

Hadil Abukwaik

## **Proactive Support for Conceptual Interoperability Analysis of Software Units**



Editor-in-Chief: Prof. Dr. Dieter Rombach  
Editorial Board: Prof. Dr. Frank Bomarius  
Prof. Dr. Peter Liggesmeyer  
Prof. Dr. Dieter Rombach

**FRAUNHOFER VERLAG**

# **PhD Theses in Experimental Software Engineering**

Volume 60

Editor-in-Chief: Prof. Dr. Dieter Rombach

Editorial Board: Prof. Dr. Frank Bomarius  
Prof. Dr. Peter Liggesmeyer  
Prof. Dr. Dieter Rombach

Hadil Abukwaik

## **Proactive Support for Conceptual Interoperability Analysis of Software Units**

Fraunhofer Verlag

Zugl.: Kaiserslautern, TU, Diss., 2017

Printing:  
Mediendienstleistungen des  
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Printed on acid-free and chlorine-free bleached paper.

All rights reserved; no part of this publication may be translated, reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. The quotation of those designations in whatever way does not imply the conclusion that the use of those designations is legal without the consent of the owner of the trademark.

© by **Fraunhofer Verlag**, 2018  
ISBN (Print): 978-3-8396-1338-2  
Fraunhofer-Informationszentrum Raum und Bau IRB  
Postfach 800469, 70504 Stuttgart  
Nobelstraße 12, 70569 Stuttgart  
Telefon +49 711 970-2500  
Telefax +49 711 970-2508  
E-Mail [verlag@fraunhofer.de](mailto:verlag@fraunhofer.de)  
URL <http://verlag.fraunhofer.de>

# Proactive Support for Conceptual Interoperability Analysis of Software Units

Beim Fachbereich Informatik  
der Technischen Universität Kaiserslautern  
zur Verleihung des akademischen Grades

**Doktor der Ingenieurwissenschaften (Dr.-Ing.)**

genehmigte Dissertation  
von

**Hadil Abukwaik, M. Sc.**

Technische Universität Kaiserslautern  
Kaiserslautern

Berichterstatter:	Prof. Dr. Dr. h.c. Dieter Rombach Prof. Dr. Ralf Reussner
-------------------	--------------------------------------------------------------

Dekan:	Prof. Dr. Stefan Deßloch
--------	--------------------------

Tag der Wissenschaftlichen Aussprache:	12.07.2017
----------------------------------------	------------



To my beloved husband Mohammed and children Adam and Hannah



# Acknowledgments

I would like to express my sincere gratitude and thanks to my advisor Professor Dr. Dr. h. c. Dieter Rombach. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your great advice on research and career development and your support for me during my new motherhood responsibilities have been priceless. I would also like to thank my second supervisor Professor Dr. Ralf Reussner for his valuable feedback and insightful discussion. In addition, I want to express my special appreciation to Dr. Jörg Dörr, Dr. Matthias Naab, and Dr. Liliana Guzmán from Fraunhofer IESE for their continuous guidance, stimulating discussions, and encouragement throughout all the phases of my Ph.D. thesis research and writing.

I acknowledge my research fellows at AGSE and Fraunhofer IESE who provided me with feedback during my dissertation work including Dr. Jens Knodel, Dr. Martin Becker, and Alexander Klaus.

I thank the AGSE Group, the Nachwuchsring Program, and the PhD Program of the Computer Science Department at the University of Kaiserslautern for funding my Ph.D. studies and my trips to scientific conferences and events.

I also thank my friends Walaa and Tina for their motivation and kind support to overcome all challenges. I am also very thankful for Prof. Dr. Katharina Zweig for the wonderful example she has provided for me as a successful female scientist, professor, and mother. Your positive energy is contagious!

A very special thanks go to my parents for supporting me spiritually throughout my thesis work. Your prayers for me were what sustained me thus far and you motivated me to always strive towards my goal.

Last but not the least, thank you my little son Adam for being patient when Mama could not play and for cheering her up with your beautiful laugh! Thank you my baby girl, Hannah, for the beautiful feeling that your kicks gave me while writing and defending my thesis. And words can never express how grateful I am to my beloved husband Mohammed for all of the sacrifices that you have made on my behalf. Your unconditional support, patience, and care for me and for our son were my relief in my sleepless nights and working weekends and holidays. Thank you for your comforting smile throughout the ups and downs of this journey!





# Abstract

Interoperability is the ability of separately developed software units to communicate and exchange data or services meaningfully. This property plays a vital role in enabling interoperation in today's systems-of-systems, cyber-physical systems, ecosystems, etc. Achieving meaningful interoperation with a software unit requires identifying its conceptual constraints (e.g., usage context, design decision, quality, etc.) to understand their associated impact and required resolution. Missing such constraints causes unexpected conceptual mismatches, leading to projects running late with costly rework. However, for black-box software providers, it is a tedious, unguided, and time-consuming task to share the conceptual constraints explicitly and comprehensively. Also, despite reuse analysis and mismatch detection approaches, third-party clients lack guidance on how to detect conceptual mismatches.

To cope with these challenges, we built a model for **Conceptual Interoperability Constraints (COINs)**, extending and enhancing existing models of interoperability and reuse. This model is the foundation for the methodical contribution of this thesis: a framework for **Conceptual Interoperability Analysis (COINA)** that supports software architects and analysts in identifying the conceptual interoperability constraints and mismatches of software units more effectively and efficiently. COINA comprises: (1) Proactive, semi-automatic, in-house preparation for interoperable software units, which helps software providers to explicitly share conceptual constraints with interested clients with the least effort. This is facilitated by our tool-supported extraction of COINs from internal architectural documents (i.e., UML diagrams) and public API documents. The output is a standard, ready-to-share COIN document for clients. We also provide guidelines for improving the conceptual content of API documents. In the long term, this proactive preparation will help software providers achieve higher business impact and better competitiveness by increasing the success rate of interoperation. (2) A systematic, algorithm-based method for mapping the conceptual constraints of systems to detect their mismatches. Our guided method directs third-party clients in their conceptual analysis of external units.

We demonstrate the feasibility of COINA preparation by implementing the COIN extraction from UML diagrams as an add-in for Enterprise Architect and from API documents as a machine learning classifier. A multiple-run controlled experiment confirmed our hypotheses that our systematic analysis method significantly increases the architects' effectiveness and efficiency in detecting conceptual mismatches. A survey of practitioners and an initial experiment confirmed that our guidelines improve the usefulness and ease-of-use of API documents.



