 0.4), (e9, 0.6)\}, \\ & \mu : \{(ORDISC, (3, 2, \epsilon))\}, \\ & \tau : \{(COMP, COMP), (ORDISC, ORDISC), (AND, AND), (XOR, XOR), (WS1, WS), \\ & \quad (WS2, WS), (WS3, WS), (WS4, WS), (WS5, WS), (WS6, WS)\}, \\ & \sigma : \{(WS1, (2, 4)), (WS2, (3, 3)), (WS3, (5, 4)), (WS4, (6, 2)), (WS5, (4, 3)), \\ & \quad (WS6, (4, 6))\} \end{aligned}$$

Next, we discuss how to calculate expected impact trees.

7.8.2 Expected impact tree

After deriving the composition tree from the Web service composition, we calculate the expected impact trees. These trees depict expected behavior of the services. Hereby, the focus is on the expected impact each service has on the composition. This expected impact is for each SLO calculated separately.

Based on the composition tree, expected runtime behavior is calculated resulting in an *expected impact tree*. This expected behavior constitutes the formalization of inter-SLO consistency constraints where trees are built that show dependency between different services (cf. Step 4). For an SLO such an expected impact tree depicts the expected impact of a service per composition invocation, i.e., the *impact factor*. In addition, every edge in the tree has a *contribution factor* that depicts the average number of times a subtree contributes per invocation.

Definition 9 (Impact factor). *Let $X \rightarrow Y$ be two vertices with a connecting edge of an impact tree for a specific SLO. If vertex Y is an input service, the impact factor of Y denotes the average contribution to the composition for the SLO value. The contribution is defined as percentage of the composition SLO value.*

¹We assume that all vertices are annotated with the same tuple of SLOs.

For example, assume the impact factor of a service on the composition is 0.5 for response time. Then the average contribution of this service to the overall response time of the composition is 50%.

Definition 10 (Contribution factor). *Let $X \rightarrow Y$ be two vertices with a connecting edge of an impact tree for a specific SLO. The contribution factor on the edge (\rightarrow) denotes the average number of times subtree Y contributes per composition invocation. (Note that this value is less than or equal to 1 unless the subtree contains a loop structure.)*

For example, assume the contribution factor of a service to the composition is 0.5 for response time. Then the average number of times this service contributes to the composition is 50% regarding response time.

Recall the semantics of the vertices in Table 7.7, explained at the beginning of Section 7.8.1. *Expected impact trees* are defined as follows:

Definition 11 (Expected impact tree). *Let \mathcal{V}_s be the set of service vertices $\{WS, COMP\}$ and let \mathcal{V}_i be the set of dependency vertices: $\{max(total), max(subset), max(one), sum(total), sum(subset), sum(one), sum(total^*)\}$. An expected impact tree is a 5-tuple.*

$\mathcal{EIT}(\mathcal{V}_i, \mathcal{V}_s) = (V, E, \rho, \tau, \sigma)$, where

- V is a set of vertices,
- $E : V \rightarrow V$ is a set of directed edges,
- $\rho : E \rightarrow \mathbb{R}$ is the probability of contribution per composition invocation,
- $\tau : V \rightarrow \mathcal{V}_i \cup \mathcal{V}_s$ specifies the type of the vertex,
- $\sigma : \{v \in V \mid \tau(v) = WS\} \mapsto \mathbb{R} \times \mathbb{R}$ Annotations of Web service vertices are in the first dimension “estimated impact”, and in the second dimension “expected SLO value”,
- $\sigma : \{v \in V \mid \tau(v) = COMP\} \mapsto \{1\} \times \mathbb{R}$ annotation of composed services specifies estimated SLO value based on composition structure. We use \mathbb{R} instead of $\{1\} \times \mathbb{R}$ for brevity in the remainder of this chapter.

The expected impact tree supports identification of services with high influence on composition behavior. The type of considered SLO determines the transformation algorithm. For example, consider the example from Figure 7.7 representing a service composition where each run invokes two services in parallel: either S1 and S2, or S1 and S3. Intuitively, expected impact for S1 is $1 \cdot 10ms = 10$ since it is invoked every invocation (i.e., 1), and responds in 10 ms. S2 has an impact of $0.5 \cdot 20ms = 10$, and S3 of $0.5 \cdot 15 = 7.5$, assuming both have 50% chance of being chosen per invocation.

However, in practice structure (i.e., expected number of invocations) and performance (i.e., SLO value) are not sufficient to describe the realized impact. It can be expected that most times, S1 finishes before the other service (faster response time). Therefore, S1 usually does not contribute to overall response time since this is done by the longer

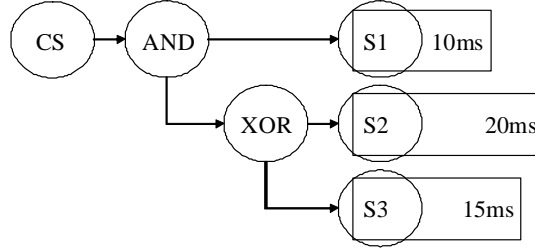


Figure 7.7: Small example: Service composition

running service. In fact, the *setting* in which the services run (e.g., running parallel with high response time services) should be taken into account as well. As a consequence, each invoked service has an impact on composition costs. However, only the slowest responding service influences composition response time. To make this explicit, we create an expected impact tree for each monitored SLO (e.g., one for response time and one for costs). In other words, the structural behavior of response time and cost differ, hence, this different behavior requires different algorithms.

We only introduce algorithms for response time, and only parts of the algorithm are provided formally (pseudo code): the remaining parts are described informally since the formalization is lengthy, and does not contribute to the clarification of our approach. For example, the calculation code is not shown for OR-split with discriminative join because the code is longer, and the general principle is shown for the AND case. The goal of the expected impact tree is threefold:

- Estimate the behavior of the overall composition based on both contracts (SLAs) with the different service providers and the structure of the composition (i.e., calculate the σ -value).
- Estimate the impact (e.g., on response time) of each subtree on the overall composition (i.e., calculate the ρ -value for the edges).
- Estimate the impact (e.g., on response time) of each Web service on the overall composition (i.e., calculate the σ -value for the Web service).

All these estimations are based on contracts with the service providers, and on the structure of the composition. Both expected QoS and structure determine estimated probability that a service is *invoked*. As described, invocation of a service does not necessarily mean it actually *contributes* to, for example, the response time (e.g., in branches running parallel, only the longest running one has an impact). Therefore, Function 1 $\text{calc}(v)$ determines the probability a service gets invoked and it actually contributes to the overall composition.

Function 1 $\text{calc}(v)$. Vertices V and edges E of the expected impact tree are equal to vertices V and edges E of the composition tree. The *structure* of both trees is the

Function 1 $\text{calc}(v)$

input : $v \in V$ & Composition Tree $CT = \{V, E, \rho_{CT}, \mu_{CT}, \tau_{CT}, \sigma_{CT}\}$
output : Expected average rt of the composition & Expected Impact Tree
 $EIT = \{V, E, \rho, \tau, \sigma\}$

```

1  switch  $\tau_{CT}(v)$  do
2      case  $COMP$ 
3           $\tau(v) = COMP;$ 
4           $e_{out} = v \rightarrow v_y \in E;$ 
5           $\rho(e_{out}) = \rho_{CT}(e_{out});$ 
6           $\sigma(v) = \text{calc}(v_y);$ 
7          return  $\sigma(v);$ 
8      case  $AND$ 
9           $\tau(v) = \max(totalI);$ 
10          $rt_{max} = 0;$ 
11          $v_{max} = v;$ 
12          $e_{in} = v_y \rightarrow v \in E;$ 
13         foreach  $e_{out} = v \rightarrow v_y \in E$  do
14              $\rho(e_{out}) = \rho(e_{in});$ 
15              $rt_y = \text{calc}(v_y);$ 
16             if  $RT_y > RT_{max}$  then
17                  $RT_{max} = RT_y;$ 
18                  $V_{max} = v_y;$ 
19         foreach  $v \rightarrow v_y \in E \setminus \{v \rightarrow V_{max}\}$  do  $\text{reassign}(v_y, 0);$ 
20         return  $rt_{max};$ 
21     case  $WS$ 
22          $\tau(v) = WS;$ 
23          $e_{in} = v_y \rightarrow v \in E;$ 
24          $\text{impact}(\sigma(v)) = \rho(e_{in}) \cdot rt(\sigma_{CT}(v));$ 
25          $rt(\sigma(v)) = rt(\sigma_{CT}(v));$ 
26         return  $rt(\sigma_{CT}(v));$ 

```

Function 2 $\text{reassign}(v, z)$

input : $v \in V$ and $z = \rho$ of incoming edge v

```

1   $e_{in} = v_y \rightarrow v \in E;$ 
2   $\rho(e_{in}) = z;$ 
3  switch  $\tau_{CT}(v)$  do
4      case  $XOR$ 
5          foreach  $e_{out} = v \rightarrow v_y \in E$  do  $\text{reassign}(v_y, \rho(e_{in}) \cdot \rho_{CT}(e_{out}));$ 
6      case  $AND$ 
7          foreach  $v \rightarrow v_y \in E$  do  $\text{reassign}(v_y, \rho(e_{in}));$ 

```

same, though *annotation* and *naming* of vertices and edges differ. The function traverses recursively through the composition tree, starting with the composition (*COMP*) vertex. Its calculations are divided into five steps.

1. When traversing the tree the *name of each vertex* (τ -value) is determined by its original type (cf. Function 1 `calc(v)` in Line 3, 9, & 22). For example, a parallel split is named `max(total)` in the expected impact tree for response time (cf. Table 7.7).
2. The *probability an edge gets invoked* (cf. Function 1 `calc(v)`, ρ value assignment in Line 5 & 14) is determined by combining its local probability with the probability its parent edge gets invoked. Now, each Web service “knows” *locally* its probability for each composition invocation to be invoked.
3. Each vertex determines its *expected average response time* based on the expected response times of its children (i.e., the expected response times of services invoked in that subtree). For example, in Lines 13-15 of Function 1 `calc(v)`, an AND vertex determines its expected response time based on the maximum response time of all its children.
4. Each vertex determines which children branches have an impact if they are chosen (i.e., *expected contribution*). For example, assume an AND vertex has one fast responding child compared to the other children. This child, most likely, does not contribute to the composition response time since it finishes before the rest (cf. `calc(v)`, Lines 16-18). Now, each vertex has *global* knowledge on the expected behavior of its children.
5. This global information is propagated through the tree, annotating each branch with the *probability it contributes* to the composition per invocation (cf. Function `reassign(vy, n)` invoked by `calc(v)` in Line 19).

As discussed above, we describe the calculation for OR-split with discriminative join informally. For each subset s of branches from the OR-split, we calculate likelihood of invocation l , and the expected response time of the subset. Since it is a discriminative join, expected response time depends on the fastest responding subset s' . For example, if four out of five branches are started, and three need to finish for the discriminative join to succeed, the theoretical minimum response time can only be the response time of the third-quickest service. Comparable calculations are done for the remaining vertex types.

Using this algorithm, the *response time impact tree* and the *cost impact tree* are calculated. A graphical representation of the trees of our illustrative example is depicted in Figure 7.8.

The response time impact tree from Figure 7.8(a) depicts expected average impact of each service on the response time of the composition. For the OR-split with discriminative join we calculate the estimated response time of each branch. For each subset of three branches (recall that three branches out of four are invoked) we calculate the invocation chance based on the estimated chance to be chosen on the branches. With the average

response time for each branch and the chance for each subset of branches to be invoked, we determine which branch, most likely, is the second to finish for each subset. Recall that the ORDISC-construct succeeds after the second branch responds. Now, we determine the average response time for the ORDISC based on the expected response time of each subset of three services and the invocation chance of that subset. For example, the first and second branch are expected never to be the second responding branch, therefore, their expected impact is 0. On average, the third branch with the XOR-split is expected to determine response time in 57% of the composition invocations. Recall from Figure 7.6 that the chance to be invoked for WS 4 and WS 5 is 0.4 and 0.6, respectively. Therefore, on average, WS 4 contributes $0.4 \cdot 0.57 = 0.23$ times to the overall response time when the composition is invoked. The impact of WS 4 (i.e., the average contribution to the total response time) is 0.31. This value is calculated using the average response time of WS 4 (6ms), the average response time of the composition (4.46 ms), and the number of times the WS is expected to contribute to the response time of the composition (0.23): $\frac{6}{4.46} \cdot 0.23 = 0.31$.

The cost impact tree in Figure 7.8(b) depicts the expected average contribution of each Web service to the composition costs (i.e., the impact factors). Furthermore, it depicts how often a service is expected to contribute to the costs per composition invocation (i.e., the values on the branches). For costs it holds that each invoked service is paid. Therefore, ORDISC results in the summation of costs of a subset of its outgoing branches (in this case 3) (sum(subset)). For each subset of three branches we calculate the invocation chance. For each branch we determine the average invocation chance by summing up the chances of each subset where the service is contained. Results are depicted on the outgoing branches, and add up to 3 since each invocation of the composition results in the invocation of 3 branches that all contribute to the cost of the composition. For the XOR-split in the third branch we calculate the invocation chance for WS 4 and WS 5 ($0.4 \cdot 0.8 = 0.32$, and $0.6 \cdot 0.8 = 0.48$, respectively). Therefore, these Web services contribute to the composition cost. The expected impact factor of each service is again calculated by dividing average costs of the service (e.g., 4 euro for WS 1) with average composition costs (€14.47). We multiply this with the chance the service actually contributes (0.73): $\frac{4}{14.47} \cdot 0.73 = 0.20$.

The formal notation of the impact trees of our example is as follows:

7.8. STEP 7: DEPENDENCY ANALYSIS

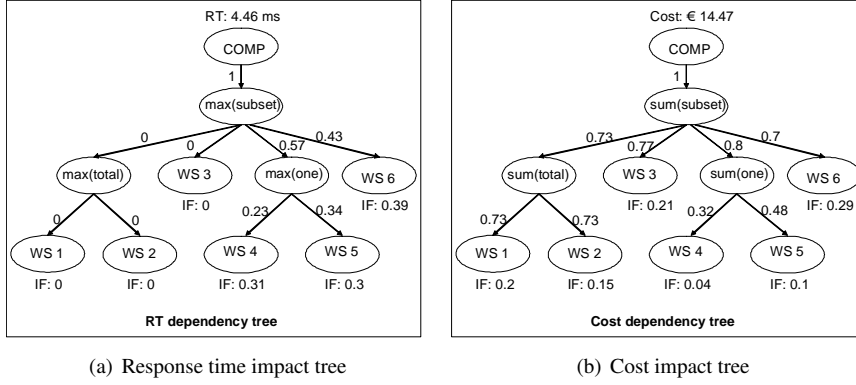


Figure 7.8: Illustrative example: Estimated impact trees

$$\begin{aligned}
 \mathcal{EIT}_{cost} = (\\
 &V : \{COMP, sum(subset), sum(total), sum(one), WS 1, WS 2, WS 3, WS 4, WS 5, WS 6\}, \\
 &E : \{e1 = \{COMP, sum(subset)\}, e2 = \{sum(subset), sum(total)\}, \\
 &\quad e3 = \{sum(subset), WS 3\}, e4 = \{sum(subset), sum(one)\}, e5 = \{sum(subset), WS 6\}, \\
 &\quad e6 = \{sum(total), WS 1\}, e7 = \{sum(total), WS 2\}, e8 = \{sum(one), WS 4\}, \\
 &\quad e9 = \{sum(one), WS 5\}\}, \\
 &\rho : \{(e1, 1), (e2, 0.73), (e3, 0.77), (e4, 0.8), (e5, 0.7), (e6, 0.73), (e7, 0.73), (e8, 0.32), \\
 &\quad (e9, 0.48)\}, \\
 &\tau : \{(COMP, COMP), (sum(subset), sum(subset)), (sum(total), sum(total)), \\
 &\quad (sum(total), sum(total)), (WS 1, WS), (WS 2, WS), (WS 3, WS), (WS 4, WS), \\
 &\quad (WS 5, WS), (WS 6, WS)\}, \\
 &\sigma : \{(WS 1, 0.2), (WS 2, 0.15), (WS 3, 0.21), (WS 4, 0.04), (WS 5, 0.1), (WS 6, 0.29)\})
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{EIT}_{rt} = (\\
 &V : \{COMP, max(subset), max(total), max(one), WS 1, WS 2, WS 3, WS 4, WS 5, WS 6\}, \\
 &E : \{e1 = \{COMP, max(subset)\}, e2 = \{max(subset), max(total)\}, \\
 &\quad e3 = \{max(subset), WS 3\}, e4 = \{max(subset), max(one)\}, e5 = \{max(subset), WS 6\}, \\
 &\quad e6 = \{max(total), WS 1\}, e7 = \{max(total), WS 2\}, e8 = \{max(one), WS 4\}, \\
 &\quad e9 = \{max(one), WS 5\}\}, \\
 &\rho : \{(e1, 1), (e2, 0), (e3, 0), (e4, 0.57), (e5, 0.43), (e6, 0), (e7, 0), (e8, 0.23), (e9, 0.34)\}, \\
 &\tau : \{(COMP, COMP), (max(subset), max(subset)), (max(total), max(total)), \\
 &\quad (max(total), max(total)), (WS 1, WS), (WS 2, WS), (WS 3, WS), (WS 4, WS), \\
 &\quad (WS 5, WS), (WS 6, WS)\}, \\
 &\sigma : \{(WS 1, 0), (WS 2, 0), (WS 3, 0), (WS 4, 0.31), (WS 5, 0.3), (WS 6, 0.39)\})
 \end{aligned}$$

For this example, we derive that response time of the composition is influenced by WS 4, WS 5, and WS 6. Furthermore, we see that they have comparable impact. Concerning the composition cost, we derive that WS 4 and WS 5 have a comparable low impact, and

WS 6 contributes to almost 30% of the cost. Furthermore, we see for response time that WS 6 contributes in 43% of the invocations, and that WS 4 and WS 5 contribute in 23% and 34% of the invocations, respectively. For the costs, WS 1, WS 2, WS 3, and WS 6 all contribute in 70 – 80%, and WS 4 and WS 5 only in 30 – 50% of the invocations.

The main results of Step 7 are as follows:

- Formalization of the service composition with SLAs,
- Formalization of the expected impact trees for SLOs (i.e., formalization of the consistency constraints), and
- Algorithm for automatic translation from composition tree into expected impact trees.

7.9 Step 8: Log analysis

In the previous steps we analyze SLAs, define consistency constraints, and formalize SLA analysis. In Step 8 we demonstrate how we analyze event logs such that we can check consistency constraints at runtime.

At runtime we gather monitoring data from event logs. These logs contain information needed to check whether the composition is behaving according to the SLA specifications. We *abstract* this information from the event logs, and refer to this as the *event log abstraction*. Each Web service invocation is an entry in the event logs, and we need to recognize to which composition invocation it belongs. For example, if a Web service invocation is responded to, then this response is stored as entry in the event log. We discuss which information is necessary to monitor SLAs.

The challenge is to abstract *useful* information, and to *structure* it in a meaningful way. Focus of our monitoring approach is on the different SLOs of a composite service. Therefore, we capture information concerning the SLO (e.g., timestamps to measure response times and payment information for cost calculations). In addition, we capture data to correlate different messages. We accomplish this correlation by analyzing different time stamps on the service invocations in combination with the service composition structure. To enable abstracting necessary data from the event logs, we need to correlate the following information:

1. Messages exchanged between provider and customer during invocation of the composite service.
2. Services (and their messages) invoked by the composite service, i.e., the services a composite service depends on. This is achieved by either matching identifiers (as available, for example, with a BPEL implementation) or by doing semantic matching.

3. Instances of services belonging to one specific type of service (e.g., all instances of a premium account).
4. Grouping classes of services belonging to one business activity (e.g., all premium and normal account instances of service X).
5. Instances of services that are exchanged with the same business partner (e.g., all instances of services exchanged with content provider A).
6. Composite services that reuse data from the same service. Again this information is correlated through identifiers or by semantic matching.

To meet the identified correlation criteria we structure our event log abstraction in the following manner. Each *composition invocation* is represented as a list of invoked Web services for that composition instance:

- Issuer of the composite service,
- Type of service issued,
- Costs of the service,
- Set of messages exchanged with the issuer, and
- Set of service instances this composite service depends on.

Each *service* invoked in the composition is a 4-tuple containing a time stamp (*ts*), the name of the invoked Web service (*ws*), its costs (*costs*), and its response time (*rt*). For brevity, we denote timestamps in relative milliseconds instead of complete date and time notation.

An example of one instance of NewsRequest by a customer in event log abstraction notation is as follows.

<i>Composite Service:</i> CS(id1)=(CustomerX, NewsRequest, 0.50, {M(id2),M(id3)}, {S(id4), S(id5)})
<i>Services:</i> S(id4)= (M(id6), M(id7), 0.20) S(id5)= (M(id8), M(id9), 0.15)
<i>Messages:</i> M(id2)= (1.0, CustomerX, Company, "Dutch politics") M(id3)= (5.5, Company, CustomerX, ContentY) M(id6)= (2.0, Company, CP1, "Dutch politics") M(id7)= (4.2, CP1, Company, ContentY) M(id8)= (2.1, Company, CP2, "Dutch politics") M(id9)= (5.1, CP2, Company, ContentZ)

Here, composite service NewsRequest CS(id1) is issued by CustomerX for the price of 0.50 euro. Between the company and CustomerX, two messages M(id2) and M(id3) are exchanged, and the two services S(id4) and S(id5) are used by the company to fulfill the composite service. Both services S(id4) and S(id5) consist of two messages, and cost 0.20 and 0.15 euro, respectively. Each message M(idX) consists of a timestamp, issuer, recipient, and content of the message. In the given case, the customer wants an update on Dutch politics. Both CP1 and CP2 deliver news items (ContentY and ContentZ). The company sends ContentY since this is the first to arrive ($4.2 \text{ ms} < 5.1 \text{ ms}$).

7.10 Step 9: Causal analysis

In the previous steps we formalize SLA analysis to identify dependencies, and to use them to calculate *estimated behavior* of the service composition. In addition, we abstract necessary information from the event log. This makes it possible to compare estimated behavior in the dependency analysis with realized behavior described in the event log. The comparison allows us to identify SLA *violations* as well as possible *causes* for these violations. Identification of causes for violations allows developing a management strategy that minimizes the number of SLA violations.

For this, we first discuss how to construct *realized impact tree* that depicts what the realized service levels for the composition are. Secondly, we discuss how to compare estimated and realized impact trees, and we create a *feedback tree* that shows deviations between the two.

7.10.1 Realized impact tree

We use analysis of dependencies between services during development phase, and analysis of the impact these services have on the composition, in order to support monitoring composite services at runtime. The event log abstraction, containing results of bilateral communication, is compared with the SLOs. For example, if the SLO states response time is on average 3 ms, achieved average response time is easily calculated using timestamps. However, opposed to traditional monitoring approaches, we also consider SLO values of other invoked services in the composition during comparison. As a result, not only bilateral SLOs are assessed: a complete picture is provided on how the composite service is performing. The resulting model is referred to as the *realized impact tree*, and is defined as follows:

Definition 12 (Realized impact tree). *Let \mathcal{V}_s be the set of service vertices $\{WS, COMP\}$ and let \mathcal{V}_i be the set of dependency vertices $\{max(total), max(subset), max(one), sum(total), sum(subset), sum(one), sum(total^*)\}$. A realized impact tree is a 5-tuple $\mathcal{RIT} = (\mathcal{V}_i, \mathcal{V}_s) = (V, E, \rho, \tau, \sigma)$, where*

- V is a set of vertices,
- $E : V \rightarrow V$ is a set of directed edges,
- $\rho : E \rightarrow \mathbb{R}$ the average contribution per composition invocation,

- $\tau : V \rightarrow \mathcal{V}_i \cup \mathcal{V}_s$ specifies the vertex type,
- $\sigma : \{v \in V \mid \tau(v) = WS\} \mapsto \mathbb{R} \times \mathbb{R}$ annotations of Web service vertices are in the first dimension: total contributed SLO value, and in the second dimension: total number of contributions,
- $\sigma : \{v \in V \mid \tau(v) = COMP\} \mapsto \mathbb{R} \times \mathbb{R}$ annotations of the composition vertex is in the first dimension: total realized SLO value, and in the second dimension: total number of invocations.

The goal of the realized impact tree is threefold:

- Determine realized behavior of the service composition over a specific period of time (i.e., calculate σ),
- Determine realized impact each subtree has on the overall composition (i.e., ρ -value of the edges), and
- Determine realized impact of each Web service on the composition (i.e., σ of the Web service vertices).

Algorithm 3: Realized impact. Vertices V and edges E of the realized impact tree are equal to vertices and edges in the composition. In other words, we assume the structure of the service composition is the same. Algorithm 3 shows implementation for response time. The code is only depicted for a subset of vertex types.

As argued before, not every Web service invocation contributes to the SLO value of the composition. Therefore, Algorithm 3 *analyzes all entries* in the event log abstraction (Line 3, Algorithm 3). Furthermore, it determines which entries (i.e., which Web service invocations) impact the composition based on composition structure and other service performance. For this, recursive Function $\text{addWS}(v)$ is invoked in Line 5 of Algorithm 3. For example, the XOR-split determines which children (cf. Line 20 of Function $\text{addWS}(v)$) contribute to the overall response time (cf. Line 22-26), and returns all contributing Web service invocations in the subtree (cf. *confirm*, Line 27).

These entries are added to the *confirmed* list (Line 6, Algorithm 3). Furthermore, all confirmed entries are used to *update* total response time and total number of invocations (i.e., σ -values) of the services (Lines 7-10, Algorithm 3).

As a last step, Algorithm 3 determines for each edge the contribution per composition invocation (i.e., ρ -value) by invoking recursive Function $\text{calcImpact}(v_{comp})$ in Line 11. This function determines the number of contributions per composition invocation for each edge (i.e., its ρ -value). Web service leafs calculate ρ -value of the incoming edge (cf. Line 16-19 of Function calcImpact) by comparing number of leaf vertex invocations with total number of composition invocations. For example, if the composition is invoked 6 times, and the leaf vertex contributes 3 times, it contributes on average 0.5 times to each composition invocation. Each structural vertex combines information of its outgoing edges, and determines the ρ -value of the incoming edge. For example, in an AND-split

the overall contribution of the subtree is the summation of ρ -values of its outgoing edges (cf. Lines 11-14).

As a result, a realized impact tree is created for each SLO, similar to the estimated impact trees (cf. Figure 7.8).

Algorithm 3: Realized impact

<p>input : Log File Model L and Composition Tree $CT = (V, E, \rho_{CT}, \mu_{CT}, \tau_{CT}, \sigma_{CT})$ output : Realized Impact Tree $EIT = (V, E, \rho, \tau, \sigma)$</p> <pre> 1 $confirmed = \emptyset;$ 2 $v_{comp} = \{v \mid \tau_{CT}(v) = COMP\};$ 3 foreach $l \in L$ do 4 $initially = l;$ 5 $(cf, rt, ts) = addWS(v_{comp});$ 6 $confirmed = confirmed \cup cf;$ 7 foreach $tuple \in confirmed$ do 8 $v = ws(tuple);$ 9 $rt(\sigma(v)) = rt(\sigma(v)) + rt(tuple);$ 10 $contribution(\sigma(v)) ++;$ 11 $\rho = calcImpact(v_{comp});$ </pre>
--

7.10.2 Feedback tree

The feedback tree depicts deviations from agreed upon SLA values by comparing estimated and realized impact trees. Colors on edges and vertices are used to visualize these deviations. Currently, red, green, yellow, darkgreen, and colorless are used (i.e., θ -values), but these colors can be extended or changed in any preferred way. Intuitively, red and yellow represent negative deviations, while green and darkgreen represent positive deviations.

The goal of the feedback tree is to support management in identifying causes for badly performing compositions. We accomplish this by giving feedback on:

1. Deviation between expected and realized behavior of the composition regarding a particular SLO (i.e., its θ -value),
2. Deviation between expected and realized contribution of each subtree to the SLO of the composition,
3. Deviation between expected and realized contribution of each Web service in the composition, and
4. Realized contribution per invocation of Web services (i.e., σ -value) and subtrees (i.e., ρ -value).

Function 4 addWS(v)

input : $v \in V$
output : ($confirm, rt, ts$): the set of contributing tuples $confirm$, with its overall rt , and the ts of the first started tuple $\in confirm$

```

1   $confirm = \emptyset$ ;
2   $rt = 0$ ;
3   $ts = Max$ ;
4  switch  $\tau_{CT}(v)$  do
5      case  $COMP$ 
6           $v \rightarrow v_{child} \in E$ ;
7           $(confirm', rt', ts') = addWS(v_{child})$ ;
8           $rt(\sigma(v)) = rt(\sigma(v)) + rt'$ ;
9           $invoc(\sigma(v)) + +$ ;
10         return ( $confirm', rt', ts'$ );
11     case  $AND$ 
12         foreach  $v \rightarrow v_{child} \in E$  do
13              $(confirm', rt', ts') = addWS(v_y)$ ;
14             if  $rt' > rt$  then
15                  $rt = rt'$ ;
16                  $confirm = confirm'$ ;
17                  $ts = ts'$ ;
18         return ( $confirm, rt, ts$ );
19     case  $XOR$ 
20         foreach  $v \rightarrow v_{child} \in E$  do
21              $(confirm', rt', ts') = addWS(v_y)$ ;
22             if  $confirm' \neq \emptyset \wedge ts' < ts$  then
23                 if  $confirm \neq \emptyset$  then  $initially = initially \cup confirm$ ;
24                  $rt = rt'$ ;
25                  $confirm = confirm'$ ;
26                  $ts = ts'$ ;
27         return ( $confirm, rt, ts$ );
28     case  $WS$ 
29         foreach  $tuple \in initially, ws(tuple) = v$  do
30             if  $ts(tuple) < ts$  then
31                  $rt = rt(tuple)$ ;
32                  $confirm = tuple$ ;
33                  $ts = ts(tuple)$ ;
34         if  $confirm \neq \emptyset$  then
35              $initially = initially \setminus confirm$ ;
36             return ( $confirm, rt, ts$ );
37         else
38             return ( $\emptyset, 0, 0$ );

```

Function 5 $\text{calcImpact}(v)$

```

input :  $v \in V$ 
1  $v_{comp} = v_x \in V, \tau_{CT}(v_x) = COMP;$ 
2  $e_{in} = v_y \rightarrow v \in E;$ 
3 switch  $\tau_{CT}(v)$  do
4   case  $COMP$ 
5      $\tau(v) = COMP;$ 
6      $\rho = \text{calcImpact}(v_{child});$ 
7     return  $\rho;$ 
8   case  $AND$ 
9      $\tau(v) = XOR;$ 
10     $contribution = 0;$ 
11    foreach  $v \rightarrow v_{child} \in E$  do
12       $\rho_{child} = \text{calcImpact}(v_{child});$ 
13       $contribution = contribution + \rho_{child};$ 
14     $\rho(e_{in}) = contribution;$ 
15    return  $\rho(e_{in});$ 
16   case  $WS$ 
17      $\tau(v) = WS;$ 
18      $\rho(e_{in}) = \frac{contribution(\sigma(v))}{invoc(\sigma(v_{comp}))};$ 
19     return  $\rho(e_{in});$ 

```

Algorithm 6: Feedback tree. Vertices V and edges E of a feedback tree are equal to vertices V and edges E of estimated and realized impact tree. Algorithm 6 calculates the feedback tree by computing for each Web service vertex what its composition impact is (e.g., Line 5, Algorithm 6). This impact depends on the number of contributions per composition invocation (i.e., ρ -value), and average SLO when invoked (i.e., σ -value). Assume Web service S1 has an average response time of 10 ms, while the composition has 20 ms response time. If S1 contributes in fifty percent of the composition invocations (i.e., $\rho = 0.50$), the impact factor of S1 is $\frac{10}{20} \cdot 0.50 = 0.25$. On average S1 determines 25% of the composition response time.