---

# Table of Contents

Acknowledgments .....	v
Abstract .....	vii
Table of Contents .....	ix
List of Figures .....	xii
List of Tables .....	xiv
<b>1 Introduction .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Problem Statement and Goals .....	5
1.3 Solution Ideas and Hypotheses .....	13
1.4 Research Context and Assumptions .....	16
1.5 Overview of Contributions .....	18
1.6 Research Method .....	22
1.7 Thesis Outline .....	23
<b>2 Background .....</b>	<b>27</b>
2.1 Introduction .....	27
2.2 Sources of Conceptual Interoperability Information .....	27
2.3 Interoperability Mismatches of Software Units .....	30
2.4 Natural Language Processing and Machine Learning Techniques .....	31
2.5 Summary .....	33
<b>3 State of the Practice .....</b>	<b>35</b>
3.1 Introduction .....	35
3.2 Survey on the State of the Practice .....	35
3.2.1 Research Methodology .....	35
3.2.2 Results and Discussion .....	38
3.2.3 Threats to Validity .....	51
3.3 Summary and Conclusion .....	52
<b>4 State of the Art .....</b>	<b>53</b>
4.1 Introduction .....	53
4.2 Scoping Study on the State of the Art .....	53
4.2.1 Research Methodology .....	53
4.2.2 Results and Discussion .....	56
4.2.3 Threats to Validity .....	64
4.3 Conceptual Interoperability Foundations .....	65
4.4 Conceptual Interoperability Analysis .....	67
4.4.1 Identification of Conceptual Interoperability Constraints .....	67

4.4.2	Identification of Software Mismatches in Black-Box Contexts .....	68
4.5	Summary and Conclusion .....	69
<b>5</b>	<b>The Conceptual Interoperability Constraint (COIN) Model .....</b>	<b>73</b>
5.1	Introduction.....	73
5.2	Model Overview.....	74
5.3	Model in Detail.....	75
5.3.1	Principal Dimensions .....	75
5.3.2	Detailed COIN Attributes .....	79
5.3.3	Conceptual Interoperability Mismatches and Their Types.....	87
5.4	Standard Documentation Templates .....	89
5.4.1	Conceptual Interoperability Constraints Template (COIN Portfolio) .....	89
5.4.2	Mismatches List Template.....	93
5.5	Summary .....	96
<b>6</b>	<b>The Conceptual Interoperability Analysis (COINA) Framework.....</b>	<b>97</b>
6.1	Introduction.....	97
6.2	Framework Overview.....	98
6.2.1	Methodical Overview: Input, Output, Activities.....	98
6.2.2	Contextual Scenario: Who, When, How .....	100
6.3	Proactive, In-House Preparation for Interoperable Software Units.....	101
6.3.1	Extracting COINs from UML Diagrams .....	102
6.3.2	Extracting COINs from the Text of API Documents .	113
6.3.3	Guidelines for Improving API Documentations .....	130
6.4	Approach for the Systematic Detection of Conceptual Mismatches .....	135
6.4.1	Perspective-based Extraction of COINs .....	136
6.4.2	Checklist-based, Algorithmic Mapping of Portfolios.	139
6.5	Summary .....	144
<b>7</b>	<b>Evaluation .....</b>	<b>145</b>
7.1	Introduction.....	145
7.2	Objectives and Hypotheses.....	145
7.3	Multi-Run Controlled Experiment.....	148
7.3.1	Objectives and Research Questions .....	148
7.3.2	Experimental Context .....	149
7.3.3	Experimental Setup .....	149
7.3.4	Analysis Results .....	156
7.3.5	Discussion .....	162
7.3.6	Threats to Validity.....	163
7.4	Survey and Initial Controlled Experiment.....	165
7.4.1	Objective and Research Questions .....	165
7.4.2	Survey Study .....	166
7.4.3	Initial Controlled Experiment.....	169

---

7.4.4	Summary .....	174
7.5	Multiple-Case Study .....	174
7.5.1	Objectives and Research Questions .....	175
7.5.2	Data Analysis and Discussion .....	175
7.6	Summary .....	176
<b>8</b>	<b>Summary .....</b>	<b>179</b>
8.1	Contributions and Results .....	179
8.2	Benefits and Limitations .....	181
8.3	Future Work.....	184
	<b>References.....</b>	<b>187</b>
	<b>Appendix.....</b>	<b>201</b>
	<b>Lebenslauf (CV).....</b>	<b>261</b>

---

## List of Figures

Figure 1	Basis for relationship between interoperability, integration, and interoperation .....	3
Figure 2	Reuse-investment relation [BB91].....	6
Figure 3	Example of detected and missed mismatches between S1 and S2.....	10
Figure 4	Overall thesis goal: improving the detection of conceptual interoperability mismatches.....	12
Figure 5	Mapping overview of problems, goals, and solution ideas .	14
Figure 6	Relations among foundation/methodical contributions and evaluation contributions .....	22
Figure 7	Summary of Research Method and Contributions .....	25
Figure 8	Questionnaire Design .....	37
Figure 9	Respondents' experience in software integration.....	39
Figure 10	The roles responsible for performing interoperability analysis.....	41
Figure 11	Cost for interoperability analysis reported by practitioners with knowledge .....	44
Figure 12	Additional project cost for resolving unexpected conceptual mismatches.....	45
Figure 13	Distribution agreement on the difficulties of interoperability analysis practices for each group of respondents .....	47
Figure 14	Distribution of agreement on perceived content problems of input artifacts for interoperability analysis per respondent group.....	49
Figure 15	Distribution of agreement on perceived presentation problems of input artifacts for interoperability analysis per respondent group.....	51
Figure 16	Year-wise distribution of selected studies .....	56
Figure 17	Interoperability-level distribution over selected studies .....	57
Figure 18	Distribution of evaluation method over selected studies ...	60
Figure 19	The main aspects of the COIN Model .....	75
Figure 20	The three dimensions of our COIN Model.....	76
Figure 21	Structural differences of data between two systems .....	82
Figure 22	An example of a COIN Portfolio (left) and one of its sheets (right) .....	91
Figure 23	Meta-model for the COIN Portfolio of Interoperable Software System.....	91
Figure 24	An example of a Mismatches List (left) and one detailed mismatch description (right) .....	94

---

Figure 25 Overview of the COINA Framework .....	101
Figure 26 COINs extraction process from UML diagrams .....	105
Figure 27 CoinsExtractor example of a list of interoperable elements .....	108
Figure 28 CoinsExtractor example of an extracted Context COIN from a use case diagram.....	109
Figure 29 CoinsExtractor example of a system's COIN Portfolio ....	110
Figure 30 CoinsExtractor example of generated COIN Portfolio .....	110
Figure 31 CoinsExtractor architectural overview .....	111
Figure 32 Multiple-case study process.....	116
Figure 33 COIN share in the Ground Truth Dataset.....	118
Figure 34 Process of the ML experiments.....	121
Figure 35 The area under FPR-TPR curve for classification algorithms after parameter tuning.....	124
Figure 36 COIN extraction process from an NL sentence of an API document .....	125
Figure 37 The selection options of the COINer tool for the COIN highlighting feature.....	127
Figure 38 A COINer tool example for an automatically highlighted Dynamic COIN .....	128
Figure 39 A COINer tool example for a COIN report generated for Structure COINs.....	128
Figure 40 Process overview of the systematic approach for detecting conceptual mismatches.....	136
Figure 41 Example of a formal modeling language and two COIN instances.....	141
Figure 42 Hypotheses for the contributions.....	148
Figure 43 Acceptance evaluation metrics .....	151
Figure 44 Experimental Design.....	152
Figure 45 Design of Experimental Sessions .....	153
Figure 46 Correlation results for the demographic characteristics of the participants.....	157



---

## List of Tables

Table 1	Frequency of problems related to undetected conceptual mismatches .....	44
Table 2	Perceived difficulties of interoperability analysis practices ..	46
Table 3	Perceived sufficiency of the input artifacts of the interoperability analysis task .....	47
Table 4	Perceived need for enhancing the content of input artifacts for interoperability analysis.....	49
Table 5	Perceived need for enhancing the presentation of input artifacts for interoperability analysis .....	50
Table 6	Overview of LCIM levels with the problems and solutions identified in the studies .....	57
Table 7	Detailed attributes of COINs .....	80
Table 8	Predefined COIN Extraction Templates .....	103
Table 9	Case share of sentences and execution time .....	117
Table 10	Example of content in the Seven-COIN and Two-COIN Corpus .....	118
Table 11	COINs identification results using different ML algorithms	123
Table 12	The results of classification after parameter tuning.....	123
Table 13	Example of a record in the COIN Cheat Sheet for UML diagrams .....	137
Table 14	Example of a snippet of a COIN Portfolio for the Smart Farm System.....	138
Table 15	Example of a snippet of a COIN Portfolio for the Smart Tractor System.....	139
Table 16	Example of a record in the Mismatches Cheat Sheet .....	142
Table 17	Example of a snippet of the Mismatches List for the Smart Farm and the Smart Tractor.....	143
Table 18	Hypotheses of the Multi-run Controlled Experiment.....	150
Table 19	Analysis results regarding Effectiveness ( $H_9$ ) and Efficiency ( $H_{10}$ ) for Run I and Run II .....	160
Table 20	Results of Acceptance ( $H_{11}$ ) for Run I and Run II.....	161
Table 21	Results of the survey on the guidelines for improving API documents .....	168
Table 22	Acceptance results from the debriefing questionnaire .....	173
Table 23	COIN Model comprehensiveness data analysis.....	176

---

# 1 Introduction

This chapter motivates the topic and the problems addressed in this thesis, provides an overview of the solution ideas, and summarizes the contributions. The chapter also presents the research scope, context, assumptions, and methodology. Finally, an outline is provided for the next chapters of the thesis.

## 1.1 Motivation

Software-intensive business

We are living in a software-intensive era in which software systems are used for various purposes in various ways. Software systems in their different forms (e.g., Information Systems (ISs), Embedded Systems (ESs), Mobile Systems (MSs), etc.) provide support and solutions in different domains including manufacturing, agriculture, banking, health, education, and military, to name but a few. Thus, these systems have become an integral part of business and influenced the way we perform processes, manage and learn from collected data, automate complex and tedious calculations, control machines and vehicles, connect markets and services, and more.

Business requires software integration

In the past, a software system used to be developed by a single organization to provide tightly focused support for certain tasks and specific purposes. However, today's software providers are urged to adopt integration solutions for independent software systems built by different organizations [BR91, BA99]. This is due to imperative needs, fast growth, and high competitiveness of modern business. Successful software integration offers several advantages such as business revenue growth through improved productivity and time-to-market for quality software, expanded sales channels, and decreased costs for software development maintenance, and operation [GC92]. Furthermore, it empowers usage innovation by allowing users to perform different activities and processes smoothly through software interoperation among integrated software units. Software interoperation is the cooperative exchange of data or services among integrated software units, which is enabled by software integration.

In this context, software integration takes place after the end of the development lifecycle for each of the integrated software units. Software integration is defined as follows:

**Definition 1 – Software Integration**

The process of bringing together separately developed software units in different contexts into one software system to enable the desired interoperation and achieve its goal.

Types of  
integrated  
software  
units

Integrated software units can not only be developed by different organizations but can also have different sizes. For example, an integrated unit can be a single software component like web service API, platform API, commercial-of-the-shelf (COTS) software, Open Source Software (OSS), etc. On a larger scale, it can be a complete software system that is intended to be a part of the recent class of large and complex software systems aimed at obtaining sophisticated capabilities with higher performance [Jam09]. Examples of such large-scale systems are: (1) systems-of-systems, which include a number of ISs (e.g., a health software system integrating a doctor application with a patient application, a pharmacy application, etc.); (2) cyber-physical systems, which include a number of ESs (e.g., a traffic control system integrating a traffic lights app, vehicle app, driver app, etc.); or (3) Ecosystems, which include a mix of ISs, ESs, and MSs all sharing the same market and the same overall system goals (e.g., a farm ecosystem integrating farm management system, smart tractors, a mobile app for farmers, etc.).

Interoper-  
ability enables  
integration

Software interoperability is a key property of software units to cope with the current business demands. This property is defined by IEEE [GKM+91] as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”. Beyond the exchange of information, we emphasize the importance of having the integrated software units aligned on the conceptual and the architectural levels. Hence, our definition of software interoperability, adapted from the aforementioned IEEE definition, has a stronger focus on the conceptual perspective as follows:

**Definition 2 – Software Interoperability**

The ability of two or more separately developed software units to communicate and exchange data or services seamlessly and in a conceptually meaningful way.

In practice, interoperability is considered as the enabler for software integration [OWR+11]. It plays a vital role in determining the readiness of a software unit for integration with other software units. Figure 1 summarizes the relationships among the terms interoperability, integration, and interoperation [OWR+11].

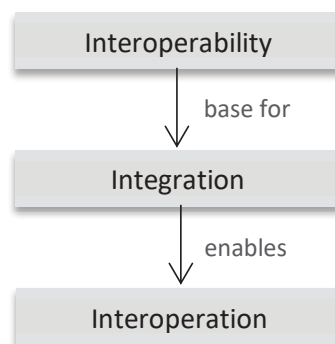


Figure 1 Basis for relationship between interoperability, integration, and interoperation

Example:  
Smart  
Farming

An example industry that we can use to demonstrate the aforementioned aspects is the farming industry, where the key business service is to produce food for people by growing crops and livestock. Delivering such a service requires a large number of business processes related to field work (e.g., fertilizing the soil, planting seeds, spraying water, harvesting crops, etc.) and management activities (e.g., tracking soil humidity, weather status, progress of on-field tasks, etc.). While in the past, most of these processes have been performed manually, today's smart farms enjoy the luxury of automation provided by different software-intensive systems. In a visit to a smart farm, one would typically find: ISs as desktop applications supporting managers in handling farm records and collected data, ESs in smart sprinklers or in tractors as screens directing the drivers on the field, or MSs as apps for farmers on their smartphones or smartwatches for tracking job assignments and reports. Without a doubt, neither have all of these various software systems been developed by the same organization, nor do they necessarily share the same business model. For example, a smart tractor ( $S_1$ ) with high-quality sensors could collect and report different types of data of high resolution and frequency, while a basic low-priced farm management system ( $S_2$ ) has limited data storage and analytic capabilities. With the growing food-production demands, it would be of great value to a smart farm ( $SF$ ) to collect and analyze data for prediction and informed decision purposes. Hence, if producers of farming-related software systems (like  $S_1$  and  $S_2$ ) or initiators of smart farming ecosystems were to aim at developing interoperable software products, they would save interested clients (like the owners of the  $SF$ ) a lot of integration costs. This would consequently be reflected in the increased competitiveness and higher long-term revenues of such software-producing and ecosystem-initiating organizations.