Furthermore, the color of each Web service and composition is determined by invoking `color(real, est)` in Line 6 and 7 of Algorithm 6. This function determines deviation between realized and estimated values as depicted in Function 7. Each edge is annotated with the realized contribution per composition invocation in Line 9, and color (θ value) is determined by calculating deviation between expected and realized contribution in Line 10.

Algorithm 6: Feedback tree

```

input :  $EIT(V, E, \rho_E, \tau_E, \sigma_E)$  and  $RIT(V, E, \rho_R, \tau_R, \sigma_R)$ 
output :  $FM(V, E, \rho, \tau, \sigma, \theta)$ 
1  $v_{comp} = v \in V, \tau_E(v) = COMP$ ;
2 foreach  $v \in V$  do
3    $\tau(v) = \tau_R(v)$ ;
4   if  $\tau(v) = WS$  then
5      $\sigma(v) = \frac{rt(\sigma_R(v))}{rt(\sigma_R(v_{comp}))} \cdot \rho_R(e_{in})$ ;
6      $\theta(v) = \text{color}(\sigma(v), \text{impact}(\sigma_E(v)))$ ;
7   if  $\tau(v) = COMP$  then  $\theta(v) = \text{color}(\frac{rt(\sigma_R(v))}{\text{invoc}(\sigma_R(v))}, \text{impact}(\sigma_E(v)))$ ;
8 foreach  $e \in E$  do
9    $\rho(e) = \rho_R(e)$ ;
10   $\theta(e) = \text{color}(\rho(e), \rho_E(e))$ ;

```

Function 7 color(real, est)

```

input : real: realized value, est: estimated value
output : color: the color of the edge or vertex
1  $deviation = \frac{real - est}{est} \cdot 100$ ;
2 if  $deviation \geq 10$  then return red;
3 if  $10 > deviation \geq 5$  then return yellow;
4 if  $5 > deviation \geq -5$  then return green;
5 if  $deviation < -5$  then return darkgreen;

```

The cost SLO of our illustrative example is evaluated and graphically represented in Figure 7.9. Using the event log abstraction, it is calculated that the average cost for invoking the composition is 14.47€. This is between 5 and 10% under performance, and,

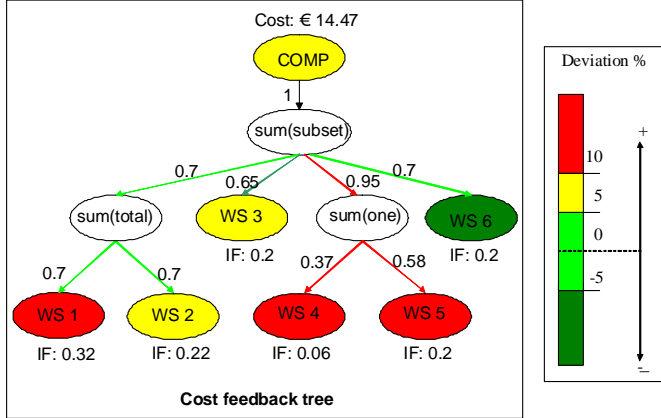


Figure 7.9: Illustrative example: Cost feedback tree

therefore, colored yellow. Furthermore, we see that only WS 6 costs on average less than expected, i.e., the service is colored darkgreen. The other services all cost on average more per invocation from that agreed upon (i.e., colored red or yellow). We see that the branch with WS 3 is invoked less often than expected, i.e., is colored darkgreen, as opposed to the branch of WS 4 and WS 5 which is invoked more often than expected, i.e., they are colored red. WS 4 has a very low impact on the overall cost (IF 0.06), while the other Web services have an impact between 0.2 and 0.32.

The result of our approach enables users to:

- Identify how well each service is performing,
- Detect whether services are invoked as often as expected, and
- Determine the impact each service has on the composition regarding its SLOs.

In the given case, we derive that the exceeded composition costs are caused by its underlying services. These services, in turn, exceed their agreed upon invocation costs. Although WS 4 violates its costs SLO, its impact is low. Furthermore, the expensive branch (with WS 4 and WS 5) concerning costs is more often invoked than expected. The cheaper branch (i.e., the branch with WS 3) concerning costs, in turn, is invoked less often, causing the overall composition to exceed its agreed upon costs.

7.11 Related work on SLAs

7.11.1 SLA models

An important *framework* for specifying composite services is the WSLA framework by Keller et al. [66]. We use their structure and requirements on SLA definitions for our

approach. Their framework enables the specification of SLAs which enables monitoring composite services although this is not treated explicitly. Another framework for specifying web services is the WSOL framework by Tomic et al. [109]. Unique in this approach is offering service monitoring in *classes* rather than per instance. Tomic et al. also consider dependencies between different services in [48].

7.11.2 Managing Web services

Menasce [77] presents response time analysis of composed services to identify impact of slowed down services. The impact on the composition is computed using a Markov chain model. The result is a measure for the overall slow down depending on statistical likelihood of a service not delivering expected response time. As opposed to our approach, Menasce performs analysis at design-time rather than providing a runtime based analysis. In addition, our work provides a framework to cover structures beyond a fork-join arrangement.

A different approach with the same goal is the virtual resource manager proposed by Burchard et al. [31]. It targets a grid environment where a calculation task is distributed among different grid vertices for individual computation jobs. If a grid vertex fails to deliver the promised service level, a domain controller first reschedules the job onto a different vertex within the same domain. If this action fails, the domain controller attempts to query other domain controllers for passing over the computation job. Although the approach covers runtime, it follows a hierarchical autonomic recovery mechanism. MoDe4SLA focusses on identifying causes for correction on the level of business operations rather than on autonomous job scheduling. Further, in *critical path analysis* (CPA) for resource management the preferred order of tasks is determined by analyzing the graph containing all possible paths [115]. However, in CPA each path shows one possible execution, while in our analysis the complete graph depicts one execution. Furthermore, in MoDe4SLA we compare estimated with realized behavior which is not done in CPA.

In the COSMA approach Ludwig et al. [72] describe a framework for the life cycle management of SLAs in composite services. They recognize the problem of managing dependencies between different SLAs. However, it is difficult to compare their approach to ours since the framework is described on a high level and no details on the implementation are given. Furthermore, their COSMA_{doc} component describes composite specific dependencies, but does not explicate what type of dependencies are considered.

The SALMon approach by Oriol et al. [91] aims at monitoring and adapting SOA systems at runtime. Monitoring is done for violations of SLAs. Furthermore, a decision component performs corrective actions so that SLAs are satisfied. This approach has similar goals to our management of SLAs for composite services approach. More specifically, they focus on the analysis and monitoring of SLAs, and based on the results management decisions are done. However, their approach does not focus on service compositions, but rather on runtime adaptability. As a consequence they are not concerned with dependencies between different SLAs.

In Moser et al. [81] an approach is described for automatically replacing services at

runtime without causing any downtime for the overall system. The BPEL processes are monitored according to their QoS attributes and replacement of services and partners is offered on various strategies. Although their approach has similarities to ours, their goals focus on runtime adaptability, and not on service compositions and their SLA dependencies.

7.11.3 Root Cause analysis

Another research community analyzes *root causes* in services. In responding approaches dependency models are used to find causes of violations *within* a company. Here, composite services are not considered, but merely services the company is responsible for on its own. For example, Agarwall et al. [5] determine the cause of a problem by using dependency graphs. Especially finding the cause of a problem when a service has an SLA with different metrics is here a challenging topic. Also Caswell et al. [34] use dependency models for managing internet services. Again, focussing on finding internal causes for problems. In our work we identify causes of violations in other services rather than internally. Furthermore, our dependencies between different services are on the same level of abstraction, while in root cause analysis one service is evaluated on different levels of abstraction.

7.11.4 Service monitoring approaches

Sahai et al. [100] aim at automated SLA *monitoring* by specifying SLAs and not only considering provider side guarantees, but focus also on distributed monitoring, taking the client side into account as well. Barbon et al. [11] enable runtime monitoring while separating the business logic from the monitoring functionality. For each instance of a process a monitor is created. Unique for this approach is the ability to also monitor *classes* of instances, enabling abstraction from an instance level. The smart monitoring approach of Baresi et al. [12] implements the monitor itself as a service. There are three types of monitors available for different aspects of the system. Their approach is developed to monitor specifically contracts with constraints. In [13] Baresi et al. present an approach to dynamically monitor BPEL processes by adding monitoring rules to the different processes. These rules are executed during runtime. Our approach does not require modifications to the process descriptions what might suit better to some application areas. An interesting approach in this direction is work by Mahbub et al. [74] who, as an exception, do consider the whole state of the system in their monitoring approach. They aim at monitoring derivations of behavior of the system. The requirements for monitoring are specified in event calculus and evaluated with runtime data. Although many of above mentioned approaches do consider third parties and also allow abstraction of results for composite services, none of them addresses how to create this abstraction in detail. Problems like matching messages from different processes as in our SubscribedNews example where databases are used, are not considered.

7.11.5 QoS-based service composition

Although our approach assumes the Web service composition is present, the process of Web service selection based on QoS criteria for the overall composition is closely related to our problem. More specifically, our approach monitors and manages the selections done in the service composition. Therefore, it is important to have an overview of widely used approaches for QoS based service composition. Many service selection approaches use linear programming methods although also alternative approaches exist.

Zeng et al. [118] propose a global planning approach for optimal service selection based on their QoS during execution of the composite service. The problem is addressed as optimization problem which is solved with a linear programming method. Berbner et al. [14] describe an approach for Web service composition taking QoS constraints into account where the authors provide a heuristic based approach. The results are computed by a relaxed integer program. Ardagna and Pernici [9] provide a Web service selection approach for service compositions using an optimization approach with mixed integer linear programming. Their approach allows specification of both local constraints on Web services as well as global constraints for the composition concerning QoS. Canfora et al. [32] are an exception to the linear programming approaches and describe an approach for Web service composition taking QoS constraints into account using Genetic Algorithms. Their claim is that although Genetic Algorithms are slower than integer programming approaches, their approach is better scalable. The aim of their approach is to do late-binding of services at runtime.

7.12 Summary

In this chapter we apply the method for managing dependency relations between models for inter-organizational cooperations (see Chapter 4) to monitoring SLAs of composite services. We show how to apply the different steps. Furthermore, we show that with using this method it becomes possible to formalize the derivation of impact trees and feedback trees so that automation and implementation become more straightforward. This case is illustrated with the use of a running example. In Section 10.3 we discuss the lessons learnt from applying our method for managing dependencies between inter-organizational models to the given case.

The main result of this chapter is the use of contribution factors, where we describe the contribution of each subtree to the composition, and the use of impact factors, where we describe the contribution of each service to a specific SLO of the composition.

The result supports the company in managing its composite services for detection and coverage of SLA violations. More specifically, decisions on whether or not to change content provider, whether or not to change certain SLAs, and which SLAs should be reconsidered are supported by depicting dependencies between services.

PROOF-OF-CONCEPT IMPLEMENTATION: SERVICE COMPOSITIONS

In Chapter 7 we illustrate the use of our MaDe4IC method for managing SLAs of composite services. In this chapter we show how the results of this analysis are implemented [24]. We provide a mechanism to identify *causes* for SLA violations of composite services. These causes are either due to under performance of services the composition depends on, or due to differences between estimated behavior of the composition and actual behavior at runtime. For this implementation we use the dependency relations and algorithms identified in Chapter 7. In Chapter 9 we then use our proof-of-concept prototype to evaluate usefulness of our solution for managing SLAs of composite services.

We demonstrate the implementation by means of an example scenario case as discussed in Section 8.1. In Section 8.2 we discuss implemented analysis techniques. These constraints are adopted from Section 7.8. In Section 8.3 we discuss how our MoDe4SLA simulator works. Furthermore, Section 8.4 shows how our MoDe4SLA simulator is used to analyze causes for composition SLA violations, and how a service manager uses visualization results from the simulator to identify these causes.

8.1 Example scenario

In the following we use an abstract example to illustrate the implementation of MoDe4SLA. The use of an abstract example allows us to focus on the approach, and illustrate the complexity of the problem without obscuring it with real-life details. Figure 8.1 depicts the composition of our example scenario where invocation of the composite service results in the invocation of a subset of 17 dependent services. Each invocation of the composition triggers the invocation of WS 1 - WS 7, WS 13, WS 16, and WS 17. In addition, either WS 8, WS 12, or the *Loop*-construct is chosen (due to the *XOR*-construct). Each outgoing edge of the *XOR*-construct is annotated with the *chance to be chosen* compared to its siblings. For example, the chances to be chosen for WS 8, Loop, and WS 12 are 0.72 : 0.15 : 0.13. When the *Loop*-construct is chosen, WS 9, WS 10, and WS 11 are invoked in a *sequence*, which is, on average, repeated 3 times. Furthermore, either WS 14 or WS 15

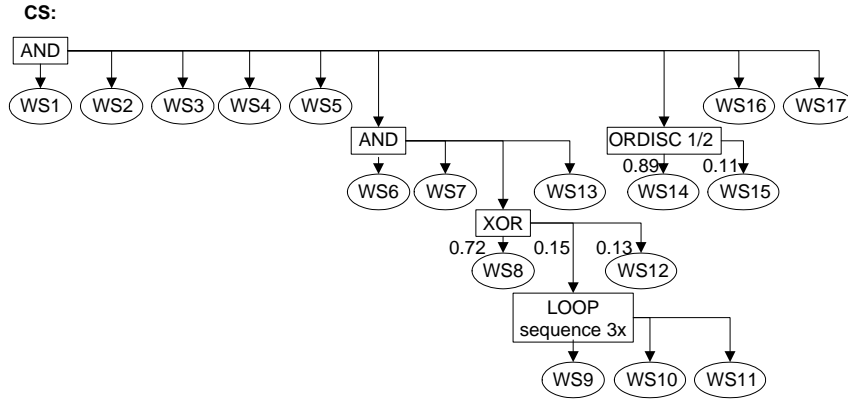


Figure 8.1: Example scenario: Service composition

is invoked (*ORDISC*-construct with ratio 0.89 : 0.11). Except for the services in the Loop sequence, all services are invoked in *parallel*.

Each service has an *agreed upon QoS level*. This QoS level is part of the SLAs between provider and customer. Figure 8.2 depicts the same composition as Figure 8.1, but focusses on the agreed upon QoS levels. Each Web service is annotated with a *minimum*, *mean*, and *maximum* agreed upon *cost* and *response time* for each invocation. These values are also given for the level of service offered to customers invoking the composite service.

8.2 Implementation

The implementation of the algorithms from Chapter 7 is discussed in this chapter. Therefore, we describe specificities of the implementation without discussing the different algorithms in detail again. However, we focus on describing how the algorithms are implemented. The algorithms formalize how to calculate *estimated impact trees* and *realized impact trees* using the composition as depicted in Figure 8.1 (with its estimated values and, at runtime, realized values for each service as monitored in the event logs). These trees show how the composite service depends on the services implementing it. The calculations used for the analysis are based on the number of times a service is invoked per service composition, and on the average SLO value the service achieved. Firstly, we describe *contribution factors* that depict the chance a branch in the composition tree contributes to the composition. Contributing branches are branches that influence the SLO value of the composition. Secondly, we describe how invoked *Web services* in a composition *contribute* to the composite service concerning costs and response time. For this we use average SLO values. Furthermore, we describe how we combine contribution factors and service contribution to an overall impact measure of each service on the composition, namely the *impact factors*.

8.2. IMPLEMENTATION

```
Composition has agreed upon QoS:
  Response Time (min,mean,max): 41.0, 117.02195, 725.0
  Costs (min,mean,max): 4415.4473, 8245.411, 22181.707
AND split and join.
_Subelement:
  Web Service No 1 with QoS:
    Response Time: ( min, mean, max): 10.0, 11.63, 18.0
    Cost: ( min, mean, max): 210.02309, 313.72, 513.5868
_Subelement:
  Web Service No 2 with QoS:
    Response Time: ( min, mean, max): 33.0, 37.69, 47.0
    Cost: ( min, mean, max): 846.5581, 963.62225, 1339.1849
_Subelement:
  Web Service No 3 with QoS:
    Response Time: ( min, mean, max): 22.0, 33.45, 34.0
    Cost: ( min, mean, max): 372.75146, 644.95825, 1061.9478
_Subelement:
  Web Service No 4 with QoS:
    Response Time: ( min, mean, max): 31.0, 43.39, 60.0
    Cost: ( min, mean, max): 736.048, 909.04694, 1338.7078
_Subelement:
  Web Service No 5 with QoS:
    Response Time: ( min, mean, max): 7.0, 12.19, 18.0
    Cost: ( min, mean, max): 100.35527, 168.35362, 198.89468
_Subelement:
  AND split and join.
  _Subelement:
    Web Service No 6 with QoS:
      Response Time: ( min, mean, max): 41.0, 59.62, 62.0
      Cost: ( min, mean, max): 442.9783, 454.52646, 591.99786
  _Subelement:
    Web Service No 7 with QoS:
      Response Time: ( min, mean, max): 36.0, 60.51, 61.0
      Cost: ( min, mean, max): 201.85391, 318.47342, 325.6623
  _Subelement:
    XOR split and XOR join.
    _Subelement, [0.72]:
      Web Service No 8 with QoS:
        Response Time: ( min, mean, max): 32.0, 60.31, 67.0
        Cost: ( min, mean, max): 525.27045, 618.8252, 664.4616
    _Subelement, [0.15]:
      Loop (Estimated: 5 times).
      _Subelement:
        Web Service No 9 with QoS:
          Response Time: ( min, mean, max): 17.0, 31.58, 33.0
          Cost: ( min, mean, max): 804.11975, 1158.6505, 1446.9174
      _Subelement:
        Web Service No 10 with QoS:
          Response Time: ( min, mean, max): 16.0, 28.44, 54.0
          Cost: ( min, mean, max): 35.12873, 53.35737, 71.33427
      _Subelement:
        Web Service No 11 with QoS:
          Response Time: ( min, mean, max): 31.0, 36.78, 59.0
          Cost: ( min, mean, max): 830.5941, 837.7012, 1149.1959
    _Subelement, [0.13]:
      Web Service No 12 with QoS:
        Response Time: ( min, mean, max): 5.0, 7.69, 11.0
        Cost: ( min, mean, max): 625.7165, 1118.7761, 1415.0142
  _Subelement:
    Web Service No 13 with QoS:
      Response Time: ( min, mean, max): 13.0, 19.05, 32.0
      Cost: ( min, mean, max): 694.0153, 1381.3864, 1965.6458
_Subelement:
  OR split, DISC join, where 1/2 services are started and 1 must finish.
  _Subelement, [0.89]:
    Web Service No 14 with QoS:
      Response Time: ( min, mean, max): 20.0, 22.43, 24.0
      Cost: ( min, mean, max): 502.37628, 567.51276, 859.6056
  _Subelement, [0.11]:
    Web Service No 15 with QoS:
      Response Time: ( min, mean, max): 26.0, 50.12, 81.0
      Cost: ( min, mean, max): 6.94027, 9.2576885, 13.704545
_Subelement:
  Web Service No 16 with QoS:
    Response Time: ( min, mean, max): 4.0, 8.1, 9.0
    Cost: ( min, mean, max): 156.71298, 285.16486, 325.40204
_Subelement:
  Web Service No 17 with QoS:
    Response Time: ( min, mean, max): 30.0, 42.86, 58.0
    Cost: ( min, mean, max): 121.94054, 171.77739, 323.83493
```

Figure 8.2: Example scenario: Agreed upon QoS

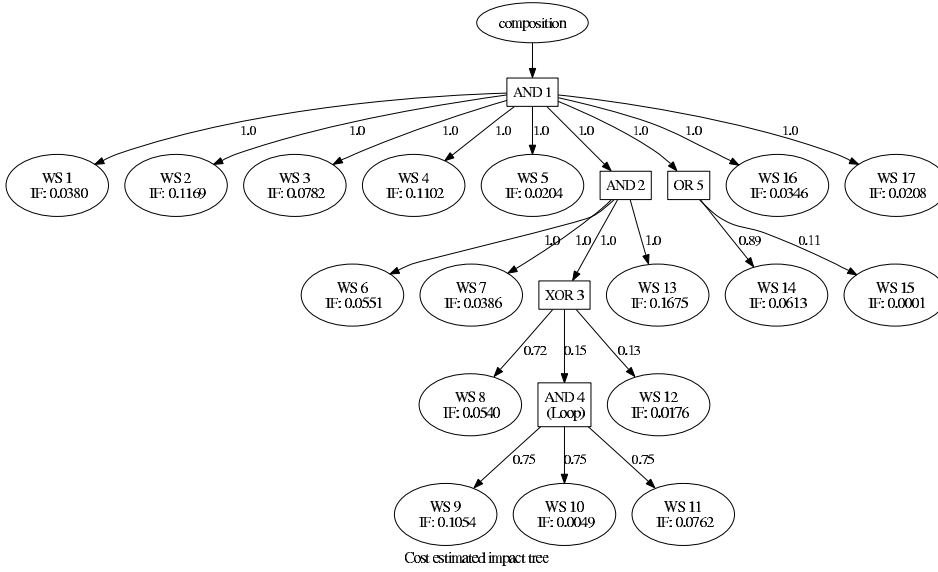


Figure 8.3: Example scenario: Estimated impact tree for cost

8.2.1 Contribution factors

The vertices and nodes of each impact tree indicate how the composition depends on the different services. In addition, certain quantifications are done to indicate how big this impact is. Here, we discuss quantification of branches in the impact trees. Each branch in the tree is annotated with a value. For each invocation, this value indicates the *number of times a branch contributes to the composition*. For estimated impact trees these values are estimations based on the composition structure and additional knowledge (e.g., preferences for invoking a certain service). For the realized impact trees these values are derived from event logs. Unless *Loop*-constructs are present in the composition, each value annotated to an edge is less than or equal to 1 since each service contributes at maximum one time per composition invocation to the composition value. Consider, for example, Figure 8.3 which depicts the *estimated impact tree* for costs based on the composition structure and estimations on number of invocations. The third construct in this tree (i.e., the XOR-construct) has three outgoing edges. Each edge has its own probability to contribute to the costs of the composition for each invocation. In this case, each composition invocation, Web service 8 (WS 8) contributes on average 0.72 times to the overall costs. In other words, the composition manager expects that WS 8 contributes about 7 out of 10 times to the composition costs.

For **costs** the chance to contribute to the composition is equal to the chance of being invoked. More precisely, if a service is invoked it also is paid for. Since estimations on which services are more likely to be chosen are present in the composition tree (cf. Figure 8.1), the contribution factor for costs is easily calculated. When there are no estimations

available on the likelihood to be chosen, each branch is awarded an equal ratio. For example, if always one out of two services are invoked, and no preference ratio is given, our implementation assumes these chances to be equal.

For **response time** the situation is a bit more complex. Not every invoked service also contributes to the overall response time. For example, in two parallel running branches, fastest responding branch does not contribute to the response time of the composition since the composition waits for the slowest responding branch anyway. As a consequence, the chance to contribute to the response time of the composition does not only depend on whether a branch is invoked, but also on the performance of branches running in parallel, i.e., on the behavior of its siblings. In the implementation, the default is that the chance a service is chosen is equal to the chance a service contributes, *except* when the parent construct informs the branch about sibling branches that are more likely to contribute. A precise description of this algorithm is found in Chapter 7 where Function `calc(v)` (see page 142) determines the probability a service gets invoked, and actually contributes to the overall composition.

8.2.2 Service contribution

For calculating *estimated* average contribution of a service to the overall response time or cost of the composition, we use average agreed upon response time and cost in the SLA. For the *realized* average contribution, we use average value of these invocations of services where the invocation actually *contributed* to the overall cost or response time of the composition. As a result, not every invoked service is considered in these calculations. Therefore, our approach does not substitute bilateral monitoring since the aim is to diagnose causes for violations in the composition.

Average realized service contribution concerning **costs** is calculated by considering each invoked service, since invocation of a service means the service is paid for. In other words, realized average costs for a service are equal to average realized contribution for a service.

Average realized service contribution concerning **response time** is again more complex to calculate since not each invocation of a service means the service contributes to the overall response time. This is, for example, the case in branches that run in parallel.

When only considering branches and services that influence the QoS of the composition, it also becomes possible to identify outliers. These outliers are services that might on average function very well (and, therefore, might satisfy its SLAs), but show great variability concerning their QoS, and, therefore, cause the overall composition to violate its SLA on a regular basis.

In our proof-of-concept implementation we, therefore, make a distinction between *unconsidered* services and *rolled back* ones. The first category contains services which are invoked and finished before the overall composition finishes, but which nevertheless do not contribute to overall response time. These services contribute to the cost since they are invoked. The second category contains services which are invoked, but did not yet finish when the overall composition finishes. This situation occurs, for example, when

only considering the fastest responding service. These services contribute to costs, but not to overall response time.

8.2.3 Impact factors

In impact trees each service is annotated with an *impact factor* (cf. IF in Figure 8.3). This impact factor is a *combination* of contribution factor and average service contribution. The value indicates the average *contribution percentage to the composition* concerning an SLO.

We calculate the impact factor by multiplying the contribution factor (i.e., the number of times a service contributes per composition invocation), with the average service contribution (i.e., the average SLO value of a service when it contributes to the composition). We divide this value with the average SLO value of the composition. Therefore, the impact factor of a service indicates its average contribution. For example, in Figure 8.3 WS 13 has an expected impact factor of 0.1675 for costs. This indicates it is expected that WS 13 contributes on average 16.75% to the overall costs of the composition. As a result, in the MoDe4SLA analysis, impact factors represent a combination of the structure of the composition (i.e., the branch value) and the SLO values of the services. Consequently, impact factors of the services in one composition add up to 1.

8.3 Generation and execution

For the MoDe4SLA approach, we implement a *simulator* to generate random service compositions with accompanying SLAs. In addition, this simulator simulates *invocations* of the composition where the different SLAs are regularly violated. These invocations are monitored by *logging* necessary information, for example, number of invocations, response times, and costs. Furthermore, our implementation provides necessary *analysis* of the compositions and event logs, and it *creates* the impact and feedback trees. For the simulation part we adapt and extend SENECA [62], an existing simulation environment for composite services.

This simulation environment implements (a) a structural model of service compositions and (b) a QoS model for handling QoS attributes of the services in the composition. The generation of compositions is performed randomly with particular input parameters. The parameters are the number of services in total and the range of the planned QoS delivered. By using the standard random number generator of the Java platform, the software then generates the following two main parts.

Firstly, the *structure of the composition* is created. At a given point, the generation software decides between placing a service or a composition structure containing services, both with equal probability. Based on this schema, compositions can result in a flat sequence of services, or contain nested structures forming a more sophisticated execution plan. There are seven basic executions patterns [62] chosen from the workflow patterns by Aalst et al. [1]. The number of services incorporated in the composition is determined by the user.

Secondly, the *QoS delivered by the service* is created. Originally, this function is used to perform simulations for optimizing QoS of service compositions [62]. We extend SENECA with the possibility of SLA *violation* at runtime by implementing some randomization. The generator builds up a data structure in application memory that allows the environment to simulate the execution of a service composition.

The discrete event *simulation* simulates the pass of a second (this is the unit of the simulated response time SLO), and tracks the progress among the services of the composition. If a service is finished, the next service is triggered according to the execution plan. Each service implements the simulation to either start or finish the work as planned. In addition, a random function allows to violate the agreed upon QoS by running longer or charging higher costs. The simulation software generates a log output that contains all events occurring during execution of the composition. For example, consider Figure 8.4 that depicts part of the event log created at runtime for our example scenario. This log file shows aggregated information on realized average values of the services, its total number of invocations, and the average number of times a loop is invoked per composition. Aside from these aggregated logs, logs with raw data are available as well. Furthermore, we add to SENECA creation of impact trees using the log after passing the total execution time for the composition. Calculations and derivations are done according to the different algorithms presented in Step 7 of Section 7.8. At the end of the simulation the user has the feedback models and monitoring results available for use.

8.4 Visualization

The goal of MoDe4SLA is to provide information on the composition performance to service developers through the graphical feedback model as generated by our MoDe4SLA simulator. This information is used to evolve the composition by tuning the structure or renegotiating SLAs of the services. The *feedback trees* (cf. Figure 8.5 and Figure 8.6) indicate differences between design time estimations based on SLAs, and realized values monitored in the event log. More particular, the feedback tree shows the *cause* of an SLA violation.

The difference between agreed upon and real-life values is depicted using *colors* for branches to indicate deviations in real life from estimated contribution factors. Colored services indicate deviations in real life from agreed upon average service contributions. As discussed, the impact factors indicate the combination of these two measures showing average contribution to the composition. Every service is annotated with its realized impact factor.