Why  
interloper-  
ability  
analysis?

Having said this, building successful integration and consequently achieving meaningful interoperation starts with early assessment and reasoning about the properties of the two software units that are to interoperate. This assessment takes place before adapting,

configuring, or glue-coding the units [BPY+03] and is performed by software architects and analysts. We call this “interoperability analysis” and define it as follows:

### **Definition 3 – Interoperability Analysis**

The process of checking the constraints and assumptions of two software units in order to find if they have any mismatches that impede their desired communication or meaningful interoperation between them.

This analysis is a cornerstone for understanding the impact of the units’ constraints, identifying their mismatches, planning the required resolution work (including design and implementation), estimating the associated cost (in terms of time and money), and consequently determining the feasibility of integration early in the project. However, performing this important interoperability analysis effectively is not a trivial task. Not only does it require identifying the units’ technical mismatches that impede the actual exchange of data and services (e.g., different network protocols, programming languages, data types, etc.); it also requires identifying their conceptual mismatches that lead to meaningless or improper interoperation results (e.g., different usage contexts, architectural constraints, semantics, qualities, etc.).

Why conceptual/architectural interoperability analysis?

According to Krueger [Kru92], organizations adopting integration report limited success due to non-technical issues. Missing conceptual mismatches and facing them unexpectedly at a later point in time leads to worthless technical integration, expensive rework, and obviously to projects running late. For example, COTS-based integration projects often suffer from significant overruns in terms of budget and schedule due to unexpected interoperability mismatches [Bhu07].

Therefore, analyzing software interoperability on the conceptual level should be performed early with high priority in order to guide decision-makers in selecting the most appropriate software units for their systems to interoperate with. The results of conceptual interoperability analysis offer a basis for deciding whether it is worth investing further effort into investigating the systems’ interoperability on the organizational level (e.g., privacy and intellectual property concerns), the managerial level (e.g., budget and time restrictions), and the technical level (e.g., network and communication protocols).

Software documentation for interoperability analysis

The documentation of a software unit is used to retain and communicate information about the various aspects of the unit to its audience [For02]. In that sense, it is essential input for the interoperability analysis, especially in the black-box integration (e.g., web service API), where no source code is available. Thus, a proper software documentation is considered a necessity for enabling

software integration as it helps to assess the software unit with reasonable effort [Sam97].

Architectural document preserves conceptual constraints

Accordingly, software architects and analysts require a comprehensive documentation that explicitly states the conceptual and architectural constraints for a software unit, in order to perform effective and efficient conceptual interoperability analysis. Such information is usually preserved in the local software architecture documentation, which maintains the abstract conceptual overview of the software structure comprising its elements, properties, and relationships [BCK03] [CGB+02]. Hence, information contained in architecture documents, among the other software documents, is of high value for the conceptual interoperability analysis task.

Efforts to support interoperability

In response to the potential advantages of software interoperability, many attempts have been made over the last few decades to support it. On the one hand, many interoperability standards, models, analysis approaches, and mismatch solutions have been proposed [CS12]. On the other hand, many providers of black-box interoperable software units publicly share documents about their units in the form of online API documentations, README files, reuse manuals, Wiki pages, etc.

Lack of support for conceptual interoperability analysis

Although the above-mentioned efforts led to improvements in integration projects, it is still hard in practice to analyze black-box software units with respect to their conceptual and architectural levels of interoperability. This, in turn, leads to the conceptual mismatches between the units to be integrated being missed, which imposes expensive rework to handle the conceptual mismatches detected late, provided such rework is even possible at all!

In this thesis, we explore the reasons behind the challenges faced in conceptual interoperability analysis in practice and investigate where the proposed solutions in the state of the art fall short in addressing them. Accordingly, we provide a framework that offers proactive support for conceptual interoperability analysis, allowing effective and efficient detection of conceptual interoperability constraints between separately developed software units.

## 1.2 Problem Statement and Goals

In this section, we start by presenting multiple-perspective practical problems (P.P) that impede software architects and analysts from performing conceptual interoperability analysis and mismatch detection. Afterwards, we will translate these practical problems into our practical goals (P.G). Next, we will describe the related research problems (R.P). These will then be translated into our research goal (R.G), which we aim to achieve in order to address the practical problems.



## Practical Problems and Practical Goals

### Relevance of the conceptual mismatches problem

Studies show that more than 40% of IT-related costs are spent on solving interoperability problems [MRPX08]. As introduced in the previous section, analyzing the interoperability of software units to detect their conceptual mismatches is critical for the success of integration projects. An example of the effect of such mismatches was reported by David Garlan [GAO95] in the context of an integration project of four separately developed software units. While the estimations indicated that the project needed six months to one person-years, it actually took two years a five person-years (i.e., four times the estimated project time and five times the estimated development effort). Garlan states that the reason behind the issue were the architectural mismatches resulting from the hidden assumptions about the structure of the integrated units. This has also been confirmed through published experiences with other integration projects [SK96]. Therefore, hidden and unstated assumptions and lack of documentation about interoperable software units are a significant problem that puts integration projects of third-party clients at high risk of producing incorrect or meaningless interoperation with under-estimated cost [SA11].

In the long run, providers of interoperable software units with such inadequate sharing of the conceptual constraints will lose clients, become less competitive and lose revenues. As illustrated in Figure 2, clients opt to build interoperation with an external software unit (reusing it) only if the providers invest enough in the unit to make the cost of reusing it lower than the cost of not reusing it [BB91].

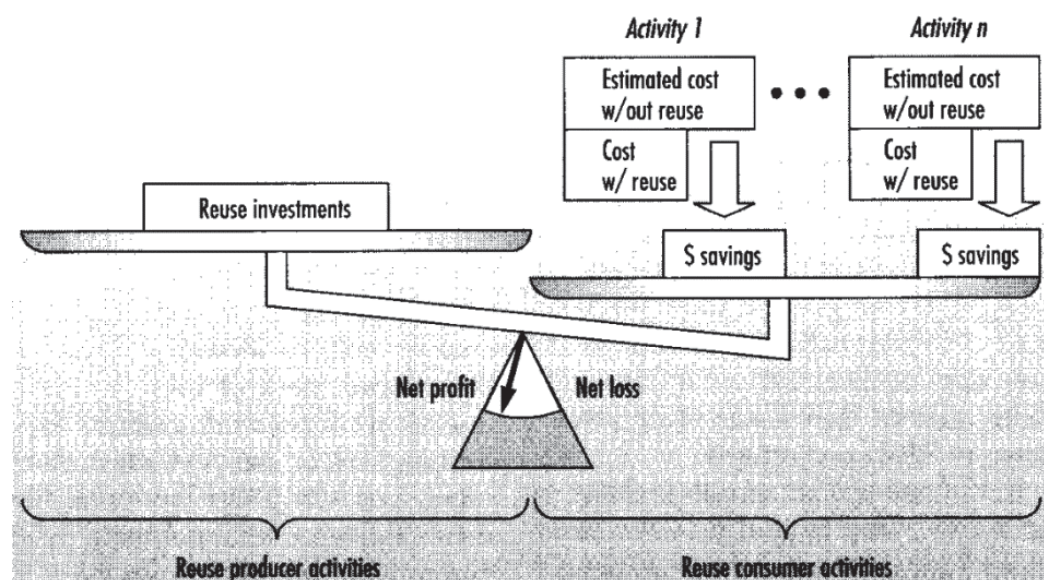


Figure 2

Reuse-investment relation [BB91]

Problem  
from the  
perspective  
of software  
providers

Hence, on the one hand, it is the responsibility of providers who offer interoperable black-box software units with which interoperation is possible to explicitly and comprehensively share the conceptual constraints for their units. Such information gets typically prepared by the architects of the software unit during its development or once it is offered for interoperation with other units. The quality of such documentation is considered to be one of the main factors that affect the cost of integration projects [AB97]. However, this is not a trivial task, as conceptual interoperability constraints are usually hidden within the internal architecture and design documents of the software unit. Add to this that these constraints are spread across multiple models (e.g., UML class diagram, sequence diagram, etc.). Therefore, it is *tedious* and *time-consuming* for the providers to manually analyze the in-house architecture documents, extract the interoperability-related conceptual constraints, and document the collected constraints for clients in a publicly shared document (e.g., an API document). This task gets harder with large software systems, especially with limited time and manpower resources. In such contexts, time pressure often leads to neglecting the documentation or to incomplete and inconsistent information [Sam97].

Furthermore, the task of sharing documentation about the conceptual and architectural constraints of an interoperable unit should be performed from the point of view of the reader (i.e., third-party clients performing the conceptual interoperability analysis) [CGB+02]. Thus, this task does, without a doubt, require the responsible architects to have *knowledge and experience* regarding the different types of conceptual interoperability constraints that need to be shared in order to allow meaningful interoperation.

**P.P1:** Explicit and comprehensive sharing of conceptual interoperability constraints for black-box software units is expensive and requires experience.

Problem  
from the  
perspective  
of third-party  
clients

On the other hand, it is the responsibility of third-party clients, who look for interoperable software units that satisfy their needs, to thoroughly analyze the conceptual interoperability between their systems and the external software units of interest. Such an analysis takes place early in the integration project and is performed by software architects or analysts. However, this is not a straightforward task; on the contrary, it requires clients to carefully identify and compare the conceptual constraints of their own software systems (from in-house documents) and of the external software units (from publicly shared information sources like API documents) in order to detect any conceptual mismatches. Thus, the conceptual interoperability analysis is tedious and time-consuming. An example study [PXZ+12] confirmed this fact by reporting that only browsing the eBay documentation for a web service called “AddFixedPriceItem”, which had about 52,657 words, took more than 10 hours without even extracting the constraints.



Apparently, the larger in size the software units under investigation and their documents, the higher the analysis cost (in terms of time and effort). Besides, this cost obviously rises as the number of external software unit candidates increases.

In addition, in order to perform the conceptual analysis task successfully, responsible architects or analysts need to have *knowledge and experience* regarding the task and the different types and impacts of conceptual constraints and mismatches. Moreover, Rubinger et al. [RB10] showed that interoperability constraints still slipped despite familiarity with the documentation, due to the verbose text and its informality, which requires manual *linguistic skills* for capturing, structuring, and saving information about constraints. Consequently, the significant effect of the integrator's experience on the analysis makes this one of the factors used in assessing the integration cost [AB97]. In other words, the success of conceptual interoperability analysis depends critically on the architects' and analysts' expertise.

Therefore, the cost and the requirements of conceptual interoperability analysis will often compel software providers to either neglect the issue of conceptual interoperability altogether or postpone it until the candidates are filtered based on how easy it is to integrate them from a technical point of view [Bhu07]. This puts projects at the risk of finding conceptual mismatches late, which defeats the desired gains of software integration.

**P.P2:** Analyzing the conceptual interoperability of software units and identifying their conceptual mismatches is expensive and requires experience.

Example from smart farming	Here we will illustrate these practical problems using our previous example from the smart farming industry. This is a very simple toy example, with respect to the description of the software units and the interoperation requirements, which we will use to shed light on the interoperability analysis problems on the conceptual level. We will also limit the conceptual interoperability constraints covered within the example at this point, but will detail them in Chapter 0.
Providing an interoperable Smart Tractor ( $S_1$ )	Imagine a company Alpha ( $\alpha$ ) that has developed a smart tractor ( $S_1$ ) with the intention of making its software system interoperable. That is, $S_1$ was meant to interoperate with other separately developed software systems (e.g., farms management systems, farmers' mobile apps, other smart machinery like harvesters and mowers, etc.). The interoperation offer includes services like <i>Auto/Remote Steering</i> , <i>Location Tracking</i> , <i>Task-Progress Monitoring</i> , <i>Field-Data Collecting</i> , <i>Usage Reporting</i> , and more. Thus, company $\alpha$ decided to invest effort into exposing the required information about $S_1$ for interested clients.

As  $\alpha$  wanted to start integrating  $S_1$  very soon due to a time-to-market constraint, Jana, the software architect responsible for  $S_1$ , was in rush during the preparation of the conceptual part of the shared documentation about  $S_1$ . She could not go through all available in-house specification, architecture, and low-level design documents of  $S_1$  to completely identify the conceptual constraints of the offered-for-interoperation elements (i.e., services and data). Hence, Jana updated the API documentation and integration manual web pages of  $S_1$  with as much main conceptual information as she could before the deadline.

Meanwhile, the owners of a smart farm ecosystem ( $SF$ ) were looking for a tractor that would boost the productivity of farmers on the field and would support informative decision-making by delivering data to the farm management system ( $S_2$ ). By searching the market, the owners of  $SF$  got interested in integrating an instance of  $S_1$  into their software ecosystem.

Are  $S_1$  and  $S_2$  conceptually interoperable?