For color coding we use *red* to indicate worse performance than expected based on the SLA, while *green* indicates proper performance of the service. *Yellow* indicates the service is not performing perfectly, but still within the boundaries set by the company, and *dark green* indicates a service runs even better than the company anticipated. Uncolored services indicate the services never contributes to the overall response time in the considered event log. Therefore, their impact is zero. The edges are colored in the same manner,

CHAPTER 8. PROOF-OF-CONCEPT IMPLEMENTATION: SERVICE COMPOSITIONS

```
Realized QoS of the composition:
  Response Time (min,mean,max): 48.0, 174.86957, 1036.0
  Costs (min,mean,max): 5590.6934, 8678.282, 25450.111
  Total number of invocations: 23
AND split and join.
  _Subelement:
    Web Service No 1 with QoS:
      Response Time: ( min, mean, max): 7.0, 12.565217, 21.0
      Cost: ( min, mean, max): 39.184357, 288.40488, 607.5144
      Total # of invocations: 23
  _Subelement:
    Web Service No 2 with QoS:
      Response Time: ( min, mean, max): 30.0, 40.434784, 63.0
      Cost: ( min, mean, max): 335.96082, 922.88495, 1502.9639
      Total # of invocations: 23
  _Subelement:
    Web Service No 3 with QoS:
      Response Time: ( min, mean, max): 20.0, 37.869564, 54.0
      Cost: ( min, mean, max): 161.43155, 574.28046, 1251.6294
      Total # of invocations: 23
  _Subelement:
    Web Service No 4 with QoS:
      Response Time: ( min, mean, max): 14.0, 40.04348, 69.0
      Cost: ( min, mean, max): 341.64148, 848.5004, 1338.3586
      Total # of invocations: 23
  _Subelement:
    Web Service No 5 with QoS:
      Response Time: ( min, mean, max): 1.0, 13.782609, 21.0
      Cost: ( min, mean, max): 62.765213, 157.46265, 242.28073
      Total # of invocations: 23
  _Subelement:
    AND split and join.
      _Subelement:
        Web Service No 6 with QoS:
          Response Time: ( min, mean, max): 33.0, 61.608696, 92.0
          Cost: ( min, mean, max): 257.6671, 449.0448, 714.92725
          Total # of invocations: 23
      _Subelement:
        Web Service No 7 with QoS:
          Response Time: ( min, mean, max): 31.0, 61.304348, 104.0
          Cost: ( min, mean, max): 172.90501, 330.1317, 451.6234
          Total # of invocations: 23
      _Subelement:
        XOR split and XOR join.
          Total # of invocations: 23
      _Subelement, [0.74]:
        Web Service No 8 with QoS:
          Response Time: ( min, mean, max): 33.0, 67.588234, 101.0
          Cost: ( min, mean, max): 456.34702, 591.18994, 754.9042
          Total # of invocations: 17
      _Subelement, [0.13]:
        Loop (Estimated: 5 times).
        On average 7.666665 iterations per invocation, with:
        7 iterations minimum
        9 iterations maximum
        Total # of invocations: 3
        Total # of iterations: 23
      _Subelement:
        Web Service No 9 with QoS:
          Response Time: ( min, mean, max): 17.0, 34.391304, 55.0
          Cost: ( min, mean, max): 655.4398, 1162.9155, 1749.987
          Total # of invocations: 23
      _Subelement:
        Web Service No 10 with QoS:
          Response Time: ( min, mean, max): 9.0, 31.782608, 66.0
          Cost: ( min, mean, max): 28.277946, 57.33591, 104.81079
          Total # of invocations: 23
      _Subelement:
        Web Service No 11 with QoS:
          Response Time: ( min, mean, max): 7.0, 40.04348, 81.0
          Cost: ( min, mean, max): 363.83832, 811.3514, 1204.1954
          Total # of invocations: 23
      _Subelement, [0.13]:
        Web Service No 12 with QoS:
          Response Time: ( min, mean, max): 3.0, 7.0, 14.0
          Cost: ( min, mean, max): 388.61462, 854.22363, 1091.9017
          Total # of invocations: 3
      _Subelement:
        Web Service No 13 with QoS:
          Response Time: ( min, mean, max): 10.0, 22.043478, 37.0
          Cost: ( min, mean, max): 203.39478, 1530.2379, 2630.816
          Total # of invocations: 23
```

Figure 8.4: Example scenario: Realized QoS

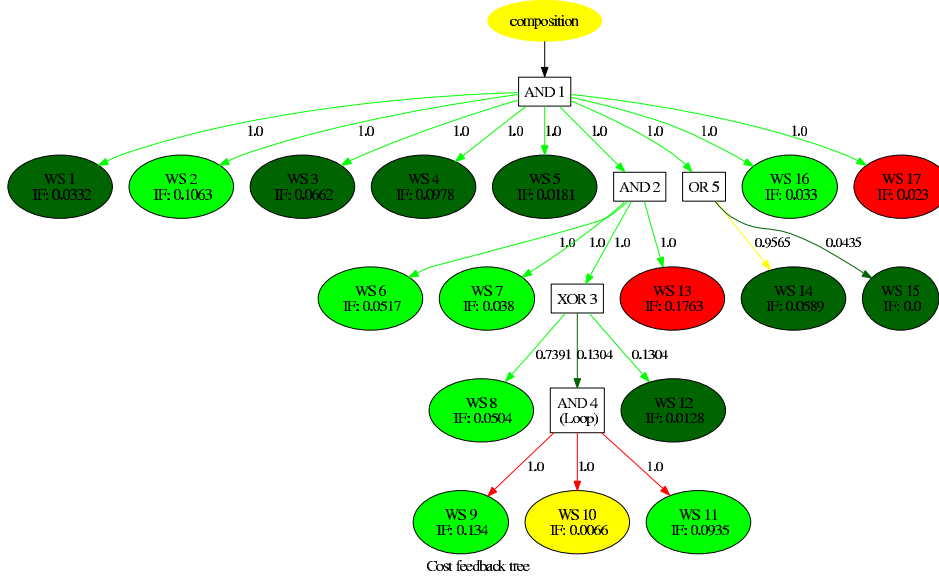


Figure 8.5: Monitoring example scenario: Cost feedback tree

for example, red indicates the edge contributes more often than expected.

Figure 8.5 depicts the cost feedback tree of our running example. The yellow colored composition indicates that the SLO for composition cost is not completely met. This lack of performance is caused by two factors. *Performance of service WS 13* is the first factor. This service is malfunctioning (i.e., violates its SLA, and is, therefore, colored red), and has an impact factor (IF) of 0.1763. The latter means that almost 18% of the overall costs are contributed by this service. *Structure of the composition* is the second factor. Services WS 9, WS 10, and WS 11 are contributing more often than expected (i.e., red incoming edges). This increased number of service invocations for each composition invocation causes elevated overall costs.

In this case, several solutions are possible. For example, badly performing WS 13 can be replaced or renegotiated. It is also possible to change the structure of the composition in such a way that WS 9, WS 10, and WS 11 are not invoked that often. Furthermore, we conclude that although WS 17 is not functioning properly, its impact factor is too low (i.e., 2%) such that it cannot be the main cause for the composition violation. Furthermore, several services are over-performing (i.e., they are dark green). If bad performance problems are solved, these over-performing services positively influence composition costs.

Figure 8.6 depicts the response time feedback tree of our running example. Response time wise, the composition is under-performing (i.e., it is colored red). The same holds for most services it depends on. The three services with highest impact factors (i.e., WS 8, WS 9, and WS 10) are colored yellow or red. Together, they are responsible for over 50% of overall response time. Furthermore, the structure of the composition shows deviations

8.5 Summary

In this chapter we show how our MoDe4SLA simulator for monitoring and analyzing causes for SLA violations is implemented and used to manage service compositions. More precisely, we show how to implement the different analysis algorithms introduced in Chapter 7, and how monitoring and analysis at runtime work. In addition to implementation of the constraints, and monitoring of the model, we depict how to use this information for managing the SLAs of the composition. More particularly, we show how to visualize deviations between estimations in the composition and runtime results in the event log. Our analysis enables identification of causes for violations that are missed in traditional bilateral monitoring approaches.

EVALUATION OF MODE4SLA: SERVICE COMPOSITIONS

MoDe4SLA identifies complex dependencies between Service Level Agreements (SLAs) in a service composition. By explicating these dependencies, causes of SLA violations of a service might be explained by malfunctioning of the services it depends on. MoDe4SLA assists managers in identifying such causes. Effectively managing compositions results in competitive service level offerings to customers with maximum profit for the business. Furthermore, insights on services run by other providers, helps managers to plan negotiation strategies concerning SLAs.

In this chapter we discuss possibilities for evaluating the MoDe4SLA approach [19]. We discuss *effectiveness* for the business if maintenance is done using our approach, and *usefulness* for managers burdened with actual maintenance of compositions. Effectiveness is evaluated by comparing runtime results of SLA management using MoDe4SLA with runtime results of unsupported management. Indicators for effectiveness are cost reduction, and increase in customer satisfaction. We discuss our evaluation plan in Section 9.1, while the evaluation itself constitutes future research. Usefulness is evaluated by asking experts to manage simulated runs of service compositions using MoDe4SLA. Their opinion on the approach is an indicator for its usefulness. This part of the evaluation and its setup are discussed in Section 9.2. Following this, we discuss the course of the evaluation and evaluation results in Sections 9.3 and 9.4, respectively.

Since we do not have access to real composition monitoring data, we use the implementation of our MoDe4SLA simulator for running composite services as discussed in Chapter 8. Experts from both industry and academia are asked to manage these generated compositions, both with and without using MoDe4SLA.

9.1 Evaluation plan for effectiveness

To evaluate effectiveness of MoDe4SLA, we plan to test performance of compositions managed by experts using MoDe4SLA. We compare these results with a control group that does not use MoDe4SLA. The latter applies when bringing actors in the composition

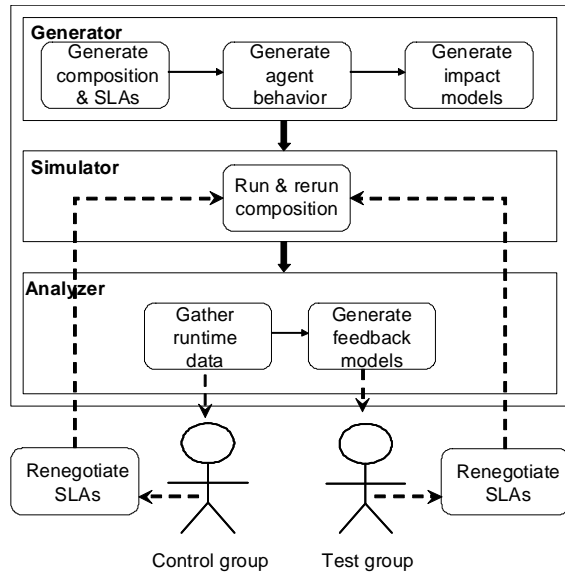


Figure 9.1: Evaluating effectiveness

closer to their goal, i.e., when the approach points to services causing bad performance of the composition. The level of *customer satisfaction* and *profits* made by the company are key indicators for evaluating effectiveness:

The MoDe4SLA approach is considered to be effective if managing a service composition using this approach leads to better results than managing the composition without it.

We define *better* in this case as having better business results. In our opinion, this is a twofold process, where both *customer satisfaction* and *profit* are maximized (cf. Table 9.1). We assume customer satisfaction is high if the number of SLA violations is low, costs for the customer are low, and response time is low. We assume violations being associated with high penalties having a more negative impact on customer satisfaction than low penalties. Therefore, customer satisfaction is measured through three indicators: payments for violations, average response time, and average costs for the composition (cf. Table 9.1).

Maximizing profit is achieved through minimizing costs and maximizing income (cf. Table 9.1). In our case, these costs consist of costs for services and payments for penalties. Income is generated through payments by customers and payments by providers for SLA violations. The total payments for services to providers and for SLA violations paid to customers are subtracted from this income.

To evaluate effectiveness, we plan to extend the implementation so that it becomes possible to *rerun* the composition after some management choices have been made by the

9.1. EVALUATION PLAN FOR EFFECTIVENESS

	Minimize	Maximize
Customer Satisfaction	violation costs	-
	average response time	-
	average costs	-
Profits	service costs	customer payments
	violation payments	violations paid by providers

Table 9.1: Effectiveness indicators

expert. Figure 9.1 depicts an overview of the effectiveness evaluation approach. Furthermore, we adapt parts of the *service behavior*, we add a *renegotiation* of SLAs, and finally we *evaluate* effectiveness of the approach.

In real life, services in a composition behave differently than expected. These differences arise, for example, due to different content of services, and different services are offered by different providers. The difference in behavior, together with the complexity of the composition structure, make it difficult to manage the services. In addition, managers can neither influence behavior of services provided by other companies nor can they predict their behavior other than by relying on SLAs. To simulate this real-life complexity, we implement *different types of behavior* for services. For example, some simulated services have low variability in response time, and never violate the SLA, while other services have higher variability in response time, and violate the SLA more frequently. Each simulated service gets assigned a behavior type at design time. Which type of behavior is assigned to the service, is unknown to the participants in the experiment.

To realize this, we implement *agents* to steer the different services in a composition. Each agent gets assigned a behavior type, for example, the *reliable* behavior type. Behavior of the service depends on both behavior type and agreed upon SLA. For example, if the agent behavior type is *reliable* and agreed upon response time is *lower than 10 ms*, the simulator will generate a random distribution of response times for the next 100.000 invocations. This distribution is created with the parameters *reliable* and *lower than 10 ms*. During the simulation, with each invocation of the service, a response time is randomly chosen from this distribution. As a result each service shows different behavior, fitting its SLA. Some services might violate their SLAs more severely, or more frequently than other services due to their assigned behavior type.

After running the composition the expert is asked to renegotiate SLAs for a subset of the services. In practice, determining which SLAs to renegotiate is particularly difficult for complex compositions, i.e., compositions with many services and constructs. Since the MoDe4SLA tool graphically pinpoints badly performing services and, moreover, indicates the *impact* these services have on the composition, it becomes possible to prioritize SLAs, decreasing management efforts.

As described, services in the simulator have unique behavior: some perform with only minor deviation in, for example, response time, while others fluctuate more. In a perfect agreement, the SLA between provider and customer reflects this behavior exactly.

In practice, however, it is necessary to monitor service behavior and to renegotiate the SLAs for badly performing services. In these negotiations provider and customer have conflicting interest since both aim at maximizing monetary benefits. To simulate this in the evaluation, the experts are asked to select a subset of services for renegotiation after the first run. Each service agent offers three new possible SLAs to the expert. For example, the original SLA has fast response time, but low violation penalties. A newly offered SLA by the agent might have slower response time, but high penalties when being violated. These new SLAs are generated taking the behavior type of the agent as well as the original SLA into account. Some of the new SLAs will fit better to the service from the customer perspective and some will fit the service better from a provider perspective. The challenge for the expert is to make choices beneficial for his company.

We use one *test group* and one *control group* for each tested service composition. The test group uses MoDe4SLA to choose services they want to renegotiate, while the control group uses log files only. MoDe4SLA is designed to assist the user in making choices on which services to renegotiate. Furthermore, MoDe4SLA pinpoints to the exact problems of the service so that choices between newly offered SLAs are easier to make. If the expert successfully identifies services with high impact on the composition and performing badly according to their SLAs, renegotiation will have to result in better runtime results of the test group in comparison to the control group (cf. Table 9.1).

9.2 Evaluating usefulness: Setup

To evaluate usefulness, we interview experts, asking them to make a statement on how useful they perceive the approach when managing the compositions. We use the following criterion to evaluate usefulness:

MoDe4SLA is considered as being useful when experts testing it perceive the feedback given by MoDe4SLA as more useful for managing and maintaining the composition than when only using bilateral monitoring results.

Common management approaches return bilateral monitoring results to the user. They do not provide information on the relation between the different services, but merely return performance of each individual service. For evaluating our MoDe4SLA approach, the simulator (cf. Section 8.3) is able to perform a discrete event simulation when running the composition with service candidates. Figure 9.2 depicts an overview of this simulator's functionality. SENECA randomly generates a *composition structure* for a given number of services. Actually considered structures in the simulator are sequence, loop, XOR-split/XOR-join, AND-split and OR-split (with either a normal join or a discriminative join). A discriminative join indicates the structure succeeds if a subset of incoming services succeeds. For example, when only considering the fastest three out of five responding services, this is a discriminative join.

We extend the implementation of an existing simulator, SENECA [62], with generating impact models and with an analyzer module (cf. Figure 9.2). This simulator randomly

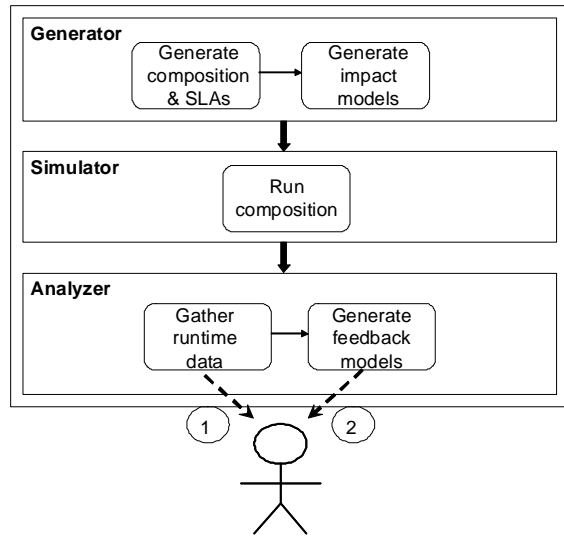


Figure 9.2: Evaluating usefulness

generates a *composition structure* for a given number of services. To each created service in the composition a randomly generated SLA is assigned. The *impact models* are derived based on the composition structure and SLAs. SENECA simulates invocation of services according to the composition structure. Accordingly, services might violate their SLAs. The simulator gathers *runtime data* and generates the *feedback models*.

The experts for our evaluation are from both academia and industry. We start each session with a training period to explain our approach and the simulator. Furthermore, we provide a presentation, and we explain two complete examples. After the training period we start the evaluation. For this, we prepare three different types of compositions with respect to complexity. The complete set of documents handed out to the experts for evaluation, including the questionnaire, is depicted in Appendix A.2. The first test case consists of five services with three constructs. The second test case consists of ten services with one OR-split and one discriminative join. The third test case consists of seventeen services connected through five constructs.

Each test case is invoked for a random number of times in the simulator. For the first test case this is 81 times, for the second one 260 times, and for the third case 23 times. For each case it holds that SLA agreements for the services are violated from time to time. These runtime results are gathered by the simulator for classical bilateral monitoring as well as for our instance-based monitoring approach. For each composition we prepare two documents. The MoDe4SLA document contains the feedback models for both response time and costs, while the control document contains performance data for each service, but does not provide information on how they are related. The main goal of our evaluation is to test the following hypothesis:

The MoDe4SLA document has a clear benefit over the control document for managing the composition.

We evaluate this hypothesis by conducting a survey among the experts where we find out how they feel about:

- RQ1** Accuracy of identifying malfunctioning services using MoDe4SLA compared to using bilateral monitoring results,
- RQ2** Efficiency when identifying malfunctioning services using MoDe4SLA compared to using bilateral monitoring results, and
- RQ3** Confidence the experts have in their answers when using MoDe4SLA compared to using bilateral monitoring results.

Aside from testing the above mentioned hypothesis, the aim of this evaluation is also to learn about following items:

- RQ4** How complex is the MoDe4SLA approach for users?
- RQ5** Which possible improvements for the MoDe4SLA approach do experts suggest?
- RQ6** Is there related work that we overlooked in our literature study which is suggested by our experts?

For this purpose, we prepare a questionnaire containing 49 questions that experts answer before, during and after the evaluation. Most questions have a typical five-level Likert item to rate the response of the experts:

1. Strongly disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

9.3 Course of evaluating usefulness

Our evaluation starts at Friday 9 January 2009 with a test run where we do the evaluation with three colleagues. This test run is intended to find out particular problems or errors

in examples, test cases, and questionnaire. The result is the addition of a front sheet that depicts the composition graphically. The minutes of this evaluation can be found in Appendix A.1. After this test run, we conduct six more sessions with in total 34 participants from several universities and companies. In general, such a session consists of three parts:

1. A PowerPoint presentation of at least 15 minutes explaining in detail the goal of the approach.
2. Discussion of two example cases where the presentation of both bilateral monitoring results and MoDe4SLA feedback model results are explained. This explanation includes a discussion on how to interpret results.
3. The actual evaluation consists of a sequence of events:
 - (a) Answer a set of introductional questions.
 - (b) Study the first test case composition with expected values.
 - (c) Answer a set of questions about the composition complexity.
 - (d) Study bilateral monitoring results.
 - (e) Answer a set of questions about these results.
 - (f) Study MoDe4SLA feedback models.
 - (g) Answer questions about these models.
 - (h) Repeat these steps for the second and third test case.
 - (i) Conclude with answering some general questions.

9.4 Conclusions from evaluating usefulness

In this section we present the most important results of our evaluation. First, we introduce some demographics after which we discuss statistics from the answers that relate to the questions listed in Section 9.2. We conclude with some interesting results when analyzing relations between different outliers in questions. A complete list of statistics on all questions can be found in Appendix A.3. Please, note that answers to Question 48 are *not* included since these are names and email addresses of our participants.

9.4.1 Demographics

Two of our experts come from industry, nine come from both industry and academia, and 23 experts come from academia. 15% of the experts have experience using tools for managing composite services, and the same percentage in developing such tools. 60% of the participants have not worked with composite services, while the remaining 40% have varying experience from less than a year to over three years. Only 9% of the participants consider themselves having a high level of expertise in managing composite services, while 32% are undecided, and 59% consider themselves having a low level of expertise.

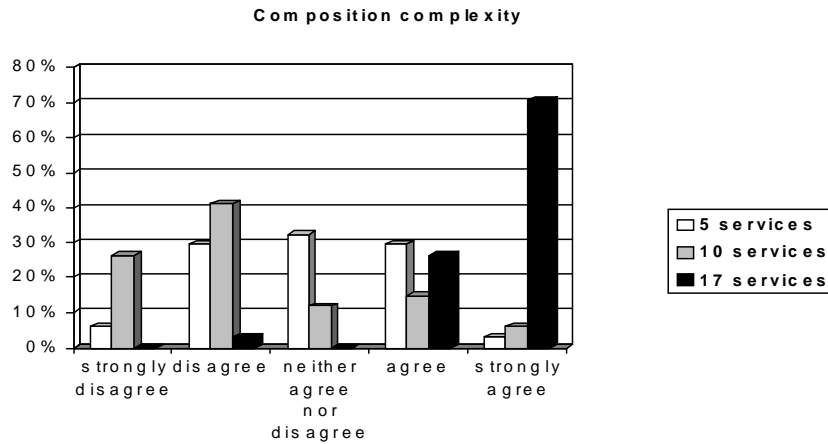


Figure 9.3: The offered composition appears to be complex.

We conclude that although our participants are familiar with service compositions, on average, their expertise in managing them is not high. The advantage is that this inexperience helps us in determining how difficult it is to learn to manage service compositions with MoDe4SLA. As disadvantage, we cannot expect much feedback on possible other approaches for managing service compositions our participants are aware off.

9.4.2 Statistics

We start each test case with a question on how complex the participants feel the test case is (Test Case 1 with 5 services: Question 8, Test Case 2 with 10 services: Question 19, Test Case 3 with 17 services: Question 30) (cf. Figure 9.3). We add this question since we assume the more complex the composition, the more useful MoDe4SLA. For example, in our opinion, Test Case 1 is relatively easy to manage since it only considers 5 Web services. Although the participants consider the different test cases to be of different complexity, and although participants appreciate using MoDe4SLA even more when considering the complex test case (i.e., Test Case 3), these differences are much lower than expected. Results are shown when evaluating questions in the following paragraphs. First we evaluate questions related to the benefits from using MoDe4SLA compared to not using our monitoring approach (i.e., RQ1, RQ2, and RQ3).

RQ1: Accuracy. We want to know whether participants feel identifying problematic services is done more accurately with than without MoDe4SLA. We have two questions giving us an insight on this.

Firstly, we ask participants for each test case, and for both response time and costs whether they perceive identifying the impact each service has on the composition easier

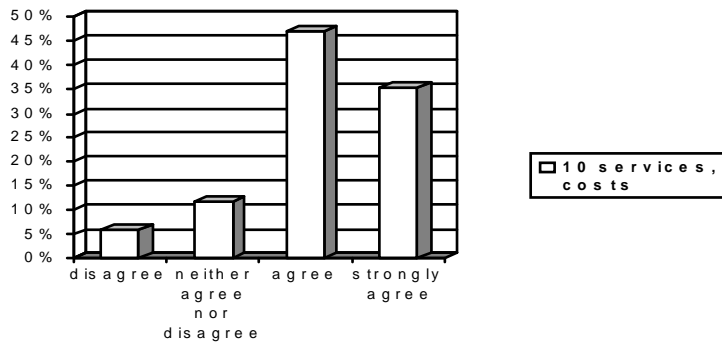


Figure 9.4: Concerning costs: It is easier to determine the impact each service has on the composition with the analysis than without it.

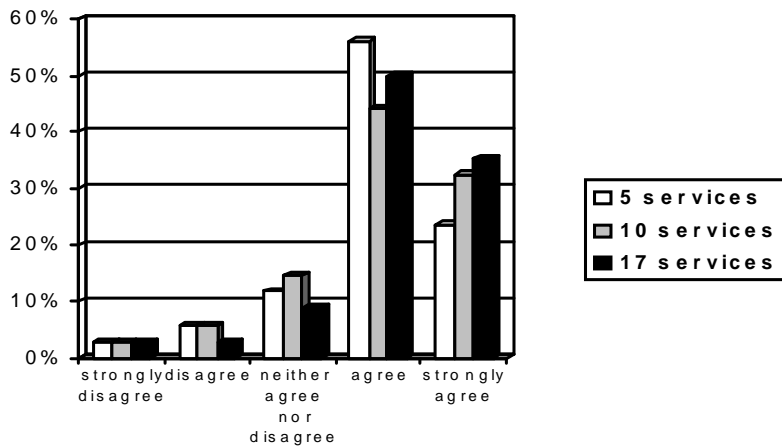


Figure 9.5: MoDe4SLA approach is helpful when managing this composition with regard to accurately depicting malfunctioning services.

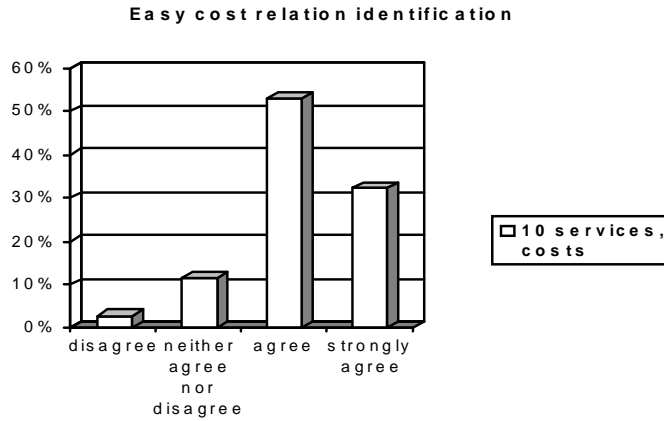


Figure 9.6: Concerning costs: It takes less time to see relations between different services and the composition.

with MoDe4SLA than without MoDe4SLA (for Test Case 1: Questions 15 and 16, for Test Case 2: Questions 26 and 27, for Test Case 3: Questions 37 and 38). Looking at results, we derive that, in general, MoDe4SLA is perceived as more useful for response time than for costs, and as more useful for Test Case 3 than for Test Cases 1 and 2. The majority perceives the use of MoDe4SLA as very helpful for easier identification. Figure 9.4 depicts the histogram with *least* positive responses for our approach. It entails over 80% of the participants agreeing or strongly agreeing to the statement. This is Question 26 for costs in Test Case 2.

Secondly, for each test case we ask participants whether they feel MoDe4SLA is helpful when managing the composition with regard to accurately depicting malfunctioning services (for Test Cases 1, 2, and 3, and Questions 18, 29, and 40). Figure 9.5 depicts results for these questions. We conclude that 75-80% of the participants agree or strongly agree that MoDe4SLA is helpful to accurately depict these services.

RQ2: Efficiency. We want to know whether participants feel that using MoDe4SLA for managing service compositions is more *efficient* than managing such compositions without our approach. Again, two sets of questions give us insights into this topic.

Firstly, for each test case, and for both response time and cost we ask participants to respond to the statement that it takes them less time to see relations between the different services in a composition when using MoDe4SLA. Since MoDe4SLA relies on identifying relations and dependencies between the services, we assume that MoDe4SLA is helpful when trying to identify these relations in the management phase. The relations are used to do a causal analysis. Here, depending on the question, 85-100% of the partic-

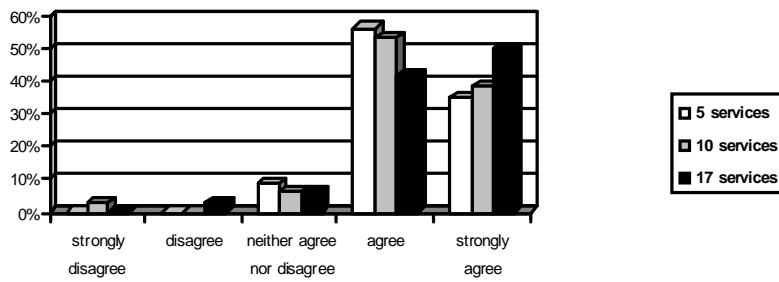


Figure 9.7: MoDe4SLA approach is helpful when managing this composition with regard to faster selecting services to renegotiate.

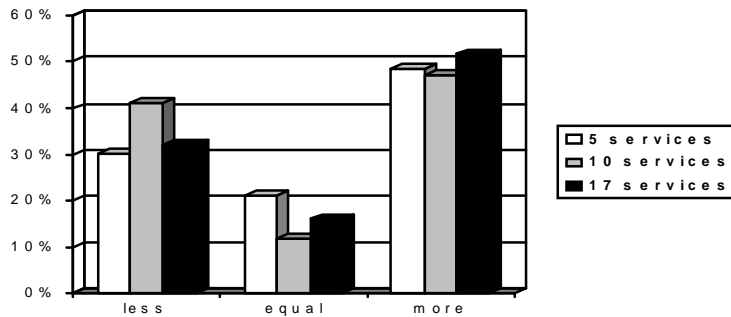


Figure 9.8: After seeing the MoDe4SLA analysis, how is your confidence about the service selection for renegotiation you made before?

Participants agree or strongly agree with this statement. In Figure 9.6 we depict the results for the least positive responses for our approach. This is Question 24 for costs in Test Case 2.

Secondly, for each test case we ask participants whether they feel MoDe4SLA is helpful when managing the composition with regard to faster selecting services to renegotiate (for Test Case 1, 2, and 3, and Questions 18, 29, and 40). Figure 9.7 depicts results for these questions. We conclude that around 90% of the participants agree or strongly agree that MoDe4SLA is helpful to faster select these services.