Hence, Noah, the software architect responsible for the ecosystem who was a junior, started assessing the feasibility of building a successful interoperation between  $S_2$  and the tractor  $S_1$ . In this scenario, Noah had to analyze the conceptual and architectural constraints for both  $S_1$  and  $S_2$  to report any conceptual mismatches to the project manager as soon as possible, before the developers started implementing the technical integration. Noah started his ad-hoc manual investigation of the text in the shared API document and integration manual of  $S_1$ . He was overwhelmed by the technical constraints and code examples, which were much more than the small amount of conceptual information he was mainly looking for. For example, he found that the offered "*Remote Steering*" function would accept steering requests as a JSON file including the ID of the tractor sent using the secure communication protocol Secure Socket Layer (SSL), which would introduce technical mismatches with the  $S_2$ , which used the Transport Layer Security (TLS) protocol to send Java objects. However, Noah did not find any user restrictions on the *Remote Steering* functionality, so due to time constraints and low experience he assumed that it is permitted for any authorized driver, even concurrently with safety-related emergencies. Similarly, the *Usage Reporting* description did not explain the frequency of such reporting and whether the report would be per driver account or per tractor, so Noah took a note on that issue to get back to it later, but under time pressure he forgot. Besides, the undirected analysis led Noah to miss questioning other important concerns about quality and behavior. For example, he did not investigate the interaction property of  $S_1$ , which was indeed asynchronous and not sending confirmations for requests, unlike the synchronous interaction of  $S_2$ . Figure 3 presents our example of the detected technical mismatches and the missed conceptual ones.

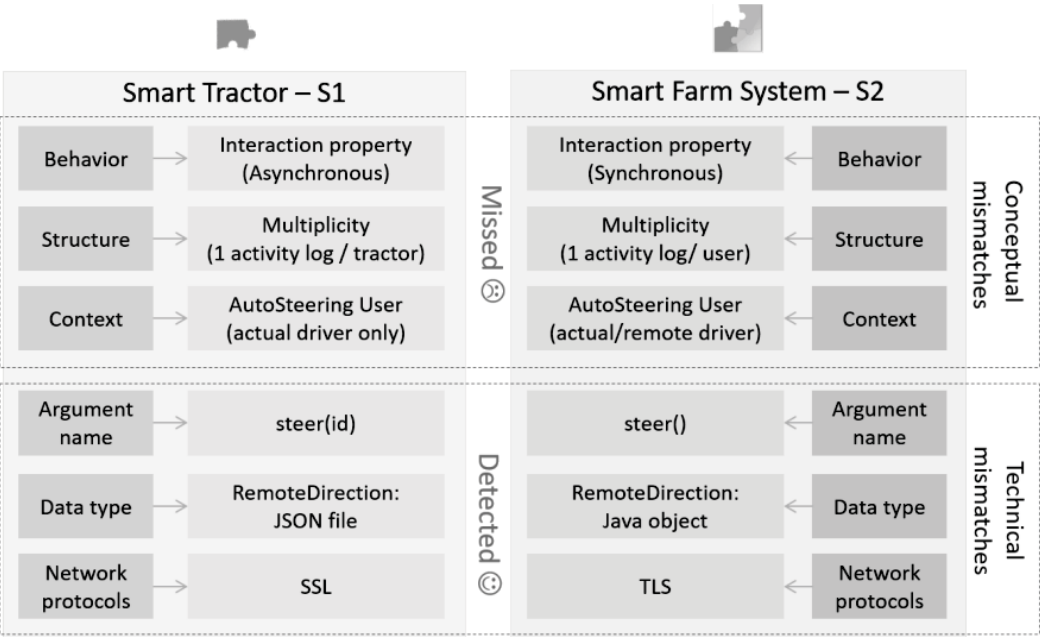


Figure 3 Example of detected and missed mismatches between S1 and S2

Ideally, if the conceptual constraints had been explicitly specified like the technical ones in the API documents, they would have alerted Noah during his analysis. However, the reality was far from ideal and these conceptual constraints were still hidden in the structural and behavioral UML diagrams of *S<sub>1</sub>*. After all the effort Noah put into the analysis, his report of the detected interoperability mismatches was incomplete and the integration project faced the risk of finding these unexpected conceptual mismatches late in the integration project.

A one possible approach for handling the missing constraints that Noah had not noticed during his analysis task would be to contact company  $\alpha$  asking for further information about *S<sub>1</sub>*. However, adopting this solution might be expensive for company  $\alpha$ , especially for repeated inquiries by different clients. Also, the waiting time might be inconvenient for the clients, leading to an unpleasant experience in integrating the product of company  $\alpha$  and affecting its reputation in the marketplace.

Practical goals

According to the aforementioned practical problems that we illustrated in the example, we formulate the practical goals that we seek to address in this thesis as follows:

- P.G1:** Increase the software architects' effectiveness and efficiency in identifying and sharing the conceptual interoperability constraints of their interoperable black-box software units.
- P.G2:** Increase the software architects' and analysts' effectiveness and efficiency in identifying the conceptual interoperability mismatches between their systems and external software units of interest.

The first step towards achieving these goals is to reveal the reasons behind the practical problems in performing the conceptual interoperability analysis. In the next section, we will therefore describe the research problems we identified that can lead to P.P1 and P.P2 with an illustrating example.

## Research Problems and Research Goals

### Related research problems

To cope with the practical problems mentioned above, different solution ideas have been proposed, each making a contribution from a specific angle. Here we outline the research problems (R.P) with regard to conceptual interoperability analysis that the state of the art has not covered yet. This leads us to state the research goal (R.G) of this thesis. Note that here we only provide a very brief overview of related work, but we will get into detail and present state of the practice and state of the art studies in Chapters 1 and 4.

We described that performing conceptual interoperability analysis is about identifying the mismatching conceptual interoperability constraints of two software units. Therefore, it is crucial to know WHAT conceptual interoperability constraints are, before looking into HOW to share and identify them. Only when this is achieved can the approaches for HOW to prepare for and perform conceptual interoperability analysis be of real benefit.

It is necessary for both the providers of interoperable software units and the clients who integrate these units into their systems to have a clear and mutual understanding of WHAT the conceptual interoperability constraints are, and to know their classification, the kind of mismatches they cause, and their impact on the planned integration. During the last few decades, many state of the art models have been proposed to classify interoperability. Although these models have established a strong basis for this property, they are abstract classifications for the concept and do not support the analysis purposes and activities. That is, none of them specify precisely what each classification level would include in terms of constraints that strict the software units that are to interoperate. Besides, the existing models do not relate the classifications to the types of mismatches they cause. Hence, these models have not found their way into practical approaches for conceptual interoperability analysis.

**R.P1:** Lack of theoretical foundation that defines the conceptual interoperability constraints and their related mismatches for software units.

Concerning HOW providers can prepare their black-box software units for interested clients, very few approaches have been proposed in literature. Most of the previous works focus on building and following interoperability standards, which are typically technology-oriented,

domain-specific, unsustainable, and absolutely unable to cover the semantics of software units. Some other reuse approaches call for manual creation and formalization of contracts and interfaces that mainly specify technical constraints rather than sufficiently including conceptual constraints as well.