RQ3: Confidence. To evaluate how confident participants are when they make a choice on which services to adapt to get a better performance of the composition, we ask three questions per test case. Firstly, we ask how confident they are making a choice *before* seeing the MoDe4SLA models, secondly, we ask how confident they are about their original choice when seeing the MoDe4SLA models, and thirdly, we ask how confident they are in making a choice when considering the MoDe4SLA models.

The aim of the second question is to find out whether participants feel MoDe4SLA

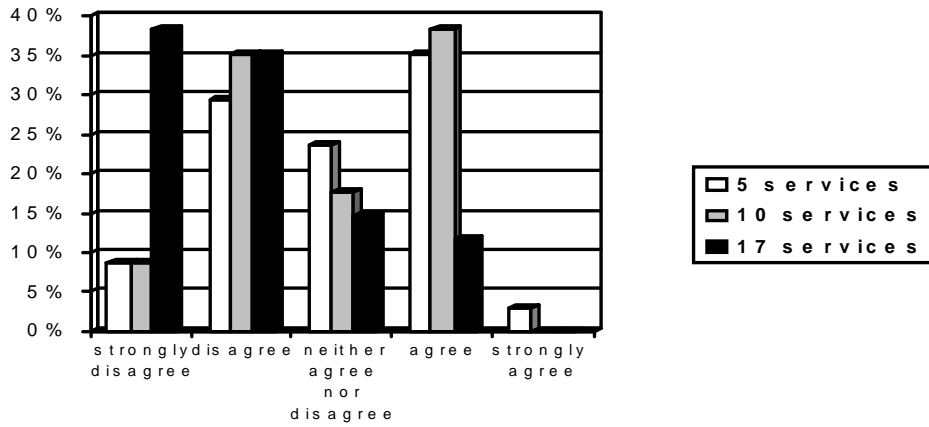


Figure 9.9: Assume only a subset of services can be renegotiated regarding their SLAs. I would feel confident in selecting services for renegotiation.

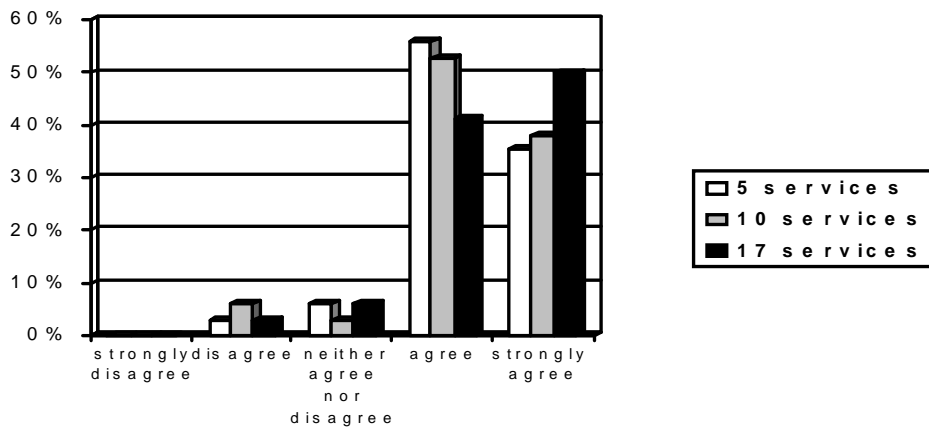


Figure 9.10: Assume only a subset of services can be renegotiated regarding their SLAs. I would feel more confident in selecting services for renegotiation with MoDe4SLA than without.

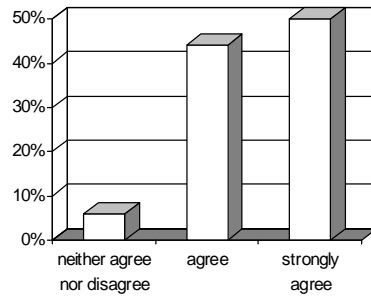


Figure 9.11: MoDe4SLA is helpful for managing service compositions.

gives additional support. If they feel more or less confident, MoDe4SLA apparently gives them additional insights. If they did not change their opinion, MoDe4SLA does not give additional insights. Furthermore, comparing the answers given to the first question and the answers given to the third question, gives insights whether participants feel more confident making choices when using MoDe4SLA than without using the approach. The change in confidence the experts have (i.e., the second question) is depicted in Figure 9.8. Here, we see that for each test case at least 80% of the participants changes their confidence level.

The confidence level of participants before considering MoDe4SLA is depicted in Figure 9.9. Here, we see that first and second test case have reasonable confidence levels, but there is no confidence in the third test case. In Figure 9.10 we see that the confidence level goes up for all test cases after participants see the MoDe4SLA files.

From these results we conclude that, on average, participants feel more confident when making choices on which services to adapt using MoDe4SLA than without it.

In the previous three paragraphs we review evaluation results concerning accuracy, efficiency and confidence levels. Besides testing usefulness of our approach with these indicators, we ask participants at the end of the survey to respond to the statement that using MoDe4SLA is useful when managing composite services. Corresponding results are depicted in Figure 9.11. We see that none of the participants disagrees or strongly disagrees with this statement. 94% of the participants agrees or strongly agrees with it.

RQ4: Complexity. Another important consideration is on how difficult MoDe4SLA is to understand. We strive to develop an *intuitive* approach that is easy to understand for users. Of course, the positive evaluation results concerning usefulness of MoDe4SLA after a short training period, supports our claim that MoDe4SLA has good usability. However, we also want to know whether participants feel the given presentation on MoDe4SLA is sufficient to use the models. The presentation takes at most one and a half hour, including discussion and questions. In Question 42 we ask participants whether they agree or not with the statement that the provided presentation gives sufficient information to un-

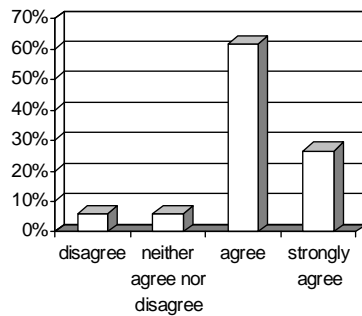


Figure 9.12: The presentation before the evaluation is sufficient to properly understand MoDe4SLA approach.

derstand the MoDe4SLA approach. Results are depicted in Figure 9.12 where we see that most participants (over 85%) agree with this statement. We conclude that for most participants the presentation is sufficient. But there is a considerable group, around 12%, that appreciates more explanation.

Furthermore, in Question 45 we ask participants to name weak points of the approach. Here, there are more indications that MoDe4SLA is less intuitive to some of the participants. 7 of them (i.e., around 20%) indicate they have problems understanding the values in the model. With these values, participants sometimes mean impact factors, but usually contribution ratios turn out to be hard to understand. The magnitude of numbers confuses some of the participants. 2 of them (i.e., around 6%) state that for them the feedback models are too complex to comprehend. In conclusion we state that over 25% of the participants have some difficulties understanding the values in the feedback models.

RQ5: Possible improvements. While discussing these last results, we come to possible improvements for the MoDe4SLA approach. We ask participants to name things they feel are most beneficial about the feedback models. Participants like the visualization part, especially with the coloring. Furthermore, impact factors and the analysis itself are beneficial for the participants.

In addition, we ask for possible improvements of our approach. From the answers we conclude some suggestions for further development of the approach. The first possible improvement is a reduction of the many numbers used in the models. As discussed in RQ4, for some participants there is simply too much information in the models. Furthermore, participants feel they are able to make proper choices with just coloring and impact factors. In other words, the additional ratios are often not necessary. Therefore, we consider filtering this information when the models are presented to users.

Secondly, participants appreciate some interpretation guidelines. For example, “what does a low impact factor indicate?”, “what does a high ratio mean?”, and “what can I derive from the combination of an impact factor and a ratio?”. Therefore, we consider

extending the presentation to participants with information on these statements.

Thirdly, related to the previous improvement, participants appreciate guidelines for decision making. It is beneficial if the models indicate which services to consider for change and why. Currently, models only provide monitoring information without any suggestions on how to improve performance. Developing such guidelines is part of our future work.

RQ6: Related work. Although we hope to get some more feedback on this question, unfortunately, we only received related work indications by two participants. The approaches mentioned are already known to us. So far, this supports our observations that there do not exist any approaches similar to MoDe4SLA.

9.4.3 Conclusions

Test cases. Although we introduce three different test cases with different complexity levels, it turns out that MoDe4SLA is already perceived as useful when managing a composition of only five services. Furthermore, our test case of seventeen services is considered as highly complex by the participants. However, when considering real-life constellations, seventeen services is not a lot. As a consequence, a proper management approach is definitely necessary in those cases.

Furthermore, Test Case 2 is often perceived as less complex than Test Case 1 although it consists of twice as many services. Reason for this is that dependencies in the structure of Test Case 1 are more complex since it contains three constructs as opposed to Test Case 2 that only contains one construct.

Cost versus response time. When analyzing the different diagrams, it becomes clear that most participants struggle more with response time dependencies than with cost dependencies. As a result, MoDe4SLA is especially appreciated in the response time models, which is also supported by answers given to Question 48 where beneficial parts are named. The reason for this is that response time of a branch depends on the interaction between the different services. Whether service A contributes, depends not only on whether it is invoked, and how fast it runs, but also on how fast its neighbor runs. Although the ratio of contribution for costs is also dependent on the cost of other services, this dependency is much less strong. Therefore, for property response time it is even more important to identify dependencies than for costs.

Overall conclusion. To support our hypothesis that MoDe4SLA models have a clear benefit over an approach that only supports bilateral monitoring (cf. Section 9.2), we investigate usefulness of our approach as perceived by experts using the models. All three sub-questions concerning accuracy (RQ1), efficiency (RQ2), and confidence (RQ3) clearly indicate that participants benefit from the MoDe4SLA models. Although there are some improvements to consider, as discussed in RQ5, participants are able to properly

understand the MoDe4SLA approach within one and a half hour. These results give us sufficient support to continue developing our management approach.

So far, we only ask participants for their *opinion*. There are no good or bad answers. Of course, we want to know whether the answers our participants give are *effective* as well. In other words, we want to know whether their decisions are better when making them with MoDe4SLA than without our approach. This is considered in our effectiveness evaluation in future research.

9.5 Summary

This chapter presents our evaluation approach consisting of an effectiveness part and a usefulness part. Actual evaluation of effectiveness constitutes future research. We conduct in several interactive sessions an extensive usefulness evaluation where 34 participants each answer 49 questions. The obtained results are so far promising since most participants consider MoDe4SLA as useful for managing composite services (particularly if they compare the approach to traditional bilateral monitoring). We identify some possible extensions and improvements for our approach that are incorporated in future research.

Part V

DISCUSSION

DISCUSSION & LESSONS LEARNT

In this thesis we present our MaDe4IC method suitable for managing different models describing inter-organizational cooperations. Firstly, we demonstrate the need for a method managing dependency relations between inter-organizational models, and we describe this method in detail. Furthermore, we evaluate our MaDe4IC method by applying it to two different scenarios. In this chapter we discuss the major research results of this thesis. We start in Section 10.1 with discussing whether and how requirements identified in Section 3.4 are fulfilled by our MaDe4IC method (cf. Chapter 4). Furthermore, we perform a cross-scenario comparison in Section 10.2. We elaborate on characteristics of the two different scenarios used to evaluate our method. In Section 10.3 we discuss the application of our method. We first discuss its application to business and coordination models. Secondly, we show how our method is applied in the context of service compositions and related Service Level Agreements. In Section 10.4 we reflect on the validation of the MoDe4SLA approach that is developed from applying our method to service compositions. After presenting the main results of this thesis, Section 10.5 continues with discussing the research questions formulated in Chapter 1. We conclude with a discussion on future research in Section 10.6.

10.1 Method requirements

In Chapter 3 we identify requirements. In detail, we formulate the following requirements:

- **R1** The method should be language-independent.
- **R2** The method should be able to handle heterogeneous models, i.e., models described in different languages.
- **R3** The method should ensure consistency through checking rather than through construction.
- **R4** The method should provide guidelines for overcoming model heterogeneity.
- **R5** The method should provide guidelines for identifying overlap between models.
- **R6** The method should provide guidelines for identifying dependencies between models.
- **R7** The method should provide guidelines for defining consistency constraints for inter-organizational cooperation models.
- **R8** The method should provide guidelines to identify the overlap between event logs and model.
- **R9** The method should provide an approach to log information necessary for consistency checking.
- **R10** The method should provide an approach to analyze an event log to abstract necessary information for consistency checking.
- **R11** The method should provide guidelines for defining consistency constraints between running system and its underlying models.
- **R12** The method should provide an approach to show consequences of model adaptations on consistency relations.
- **R13** The method should provide an approach to provide monitoring results as feedback.

As we discuss in the following, our MaDe4IC method, as defined in Chapter 4, fulfills these requirements.

It is (R1) *language-independent*. It does not make any assumptions on the type of language used to describe models, aside from the assumption that they are applied to conceptual models. This is also supported by the two scenarios to which we apply our method (cf. Chapters 5 and 7). These scenarios describe models in several different modelling languages.

The developed method is (R2) *suitable for heterogeneous models*. On the one hand, this is supported by providing homogenization guidelines in Step 2 of the method to enable model comparison of heterogeneous models. On the other hand, this is supported by providing a language-independent abstraction of related model parts in Step 7 where we suggest the use of either graphs or sets. Furthermore, the scenarios to which our method is applied (cf. Chapters 5 and 7) consist of several heterogeneous models. In both scenarios our method is successfully applied.

Requirement R3 states that the method should rely on consistency *checking* rather than on ensuring consistency through *construction*. The method also complies to this requirement since it assumes the existence of (possibly inconsistent) models from the start. These models then form the basis for the management approach. Furthermore, in both scenarios, the models are checked for consistency rather than that they are constructed maintaining consistency.

In the fourth requirement (R4) we state the necessity for guidelines to overcome *model heterogeneity*. In Step 1 of our method (cf. Figure 4.1), each model is analyzed. The result is a set of model characteristics. These characteristics support handling heterogeneity between the models in Step 2. The guidelines in Step 2 provide a structured approach in identifying heterogeneity between the models. Furthermore, it provides guidelines to either homogenize the models, or to handle the differences between them in subsequent steps. In addition, at the end of the analysis phase, the combined dependency analysis (Step 7) overcomes any other heterogeneity problems by representing relevant model parts in a language-independent formalism.

Both requirements R5 and R6 are ensured in Step 3 of our method. R5 states the necessity for inter-model *overlap detection*, while R6 states the necessity to identify *inter-model relations*. The overlap is mainly present in viewpoint models, while relations are identified between viewpoints and between partial models.

Requirement R7 expresses the need for an approach to define *consistency constraints* that support checking and ensuring consistency within and between models. Our method provides guidelines to define inter-model consistency constraints in Step 4.

Requirements R8, R9, and R10 are covered in Step 8 of our method. Here, we first specify how to log necessary information (R9), after which we provide an approach to analyze information from event logs to abstract necessary information for consistency checking (R10). Finally, we describe how these event logs are related to the different models, and how they are used to perform monitoring (R8).

Requirement R11 states the necessity for guidelines on how to define *consistency constraints* between running system and its underlying models. Our method provides guidelines to define intra-model consistency constraints in Step 6. These constraints state which information on the behavior of the system should be present in the event logs for the models to be considered being consistent with the running system.

Furthermore, in R12 we express the necessity for identifying *consequences* of model changes with respect to consistency relations. This part is covered by Step 9 of the method where we do a causal analysis based on the different relations identified within and between the models.

Finally, R13 states we should provide an approach to show monitoring results to users in order to manage the models. For this, our method suggests to use the *management models* that result from Step 8. These models show all relations, dependencies, and constraints between and within the models. In addition, our method suggests *coloring* and *annotating* parts of the models for easy distinction between good and bad performance. This is applied in both scenarios where the implementation (cf. Chapter 6 and Chapter 8) offers graphical feedback to its users based on the different dependencies.

We develop the method in this thesis with the identified solution requirements in mind, and we conclude that the method indeed complies with these requirements. This is supported by both scenarios to which we successfully apply our MaDe4IC method.

10.2 Cross-scenario discussion

In Part III of this thesis we evaluate our MaDe4IC method by applying it to *business models* and *coordination models*. More specifically, we use e³-value modelling ontology for business modelling and Petri Nets for coordination modelling. The value models are used to discuss profitability of a cooperation among involved actors. The coordination models are used to discuss the order in which message exchanges are accomplished. The challenge is to relate these two different models, and to manage them both at design time and at runtime of the cooperation.

In Part IV of this thesis we extend our evaluation by applying our method to monitoring Service Level Agreements (SLAs) for service compositions. Here, we use SLA models to describe the quality of each service in the composition. More specifically, we focus on monitoring response time and costs of the services. The challenge is to relate the different SLAs, taking the service composition into account. Furthermore, we develop an approach to manage the SLAs at runtime.

Based on this informal summary of the two evaluated scenarios, it becomes clear that they differ greatly. For us, it is important to choose scenarios that are as diverse as possible: (1) To show as much of the functionality of our method as possible, and (2) to show the applicability of our method to a variety of scenarios and cases. The diversity of the scenarios becomes clear when considering the different characteristics of inter-organizational models as identified in Chapter 3:

- Our two scenarios depict their models in different languages. This supports our claim that our method is *language-independent*.
- In Section 3.1.3 we describe different types of pragmatic heterogeneity that influence the approach for managing models of a cooperation:
 - We identify a difference between *perspectives* of a model. We distinguish *bird's eye view* and *single actor* models. The first scenario, using business and coordination models, describes models from a bird's eye perspective, while our second scenario describes models from a single actor perspective.

- We discuss the difference with respect to the *focus* models have. We distinguish between viewpoint and partial models. Viewpoint models describe the complete cooperation with a specific focus, while partial models describe only a part of the cooperation, but do this in full detail. Our first scenario contains viewpoint models where the business model describes the complete cooperation with a value focus, while the coordination model describes the complete cooperation with an ordering focus. Our second scenario describes several partial models; each SLA model describes part of the service composition, but does this in detail (i.e., for both costs and response time).
- We discuss the distinction between *instance-based* models, and models describing a *period of time*. In the first scenario, for example, the business model describes a period of time where value is calculated for the upcoming three months, while the coordination model describes an instance-based model where the order of messages for each model instance is described. In other words, the coordination model describes for each business transaction how to execute it. In our second scenario, the SLAs describe a period of time where averages of costs and response time are modelled. However, the overall Web service composition describes the cooperation in an instance-based manner where the model shows the interactions for one Web service invocation.

The two scenarios differ in *language*, *perspective*, *focus*, and *time frame*. We identify these characteristics in our problem investigation (cf. Chapter 3) as the main characteristics of inter-organizational models. More specifically, we did not identify any other characteristic differences in inter-organizational models. Therefore, incorporating these differences provides the complexity for building a method that is suitable for coping with a variety of conceptual models. Our MaDe4IC method is built to handle these different characteristics. By demonstrating the applicability of our method to this variety of aspects, we show its applicability to conceptual models of inter-organizational cooperations in general.

10.3 Applying our MaDe4IC method

We discuss our experiences in applying our MaDe4IC method for managing dependencies between inter-organizational models to both scenarios. We discuss the application of each step in our method. We conclude with a summary of the results in Section 10.3.10.

10.3.1 Step 1: Model analysis

In Step 1, model analysis, we determine *focus*, *perspective*, *time frame*, and *property type* of the models. For both Scenario 1 (business and coordination models) and Scenario 2 (Service Level Agreements for service compositions) this is a straightforward exercise.

10.3.2 Step 2: Homogenization

In Step 2 we homogenize the models on three different levels, i.e., on *syntactic*, *semantic*, and *pragmatic* level.

Syntactic homogenization. Business and coordination models are described in different languages (i.e., they are syntactically heterogeneous). We identify differences and correspondences between the two modelling languages (i.e., between e³-value and BPMN). This constitutes a precise, but straightforward exercise. Although the languages are different, they both describe conceptual models for inter-organizational cooperations, and use similar constructs to build the models. This enables matching the language constructs.

The Service Level Agreements considered in the second scenario, are described in the same language. Therefore, there is no need for homogenization.

Semantic homogenization. Both scenarios are semantically homogeneous by construction.

Pragmatic homogenization. In both scenarios *focus*, *granularity* and *property type* are the same for the considered models. In business and coordination models the *perspective* of the models differs. The identification of this heterogeneity enables easier identification of inter-model relations in the following steps. In both scenarios there is a heterogeneity in *time frames*. In both scenarios we choose to do instance-based comparisons between the models. In other words, we *shorten* the time frame for models describing a longer period of time. As a result, monitoring is done on an *instance level* which proves very beneficiary. Especially in the second scenario, instance-based monitoring supports determining the impact separate services have on the composition (cf. Section 7.5.1).

10.3.3 Step 3: Inter-model relation detection

Inter-model relations are identified between *concepts* and *properties* in different models. These relations describe dependencies that are *symmetric* or *asymmetric*.

Scenario 1: Business and coordination models. The inter-model relations in business and coordination models are between *concepts*. The key is to identify concepts in the models that describe the same real-world entity. Since the models are *viewpoints* (i.e., they focus on a specific characteristic in the cooperation), they describe different characteristics. Therefore, the concepts in the models describe different *properties*. In general, we state that viewpoint models do not have inter-model relations on property level. This makes inter-model relation detection comparably easy for viewpoint models, since property dependencies require formulas describing *how* properties are related. By contrast, for concept relations the relation simply describes *that* they are related.

Scenario 2: Service Level Agreements for service compositions. The inter-model relations in SLAs are between *properties*. We do not only determine which concepts describe the same entities, but also *how* these entities are related. For example, the monetary value of a product depends on the monetary value of its parts. Since SLAs are *partial models*, they describe a set of characteristics. Each characteristic type relates concepts from different models. In general, we state that partial models typically have many inter-model property relations. This makes inter-model relation detection comparably complex for partial models.

10.3.4 Step 4: Inter-model consistency constraints

We formulate consistency constraints for these relations in Step 4 using the detected inter-model relations in Step 3.

Scenario 1 (Business and coordination models). For each type of inter-model relation our method provides a matching consistency constraint. Therefore, formulating the inter-model consistency constraint for business and coordination models is straightforward.

Scenario 2 (Service Level Agreements for service compositions). Although our method enables matching consistency constraints for each dependency, formulating such constraints constitutes a more complex task for property dependencies than for concept dependencies. This complexity is caused by the formulas describing *how* the properties are related. For the SLAs, each property type (e.g., costs) shows up in each SLA. All these properties are inter-related over the different SLAs. For example, the costs of each service influences the costs of one or more services in the composition. As a consequence, a formula is created for each property type describing the inter-model consistency constraint.

In general, defining inter-model consistency constraints constitutes a more complex task when considering partial models than when considering viewpoint models. This increased complexity is caused by the existence of property dependencies between partial models.

10.3.5 Step 5: Intra-model relation detection

Intra-model relations are identified between *concepts* and *properties* within each model. Identifying intra-model relations is less complex than identifying inter-model relations. Intra-model relations are already depicted in the considered models as opposed to inter-model relations that are constructed between the models. The most challenging part of this task is to determine what type of dependencies exists. Classifying the different dependencies in *concept* and *property* dependencies, and in *asymmetric* and *symmetric* dependencies constitutes the most challenging task. Identifying these dependencies in our scenarios was a comparable effort for business models, for coordination models, and for SLAs.

10.3.6 Step 6: Intra-model consistency constraints

Scenario 1 (Business and coordination models). Viewpoint models (e.g., business and coordination models) describe the complete cooperation for one characteristic (e.g., for costs). As a consequence, each model contains many concepts that refer to different real-world entities. Therefore, the concepts in the models are related, and there exist many dependencies between them. These relations are either property or existence dependencies. Especially many concepts referring to different entities make intra-model consistency constraint definition complex. In addition, model-specific constraints are developed independently from the identified intra-model relations.

Scenario 2 (Service Level Agreements for service compositions). Intra-model relations in *partial* models (e.g., SLAs) are typically property dependencies. A partial model describes one part of the cooperation in detail. Therefore, such models often contain a small set of concepts with many properties. As a consequence, the intra-model relations are not complex. In addition, model-specific constraints are developed independently from the identified intra-model relations.

10.3.7 Step 7: Dependency analysis

In Step 7 we use consistency constraints formulated in Steps 4 and 6 to *formalize* them. In addition, we formalize those parts of the models that influence these consistency constraints. Formalization enables easy implementation for automatic consistency checking. In both scenarios we *abstract* necessary information from the models. In the first scenario we use *sets* to represent this information, while we use *graphs* in the second scenario.

Scenario 1 (Business and coordination models). Formalizing models and consistency constraints, constitutes a straightforward task for business and coordination models. Steps 1 - 6 provide sufficient preparation to identify what should be formalized, and how this should be done.

Scenario 2 (Service Level Agreements for service compositions). Formalization of the Service Level Agreements constitutes a more challenging task. We choose to construct a model for each characteristic. Therefore, we formalized all *inter-model* dependency relations, and represent them in one dependency model. As opposed to the first scenario, where we formalize *intra-model* dependencies and represent those in one model, all inter-model *property* dependencies are described by a formula. Each of these formulas is formalized in algorithms for automatic derivation of the dependency models. Especially constructing the algorithms constitutes a challenging task.

10.3.8 Step 8: Log analysis

The log analysis in Step 8 is comparable in the two scenarios. Although the necessary information is different, the approach to identify and abstract necessary information is

comparable. In both scenarios it is important to identify which actors are involved in a transaction, what the identifier of each transaction is, and what property values are exchanged.

10.3.9 Step 9: Causal analysis

In both scenarios we use the formalized models from Step 7 and the log analysis from Step 8 to do a causal analysis at runtime. We compare predictions and agreed upon values in the models with the realized values in the event logs. In both scenarios we use intra-model as well as inter-model dependencies to reason over the compared values. For example, we reason over the question why realized values differ from the agreed upon values. Furthermore, we use the models and dependencies to predict consequences of adapting parts of the models. In both scenarios the preparations done in Steps 1 - 8 are sufficient to do a causal analysis for violations as well as for analyzing consequences of changes.

10.3.10 Lessons learnt

Although the two considered scenarios are different in many respects, when applying our method the largest influence is caused by a difference in *focus*. Managing dependencies in viewpoint models and those in partial models is very different. Main differences appear in the inter-model relation detection (Step 3), the inter-model consistency constraint definition (Step 4), and the dependency analysis (Step 7):

Step 3: Inter-model relation detection. Property dependencies are more complex than concept dependencies since concept dependencies simply state that the existence of one concept depends on the other. Property dependencies do not only state they are existence dependent, but also how this dependency looks like. Since inter-model relations for partial models are typically on a property level (as opposed to viewpoint models), describing these relations is more difficult than it is for describing them in viewpoint models.

Step 4: Inter-model consistency constraint definition. In general, defining inter-model consistency constraints constitutes a more complex task when considering partial models than when considering viewpoint models. This increased complexity is caused by the existence of property dependencies between partial models. These dependencies are translated into complex consistency constraints containing formulas.

Step 7: Dependency analysis. In the dependency analysis of Step 7 we formalize those parts of the models that influence the consistency constraints. For every characteristic (e.g., response time) we create a separate formal dependency model. Especially when there are many property dependencies, creating these formal models constitutes a challenging task.

10.4 Validating MoDe4SLA

In addition to the proof-of-concept implementation we build for both scenarios, we validate *usefulness* of the results of the second scenario, and explicate the necessity for validating *effectiveness* of our solution. We decide to validate usefulness by carrying out an interactive survey. We start each session with a presentation of our approach, give examples of our management tool in comparison to traditional bilateral monitoring, and conclude with the actual survey. In this survey we ask our participants to manage different compositions both with and without using our management tool. We ask them for their opinion on the approach. The goal of this validation is fourfold:

1. We want to know whether our participants evaluate the approach as *useful*,
2. We want to know how *complex* our approach is to understand for users,
3. We want to find additional related work, and
4. We ask suggestions for possible improvements of our approach.

Participants evaluate MoDe4SLA as being very useful when managing service compositions, especially its way of identifying dependencies is appreciated. Furthermore, our approach is well understandable and usable after proper presentation. There are several suggestions for possible improvements of which some (i.e., interpretation guidelines, decision guidelines, and less numbers in the models) are considered important future work.

From our interactive survey we learn that most participants feel that managing response time benefits more from the dependency analysis than management of composition costs. This difference is explained by the different influence that response time and costs have on the performance of a branch. Here, it becomes clear that *dependencies* between services influence the impact that response time of a service has on the composition. However, these dependencies do not influence the impact costs of a service have on the composition.

The number of constructs influences significantly the perceived complexity level for managing the cooperation. This influence is more significant than the influence the number of services has. It appears that the composition itself determines to a large extent the perceived complexity of a composition.

In conclusion, we state that conducting such interactive survey is very useful for getting additional suggestions for further development of the approach. Furthermore, it provides valuable confirmation that the development of the approach is useful in general. The results validate the necessity for dependency analysis, and stimulate us to further develop the approach, especially in the directions identified by our participants. In addition, we conclude that necessity of identifying dependencies differs between the different properties to be monitored. For example, the necessity is bigger for response time than for costs. Therefore, it is interesting to evaluate this also for other properties (e.g., for availability of a service composition). Although necessity of the approach in general is already clear in small service compositions (as identified by our participants), we see a growing complexity for the user with the addition of constructs, rather than with the addition of services.

It is interesting to further investigate this correlation. Especially since we aim at applying this approach to complex service compositions. Therefore, it is useful to identify a measurement to determine when a service composition is considered complex.

10.5 Answering research questions

In previous sections we discuss the different topics treated in this thesis separately. Here, we discuss the results of the research questions we formulated in Chapter 1.

Research Question 1: What are solution criteria that a method for checking and ensuring consistency in inter-organizational cooperations should satisfy?

Q1a: What are characteristics of models and running system in the context of inter-organizational cooperations?

Q1b: What are criteria for a method that checks and ensures consistency in such models?

We answer the first question by identifying criteria for our method to manage dependencies between inter-organizational models. For this purpose, we position our work in a conceptual framework in Chapter 2. In addition, the *problem investigation* in Chapter 3 identifies typical characteristics of models for inter-organizational cooperations. Based on these characteristics, we define a set of *solution requirements* for our method (cf. Section 3.4).

Research Question 2: What is state of the art on maintaining consistency relations in inter-organizational models?

Q2a: How is consistency checked at design time in existing solutions?

Q2b: How is consistency ensured during runtime in existing solutions?

Q2c: How suitable are current solutions with regard to the criteria defined in question Q1b?

The second research question concerns *state-of-the-art* research. We are interested in how consistency is checked and ensured in existing solutions. In Chapter 2 we answer this question by considering several existing approaches. We determine that majority of the approaches focus on one or more specific models for which they develop an approach to maintain consistency. Although such approach is very useful when managing models in the specified language, it is not usable when managing models expressed in different languages. The remaining approaches we identified, are suitable for managing models in general. However, these approaches are too high level for our purpose. Their guidelines provide a starting point for the user. However, no detailed information is provided

on how to actually manage the models. Furthermore, many approaches do not go beyond ensuring consistency. In other words, they do not provide an approach to manage consistency at runtime, but merely provide an approach to deal with respective issues at design time. We conclude from this review that current state-of-the-art research does not provide an approach that covers the complete set of solution requirements. Therefore, there are sufficient grounds to develop a new method for managing inter-organizational cooperations.

Research Question 3: How can a solution method for checking and ensuring inter-model consistency be built?

Q3a: How can inter-model consistency be ensured?

- How can inter-model dependencies be detected?
- How can consistency constraints be defined using these inter-model dependencies?

Q3b: How can intra-model consistency be ensured?

- What intra-model dependencies exist?
- How can consistency constraints be defined using these dependencies?

Q3c: How can consistency between a running system and its underlying models be checked?

Q3d: How can consistency between a running system and its underlying models be efficiently maintained?

The third research question concerns the development of our method itself. We investigate how the different aspects of checking and ensuring consistency is accomplished in Chapter 4, which also presents our MaDe4IC method. We distinguish between inter-model and intra-model consistency relations which we treat separately in our method (cf. Figure 4.1). Furthermore, we discuss how to check consistency after defining the different constraints, and how to maintain consistency of models with a running system. Here, we use a causal analysis to assist this management process (cf. Figure 4.1).

Research Question 4: How can the solution method be validated?

Q4a: How well applicable is the solution method in different scenarios according to the criteria identified by answering Research Question 1?

Q4b: How good are the developed solutions when applying the method?

- Validate the solution of both scenarios with a proof-of-concept implementation.
- Validate the implementation of one of the scenarios through a usability survey.

In our fourth and last research question we discuss possibilities for validating our created method. Firstly, we validate our MaDe4IC method in different scenarios. We use our method for managing business and coordination models in the first scenario (cf. Chapter 5), and we use our method for managing Service Level Agreements for composite services in the second scenario (cf. Chapter 7). Especially since the two scenarios differ significantly (cf. Section 10.2), they provide proper support for the applicability of our method in inter-organizational models in general. Secondly, we validate the solutions developed using our method by means of a proof-of-concept implementation. These evaluations are described in Chapters 6 and 8, respectively. Furthermore, we perform a usability survey among 34 participants to evaluate usefulness of the implementation for managing service compositions in Chapter 9.

10.6 Future research

Although this thesis presents a complete method managing models of inter-organizational cooperations, there are several topics we intend to investigate in future research.

10.6.1 Our MaDe4IC method for managing dependencies

Although our method proves to be applicable to different scenarios, we plan to validate it in additional scenarios. Furthermore, it is useful to investigate management efforts for model developers when using our method. Currently, we do not make predictions on the complexity of the resulting management models when applying our method. However, the two scenarios indicate there exist many dependencies between numerous entities, and the effect of these dependencies differs significantly. For example, dependencies between entities modelling response time are more complex than the ones between entities modelling costs. Furthermore, the scenarios we consider are relatively small with respect to the number of dependencies and the number of entities. Based on these facts, we formulate the following interesting questions:

1. How are complexity of management models and number of dependencies in and between models related?

2. How well does our method scale up when it is applied to large scenarios?
3. Does the type of entity influence the complexity of dependencies, and, if so, can we provide a list of typical properties and their complexity factor?
4. Does the type of dependency relation influence complexity of the management model, and, if so, can we provide a classification?

10.6.2 Our approach for managing SLAs of composite services: MoDe4SLA

The resulting approach for managing Service Level Agreements (i.e., MoDe4SLA) is considered promising based on the interactive survey we conducted. Users participating in the survey are enthusiastic about its potential. Therefore, we pursue this research topic and extend the approach in several directions [18]:

1. We plan to add more *intra-model dependencies* since currently we focus on inter-model dependency relations.
2. We plan to extend the approach with *additional properties* (e.g., availability).
3. We plan to support managing *ranges of property values*.
4. We plan to support users with *guidelines for interpreting feedback models*.
5. We plan to pursue the *effectiveness evaluation* as discussed in Chapter 9.1.

1. Intra-model dependencies. Currently, we assume attributes are mutually independent. However, in real-life SLAs dependencies within SLAs are common. Therefore, we plan to extend our approach with intra-model dependencies. Consider the following SLA:

Every month, response time will be within 3 ms at least 99% of the invocations. Costs are 3 euro when the service responds within 2 ms, while a response time of 2 – 3 ms costs 2 euro. If less than 99% of the invocations have response time within 3 ms a penalty of 1000 euro is paid.

In this SLA, cost directly depends on response time. The challenge is to represent what this means for the relative contribution of component service performance on the composition cost. For example, to diagnose the cause of an increase in composition cost, we need to show the influence of decreasing response times to the cost (i.e., costs rise to 3 euro). To solve this, we intend to represent dependencies *between* models by constructing links between attributes. These links should be annotated with some contribution function, similar to causal arrows in causal loop diagrams [104]. Our goal is to represent causal influence between attributes as accurately as necessary for meaningful diagnosis. For example, in the above example it may be sufficient to represent that a decrease in response time causes an increase in cost.

2. Additional properties. We plan to extend the approach with additional properties. For example, we consider extending the approach with availability.

3. Ranges of property values. So far, we ignore property value ranges, but in real life, SLAs use these ranges. Consider the following examples:

Response time of service A will be 7 – 9 ms with an average of 8 ms.

Response time of service B will be 1 – 15 ms with an average of 8 ms.

We currently treat these performance requirements as identical since we consider *averages*. For a proper diagnosis of SLA violations, we need to reason about value *ranges*. Therefore, we will incorporate *variance* in the impact analysis computation, where the impact of services with high variance is greater than the impact of a stable service. We need to experiment with various ways of doing this while keeping complexity of dependency and diagnosis computation low.

4. Interpretation guidelines. The results from the interactive survey show that participants appreciate additional support on *how* to interpret the feedback models. Even though the semantics of the edges, nodes, colors, and values is clear, most participants appreciate guidelines on how to interpret the model as whole. For example, participants struggle to prioritize colored nodes and edges. An improvement of our MoDe4SLA implementation is a set of guidelines on how to interpret these models.

5. Effectiveness evaluation. In our effectiveness evaluation we will not only ask users for their opinion on our approach, but we will test the benefits of managing compositions with our approach as well (cf. Section 9.1). For example, we want to know whether better management decisions are made if our models are used.

EVALUATION

A.1 Transcript

Friday 9 January 2009 the first group of experts evaluated usefulness of MoDe4SLA. It was a try out with only three experts. The complete session was around one hour and 45 minutes which is divided in a *presentation part*, an *explanation part* with two examples, and the actual *evaluation part* with three test cases. A transcript of the exact times is depicted in Table A.1.

Although each participant is employed in an information technology environment, not everyone is familiar with service compositions. Therefore, the *presentation* comprises an explanation of the problem and what the exact research gap is. The first part of the presentation discusses the necessity of identifying dependencies between different services in a composition, why identifying these dependencies is not straightforward. The second part of the presentation is on the MoDe4SLA approach in which is explained how we identify these dependencies and solve the problem. Since the first group already participated in previous presentations on MoDe4SLA, the time frame of 15 minutes for the presentation should be considered a minimum.

In the second part is through two examples explained how the survey will be conducted. Both examples have the same structure as the three test cases. The goal of introducing these examples is to allow the participants to get familiar with the MoDe4SLA approach. First, the representation of the service composition and its parameters (e.g., average response times) in the bilateral documents for both the estimations and the realized values is discussed. Second, the analysis done with MoDe4SLA on the realized values of the composition is discussed. Together with a legend the participants discuss how to use both the bilateral and the analysis documents.

The last part is done by the participants separately, without interference of the presenter. First the introductory questions are answered after which the participants go through the three test cases. After the test cases the concluding questions are answered. Interested participants receive an evaluation of the three test cases on how to read the analysis done through MoDe4SLA.

Time	Subject	Minutes
15:09-15:26	Presentation	15
15:27-15:40	Example 1	13
15:40-15:58	Example 2	18
16:00-16:04	Before evaluation: Q1-Q7	4
16:04-16:15	Test Case 1: 5 Services • Q8-Q11 without MoDe4SLA: 4min • Q12-Q18 with MoDe4SLA: 7min	11
16:15-16:28	Test Case 2: 10 Services • Q19-Q22 without MoDe4SLA: 5min • Q23-Q29 with MoDe4SLA: 8min	13
16:28-16:43	Test Case 3: 17 Services • Q30-Q33 without MoDe4SLA: 8min • Q34-Q40 with MoDe4SLA: 7min	15
16:43-16:53	After evaluation: Q41-Q47	10
15:09-16:53	Total time	104

Table A.1: Time Transcript

A.2 Hand-out

This Appendix contains the complete hand-out for participants of the evaluation. This starts with a cover sheet and a legend, after which the two examples and three test cases are given. The hand-out concludes with the survey itself and some suggested answers to the presented problems.

Monitoring Dependencies for Service Level Agreements: MoDe4SLA Evaluation

Contact:

Lianne Bodenstaff
University of Twente
l.bodenstaff@utwente.nl

Website:

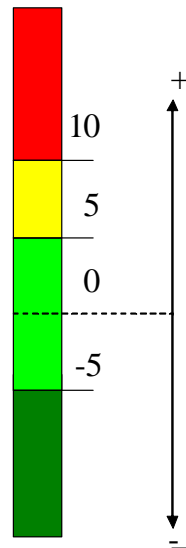
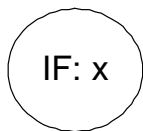
<http://www.ewi.utwente.nl/~bodenstaffl/mode4sla>

Estimations:

[x] = chance to be chosen. All chances within one construct add up to one.

Realized:

[x] = ratio a branch was chosen. All chances within one construct add up to one.

Deviation %**Analysis:**

Red: costs/response time were higher than agreed upon.

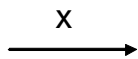
No color: did not contribute at all.

Ratio of service contribution
(= branch value)

IF= $\frac{X}{\text{Its average costs/response time}}$

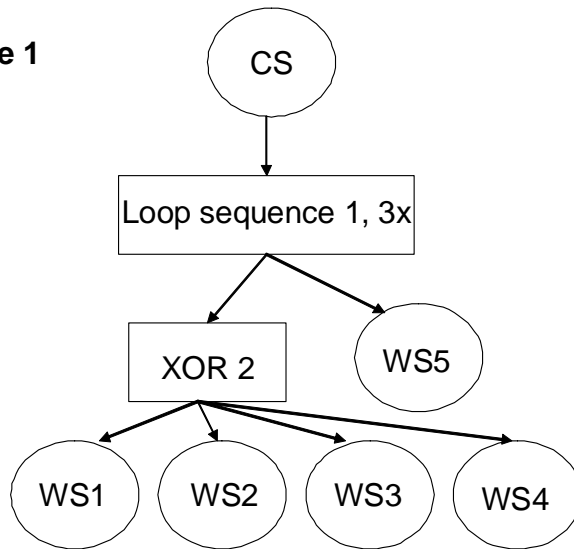
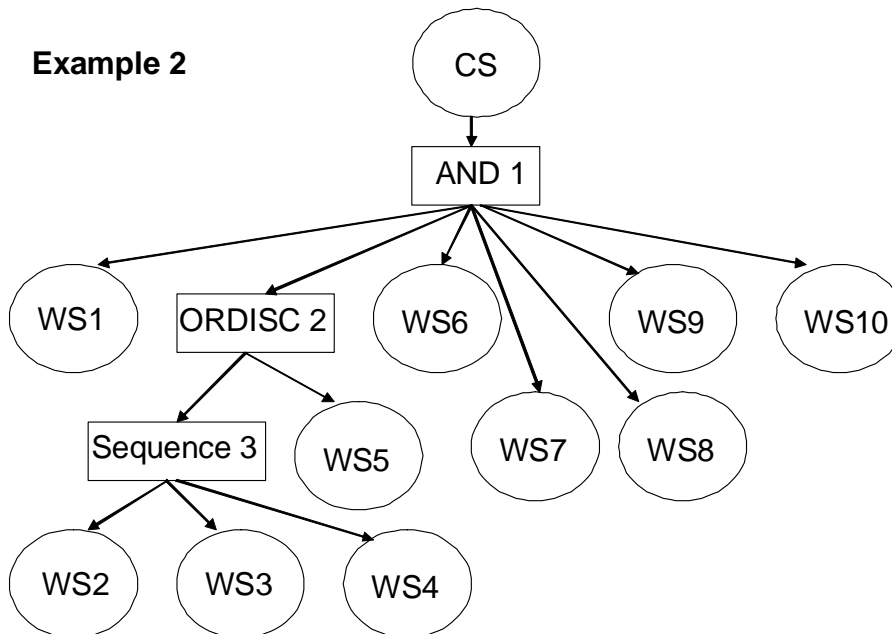


Type of dependency relation



Red: branch contributed more often than expected.

X: number of times per composition invocation that the branch contributed to the overall costs/response time.

Example 1**Example 2**

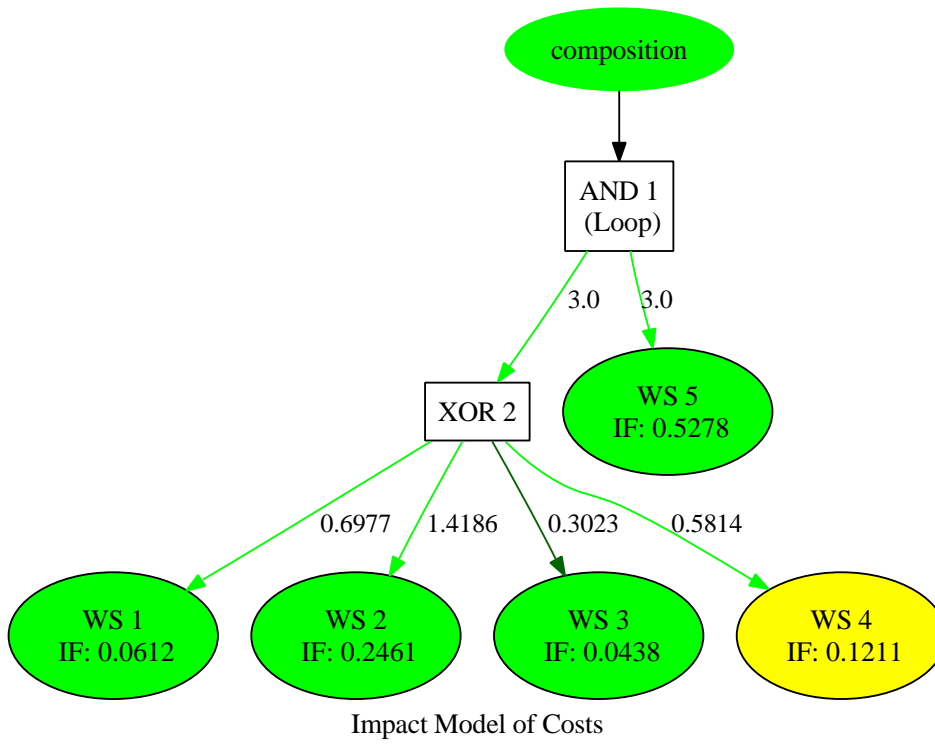
APPENDIX A. EVALUATION

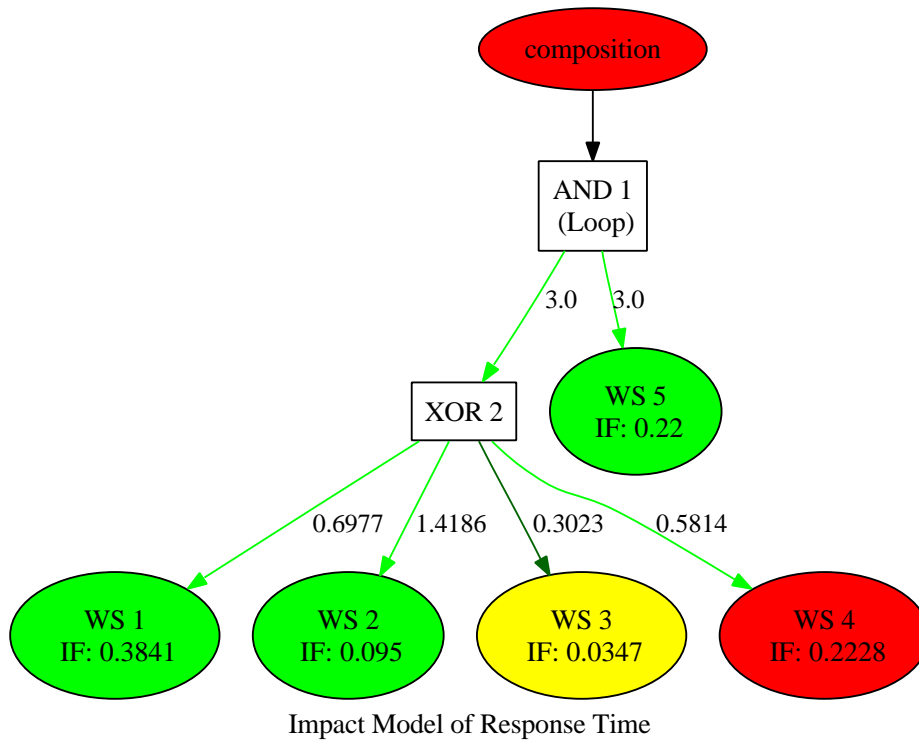
```
# -----
#
# date: 2009-01-05
#
# These are the design time estimations of the setup.
#
# -----
Composition has estimations:
  Response Time (min,mean,max): 37.0, 104.65268, 350.0
  Costs (min,mean,max):      1953.8778, 4003.2437, 6294.766
Loop (Estimated: 3 times). 1
|_Subelement:
|  XOR split and XOR join. 2
|  |_Subelement, [0.23]:
|  |  Web Service No 1 with QoS:
|  |    Response Time: ( min, mean, max): 55.0, 64.12, 103.0
|  |    Cost:          ( min, mean, max): 179.98412, 346.907, 446.46692
|  |_Subelement, [0.47]:
|  |  Web Service No 2 with QoS:
|  |    Response Time: ( min, mean, max): 6.0, 7.38, 9.0
|  |    Cost:          ( min, mean, max): 449.4833, 683.2266, 841.07556
|  |_Subelement, [0.11]:
|  |  Web Service No 3 with QoS:
|  |    Response Time: ( min, mean, max): 9.0, 12.53, 13.0
|  |    Cost:          ( min, mean, max): 306.47635, 575.76666, 1022.1091
|  |_Subelement, [0.19]:
|  |  Web Service No 4 with QoS:
|  |    Response Time: ( min, mean, max): 23.0, 37.96, 63.0
|  |    Cost:          ( min, mean, max): 652.1204, 795.49695, 1257.2106
|_Subelement:
  Web Service No 5 with QoS:
  Response Time: ( min, mean, max): 7.0, 8.08, 14.0
  Cost:          ( min, mean, max): 471.3085, 719.03064, 841.0449
```

```

# -----
#
# date: 2009-01-05
#
# These are the realized values of the setup.
# -----
Realized values of the composition:
  Response Time (min,mean,max): 10.0, 115.31396, 280.0
  Costs (min,mean,max):      807.55615, 4087.1555, 7654.6367
  Total number of invocations: 86
Loop (Estimated: 3 times). 1
On average 3.0 iterations per invocation, with;
1 iterations minimum
5 iterations maximum
Total # of invocations: 86
Total # of iterations: 258
|_Subelement:
|  XOR split and XOR join. 2
|  Total # of invocations: 258
|  |_Subelement, [0.23]
|  |  Web Service No 1 with QoS:
|  |    Response Time: ( min, mean, max): 5.0, 63.483334, 135.0
|  |    Cost:          ( min, mean, max): 4.4063416, 358.34943, 696.63635
|  |    Total # of invocations: 60
|  |_Subelement, [0.47]:
|  |  Web Service No 2 with QoS:
|  |    Response Time: ( min, mean, max): 2.0, 7.7213116, 11.0
|  |    Cost:          ( min, mean, max): 310.07037, 709.13855, 1258.3787
|  |    Total # of invocations: 122
|  |_Subelement, [0.1]:
|  |  Web Service No 3 with QoS:
|  |    Response Time: ( min, mean, max): 10.0, 13.230769, 17.0
|  |    Cost:          ( min, mean, max): 82.50397, 591.9862, 995.69275
|  |    Total # of invocations: 26
|  |_Subelement, [0.19]:
|  |  Web Service No 4 with QoS:
|  |    Response Time: ( min, mean, max): 7.0, 44.2, 80.0
|  |    Cost:          ( min, mean, max): 123.56213, 851.59, 1353.6824
|  |    Total # of invocations: 50
|_Subelement:
Web Service No 5 with QoS:
  Response Time: ( min, mean, max): 1.0, 8.457364, 19.0
  Cost:          ( min, mean, max): 224.15494, 719.02466, 1178.1434
  Total # of invocations: 258

```





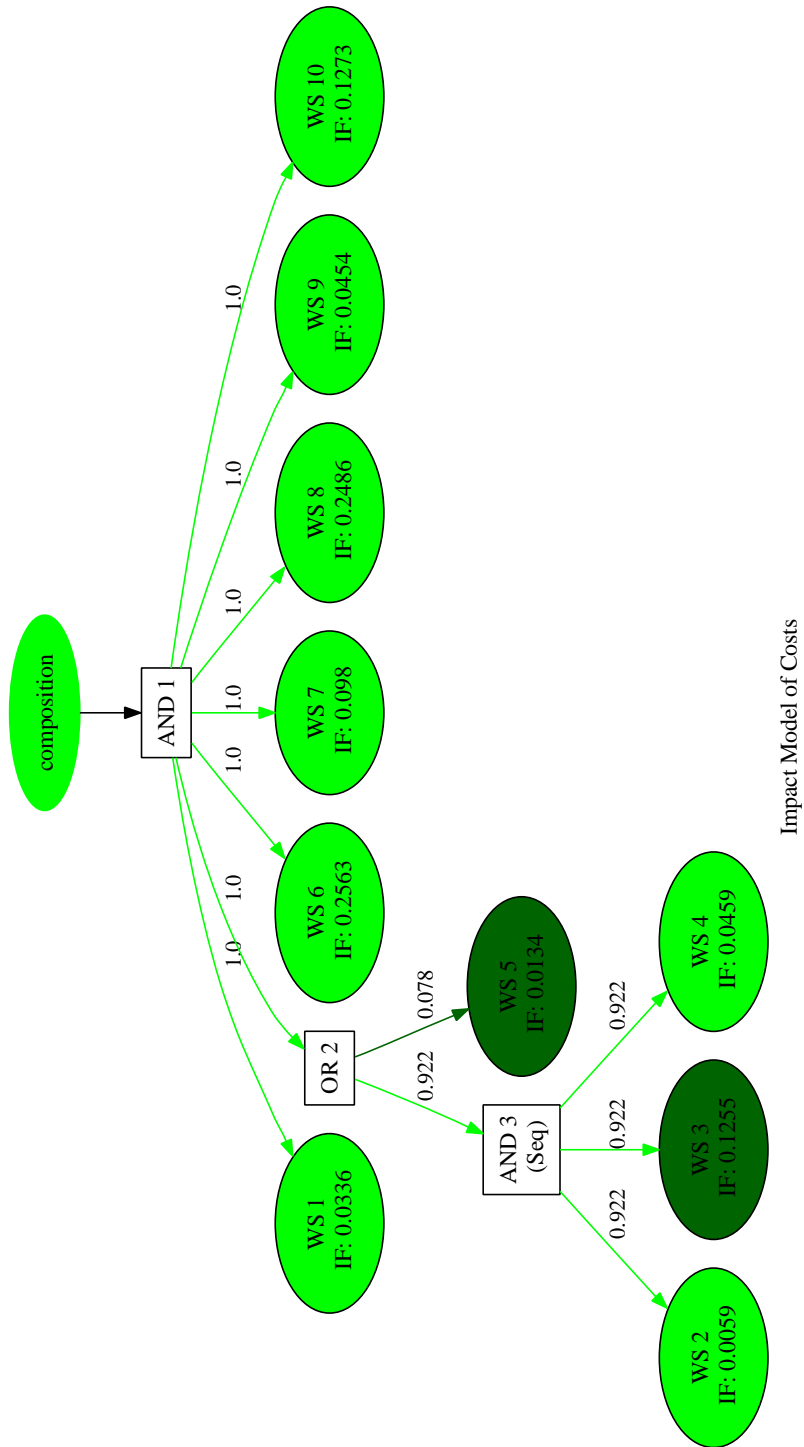
APPENDIX A. EVALUATION

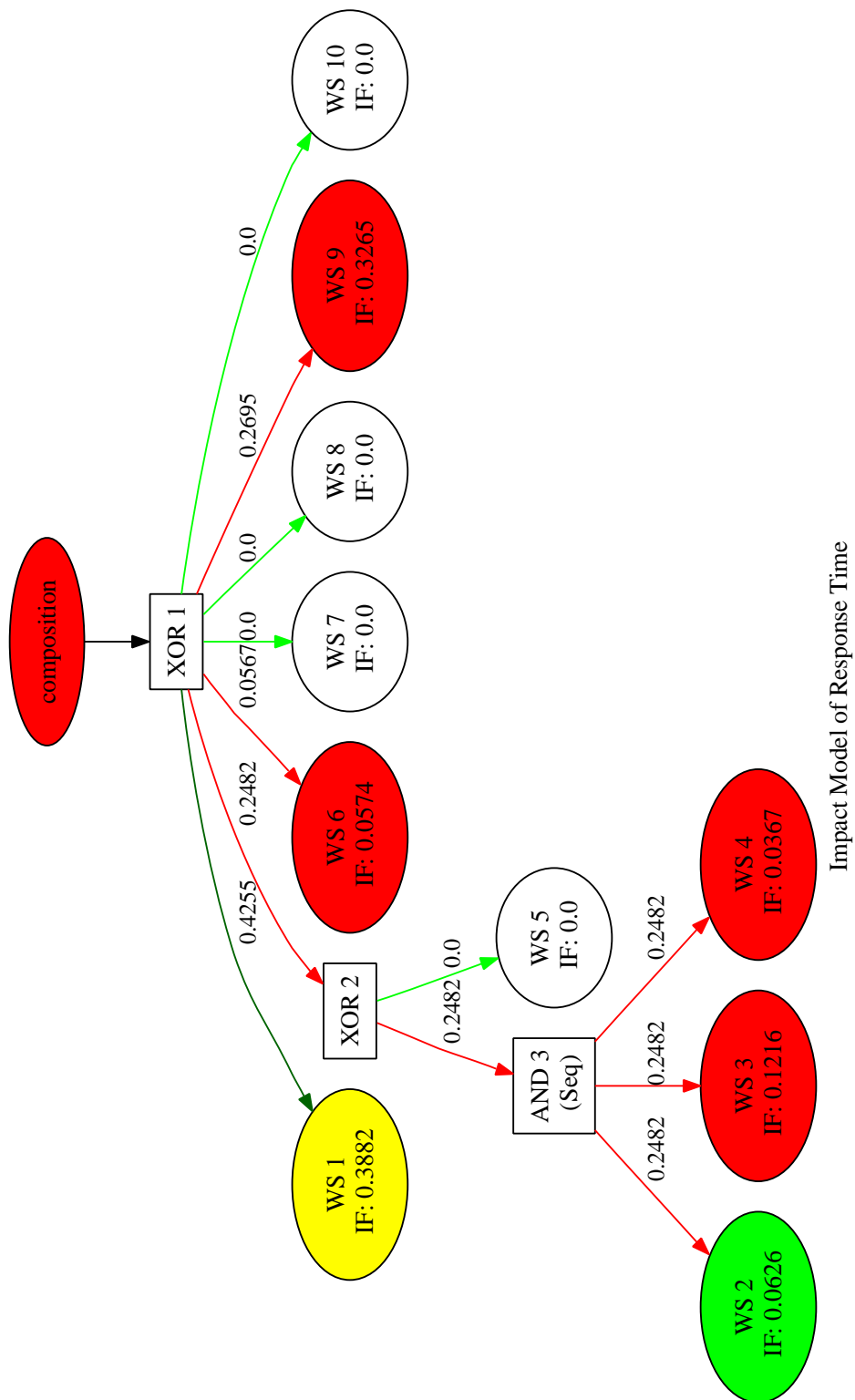
```
# -----
#
# date: 2009-01-05
#
# These are the design time estimations of the setup.
#
# -----
Composition has estimations:
  Response Time (min,mean,max): 60.0, 60.07926, 87.0
  Costs (min,mean,max):      4478.543, 5904.4717, 7152.936
AND split and join. 1
|_Subelement:
|  Web Service No 1 with QoS:
|    Response Time: ( min, mean, max): 60.0, 60.08, 71.0
|    Cost:          ( min, mean, max): 98.43849, 187.82, 276.05942
|_Subelement:
|  OR split, DISC join, where 1/2 services are started and 1 must finish. 2
|  |_Subelement, [0.91]:
|  |  Ordered or Unorderd Sequence. 3
|  |  |_Subelement:
|  |  |  Web Service No 2 with QoS:
|  |  |    Response Time: ( min, mean, max): 17.0, 17.18, 20.0
|  |  |    Cost:          ( min, mean, max): 28.15143, 36.083733, 62.51728
|  |  |  |_Subelement:
|  |  |  Web Service No 3 with QoS:
|  |  |    Response Time: ( min, mean, max): 27.0, 29.12, 39.0
|  |  |    Cost:          ( min, mean, max): 844.66565, 849.76044, 1207.5106
|  |  |  |_Subelement:
|  |  |  Web Service No 4 with QoS:
|  |  |    Response Time: ( min, mean, max): 8.0, 8.81, 14.0
|  |  |    Cost:          ( min, mean, max): 214.26782, 287.72424, 405.6331
|  |  |_Subelement, [0.09]:
|  |  |  Web Service No 5 with QoS:
|  |  |    Response Time: ( min, mean, max): 34.0, 39.99, 65.0
|  |  |    Cost:          ( min, mean, max): 917.90076, 1117.3857, 1208.3107
|  |_Subelement:
|  |  Web Service No 6 with QoS:
|  |  |  Response Time: ( min, mean, max): 31.0, 31.92, 59.0
|  |  |  Cost:          ( min, mean, max): 1376.125, 1495.7457, 1681.4296
|  |  |_Subelement:
|  |  |  Web Service No 7 with QoS:
|  |  |  Response Time: ( min, mean, max): 13.0, 20.83, 24.0
|  |  |  Cost:          ( min, mean, max): 509.5893, 567.05164, 768.5949
|  |  |_Subelement:
|  |  |  Web Service No 8 with QoS:
|  |  |  Response Time: ( min, mean, max): 19.0, 28.67, 45.0
|  |  |  Cost:          ( min, mean, max): 879.7758, 1452.2764, 1639.7931
|  |  |_Subelement:
|  |  |  Web Service No 9 with QoS:
|  |  |  Response Time: ( min, mean, max): 41.0, 51.99, 87.0
|  |  |  Cost:          ( min, mean, max): 224.44736, 271.22287, 344.06854
|  |  |_Subelement:
|  |  |  Web Service No 10 with QoS:
|  |  |  Response Time: ( min, mean, max): 30.0, 32.78, 44.0
|  |  |  Cost:          ( min, mean, max): 472.2663, 761.843, 767.32996
```

```

# -----
#
# date: 2009-01-05
#
# These are the realized values of the setup.
#
# -----
Realized values of the composition:
  Response Time (min,mean,max): 48.0, 69.95744, 118.0
  Costs (min,mean,max):      4461.6094, 5837.4336, 7363.0635
  Total number of invocations: 141
AND split and join. 1
|_Subelement:
|  Web Service No 1 with QoS:
|  | Response Time: ( min, mean, max): 38.0, 59.737587, 83.0
|  | Cost:         ( min, mean, max): 2.1773071, 196.4023, 506.32004
|  | Total # of invocations: 141
|_Subelement:
|  OR split, DISC join, where 1/2 services are started and 1 must finish. 2
|  Total # of invocations: 141
|  |_Subelement, [0.92]:
|  | Ordered or Unorderd Sequence. 3
|  | |_Subelement:
|  | | Web Service No 2 with QoS:
|  | | | Response Time: ( min, mean, max): 13.0, 17.46923, 24.0
|  | | | Cost:         ( min, mean, max): 0.64427567, 37.398582, 75.619675
|  | | | Total # of invocations: 130
|  | |_Subelement:
|  | | Web Service No 3 with QoS:
|  | | | Response Time: ( min, mean, max): 11.0, 29.476923, 46.0
|  | | | Cost:         ( min, mean, max): 221.93439, 794.90106, 1471.3582
|  | | | Total # of invocations: 130
|  | |_Subelement:
|  | | Web Service No 4 with QoS:
|  | | | Response Time: ( min, mean, max): 1.0, 9.2615385, 19.0
|  | | | Cost:         ( min, mean, max): 39.46582, 290.72437, 552.23596
|  | | | Total # of invocations: 130
|  |_Subelement, [0.08]:
|  | Web Service No 5 with QoS:
|  | | Response Time: ( min, mean, max): 14.0, 40.18182, 60.0
|  | | Cost:         ( min, mean, max): 746.75745, 1001.19104, 1312.5217
|  | | Total # of invocations: 11
|_Subelement:
|  Web Service No 6 with QoS:
|  | Response Time: ( min, mean, max): 1.0, 32.19858, 82.0
|  | Cost:         ( min, mean, max): 1139.2125, 1496.3722, 1959.8899
|  | Total # of invocations: 141
|_Subelement:
|  Web Service No 7 with QoS:
|  | Response Time: ( min, mean, max): 8.0, 21.021276, 38.0
|  | Cost:         ( min, mean, max): 199.32922, 571.8603, 848.80597
|  | Total # of invocations: 141
|_Subelement:
|  Web Service No 8 with QoS:
|  | Response Time: ( min, mean, max): 1.0, 28.269503, 61.0
|  | Cost:         ( min, mean, max): 437.3061, 1451.3611, 2627.58
|  | Total # of invocations: 141
|_Subelement:
|  Web Service No 9 with QoS:
|  | Response Time: ( min, mean, max): 3.0, 52.80142, 118.0
|  | Cost:         ( min, mean, max): 75.968735, 264.92352, 463.69916
|  | Total # of invocations: 141
|_Subelement:
|  Web Service No 10 with QoS:
|  | Response Time: ( min, mean, max): 10.0, 32.67376, 57.0
|  | Cost:         ( min, mean, max): 171.70697, 742.9943, 1202.0015
|  | Total # of invocations: 141