**R.P2:** Lack of proactive approaches and automated solutions for guiding providers of interoperable black-box software units in identifying and sharing conceptual interoperability constraints for their units.

Similarly, with regard to HOW third-party clients analyze the interoperability between their systems and software units under investigation, many approaches aim at identifying mismatches. Examples of such approaches include contract-based conformance checking, testing-based techniques, and integration prototype analysis. Most of these activities target technical mismatches and some specific types of architectural ones. Furthermore, some of these analysis approaches are not systematic in documenting the results, and some are considered expensive as they depend on analyzing the executions after actually glue-coding the units. Besides, some approaches cannot be applied in the black-box integration context as they depend on analyzing and re-engineering the code.

**R.P3:** Lack of systematic analysis approaches that guide interoperability analysts in identifying the conceptual constraints of two software units and detecting their mismatches.

**Research goal** This motivated us to aim at tackling the stated research problems in order to overcome the practical problems. Our overall research goal, which is depicted in Figure 4, serves our previously stated practical goals. Accordingly, we state the research goal in our thesis as follows:

**R.G:** Provide proactive and systematic support for improving conceptual interoperability analysis practices.

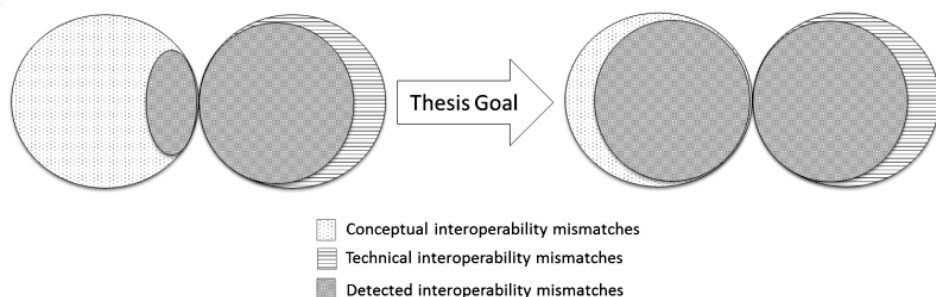


Figure 4 Overall thesis goal: improving the detection of conceptual interoperability mismatches



## 1.3 Solution Ideas and Hypotheses

In the following, we will describe our solution ideas (S.I) and map to the research problems previously stated in Section 1.2. We will provide an in-depth presentation of these ideas later in Chapters 0 and 6.

**S.I1:** A comprehensive model for Conceptual Interoperability Constraints (COINs) and related conceptual mismatches.

We started with addressing the theoretical research problem (R.P1) by extending and enhancing existing models of interoperability and reuse with a particular focus on the conceptual non-technical characteristics of interoperable software units. In particular, we established the relationships among different types of conceptual constraints, interoperable elements (i.e., system, data, and functions), and software unit types (e.g., IS, ES, etc.). The model also connects the conceptual constraints with the resulting conceptual mismatches.

**S.I2:** A framework for Conceptual Interoperability Analysis (COINA) that supports software architects and analysts in identifying the conceptual interoperability constraints and mismatches of software units more effectively and efficiently.

Founded on the aforementioned model, we contribute a supportive and guiding framework for engineering activities related to conceptual interoperability analysis. The COINA Framework comprises two methodical components to assist providers and clients of interoperable software units.

The first component of COINA addresses the methodical research problem (R.P2) that providers of interoperable black-box software units face in sharing conceptual constraints with clients.

**S.I2.1:** Proactive, semi-automatic, in-house preparation for interoperable software units, which helps software providers to explicitly share conceptual constraints with interested clients with the least effort.

This component helps providers taking charge of both extracting and documenting the COINs of interoperable software units. This proactive step makes black-box units ready for proper analysis by potential clients. The actual output of this component is a standard ready-to-share COINs document for clients. We facilitate this through our COIN templates, each of which is a rule. If it is satisfied, a COIN instance is extracted and documented. Our templates cover certain conceptual constraints of a software unit from its relevant architectural documents (i.e., UML structural and behavioral diagrams). We aid this template-based extraction by implementing an add-in for the Enterprise Architect [Spa] modeling tool.

Furthermore, we utilized machine learning capabilities in extracting existing COINs from the natural language text of public API documents related to the respective software unit. We also tool-support this extraction with an add-in for the Chrome web browser, which embeds the machine learning COIN-Classification Model that is our contribution. In addition, we provide documentation guidelines for improving the content and presentation of COINs in API documents.

The second component of COINA addresses the methodical research problem (R.P3) faced by third-party clients who are interested in external black-box software units while analyzing them.

**S.I2.2:** A systematic, algorithm-based method for mapping the conceptual constraints of system, which assists architects and analysts in detecting conceptual mismatches between software units.

This component systematically guides third-party clients in their mission to identify conceptual mismatches between their software units and external black-box ones. This is facilitated through our COIN Cheat Sheets, which provide guidance for the identification of different conceptual constraint types, and our Mismatches Cheat Sheet, which provide guidance for the identification of different conceptual mismatches between the constraints of two software units. Our algorithm-based mapping can be automated by formalizing the COINs of the two units. However, formalizing all the constraints does not seem to be a practical approach, so we offer guidelines on how to map them manually.

Furthermore, a systematic analysis with consistent documentation of the results of each step towards the final decision is of great importance. This especially applies in cases of comparing multiple candidate units, reflecting on reasons for rejecting candidate units, or learning from and saving analysis experiences. Therefore, our framework proposes standard documentation templates for saving the results of the identified COINs for each system as well as their detected mismatches.

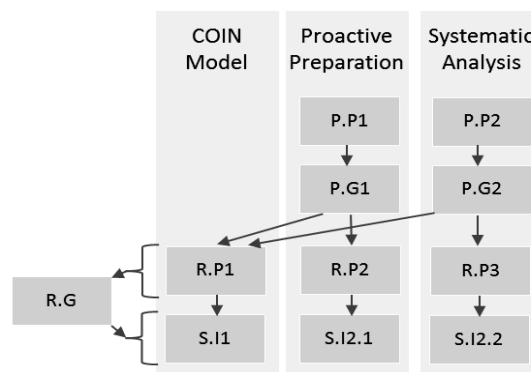


Figure 5

Mapping overview of problems, goals, and solution ideas

Figure 5 provides a big picture overview that summarizes and maps our stated practical problems, practical goals, research problems, research goal, and solution ideas.

## Research Hypotheses

For each of our proposed solution ideas, we expect to gain practical benefits that will translate into a number of scientific hypotheses. We present a very abstract view of these hypotheses here, but will refine and evaluate some of them in empirical experiments and case studies in the course of this thesis. Our hypotheses regarding the short-term practical benefits of the COINA Framework cover aspects of effectiveness, efficiency, and acceptance for both providers of interoperable software units and interested third-party clients, as follows:

**H<sub>Effectiveness</sub>:** Using the COINA Framework **increases** the **effectiveness** of architects in identifying and documenting the conceptual interoperability constraints and conceptual mismatches of software units that are intended to interoperate when compared to ad-hoc approaches.