```





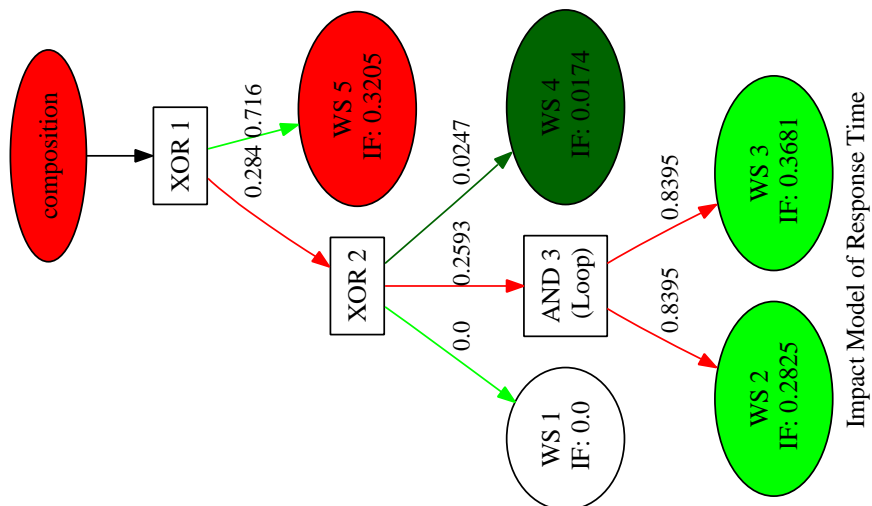
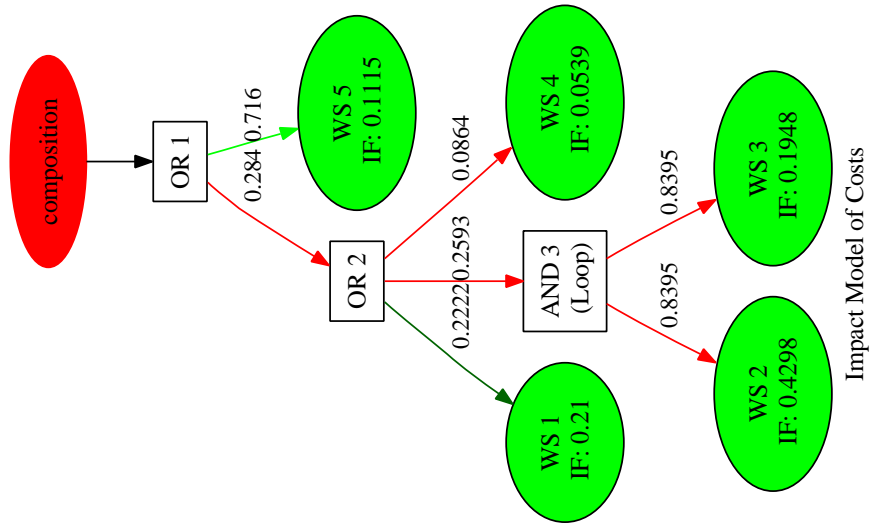
APPENDIX A. EVALUATION

```
# -----
#
# date: 2009-01-05
#
# These are the design time estimations of the setup.
#
# -----
Composition has estimations:
  Response Time (min,mean,max): 31.0, 101.255066, 346.0
  Costs (min,mean,max):      156.48984, 1128.1544, 5314.09
OR split, DISC join, where 1/2 services are started and 1 must finish. 1
|_Subelement, [0.25]:
|  OR split, OR join, where 2/3 services are started. 2
|  |_Subelement, [0.13]
|  |  Web Service No 1 with QoS:
|  |    Response Time: ( min, mean, max): 37.0, 41.94, 54.0
|  |    Cost:          ( min, mean, max): 1016.39996, 1294.8214, 1370.5468
|  |_Subelement, [0.1]:
|  |  Loop (Estimated: 3 times). 3
|  |  |_Subelement:
|  |  |  Web Service No 2 with QoS:
|  |  |    Response Time: ( min, mean, max): 36.0, 43.24, 49.0
|  |  |    Cost:          ( min, mean, max): 452.08832, 682.67126, 924.9143
|  |  |_Subelement:
|  |  |  Web Service No 3 with QoS:
|  |  |    Response Time: ( min, mean, max): 29.0, 53.36, 66.0
|  |  |    Cost:          ( min, mean, max): 229.46642, 328.3388, 389.59998
|  |_Subelement, [0.03]:
|  |  Web Service No 4 with QoS:
|  |    Response Time: ( min, mean, max): 71.0, 95.51, 97.0
|  |    Cost:          ( min, mean, max): 593.22174, 872.5337, 968.7437
|_Subelement, [0.75]:
  Web Service No 5 with QoS:
    Response Time: ( min, mean, max): 31.0, 49.91, 90.0
    Cost:          ( min, mean, max): 156.48984, 205.53635, 284.83356
```

```

# -----
#
# date: 2009-01-05
#
# These are the realized values of the setup.
#
# -----
Realized values of the composition:
  Response Time (min,mean,max): 2.0, 124.75309, 552.0
  Costs (min,mean,max): 72.18178, 1367.3472, 6921.927
  Total number of invocations: 81
OR split, DISC join, where 1/2 services are started and 1 must finish. 1
Total # of invocations: 81
|_Subelement, [0.28]:
|  OR split, OR join, where 2/3 services are started. 2
|  Total # of invocations: 23
|  |_Subelement, [0.39]:
|  |  Web Service No 1 with QoS:
|  |  |  Response Time: ( min, mean, max): 25.0, 44.11111, 59.0
|  |  |  Cost: ( min, mean, max): 917.0228, 1291.9982, 1716.5248
|  |  |  Total # of invocations: 18
|  |  |_Subelement, [0.46]:
|  |  |  Loop (Estimated: 3 times). 3
|  |  |  On average 3.2380953 iterations per invocation, with;
|  |  |  1 iterations minimum
|  |  |  5 iterations maximum
|  |  |  Total # of invocations: 21
|  |  |  Total # of iterations: 68
|  |  |_Subelement:
|  |  |  Web Service No 2 with QoS:
|  |  |  |  Response Time: ( min, mean, max): 25.0, 41.985294, 57.0
|  |  |  |  Cost: ( min, mean, max): 148.31451, 700.11615, 1286.4136
|  |  |  |  Total # of invocations: 68
|  |  |_Subelement:
|  |  |  Web Service No 3 with QoS:
|  |  |  |  Response Time: ( min, mean, max): 2.0, 54.705883, 96.0
|  |  |  |  Cost: ( min, mean, max): 133.13046, 317.24414, 494.24457
|  |  |  |  Total # of invocations: 68
|  |_Subelement, [0.15]:
|  |  Web Service No 4 with QoS:
|  |  |  Response Time: ( min, mean, max): 66.0, 86.42857, 102.0
|  |  |  Cost: ( min, mean, max): 646.3866, 852.32965, 1006.03174
|  |  |  Total # of invocations: 7
|  |_Subelement, [0.72]:
|  |  Web Service No 5 with QoS:
|  |  |  Response Time: ( min, mean, max): 2.0, 55.844826, 134.0
|  |  |  Cost: ( min, mean, max): 72.18178, 212.97133, 314.7603
|  |  |  Total # of invocations: 58

```



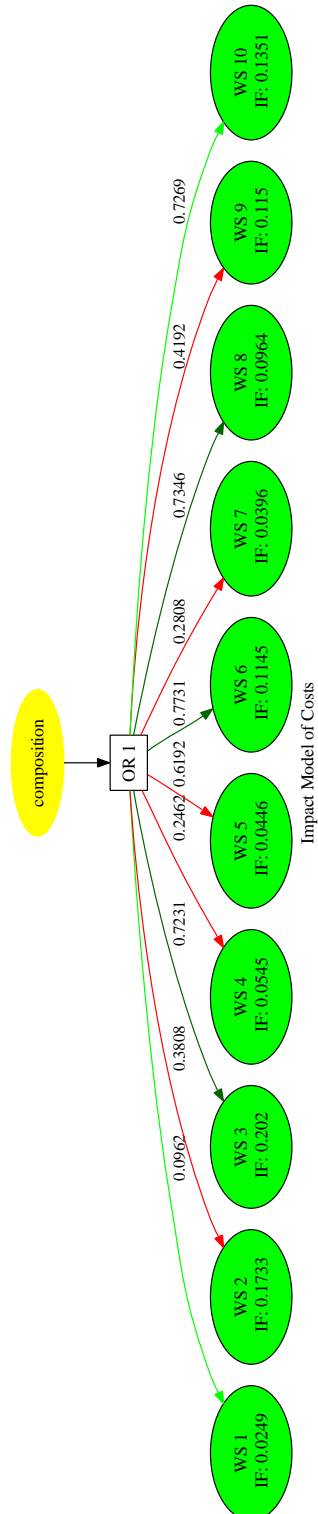

```

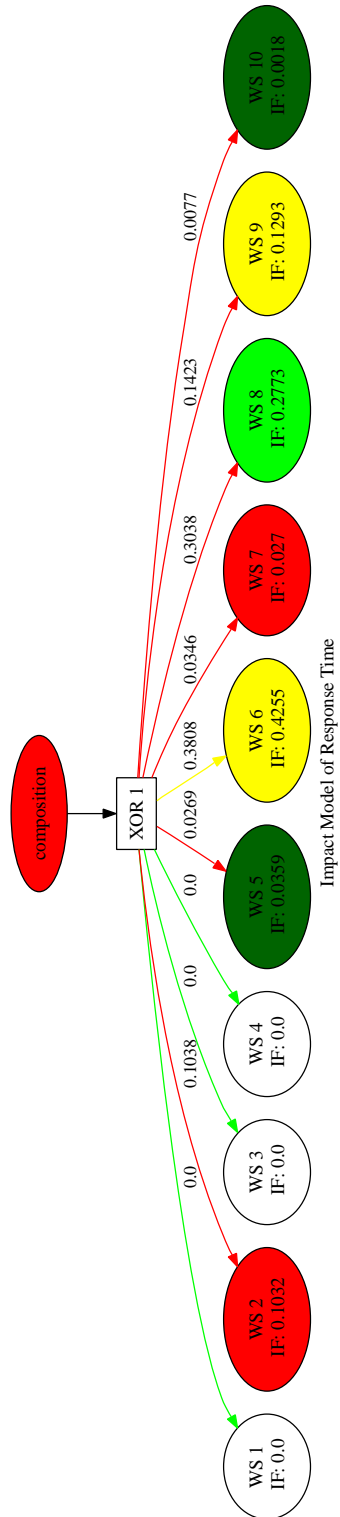
# -----
#
# date: 2009-01-05
#
# These are the design time estimations of the setup.
#
# -----
Composition has estimations:
  Response Time (min,mean,max): 8.0, 25.232832, 57.0
  Costs (min,mean,max):      1311.291, 2551.4988, 5316.124
OR split, DISC join, where 5/10 services are started and 4 must finish. 1
|_Subelement, [0.02]:
|   Web Service No 1 with QoS:
|     Response Time: ( min, mean, max): 3.0, 5.49, 10.0
|     Cost:          ( min, mean, max): 650.1171, 697.0652, 742.9949
|_Subelement, [0.06]:
|   Web Service No 2 with QoS:
|     Response Time: ( min, mean, max): 16.0, 30.5, 34.0
|     Cost:          ( min, mean, max): 910.12067, 1248.0664, 1479.9344
|_Subelement, [0.17]:
|   Web Service No 3 with QoS:
|     Response Time: ( min, mean, max): 7.0, 7.4, 13.0
|     Cost:          ( min, mean, max): 678.38586, 765.3185, 992.6084
|_Subelement, [0.04]:
|   Web Service No 4 with QoS:
|     Response Time: ( min, mean, max): 7.0, 10.01, 14.0
|     Cost:          ( min, mean, max): 340.1327, 616.41974, 820.01715
|_Subelement, [0.11]:
|   Web Service No 5 with QoS:
|     Response Time: ( min, mean, max): 56.0, 70.54, 92.0
|     Cost:          ( min, mean, max): 166.74295, 194.34271, 246.28497
|_Subelement, [0.17]:
|   Web Service No 6 with QoS:
|     Response Time: ( min, mean, max): 32.0, 38.66, 57.0
|     Cost:          ( min, mean, max): 383.56046, 412.27762, 652.6
|_Subelement, [0.04]:
|   Web Service No 7 with QoS:
|     Response Time: ( min, mean, max): 11.0, 20.22, 31.0
|     Cost:          ( min, mean, max): 214.47679, 379.56845, 542.52026
|_Subelement, [0.17]:
|   Web Service No 8 with QoS:
|     Response Time: ( min, mean, max): 30.0, 33.69, 39.0
|     Cost:          ( min, mean, max): 206.37807, 344.82913, 683.942
|_Subelement, [0.06]:
|   Web Service No 9 with QoS:
|     Response Time: ( min, mean, max): 20.0, 32.39, 40.0
|     Cost:          ( min, mean, max): 664.5739, 756.0107, 1280.5693
|_Subelement, [0.15]:
|   Web Service No 10 with QoS:
|     Response Time: ( min, mean, max): 8.0, 10.01, 15.0
|     Cost:          ( min, mean, max): 453.9847, 503.38693, 566.20245

```

APPENDIX A. EVALUATION

```
# -----
#
# date: 2009-01-05
#
# These are the realized values of the setup.
#
# -----
Realized values of the composition:
  Response Time (min,mean,max): 9.0, 37.603848, 77.0
  Costs (min,mean,max): 1385.0469, 2721.055, 4769.6475
  Total number of invocations: 260
OR split, DISC join, where 5/10 services are started and 4 must finish. 1
Total # of invocations: 260
|_Subelement, [0.02]:
|  Web Service No 1 with QoS:
|    Response Time: ( min, mean, max): 1.0, 6.64, 15.0
|    Cost: ( min, mean, max): 638.6256, 705.8908, 822.35046
|    Total # of invocations: 25
|_Subelement, [0.08]:
|  Web Service No 2 with QoS:
|    Response Time: ( min, mean, max): 1.0, 29.876404, 53.0
|    Cost: ( min, mean, max): 464.2979, 1238.5753, 2147.8433
|    Total # of invocations: 99
|_Subelement, [0.14]:
|  Web Service No 3 with QoS:
|    Response Time: ( min, mean, max): 1.0, 7.208556, 16.0
|    Cost: ( min, mean, max): 336.78345, 760.00507, 1241.7505
|    Total # of invocations: 188
|_Subelement, [0.05]:
|  Web Service No 4 with QoS:
|    Response Time: ( min, mean, max): 3.0, 10.78125, 16.0
|    Cost: ( min, mean, max): 120.918, 602.12177, 1098.9824
|    Total # of invocations: 64
|_Subelement, [0.12]:
|  Web Service No 5 with QoS:
|    Response Time: ( min, mean, max): 19.0, 45.22222, 66.0
|    Cost: ( min, mean, max): 76.74548, 196.04285, 284.44122
|    Total # of invocations: 161
|_Subelement, [0.15]:
|  Web Service No 6 with QoS:
|    Response Time: ( min, mean, max): 5.0, 35.953335, 77.0
|    Cost: ( min, mean, max): 21.71933, 402.92, 892.9569
|    Total # of invocations: 201
|_Subelement, [0.06]:
|  Web Service No 7 with QoS:
|    Response Time: ( min, mean, max): 1.0, 19.12676, 42.0
|    Cost: ( min, mean, max): 72.82269, 384.16068, 767.1182
|    Total # of invocations: 73
|_Subelement, [0.15]:
|  Web Service No 8 with QoS:
|    Response Time: ( min, mean, max): 21.0, 33.520958, 42.0
|    Cost: ( min, mean, max): 12.257751, 357.24112, 918.28345
|    Total # of invocations: 191
|_Subelement, [0.08]:
|  Web Service No 9 with QoS:
|    Response Time: ( min, mean, max): 3.0, 30.539326, 47.0
|    Cost: ( min, mean, max): 64.76129, 746.33026, 2026.8804
|    Total # of invocations: 109
|_Subelement, [0.15]:
|  Web Service No 10 with QoS:
|    Response Time: ( min, mean, max): 1.0, 10.772487, 21.0
|    Cost: ( min, mean, max): 357.44855, 505.89777, 668.23944
|    Total # of invocations: 189
```





```

# -----
#
# date: 2009-01-05
#
# These are the design time estimations of the setup.
# -----
Composition has estimations:
  Response Time (min,mean,max): 41.0, 117.02195, 725.0
  Costs (min,mean,max):      4415.4473, 8245.411, 22181.707
AND split and join. 1
|_Subelement:
|  Web Service No 1 with QoS:
|    Response Time: ( min, mean, max): 10.0, 11.63, 18.0
|    Cost:          ( min, mean, max): 210.02309, 313.72, 513.5868
|_Subelement:
|  Web Service No 2 with QoS:
|    Response Time: ( min, mean, max): 33.0, 37.69, 47.0
|    Cost:          ( min, mean, max): 846.5581, 963.62225, 1339.1849
|_Subelement:
|  Web Service No 3 with QoS:
|    Response Time: ( min, mean, max): 22.0, 33.45, 34.0
|    Cost:          ( min, mean, max): 372.75146, 644.95825, 1061.9478
|_Subelement:
|  Web Service No 4 with QoS:
|    Response Time: ( min, mean, max): 31.0, 43.39, 60.0
|    Cost:          ( min, mean, max): 736.048, 909.04694, 1338.7078
|_Subelement:
|  Web Service No 5 with QoS:
|    Response Time: ( min, mean, max): 7.0, 12.19, 18.0
|    Cost:          ( min, mean, max): 100.35527, 168.35362, 198.89468
|_Subelement:
|  AND split and join. 2
|    |_Subelement:
|      |  Web Service No 6 with QoS:
|      |    Response Time: ( min, mean, max): 41.0, 59.62, 62.0
|      |    Cost:          ( min, mean, max): 442.9783, 454.52646, 591.99786
|      |_Subelement:
|        |  Web Service No 7 with QoS:
|        |    Response Time: ( min, mean, max): 36.0, 60.51, 61.0
|        |    Cost:          ( min, mean, max): 201.85391, 318.47342, 325.6623
|        |_Subelement:
|          XOR split and XOR join. 3
|            |_Subelement, [0.72]:
|              |  Web Service No 8 with QoS:
|              |    Response Time: ( min, mean, max): 32.0, 60.31, 67.0
|              |    Cost:          ( min, mean, max): 525.27045, 618.8252, 664.4616
|              |_Subelement, [0.15]:
|                Loop (Estimated: 5 times). 4
|                  |_Subelement:
|                    |  Web Service No 9 with QoS:
|                    |    Response Time: ( min, mean, max): 17.0, 31.58, 33.0
|                    |    Cost:          ( min, mean, max): 804.11975, 1158.6505, 1446.9174
|                    |_Subelement:
|                      Web Service No 10 with QoS:
|                        Response Time: ( min, mean, max): 16.0, 28.44, 54.0
|                        Cost:          ( min, mean, max): 35.12873, 53.35737, 71.33427

```

APPENDIX A. EVALUATION

```
| | | | _Subelement:
| | | |   Web Service No 11 with QoS:
| | | |     Response Time: ( min, mean, max): 31.0, 36.78, 59.0
| | | |     Cost:         ( min, mean, max): 830.5941, 837.7012, 1149.1959
| | | | _Subelement, [0.13]:
| | | |   Web Service No 12 with QoS:
| | | |     Response Time: ( min, mean, max): 5.0, 7.69, 11.0
| | | |     Cost:         ( min, mean, max): 625.7165, 1118.7761, 1415.0142
| | | | _Subelement:
| | | |   Web Service No 13 with QoS:
| | | |     Response Time: ( min, mean, max): 13.0, 19.05, 32.0
| | | |     Cost:         ( min, mean, max): 694.0153, 1381.3864, 1965.6458
| | | | _Subelement:
| | | |   OR split, DISC join, where 1/2 services are started and 1 must finish. 5
| | | | _Subelement, [0.89]:
| | | |   Web Service No 14 with QoS:
| | | |     Response Time: ( min, mean, max): 20.0, 22.43, 24.0
| | | |     Cost:         ( min, mean, max): 502.37628, 567.51276, 859.6056
| | | | _Subelement, [0.11]:
| | | |   Web Service No 15 with QoS:
| | | |     Response Time: ( min, mean, max): 26.0, 50.12, 81.0
| | | |     Cost:         ( min, mean, max): 6.94027, 9.2576885, 13.704545
| | | | _Subelement:
| | | |   Web Service No 16 with QoS:
| | | |     Response Time: ( min, mean, max): 4.0, 8.1, 9.0
| | | |     Cost:         ( min, mean, max): 156.71298, 285.16486, 325.40204
| | | | _Subelement:
| | | |   Web Service No 17 with QoS:
| | | |     Response Time: ( min, mean, max): 30.0, 42.86, 58.0
| | | |     Cost:         ( min, mean, max): 121.94054, 171.77739, 323.83493
```

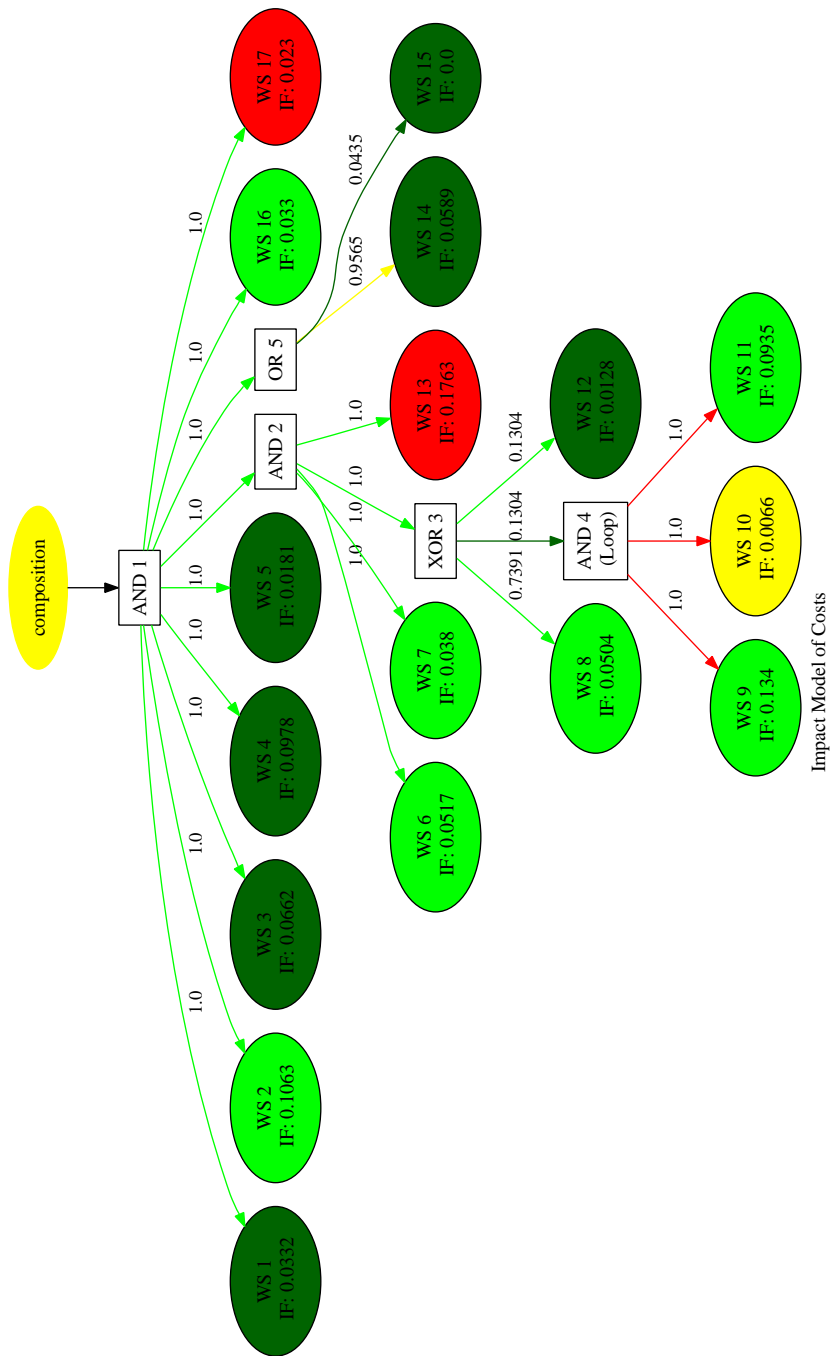
```

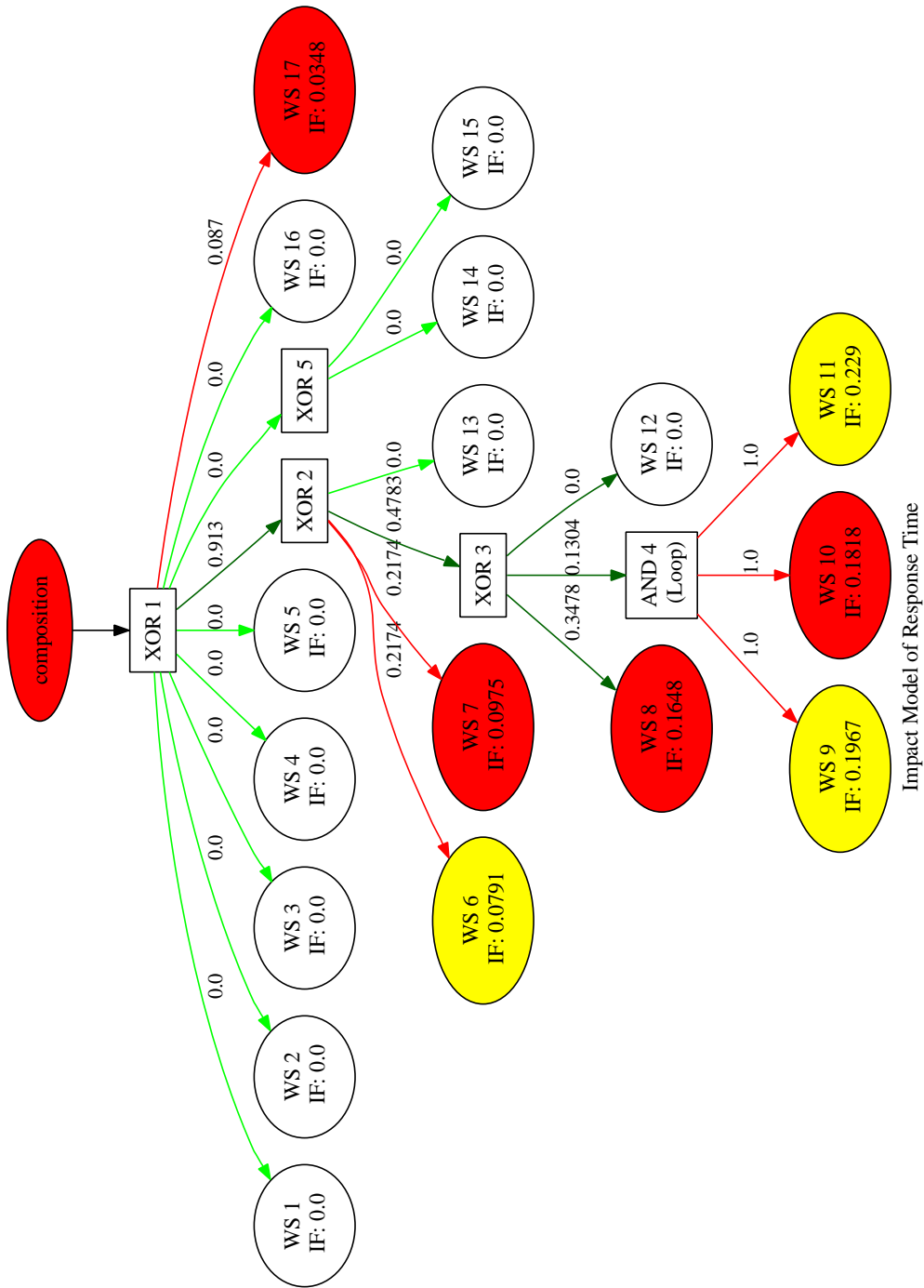
# -----
#
# date: 2009-01-05
#
# These are the realized values of the setup.
# -----
Realized values of the composition:
  Response Time (min,mean,max): 48.0, 174.86957, 1036.0
  Costs (min,mean,max): 5590.6934, 8678.282, 25450.111
  Total number of invocations: 23
AND split and join. 1
|_Subelement:
|  Web Service No 1 with QoS:
|    Response Time: ( min, mean, max): 7.0, 12.565217, 21.0
|    Cost: ( min, mean, max): 39.184357, 288.40488, 607.5144
|    Total # of invocations: 23
|_Subelement:
|  Web Service No 2 with QoS:
|    Response Time: ( min, mean, max): 30.0, 40.434784, 63.0
|    Cost: ( min, mean, max): 335.96082, 922.88495, 1502.9639
|    Total # of invocations: 23
|_Subelement:
|  Web Service No 3 with QoS:
|    Response Time: ( min, mean, max): 20.0, 37.869564, 54.0
|    Cost: ( min, mean, max): 161.43155, 574.28046, 1251.6294
|    Total # of invocations: 23
|_Subelement:
|  Web Service No 4 with QoS:
|    Response Time: ( min, mean, max): 14.0, 40.04348, 69.0
|    Cost: ( min, mean, max): 341.64148, 848.5004, 1338.3586
|    Total # of invocations: 23
|_Subelement:
|  Web Service No 5 with QoS:
|    Response Time: ( min, mean, max): 1.0, 13.782609, 21.0
|    Cost: ( min, mean, max): 62.765213, 157.46265, 242.28073
|    Total # of invocations: 23
|_Subelement:
|  AND split and join. 2
|    |_Subelement:
|      Web Service No 6 with QoS:
|        Response Time: ( min, mean, max): 33.0, 61.608696, 92.0
|        Cost: ( min, mean, max): 257.6671, 449.0448, 714.92725
|        Total # of invocations: 23
|      |_Subelement:
|        Web Service No 7 with QoS:
|          Response Time: ( min, mean, max): 31.0, 61.304348, 104.0
|          Cost: ( min, mean, max): 172.90501, 330.1317, 451.6234
|          Total # of invocations: 23
|      |_Subelement:
|        XOR split and XOR join. 3
|        Total # of invocations: 23
|        |_Subelement, [0.74]:
|          Web Service No 8 with QoS:
|            Response Time: ( min, mean, max): 33.0, 67.588234, 101.0
|            Cost: ( min, mean, max): 456.34702, 591.18994, 754.9042
|            Total # of invocations: 17

```

APPENDIX A. EVALUATION

```
| | | | _Subelement, [0.13]:
| | | |   Loop (Estimated: 5 times). 4
| | | |   On average 7.6666665 iterations per invocation, with;
| | | |   7 iterations minimum
| | | |   9 iterations maximum
| | | |   Total # of invocations: 3
| | | |   Total # of iterations: 23
| | | | | _Subelement:
| | | | |   Web Service No 9 with QoS:
| | | | |     Response Time: ( min, mean, max): 17.0, 34.391304, 55.0
| | | | |     Cost:         ( min, mean, max): 655.4398, 1162.9155, 1749.987
| | | | |     Total # of invocations: 23
| | | | | _Subelement:
| | | | |   Web Service No 10 with QoS:
| | | | |     Response Time: ( min, mean, max): 9.0, 31.782608, 66.0
| | | | |     Cost:         ( min, mean, max): 28.277946, 57.33591, 104.81079
| | | | |     Total # of invocations: 23
| | | | | _Subelement:
| | | | |   Web Service No 11 with QoS:
| | | | |     Response Time: ( min, mean, max): 7.0, 40.04348, 81.0
| | | | |     Cost:         ( min, mean, max): 363.83832, 811.3514, 1204.1954
| | | | |     Total # of invocations: 23
| | | | | _Subelement, [0.13]:
| | | | |   Web Service No 12 with QoS:
| | | | |     Response Time: ( min, mean, max): 3.0, 7.0, 14.0
| | | | |     Cost:         ( min, mean, max): 388.61462, 854.22363, 1091.9017
| | | | |     Total # of invocations: 3
| | | | | _Subelement:
| | | | |   Web Service No 13 with QoS:
| | | | |     Response Time: ( min, mean, max): 10.0, 22.043478, 37.0
| | | | |     Cost:         ( min, mean, max): 203.39478, 1530.2379, 2630.816
| | | | |     Total # of invocations: 23
| | | | | _Subelement:
| | | | |   OR split, DISC join, where 1/2 services are started and 1 must finish. 5
| | | | | Total # of invocations: 23
| | | | | | _Subelement, [0.96]:
| | | | | |   Web Service No 14 with QoS:
| | | | | |     Response Time: ( min, mean, max): 19.0, 22.545454, 26.0
| | | | | |     Cost:         ( min, mean, max): 67.887665, 534.0669, 822.7224
| | | | | |     Total # of invocations: 22
| | | | | | _Subelement, [0.04]:
| | | | | |   Web Service No 15 with QoS:
| | | | | |     Response Time: ( min, mean, max): 21.0, 21.0, 21.0
| | | | | |     Cost:         ( min, mean, max): 3.080326, 3.080326, 3.080326
| | | | | |     Total # of invocations: 1
| | | | | | _Subelement:
| | | | | |   Web Service No 16 with QoS:
| | | | | |     Response Time: ( min, mean, max): 3.0, 8.913043, 18.0
| | | | | |     Cost:         ( min, mean, max): 128.24422, 286.44473, 454.65692
| | | | | |     Total # of invocations: 23
| | | | | | _Subelement:
| | | | | |   Web Service No 17 with QoS:
| | | | | |     Response Time: ( min, mean, max): 26.0, 49.608696, 75.0
| | | | | |     Cost:         ( min, mean, max): 25.330856, 199.91945, 365.5691
| | | | | |     Total # of invocations: 23
```



SurveyGizmo: Practical Online Survey Software. Create web surveys ea...

http://app.sgzmo.com/surveybuilder/survey_editor.php?id=88090Survey: **Validating Usefulness of MoDe4SLA**Status: **Launched** (survey active)**1. Before starting the evaluation**

Copy page • Delete page •

Before starting the evaluation, please answer the following questions.

1. Do you consider yourself to be from academia or from industry?

- ☐ Industry
- ☐ Academia
- ☐ Both

2. Do you have experience using tools to support management of composite services?

- ☐ Yes
- ☐ No
- ☐ Don't know

3. How do you currently manage composite services?

4. Have you ever developed an approach for managing composite services?

- ☐ Yes
- ☐ No

5. How many years have you worked using composite services?

- ☐ None
- ☐ Up to 1 year
- ☐ 1-2 years
- ☐ 2-3 years
- ☐ More than 3 years

6. My level of expertise concerning management of composite services is high.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

7. I consider research concerning management of composite services as necessary.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

2. Evaluation without MoDe4SLA: 5 €

Copy page • Delete page •

Evaluation for the composition with **5 services**.

Answer the following question after seeing only **the structure** (= expected values).

8. The offered composition appears to be complex.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

After seeing runtime results **without MoDe4SLA**, please answer the following questions:

9. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

10. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

11.

Assume only a subset of services can be renegotiated regarding their SLAS.

I would feel confident in selecting services for renegotiation.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

3. Evaluation with MoDe4SLA: 5 Serv

Copy page • Delete page •

After seeing runtime results **with MoDe4SLA**, please answer the following questions:

12. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

- ☐ Less confident ☐ Equally confident ☐ More confident

13. Concerning Costs: I need less time to see relations between the different services and the composition.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

14. Concerning Response time: I need less time to see relations between the different services and the composition.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

15. Concerning Costs: I find it easier to determine the impact each service has on the composition than without the analysis.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

16. Concerning Response time: I find it easier to determine the impact each service has on the composition than without the analysis.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

17. Assume only a subset of services can be renegotiated regarding their SLAs. I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

18. MoDe4SLA approach is helpful when managing this composition with regard to:

- | | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|--|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Accuracy (accurate depicting of malfunctioning services) | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| Efficiency (faster depicting of these services) | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

4. Evaluation without MoDe4SLA: 10

Copy page • Delete page •

Evaluation for the composition with **10 services**.

Answer the following question after seeing only **the structure** (= expected values).

19. The offered composition appears to be complex.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

After seeing runtime results **without MoDe4SLA**, please answer the following questions:

20. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

21. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

22.

Assume only a subset of services can be renegotiated regarding their SLAS.
I would feel confident in selecting services for renegotiation.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

5. Evaluation with MoDe4SLA: 10 Ser

Copy page • Delete page •

After seeing runtime results **with MoDe4SLA**, please answer the following questions:

23. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

- ☐ Less confident ☐ Equally confident ☐ More confident

24. Concerning Costs.

I need less time to see relations between the different services and the composition.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

25. Concerning Response time.

I need less time to see relations between the different services and the composition.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

26. Concerning Costs.

I find it easier to determine the impact each service has on the composition than without the analysis.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

27. Concerning Response time.

I find it easier to determine the impact each service has on the composition than without the analysis.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

28. Assume only a subset of services can be renegotiated regarding their SLAs.

I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

- | | | | | |
|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

29. MoDe4SLA approach is helpful when managing this composition with regard to:

- | | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|--|-----------------------|-----------------------|----------------------------|-----------------------|-----------------------|
| Accuracy (accurate depicting of malfunctioning services) | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| Efficiency (faster depicting of these services) | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

6. Evaluation without MoDe4SLA: 17

Copy page • Delete page •

Evaluation for the composition with **17 services**.

Answer the following question after seeing only **the structure** (= expected values).

30. The offered composition appears to be complex.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

After seeing runtime results **without MoDe4SLA**, please answer the following questions:

31. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

32. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

33.

Assume only a subset of services can be renegotiated regarding their SLAS.

I would feel confident in selecting services for renegotiation.

Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

☐ ☐ ☐ ☐ ☐

7. Evaluation with MoDe4SLA: 17 Ser

Copy page • Delete page •

After seeing runtime results **with MoDe4SLA**, please answer the following questions:

34. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

☐ Less confident ☐ Equally confident ☐ More confident

35. Concerning Costs.

I need less time to see relations between the different services and the composition.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

36. Concerning Response time.

I need less time to see relations between the different services and the composition.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

37. Concerning Costs.

I find it easier to determine the impact each service has on the composition than without the analysis.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

38. Concerning Response time.

I find it easier to determine the impact each service has on the composition than without the analysis.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

39. Assume only a subset of services can be renegotiated regarding their SLAs.

I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

40. MoDe4SLA approach is helpful when managing this composition with regard to:

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Accuracy (accurate depicting of malfunctioning services)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Efficiency (faster depicting of these services)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. After all compositions are evaluated

Copy page • Delete page •

After evaluating all compositions, please answer the following questions.

41. I consider research concerning management of composite services as necessary.

Strongly disagree ☐ Disagree ☐ Neither agree nor disagree ☐ Agree ☐ Strongly agree ☐

42. The presentation before the evaluation was sufficient to properly understand the MoDe4SLA approach.

Strongly disagree ☐ Disagree ☐ Neither agree nor disagree ☐ Agree ☐ Strongly agree ☐

43. MoDe4SLA is helpful for managing service compositions.

Strongly disagree ☐ Disagree ☐ Neither agree nor disagree ☐ Agree ☐ Strongly agree ☐

44. What part of MoDe4SLA do you consider most beneficial?

45. What are the weak points of MoDe4SLA (possible improvements)?

46. Are you aware of any other approaches supporting management of service compositions? If so, what is the name of the approach?

☐ No
☐ Yes

47. MoDe4SLA is more helpful than the above mentioned approaches.

Yes ☐ No ☐ Not applicable ☐

48. Please, fill out your name and email so that we can get back to you if we have some questions and so that we can keep you informed about the results of this evaluation.

Please, note that we will not use your name and email for anything other than above mentioned purposes.

Name:

Email:

Test 1**Concerning Costs:**

The costs are higher than expected because the expensive loop branch (construct 3) is chosen more often than expected. As a result more services are invoked per composition invocation. Therefore overall costs are higher. Either the structure of the composition should be changed or the SLA of the composition itself has to be reconsidered.

Concerning Response Time (RT):

Also the RT is higher than expected because the loop branch is chosen more often than expected. In addition WS 5 has a high impact on the composition (32% of the overall response time is contributed by this service). The service is not chosen more often than expected but is performing bad. So, either the structure has to be improved and/or the SLA of WS 5 concerning RT should be renegotiated.

Test 2**Concerning Costs:**

Although all services are performing within boundaries, the overall composition is exceeding its costs. Considering red branches in combination with high impact factors, leads to the following conclusion: WS 2 and WS 9 have a high impact and contribute more than expected. They cause a big part of the overall costs. Either the structure should be changed, WS 2 and/or WS 9 should be renegotiated or replaced by cheaper services, or the SLA of the composition should be adapted.

Concerning RT:

The composition is performing badly because of several factors. WS 6 is contributing a lot to the overall RT (43%). This is more contribution and more violation than expected. This service should be reconsidered. In addition both WS 2 and WS 9 are contributing more and are not performing well. Also these can be reconsidered. Although WS 7 is not doing well, its impact is relatively low. WS 8 is performing fine but since its impact is high and it is contributing more than expected, it might be worth replacing it with a faster one. Due to the numerous violations no single recommendation can be appointed without considering additional criteria from the application domain, which are not known here.

Test 3**Concerning Costs:**

The bad performance cost wise is mainly caused by WS 13 and by the loop which is performed more often than expected. Renegotiating WS 13, changing the loop structure, and replacing WS 9 and WS 11 with cheaper services, are good options. The impact of WS 17 and WS 10 are too low to make a real difference when renegotiated and can therefore be neglected.

Concerning RT:

The overall bad performance RT wise is due to all red and yellow services. Especially since also all the branches reaching them are red. Therefore choose the ones with the highest impact: WS 11, WS 10, and WS 9 are good option. Also WS 8 might be considered because of the many branches. Since so many branches are red, reconsidering the overall structure is also a good choice.

A.3 Survey results



Online Surveys
Data Collection
and Integration

www.surveymonkey.com

Report: Response Summary Report #7

Survey: Validating Usefulness of MoDe4SLA

Compiled: 07/30/2009

1. Do you consider yourself to be from academia or from industry?

Summary

Value	Count	Percent %
Industry	2	5.88%
Academia	23	67.65%
Both	9	26.47%

Statistics

Choices Selected: 34

Total Responses: 34

2. Do you have experience using tools to support management of composite services?

Summary

Value	Count	Percent %
Yes	5	14.71%
No	28	82.35%
Don't know	1	2.94%

Statistics

Choices Selected: 34

Total Responses: 34

3. Have you ever developed an approach for managing composite services?

Summary

Value	Count	Percent %
-------	-------	-----------

Summary

Value	Count	Percent %
Yes	5	14.71%
No	29	85.29%

Statistics

Choices Selected: 34

Total Responses: 34

4. How many years have you worked using composite services?

Summary

Value	Count	Percent %
None	20	58.82%
Up to 1 year	4	11.76%
1-2 years	1	2.94%
2-3 years	5	14.71%
More than 3 years	4	11.76%

Statistics

Choices Selected: 34

Total Responses: 34

5. My level of expertise concerning management of composite services is high.

Summary

Value	Count	Percent %
Strongly disagree	10	29.41%
Disagree	10	29.41%
Neither agree nor disagree	11	32.35%
Agree	2	5.88%
Strongly agree	1	2.94%

Statistics

Choices Selected: 34

Total Responses: 34

6. I consider research concerning management of composite services as necessary.

Summary

APPENDIX A. EVALUATION

Value	Count	Percent %
Neither agree nor disagree	3	8.82%
Agree	18	52.94%
Strongly agree	13	38.24%
Statistics		
Choices Selected:	34	
Total Responses:	34	

7. The offered composition appears to be complex.

Summary		
Value	Count	Percent %
Strongly disagree	2	5.88%
Disagree	10	29.41%
Neither agree nor disagree	11	32.35%
Agree	10	29.41%
Strongly agree	1	2.94%
Statistics		
Choices Selected:	34	
Total Responses:	34	

8. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Summary		
Value	Count	Percent %
Strongly disagree	3	8.82%
Disagree	10	29.41%
Neither agree nor disagree	10	29.41%
Agree	11	32.35%
Statistics		
Choices Selected:	34	
Total Responses:	34	

9. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Summary

Value	Count	Percent %
Strongly disagree	4	11.76%
Disagree	14	41.18%
Neither agree nor disagree	8	23.53%
Agree	8	23.53%

Statistics
 Choices Selected: 34
 Total Responses: 34

10. Assume only a subset of services can be renegotiated regarding their SLAS.
 I would feel confident in selecting services for renegotiation.

Summary		
Value	Count	Percent %
Strongly disagree	3	8.82%
Disagree	10	29.41%
Neither agree nor disagree	8	23.53%
Agree	12	35.29%
Strongly agree	1	2.94%

Statistics
 Choices Selected: 34
 Total Responses: 34

11. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

Summary		
Value	Count	Percent %
Less confident	10	30.30%
Equally confident	7	21.21%
More confident	16	48.48%

Statistics
 Choices Selected: 33
 Total Responses: 33

12. Concerning Costs: I need less time to see relations between the different services and the composition.

Summary		
Value	Count	Percent %
Disagree	1	2.94%
Agree	19	55.88%
Strongly agree	14	41.18%
Statistics		
Choices Selected:	34	
Total Responses:	34	

13. Concerning Response time: I need less time to see relations between the different services and the composition.

Summary			
Value	Count	Percent	%
Disagree	2	5.88%	
Agree	17	50.00%	
Strongly agree	15	44.12%	
Statistics			
Choices Selected:		34	
Total Responses:		34	
Report from www.SurveyGizmo.com			

14. Concerning Costs: I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Disagree	2	5.88%
Neither agree nor disagree	2	5.88%
Agree	18	52.94%
Strongly agree	12	35.29%
Statistics		
Choices Selected:	34	
Total Responses:	34	

15. Concerning Response time: I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Neither agree nor disagree	2	5.88%
Agree	17	50.00%
Strongly agree	15	44.12%
Statistics		
Choices Selected:	34	
Total Responses:	34	

16. Assume only a subset of services can be renegotiated regarding their SLAs.
I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

Summary		
Value	Count	Percent %
Disagree	1	2.94%
Neither agree nor disagree	2	5.88%
Agree	19	55.88%
Strongly agree	12	35.29%
Statistics		
Choices Selected:	34	
Total Responses:	34	

17. MoDe4SLA approach is helpful when managing this composition with regard to:

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Accuray (accurate depicting of malfunctioning services)	2.9%1	5.9%2	11.8%4	55.9%19	23.5%8	34
Efficiency (faster depicting of these services)	-	-	8.8%3	55.9%19	35.3%12	34

APPENDIX A. EVALUATION

Average %: 1.5% 2.9% 10.3% 55.9% 29.4% 68
Total Responses: 34

18. The offered composition appears to be complex.

Summary			
Value	Count	Percent	%
Strongly disagree	9	26.47%	
Disagree	14	41.18%	
Neither agree nor disagree	4	11.76%	
Agree	5	14.71%	
Strongly agree	2	5.88%	
Statistics			
Choices Selected:	34		
Total Responses:	34		

19. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Summary			
Value	Count	Percent	%
Strongly disagree	2	5.88%	
Disagree	12	35.29%	
Neither agree nor disagree	11	32.35%	
Agree	8	23.53%	
Strongly agree	1	2.94%	
Statistics			
Choices Selected:	34		
Total Responses:	34		
Report from www.SurveyGizmo.com			

20. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Summary			
Value	Count	Percent	%
Strongly disagree	4	11.76%	
Disagree	12	35.29%	
Neither agree nor disagree	11	32.35%	

Summary		
Value	Count	Percent %
Agree	5	14.71%
Strongly agree	2	5.88%
Statistics		
Choices Selected:	34	
Total Responses:	34	

21. Assume only a subset of services can be renegotiated regarding their SLAS.
I would feel confident in selecting services for renegotiation.

Summary		
Value	Count	Percent %
Strongly disagree	3	8.82%
Disagree	12	35.29%
Neither agree nor disagree	6	17.65%
Agree	13	38.24%
Statistics		
Choices Selected:	34	
Total Responses:	34	

22. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

Summary		
Value	Count	Percent %
Less confident	14	41.18%
Equally confident	4	11.76%
More confident	16	47.06%
Statistics		
Choices Selected:	34	
Total Responses:	34	

23. Concerning Costs.
I need less time to see relations between the different services and the composition.

Summary		
Value	Count	Percent %
Disagree	1	2.94%
Neither agree nor disagree	4	11.76%
Agree	18	52.94%
Strongly agree	11	32.35%
Statistics		
Choices Selected:	34	
Total Responses:	34	

24. Concerning Response time.

I need less time to see relations between the different services and the composition.

Summary		
Value	Count	Percent %
Disagree	2	5.88%
Agree	13	38.24%
Strongly agree	19	55.88%
Statistics		
Choices Selected:	34	
Total Responses:	34	

25. Concerning Costs.

I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Disagree	2	5.88%
Neither agree nor disagree	4	11.76%
Agree	16	47.06%
Strongly agree	12	35.29%
Statistics		
Choices Selected:	34	
Total Responses:	34	

26. Concerning Response time.

I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Neither agree nor disagree	1	2.94%
Agree	15	44.12%
Strongly agree	18	52.94%
Statistics		
Choices Selected:	34	
Total Responses:	34	

27. Assume only a subset of services can be renegotiated regarding their SLAs.
I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

Summary		
Value	Count	Percent %
Disagree	2	5.88%
Neither agree nor disagree	1	2.94%
Agree	18	52.94%
Strongly agree	13	38.24%
Statistics		
Choices Selected:	34	
Total Responses:	34	

28. MoDe4SLA approach is helpful when managing this composition with regard to:

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Accuracy (accurate depicting of malfunctioning services)	2.9%1	5.9%2	14.7%5	44.1%15	32.4%11	34
Efficiency (faster depicting of these services)	2.9%1	-	5.9%2	52.9%18	38.2%13	34
Average %:	2.9%	2.9%	10.3%	48.5%	35.3%	68
Total Responses:	34					

29. The offered composition appears to be complex.

Summary			
Value	Count	Percent	%
Disagree	1	2.94%	
Agree	9	26.47%	
Strongly agree	24	70.59%	
Statistics			
Choices Selected:		34	
Total Responses:		34	

30. Concerning Costs.

I can easily determine how much impact each service has on the composition.

Summary			
Value	Count	Percent	%
Strongly disagree	11	32.35%	
Disagree	16	47.06%	
Neither agree nor disagree	4	11.76%	
Agree	3	8.82%	
Statistics			
Choices Selected:		34	
Total Responses:		34	

31. Concerning Response time.

I can easily determine how much impact each service has on the composition.

Summary			
Value	Count	Percent	%
Strongly disagree	13	38.24%	
Disagree	14	41.18%	
Neither agree nor disagree	4	11.76%	
Agree	3	8.82%	
Statistics			
Choices Selected:		34	
Total Responses:		34	

32. Assume only a subset of services can be renegotiated regarding their SLAS.

I would feel confident in selecting services for renegotiation.

Summary		
Value	Count	Percent %
Strongly disagree	13	38.24%
Disagree	12	35.29%
Neither agree nor disagree	5	14.71%
Agree	4	11.76%
Statistics		
Choices Selected:	34	
Total Responses:	34	

33. After seeing the MoDe4SLA analysis, how is your confidence about the selection of services for renegotiation you made before?

Summary		
Value	Count	Percent %
Less confident	10	32.26%
Equally confident	5	16.13%
More confident	16	51.61%
Statistics		
Choices Selected:	31	
Total Responses:	31	

34. Concerning Costs.

I need less time to see relations between the different services and the composition.

Summary		
Value	Count	Percent %
Neither agree nor disagree	1	2.94%
Agree	16	47.06%
Strongly agree	17	50.00%
Statistics		
Choices Selected:	34	
Total Responses:	34	

35. Concerning Response time.

I need less time to see relations between the different services and the composition.

Summary		
Value	Count	Percent %
Agree	11	32,35%
Strongly agree	23	67,65%

Statistics
Choices Selected: 34
Total Responses: 34

36. Concerning Costs.

I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Disagree	1	2,94%
Neither agree nor disagree	4	11,76%
Agree	15	44,12%
Strongly agree	14	41,18%

Statistics
Choices Selected: 34
Total Responses: 34

37. Concerning Response time.

I find it easier to determine the impact each service has on the composition than without the analysis.

Summary		
Value	Count	Percent %
Strongly disagree	1	2,94%
Neither agree nor disagree	1	2,94%
Agree	14	41,18%
Strongly agree	18	52,94%

Statistics
Choices Selected: 34
Total Responses: 34

38. Assume only a subset of services can be renegotiated regarding their SLAs.
I would feel more confident in selecting services for renegotiation than without MoDe4SLA.

Summary			
Value	Count	Percent	%
Disagree	1	2.94%	
Neither agree nor disagree	2	5.88%	
Agree	14	41.18%	
Strongly agree	17	50.00%	
Statistics			
Choices Selected:	34		
Total Responses:	34		

39. MoDe4SLA approach is helpful when managing this composition with regard to:

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Accuracy (accurate depicting of malfunctioning services)	2.9%1	2.9%1	8.8%3	50.0%17	35.3%12	34
Efficiency (faster depicting of these services)	-	2.9%1	5.9%2	41.2%14	50.0%17	34
Average %:	1.5%	2.9%	7.4%	45.6%	42.6%	68
Total Responses:	34					

40. I consider research concerning management of composite services as necessary.

Summary			
Value	Count	Percent	%
Neither agree nor disagree	1	2.94%	
Agree	15	44.12%	
Strongly agree	18	52.94%	
Statistics			
Choices Selected:	34		

Summary		
Value	Count	Percent %
Total Responses:	34	

41. The presentation before the evaluation was sufficient to properly understand the MoDe4SLA approach.

Summary		
Value	Count	Percent %
Disagree	2	5.88%
Neither agree nor disagree	2	5.88%
Agree	21	61.76%
Strongly agree	9	26.47%
Statistics		
Choices Selected:	34	
Total Responses:	34	

42. MoDe4SLA is helpful for managing service compositions.

Summary		
Value	Count	Percent %
Neither agree nor disagree	2	5.88%
Agree	15	44.12%
Strongly agree	17	50.00%
Statistics		
Choices Selected:	34	
Total Responses:	34	

43. Are you aware of any other approaches supporting management of service compositions? If so, what is the name of the approach?

Summary			
Value	Count	Percent %	
No	32	94.12%	
Yes	2	5.88%	
Other Values: Yes			

Value	Count
Q-WSDL, WSLA	1

work in Leyman group 1

Statistics

Choices Selected:

Total Responses: 34

44. MoDe4SLA is more helpful than the above mentioned approaches.

Summary

Value	Count	Percent %
Yes	2	6.06%
No	1	3.03%
Not applicable	30	90.91%

Statistics

Choices Selected: 33

Total Responses: 33

Appendix 1:

How do you currently manage composite services?

Data

Code	Value
------	-------

19945154 Using workflow engine.

19945429 Not

19945595 Not

23159394 Based on QoS-aspects; "management", i.e., replanning of compositions is automatically done if a SLA-deviation is detected (in fact, it's more self organization than management).

23159917 With the help of algorithms and heuristics.

23160289 In theory

23160947 Operating system does it

23162141 The thumb rule

23990507 Process engines and execution logs. For example BPEL engine

23991571 Ad hoc, look at the end result

Appendix 2:

What part of MoDe4SLA do you consider most beneficial?

APPENDIX A. EVALUATION

Code	Data Value
19945154	1. It can quickly identify and visualize the problematic parts. 2. The result can work nicely as a starting point for improving service compositions.
19945429	The impact model of response time
19945595	1. With the response time you directly see which services you do not have to look at at all. 2. Makes it much easier and faster to get the relation between the different services 3. Reduces errors from one-by-one comparison of values.
23155349	- the graph visualisation - the colors
23157347	color coding is intuitive; good services (and bad ones respectively) can be recognized very fast.
23157745	visualization
23158420	The coloring of the services is most helpful to see which services should be considered in a more thorough analysis and which can be ignored.
23158789	Coloring information Information on AND Response Time
23159075	The coloring -> you can get the information you need in very short time.
23159394	The more complex a composition gets, the more difficult is it to understand the impact of the different services; here, MoDe4SLA helps a lot
23159917	the color of the services in combination with the impact factor
23160289	- Structure visualization - coloring - impact factors
23160692	coloring
23160947	The graphical part
23161329	the diagrams are after some study easy to understand
23161861	Structured, systematic approach Graphical representation of the results
23162141	Impact Factor
23985138	Drawing the focus of the composition to the important things
23985265	The model concerning response time
23985623	I like the graphical part
23986545	The overview it gives, to see instantly where possible weak links are. Compare with gripcards for business processes of bizz design.
23987219	visualisation of real impact. Incorporates effect of composition structure and relationships.
23988240	Visualisation and "instance" approach are both usefull
23988630	Determining web services that have no impact at all.
23989164	I like the fact that the tool unveil how services interact and how compositions affect the overall efficiency.
23989619	Analysis of response time in complex service composition structured
23989786	Coloring Mentioning of IF
23990027	the time analysis
23990280	To see the impact of each service!
23990507	Intuitive visualization Trouble makers can be easily identified dependencies of the composite service on the used ones becomes more

Code	Data	Value
		obvious more efficient and accurate than manual analysis.
23990868	The approach in general. With all parts like coloured overview.	
23991193	Graphical representation instance based evaluation	
23991378	Colored representations of relations and services	
23991571	Graphical representation	

Appendix 3: What are the weak points of MoDe4SLA (possible improvements)?

Code	Data	Value
		1. legends are needed in the examples to better understand what the numbers, colours, arrows, and their combination would mean. 2. It might be helpful to visualize the service composition structure of the cases.
19945154		
19945429	Could not find any	
		1. provide a legend with explanations of colours. 2. colours might mislead you. E.g. T2 Cost has only green bubbles but yellow composition. The red arrows do not directly "spring out", compared to the bubbles.
19945595		
23155349	= colors are a little bit together (greens)	
		I needed pretty much time to understand the different values annotated in the graphs, ratio of contribution seems to be captured two times with the different perspectives. Remark at 46: 1. is process simulation of block-structured processes related to that? 2. visualisation with the use of feature trees?
23157347		
23158420	The Impacts of a loop, especially of the iteration count aren't very easy to discover. You still need to have deep look into the data sheets.	
23158789	Additional information about delta of estimated (cost/time) at runtime	
23159075	Layout of the graph in Test1 after the OR2 : The numbers of the branches are too close together	
		The way the deviations are depicted is not that intuitive; these are still 3 attributes (IF, color of the arrow, color of the bubble) and you need to understand the AND/OR/LOOP constructs.
23159394		
23159917	Interpretation of Impact Factors	
		- does it automatically arrange the bubbles? :-) - colors for arrows vs. colors for bubbles: different meaning but same colors -> maybe choose different colors.
23160289		
23160692	better explanation of the values	
		The calculations of the impact cannot be clearly seen from the graph.
23160947	Impact = prod of # invocations and agreed cost <- it will be good to include this value in the graph (value of cost)	

APPENDIX A. EVALUATION

	Data
Code	Value
23161329	I am no expert in the analysis of composite web services but I have the feeling that it would be a good t state the assumptions under which this analysis can be useful and whether they are found in practice. So these service compositions are generated ... for the validation but how do people nowadays construct these services Is it fair to assume that services are independent given network topologies congestion etc. For example: if I download from two providers A and B then the maximum speed/response time is correlated -> should network be modeled as a service?
23161861	Tree structure useful for analysis, perhaps less so for presenting results. Guidelines needed to interpret results.
23162141	There can be situations in which only arrows are the problem -> only the choice of which service to invoke is the problem. The model does not help in this case (not explicitly)
23985138	The name, Aligned to one SLA type. Adaptation probably possible, probably mixing the models (cost and response time) to give reader more information in less time, so give an IF regarding two parameters to the manager for easy analysis of renegotiating.
23985265	I found it difficult to understand what the numbers on the arrows mean, lack of connection to penalties if services violate their sla. it may even be profitable if an individual service runs bad but doesn't influence the composition outcome.
23986545	impact factor also very important but not visually easy to find high scores, more complex cost models more intuitive or clear explanation of depending relationships. Bilateral visualisation comparing to MoDe4SLA might be interesting.
23987219	the absolute numbers (runtime, cost) still need to be obtained from the text file.
23988630	You need to be an expert to use the tool but i guess usability would be hard to improve.
23989164	More clear graphical notation
23989786	Still too complex. It is not explainable in 30 minutes.
23990027	I could not understand the cost models well.
23990280	It's difficult to see the connection between the impact of a service and how good or bad the service is. And it's difficult to see the relation between costs and response time.
23990507	In a tool implementation one can add explanatory components (and also recommendations) on how to adapt a composition (e.g. which services to exchange or even more advanced "how to structurally adapt" the composition?)
23990868	Due to your decision graph-nothing. More look on the nature of the services. I know this is not your focus.
23991193	If possible: can we derive same strategies from the MoDe4SLA models, i.e. if two or more service behave bad. Can we give the user some hint which could be the best improvement?
	Data
Code	Value
23991378	Colors shall be determined based on influence overall cost and response time
23991571	colouring of branches (not intention to have same colour as bubbles) if mean something different? Why is the deviation of the branches important? Isn't only the end result for the CS important? Combination of QoS models? With different QoS parameters? Are there more QoS parameters? Get different models? Visualize the SLA (design time) on the structure.

RELATED PUBLICATIONS BY THE AUTHOR

Lianne Bodenstaff, Roel Wieringa, Andreas Wombacher, Manfred Reichert: Towards Management of Complex Service Compositions - Position Paper -; In Proceedings *IEEE International Workshop on Services Computing for B2B (SC4B2B)*, co-located with IEEE SCC, 2009

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, Michael C. Jaeger: Analyzing Impact Factors on Composite Services; In Proceedings *IEEE International Conference on Services Computing (SCC)*, 2009

Lianne Bodenstaff, Andreas Wombacher, Michael C. Jaeger, Manfred Reichert, Roel Wieringa: Monitoring Service Compositions in MoDe4SLA: Design of Validation; In Proceedings *11th International Conference on Enterprise Information Systems (ICEIS)*, 2009

Lianne Bodenstaff, Paolo Ceravolo, Ernesto Damiani, Cristiano Fugazza, Karl Reed, Andreas Wombacher: Representing and Validating Digital Business Processes; In *Advances in Web Semantics I*, LNCS 4891, Book Chapter, 2008.

Roel Wieringa, Vincent Pijpers, Lianne Bodenstaff, Jaap Gordijn: Value-Driven Coordination Process Design Using Physical Delivery Models; In Proceedings *27th International Conference on Conceptual Modeling (ER)*, 2008

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, Roel Wieringa: An Approach for Maintaining Models of an E-Commerce Collaboration; In Proceedings *IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE)*, 2008

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, Michael C. Jaeger: Monitoring Dependencies for SLAs: The MoDe4SLA Approach; In Proceedings *IEEE International Conference on Services Computing (SCC)*, 2008

APPENDIX B. RELATED PUBLICATIONS BY THE AUTHOR

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert: Formal Consistency Results between Value and Coordination Models; Technical Report, University of Twente, The Netherlands, 2007

Lianne Bodenstaff, Paolo Ceravolo, Cristiano Fugazza, Andreas Wombacher: Toward Semantic-aware Representation of Digital Business Processes; In *The Second Knowledge Management in Organization Conference (KMO)*, 2007

Lianne Bodenstaff, Manfred Reichert, Roel Wieringa: Towards the Integration of Value and Coordination Models - Position Paper -; In Proceedings *Workshops and Doctoral Consortium of the 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, 2007

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, Roel Wieringa: Monitoring Collaboration from a Value Perspective; In Proceedings *IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, 2007

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert: Dynamic Consistency between Value and Coordination Models - Research Issues; In Proceedings *International Workshop on Modeling Inter-Organizational Systems (MIOS-CIAO'06)*, 2006

Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert: Dynamic Consistency between Value and Coordination Models - Research Issues; Technical Report No. TR-CTIT-06-50, University of Twente, The Netherlands, 2006

BIBLIOGRAPHY

- [1] Workflow patterns. <http://www.workflowpatterns.com>.
- [2] W.M.P. van der Aalst. Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. *Information and Management*, 37(2):67–75, 2000.
- [3] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.
- [4] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
- [5] M. K. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem determination using dependency graphs and run-time behavior models. In *Proceedings of the International Workshop on Distributed Systems: Operations & Management (DSOM)*, pages 171–182, 2004.
- [6] B. Andersson, M. Bergholtz, A. Edirisuriya, T. Ilayperuma, and P. Johannesson. A declarative foundation of process models. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 233–247, 2005.
- [7] B. Andersson, M. Bergholtz, B. Grégoire, P. Johannesson, M. Schmitt, and J. Zdravkovic. From business to process models - a chaining methodology. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 211–218, 2006.
- [8] J.H. Andrews and Y. Zhang. General test result checking with log file analysis. *IEEE Transactions on Software Engineering*, pages 634–648, 2003.
- [9] D. Ardagna and B. Pernici. Global and local QoS guarantee in Web service selection. In *Proceedings of the International Workshop on Business Processes and Services (BPS)*. Springer, 2006.

- [10] E. Astesiano and G. Reggio. An attempt at analysing the consistency problems in the UML from a classical algebraic viewpoint. In *Proceedings of the International Workshop on Algebraic Development Techniques (WADT)*, pages 56–81. Springer, 2003.
- [11] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of Web service compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 63–71, 2006.
- [12] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *Proceedings of the International Conference on Service Oriented Computing (ICSOC)*, pages 193–202, 2004.
- [13] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *Proceedings of the International Conference on Service Oriented Computing (ICSOC)*, pages 269–282, 2005.
- [14] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Heuristics for QoS-aware Web service composition. In *Proceedings of the International Conference on Web Services (ICWS)*, pages 72–82, 2006.
- [15] Y. Bishr. Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science and Systems*, 12(4):427, 2006.
- [16] P. C. Blumenfeld, R. W. Marx, E. Soloway, and J. Krajcik. Learning with peers: From small group cooperation to collaborative communities. *Educational Researcher*, 25(8):37–40, 1996.
- [17] L. Bodenstaff, M. Reichert, and R. Wieringa. Towards the integration of value and coordination models - position paper -. In *Proceedings of the Workshop on Business Process Modeling, Development, and Support (BPMDS)*, pages 291–298, 2007.
- [18] L. Bodenstaff, R. Wieringa, A. Wombacher, and M. Reichert. Towards management of complex service compositions - position paper -. In *Proceedings of the International Workshop on Services Computing for B2B (SCB2B)*, 2009.
- [19] L. Bodenstaff, A. Wombacher, M. C. Jaeger, M. Reichert, and R. Wieringa. Monitoring service compositions in MoDe4SLA: Design of validation. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, 2009.
- [20] L. Bodenstaff, A. Wombacher, and M. Reichert. On formal consistency between value and coordination models. Technical Report TR-CTIT-07-91, University of Twente.

- [21] L. Bodenstagf, A. Wombacher, and M. Reichert. Dynamic consistency between value and coordination models - research issues. In *Proceedings of the International Workshop on Modeling Inter-Organizational Systems (MIOS-CIAO)*, 2006.
- [22] L. Bodenstagf, A. Wombacher, and M. Reichert. Dynamic consistency between value and coordination models - research issues. Technical Report 06-50, CTIT: Centre for Telematics and Information Technology, 2006.
- [23] L. Bodenstagf, A. Wombacher, M. Reichert, and M. C. Jaeger. Monitoring dependencies for SLAs: The MoDe4SLA approach. In *Proceedings of the International Conference on Services Computing (SCC)*, pages 21–29, 2008.
- [24] L. Bodenstagf, A. Wombacher, M. Reichert, and M. C. Jaeger. Analyzing impact factors on composite services. In *Proceedings of the International Conference on Services Computing (SCC)*, 2009.
- [25] L. Bodenstagf, A. Wombacher, M. Reichert, and R. Wieringa. Monitoring collaboration from a value perspective. In *Proceedings of the Conference on Digital Ecosystems (DEST)*, pages 134–140, 2007.
- [26] L. Bodenstagf, A. Wombacher, M. Reichert, and R. Wieringa. An approach for maintaining models of an e-commerce collaboration. In *Proceedings of the Conference on Enterprise Computing, E-Commerce, and E-Services (EEE)*, pages 239–246, 2008.
- [27] B. Boehm. Value-based software engineering: Reinventing. *SIGSOFT*, 28(2), 2003.
- [28] B. Boehm and K. J. Sullivan. Software economics: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 319–343. ACM, 2000.
- [29] H. Bowman, E. Boiten, J. Derrick, and M. Steen. Viewpoint consistency in ODP, a general interpretation. In *Proceedings of the International Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 189–204, 1996.
- [30] H. Bowman, M. Steen, E. Boiten, and J. Derrick. A formal framework for viewpoint consistency. *Formal Methods in System Design*, 21(2):111–166, 2002.
- [31] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert. The Virtual Resource Manager: An architecture for SLA-aware resource management. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pages 126–133, 2004.
- [32] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2005.

- [33] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and Web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, 2004.
- [34] D. Caswell and S. Ramanathan. Using service models for management of internet services. *IEEE Journal on Selected Areas in Communications*, 18(5):686–701, 2000.
- [35] K. Chang, J. Jackson, and V. Grover. E-commerce and corporate strategy: An executive perspective. *Information & Management*, 40(7):663–675, 2003.
- [36] T. K. Das and B.-S. Teng. Between trust and control: Developing confidence in partner cooperation in alliances. *The Academy of Management Review*, 23(3):491–512, 1998.
- [37] J. Derrick, E. Boiten, H. Bowman, and M. Steen. Supporting ODP – translating LOTOS to Z. In *Proceedings of the International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS)*, pages 399–406, 1996.
- [38] Z. Derzsi, J. Gordijn, K. Kok, H. Akkermans, and Y. Tan. Assessing feasibility of IT-enabled networked value constellations: A case study in the electricity sector. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 4495. Springer, 2007.
- [39] P. Dillenbourg, M. Baker, A. Blaye, and C. O’Malley. The evolution of research on collaborative learning. *Learning in Humans and Machine: Towards an interdisciplinary learning science*, pages 189–211, 1995.
- [40] S. Dustdar. Caramba-A process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases*, 15:45–66, 2004.
- [41] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 411–420, 2001.
- [42] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Co-ordinating distributed viewpoints: The anatomy of a consistency check. Technical Report 94/7, 1994.
- [43] D. Edmond and M. Papazoglou. Reflection is the essence of cooperation. In *Co-operative Information Systems: Current Trends and Directions*, pages 233–262. Academic Press, 1998.
- [44] A. Egyed and N. Medvidovic. A formal approach to heterogeneous software modeling. In *Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE)*, pages 178–192. Springer, 2000.

-
- [45] G. Engels, J. Hausmann, R. Heckel, and S. Sauer. Testing the consistency of dynamic UML diagrams. In *Proceedings of the International Conference on Integrated Design and Process Technology (IDPT)*, 2002.
 - [46] G. Engels, R. Heckel, J. M. Küster, and L. Groenewegen. Consistency-preserving model evolution through transformations. In *Proceedings of the International Conference on Model Engineering, Concepts, and Tools*, pages 212–226. Springer, 2002.
 - [47] G. Engels, J.M. Küster, R. Heckel, and L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. *SIGSOFT*, 26(5):186–195, 2001.
 - [48] B. Esfandiari and V. Tosic. Towards a Web service composition management framework. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 419–426, 2005.
 - [49] M.S. Feather. Rapid application of lightweight formal methods for consistency analyses. *IEEE Transactions on Software Engineering*, 1998.
 - [50] A. Finkelstein, D. M. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. In *Proceedings of the International Conference on European Software Engineering*, pages 84–99, 1993.
 - [51] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58, 1992.
 - [52] P. Fradet, D. Le Métayer, and M. Périn. Consistency checking for multiple view software architectures. *SIGSOFT*, 24(6):410–428, 1999.
 - [53] U. Frank. Conceptual modelling as the core of the information systems discipline - perspectives and epistemological challenges. In *Proceedings of the Americas Conference on Information Systems (AMCIS)*, pages 13–15, 1999.
 - [54] G. Giaglis, R. Paul, and G. Doukidis. Dynamic modelling to assess the business value of electronic commerce. In *Proceedings of the International Electronic Commerce Conference*, 1998.
 - [55] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: An algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75. Springer, 2004.
 - [56] J. Gordijn, H. Akkermans, and H. van Vliet. Business modelling is not process modelling. In *Proceedings of the Workshop on Conceptual Modeling for E-Business and the Web (eCOMO)*, pages 40–51. Springer, 2000.

- [57] J. Gordijn and J. M. Akkermans. Value-based requirements engineering: Exploring innovative e-commerce ideas. *Requirements Engineering*, 8(2):114–134, 2003.
- [58] R. M. Greenwood, D. Balasubramaniam, G. N. C. Kirby, K. Mayes, R. Morrison, W. Seet, B. Warboys, and E. Zirintsis. Reflection and reification in process system evolution: Experience and opportunity. In *Proceedings of the European Workshop on Software Process Technology (EWSPT)*, pages 27–38. Springer, 2001.
- [59] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan. Business Process Intelligence. *Computers in Industry*, 53(3):321–343, 2004.
- [60] J. B. Heide. Interorganizational governance in marketing channels. *The Journal of Marketing*, 58:71–85, 1994.
- [61] A. Hunter and B. Nuseibeh. Managing inconsistent specifications: Reasoning, analysis, and action. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367, 1998.
- [62] M. C. Jaeger and G. Rojec-Goldmann. SENECA - simulation of algorithms for the selection of Web services for compositions. In *Proceedings of the International Workshop on Technologies for E-Services (TES)*, pages 84–97, 2005.
- [63] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. QoS aggregation in Web service compositions. In *Proceedings of the International Conference on e-Technology, e-Commerce and e-Service (EEE)*, pages 181–185, 2005.
- [64] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Springer, 1997. Three Volumes.
- [65] R. Kazhamiakin, M. Pistore, and M. Roveri. A framework for integrating business processes and business requirements. In *Proceedings of the Enterprise Distributed Object Computing Conference (EDOC)*, pages 9–20, 2004.
- [66] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring Service Level Agreements for Web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.
- [67] S.T. King and P.M. Chen. Backtracking intrusions. *ACM Transactions on Computer Systems*, 23(1):76, 2005.
- [68] S.C. Kleene. *Mathematical Logic*. Dover Publications, 2002.
- [69] J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: A case for formal verification methods. In *Proceedings of the Conference on Enterprise Distributed Object Computing (EDOC)*, 2002.
- [70] T. B. Lawrence, C. Hardy, and N. Phillips. Institutional effects of interorganizational collaboration: The emergence of proto-institutions. *The Academy of Management Journal*, 45(1):281–290, 2002.

- [71] P. Loucopoulos and R. Zicari. *Conceptual modeling, Databases, and CASE: An Integrated View of Information Systems Development*. John Wiley & Sons, 1992.
- [72] A. Ludwig and B. Franczyk. Managing dynamics of composite Service Level Agreements with COSMA. In *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery*, volume 4, 2008.
- [73] I.R. Macneil. *The New Social Contract: An Inquiry into Modern Contractual Relations*. Yale University Press, New Haven, 1980.
- [74] K. Mahbub and G. Spanoudakis. Run-time monitoring of requirements for systems composed of Web-services: Initial implementation and evaluation experience. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 257–265, 2005.
- [75] R.S. Matthews, J.L. Cooper, N. Davidson, and P. Hawkes. Building bridges between cooperative and collaborative learning. *Change*, pages 34–40, 1995.
- [76] W.E. McCarthy. The REA accounting model: a generalized framework for accounting systems in a shared data environment. In *Accounting Review*, volume 57, pages 554–578, 1982.
- [77] D. A. Menascé. Response-time analysis of composite Web services. *IEEE Internet Computing*, 8(1):90–92, 2004.
- [78] T. Mens, R. Van Der Straeten, and J. Simmonds. Maintaining consistency between UML models with description logic tools. In *Proceedings of the Workshop on Consistency Problems in UML-based Software Development*, 2003.
- [79] T. Mens, R. Van Der Straeten, and J. Simmonds. A framework for managing consistency of evolving UML models. In *Software Evolution with UML and XML*, pages 1–31, 2005.
- [80] T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.
- [81] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *Proceedings of the International Conference on World Wide Web (WWW)*, 2008.
- [82] D. Müller, M. Reichert, and J. Herbst. Data-driven modeling and coordination of large process structures. In *Proceedings of the IFCIS Conference on Cooperative Information Systems (CoopIS)*, pages 131–149. Springer, 2007.
- [83] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 48–63. Springer, 2008.

- [84] B. Mutschler, M. Reichert, and J. Bumiller. An approach to quantify the costs of Business Process Intelligence. In *Proceedings of the International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA)*, pages 152–165, 2005.
- [85] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: A consistency checking and smart link generation service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.
- [86] C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency management with repair actions. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 455–464, 2003.
- [87] N.F. Noy and M.A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing*, pages 63–70, 2001.
- [88] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. *Computer*, 33(4):24–29, 2000.
- [89] Association of German Automobile Manufacturers (VDA). VDA Recommendation 4965 T1: Engineering Change Management (ECM) - Part 1: Engineering Change Request (ECR) Version 1.1, 2005.
- [90] OMG. OMG UML specification, 2009. <http://www.omg.org/technology/documents/formal/uml.htm>.
- [91] M. Oriol, J. Marco, X. Franch, and D. Ameller. Monitoring adaptable SOA-systems using SALMon. In *Proceedings of the Workshop on Service Monitoring, Adaptation and Beyond*, 2008.
- [92] A. Osterwalder and Y. Pigneur. An e-business model ontology for modeling e-business. In *Proceedings of the Bled Electronic Commerce Conference*, pages 17–19, 2002.
- [93] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic matching of Web services capabilities. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 333–347. Springer, 2002.
- [94] N. Phillips, T. B. Lawrence, and C. Hardy. Inter-organizational collaboration and the dynamics of institutional fields. *Journal of Management Studies*, 37(1), 2000.
- [95] S. Pokraev. *Model-Driven Semantic Integration of Service-Oriented Applications*. PhD thesis, University of Twente, 2009.
- [96] P.A. Porras and P.G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the National Information Systems Security Conference*, pages 353–365, 1997.

- [97] W. W. Powell. Neither market nor hierarchy: Network forms of organization. *Research in Organizational Behavior*, 12:295–336, 1990.
- [98] S.A. Rosenfeld. Does cooperation enhance competitiveness? Assessing the impacts of inter-firm collaboration. *Research Policy*, 25(2):247 – 263, 1996.
- [99] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
- [100] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati. Automated SLA monitoring for Web services. In *Proceedings of the International Workshop on Distributed Systems: Operations & Management (DSOM)*, pages 28–41, 2002.
- [101] M. Sefika, A. Sane, and R.H. Campbell. Monitoring compliance of a software system with its high-level design models. In *International Conference on Software Engineering (ICSE)*, volume 18, pages 387–396, 1996.
- [102] R. Smith. Aristotle’s logic. *Stanford Encyclopedia of Philosophy*, 2007.
- [103] G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. In *Handbook of Software Engineering and Knowledge Engineering*, volume 1, pages 24–29, 2001.
- [104] J.D. Sterman. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. McGraw-Hill, 2000.
- [105] L. Stoimenov and S. Djordjević-Kajan. Realization of GIS semantic interoperability in local community environment. *Proceedings of the AGILE Conference on Geographic Information Science*, pages 73–80, 2003.
- [106] K.J. Sullivan, P. Chalasani, S. Jha, and V. Sazawal. Software design as an investment activity: A real options perspective. *Real Options and Business Strategy: Applications to Decision Making*, pages 215–261, 1999.
- [107] T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended Entity-Relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.
- [108] P. Timmers. Business models for electronic markets. *Electronic Markets*, 8:3–8, 1998.
- [109] V. Tasic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management applications of the Web Service Offerings Language (WSOL). *Information Systems*, 30(7):564–586, 2005.
- [110] S. Uchitel and M. Checkik. Merging partial behavioural models. *SIGSOFT*, 29(6):43–52, 2004.

- [111] D. Varró and A. Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. *Journal of Software and Systems Modelling*, 2(3):187–210, 2003.
- [112] Y. Wand and R. Weber. Research commentary: Information systems and conceptual modeling - a research agenda. *Information Systems Research*, 13(4):363–376, 2002.
- [113] S.A. White. Business Process Modelling Notation (BPMN). <http://www.bpmn.org/Documents/BPMN> Visited: November 2, 2009.
- [114] R. Wieringa, V. Pijpers, L. Bodestaff, and J. Gordijn. Value-driven coordination process design using physical delivery models. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 216–231, 2008.
- [115] R.J. Willis. Critical path-analysis and resource constrained project scheduling - theory and practice. *European Journal of Operational Research*, 21(2):149–155, 1985.
- [116] C. Wymbs. How e-commerce is transforming and internationalizing service industries. *Journal of Services Marketing*, 14(6):463–477, 2000.
- [117] Y.Y. Yao. Granular computing. *Computer Science*, 31:1–5, 2004.
- [118] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality driven Web services composition. In *Proceedings of the International Conference on World Wide Web*, pages 411–421, 2003.
- [119] X. Zhao, J. Xie, and J. Leung. The impact of forecasting model selection on the value of information sharing in a supply chain. *European Journal of Operational Research*, 142:321–344, 2002.
- [120] Z. Zlatev and A. Wombacher. Consistency between e³-value models and activity diagrams in a multi-perspective development method. In *Proceedings of IFCIS Conference on Cooperative Information Systems (CoopIS)*, pages 520–538, 2005.

SAMENVATTING

Het gebruik van meerdere modellen waarbij ieder model een specifiek aspect of deel van het software systeem beschrijft, komt veelvuldig voor in onderzoeksgebieden zoals software development, information systems development en e-business development. In dit proefschrift richten wij ons op modelgebaseerde methoden die het beschrijven van de samenwerking tussen verschillende bedrijven ondersteunen. Deze samenwerkingsverbanden zijn gewoonlijk complex wat betreft coördinatie, overeenkomsten en waardecreatie voor de betrokken partijen. In de ontwerpfase moet we ervoor zorgen dat de verschillende modellen consistent zijn met elkaar, dat wil zeggen dat ze hetzelfde systeem beschrijven. In de uitvoeringsfase moeten we er daarnaast rekening mee houden dat het software systeem zich anders kan gedragen dan oorspronkelijk overeengekomen is. Dit afwijkende gedrag kan bijvoorbeeld veroorzaakt worden doordat partijen zich niet aan de overeenkomst houden. Daarom behoren het verzekeren van consistentie in de ontwerpfase en het monitoren van het systeem in de uitvoeringsfase zodat inconsistenties met de modellen gedetecteerd worden, tot de grootste uitdagingen van dit onderzoek.

Tijdens het managen van complexe samenwerkingsverbanden is het daarnaast ook van belang om de modellen die de samenwerking beschrijven te onderhouden zodat het succes van de samenwerking in de gaten gehouden kan worden. Het aanpassen van één model om de consistentie met het lopende systeem te behouden, kan leiden tot nieuwe inconsistenties tussen de verschillende modellen. De consequentie hiervan is dat de onderhoudsfase van de modellen tijdrovend is en groeit in complexiteit wanneer het aantal modellen toeneemt.

In dit proefschrift wordt een methode beschreven die het verzekeren en onderhouden van consistentie tussen het lopende systeem en onderliggende modellen voor samenwerkingsverbanden ondersteunt. We presenteren een gestructureerde en modelonafhankelijke methode voor het checken en onderhouden van consistentie. Daarbij ligt de focus op het identificeren en onderhouden van relaties tussen de verschillende modellen.

We valideren onze methode door middel van twee case studies in twee verschillende onderzoeksgebieden. Het eerste scenario gebruikt business- en coördinatiemodellen, terwijl het tweede scenario over Web service composities gaat. Verder worden beide scenario's geïmplementeerd als proof-of-concept evaluatie. We sluiten af met een empirische

BIBLIOGRAPHY

validatie van het Web service compositie scenario door middel van een uitgebreid en interactief onderzoek onder 34 deelnemers. Dit onderzoek bevestigt de geschiktheid van onze managementoplossing voor praktisch gebruik.

SIKS Dissertatiereeks

====
1998
====

- 1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
- 1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
- 1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations
within the Language/Action Perspective
- 1998-4 Dennis Breuker (UM)
Memory versus Search in Games
- 1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

- 1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products
- 1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
- 1999-3 Don Beal (UM)
The Nature of Minimax Search
- 1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects
- 1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven
Specification of Network Information Systems
- 1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
- 1999-7 David Spelt (UT)
Verification support for object database design
- 1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent
Mechanism for Discrete Reallocation.

====
2000
====

- 2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

- 2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
- 2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.
- 2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
- 2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.
- 2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
- 2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
- 2000-8 Veerle Coup (EUR)
Sensitivity Analysis of Decision-Theoretic Networks
- 2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
- 2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture
- 2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

- 2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
- 2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
- 2001-3 Maarten van Someren (UvA)
Learning as problem solving
- 2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets
- 2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
- 2001-6 Martijn van Welie (VU)
Task-based User Interface Design
- 2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
- 2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology:
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction
between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerd (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to

Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====

2004

====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar
abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale
politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

- 2004-17 Mark Winands (UM)
Informed Search in Complex Games
- 2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
- 2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
- 2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005
====

- 2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
- 2005-02 Erik van der Werf (UM))
AI techniques for the game of Go
- 2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
- 2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
- 2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing
- 2005-06 Pieter Spronck (UM)
Adaptive Game AI
- 2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
- 2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
- 2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
- 2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
- 2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
- 2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
- 2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
- 2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
- 2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)

Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

- 2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
- 2007-07 Natasa Jovanovic* (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
- 2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
- 2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
- 2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
- 2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
- 2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
- 2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
- 2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
- 2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
- 2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
- 2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
- 2007-18 Bart Orriens (UvT)
On the development and management of adaptive business collaborations
- 2007-19 David Levy (UM)
Intimate relationships with artificial partners
- 2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
- 2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
- 2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns
- 2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems
- 2007-24 Georgina Ramirez Camps (CWI)
Structural Features in XML Retrieval
- 2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====

2008

=====

- 2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach
- 2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations
- 2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach
- 2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration
- 2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
- 2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
- 2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning
- 2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
- 2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
- 2008-10 Wauter Bosma (UT)
Discourse oriented summarization
- 2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
- 2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
- 2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
- 2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
- 2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.
- 2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective
- 2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
- 2008-18 Guido de Croon (UM)
Adaptive Active Vision
- 2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
- 2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

- 2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
- 2008-22 Henk Koning (UU)
Communication of IT-Architecture
- 2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
- 2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
- 2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
- 2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
- 2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
- 2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
- 2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
- 2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
- 2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval
- 2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
- 2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues
- 2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining
- 2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

- 2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
- 2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
- 2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
- 2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controlling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

- 2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
- 2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
- 2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
- 2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
- 2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
- 2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
- 2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
- 2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
- 2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
- 2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
- 2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
- 2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
- 2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
- 2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets
- 2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
- 2009-41 Igor Bereznyy (UvT)
Digital Analysis of Paintings
- 2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
- 2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
- 2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
- 2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
- 2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
A Gesture interaction at a Distance