**H<sub>Efficiency</sub>:** Using the COINA Framework **increases** the **efficiency** of architects in identifying and documenting the conceptual interoperability constraints and conceptual mismatches of software units that are intended to interoperate when compared to ad-hoc approaches.

**H<sub>Acceptance</sub>:** Using the COINA Framework as proposed is **accepted** by architects for identifying and documenting the conceptual interoperability constraints and conceptual mismatches of software units that are intended to interoperate.

In the long term, proactive preparation with COINA will help software providers achieve higher business impact and better competitiveness by increasing the success rate for interoperation. It is a one-time effort that will save costs in each future interoperation. Also, the COINA systematic analysis offers third-party clients engineering benefits such as traceability (e.g., linking mismatches to the conceptual constraints that cause them and verifying the resolution cycle from mismatch resolution requirements to resolution code), repeatability of the analysis results (which offers higher confidence in the analysis results and better estimation for conceptual mismatch-related risks), and experience reuse (which results from mining the accumulative cross-project interoperability analysis results). This also allows making informed decision and trade-offs among candidates. However, we do not list these long-term hypotheses in our thesis as they would require collecting data and monitoring results over years and across projects, which makes it unfeasible within the time period available for this thesis.



## 1.4 Research Context and Assumptions

Now that we have described the ideas of this thesis, we will establish the boundaries for our research and its contributions. That is, we will define the context and the assumptions under which our aforementioned benefits are expected to be realized.

Thesis domain	This thesis is dedicated to advancing the knowledge of <i>software engineering</i> and its contributions are aligned with the practical needs described in Section 1.2. The work focuses on <i>software interoperability</i> as an important quality attribute for systems with meaningful interoperation needs and interests. The proposed work is mainly dedicated to supporting the analysis of this property between two systems that are intended to interoperate.
Types of interoperable software units	We have described earlier in the motivation in Section 1.1 and the example in Section 1.2 that our research focus is on supporting the conceptual analysis of external <i>black-box software units</i> and we have focused our empirical evaluation on this type of units. However, it is possible to scale up the usage of our proposed proactive preparation and systematic analysis solutions to white-box software units, which offer third-party clients access to additional sources of information (e.g., public architecture documents or code). In addition, our ideas can be applied to support conceptual interoperability analysis for software units of <i>different types</i> (i.e., IS, ES, MS, etc.), as we will show later in the evaluation part of this thesis. Although not empirically evaluated, we believe that our solution benefits should not be limited to a specific software size (i.e., the units can vary in size from a single software API service to a complex software ecosystem consisting of multiple systems and services).
Level of interoperability	As mentioned above, the focus of this thesis is on the <i>conceptual interoperability</i> of software units and not on their technical interoperability. We give this priority to the conceptual level on the basis of its critical driving role for integration projects and because of the lack of support it has in the literature. However, this does not suggest that the other interoperability levels (e.g., technical, organizational, etc.) are not important. Accordingly, our contributed model and framework both focus on the conceptual aspects of interoperability constraints with the <i>architecture</i> -related ones being at the core. Therefore, we are after the conceptual constraints and mismatches of software units.
Software integration time	In the context of this thesis, the business need to build interoperation between a software unit and another external black-box software unit arises in the future, after both units have already been developed. Hence, integration requirements are <i>potential and not imperative</i> . In other words, the interoperable units have not been developed with prior requirements or concrete plans to match with any other specific units.

Hence, the interoperation between two software units is *not enabled by design, but rather by construction* in a later integration project.

Conceptual  
interoper-  
ability  
analysis time

Accordingly, our contributions assist interested third-party clients in analyzing the conceptual interoperability between their systems and the external units only when they are *already developed* and *after the emergence of integration requirements* based on business needs. However, within the integration project itself, the conceptual interoperability analysis reserves a place in the *early stages*. The results of the conceptual analysis determine the feasibility of the project and the requirements to resolve the conceptual mismatches. Therefore, this analysis comes *before the technical interoperability analysis* and *before designing and implementing* the integration.

Proactive  
preparation  
time

Although specific interoperation requirements are not in the picture while developing interoperable software units, our research ideas encourage *early proactive preparation* for potential interoperations. This preparation, which results in documentation for the conceptual interoperability constraints of an interoperable unit, takes place either *directly after developing* the software unit or *during its development life-cycle*, with attention given to updating the document whenever a change is introduced to the unit under development. This proactive preparation is considered as an investment that requires *one-time effort* to serve *many potential integrations*. Of course, such prepared documents need to be maintained in order to keep them *up to date and aligned* with software-unit-related changes. Overall, this preparation applies for both providers of black-box units and owners of software systems with interoperation interest in mind (e.g., establishers of a software ecosystem). In general, we expect such proactively prepared documents to reduce the cost and increase the quality of the conceptual analysis results.

Software  
engineering  
actors

The methodical contributions of this thesis (i.e., proactive preparation and systematic analysis) will aid *software architects and analysts* of interoperable software units. We expect our contributions to have strong effects even when serving *inexperienced* architects or general software engineers due to our detailed guidance and automation support. Great architectural knowledge and expertise would be prerequisites if no automated support was used for analyzing low-level design documents (i.e., UML diagrams).

Software  
engineering  
activities  
Level of  
automation

The main software engineering activity supported by our contributions is *analyzing* conceptual and architectural documents (e.g., API and UML documents) of software units. This particularly includes *extracting* the conceptual interoperability constraints of a software unit, *documenting* and *sharing* the constraints, *comparing* two software units' constraints to detect their mismatches, and *reporting* the detected mismatches.

For our proposed proactive preparation of the COINs document for interoperable software units, we offer semi-automatic tool support for COIN extraction from UML diagrams. As previously described in Section 1.3, the tool automates the extraction based on our predefined COIN templates; however, it requires architects to select the interoperable elements of the software units under analysis. On the other hand, the ML-based extraction of COINs from NL text in API documents can be totally automated if the corpus is enriched with more labeled sentences from more API documents in order to increase its reliability in detecting the COINs accurately.

**Assumptions** We have a number of critical assumptions on which we base our contributions. If they do not hold, then the results produced in each analysis step cannot be guaranteed to be correct or complete. First, during the interoperability analysis of software units, these units are *stable* and have *available up-to-date documentations* (e.g., the architecture, low-level design, or API documents). Second, these documentations also need to be *consistent* with each other. This consistency is assumed among UML diagrams of the same structural type (e.g., both class diagram and object diagram agree on the multiplicity relationships), the same behavioral type (e.g., both sequence diagram and data flow diagram are aligned in representing the process), and the different types (e.g., a structural class diagram and a behavioral sequence diagram both agree on the functions a specific class can issue). Similarly, the different documents are assumed to hold consistent information (e.g., information in the API documentation complies with the UML diagrams). Third, we suppose that the UML notations are used in the *correct* way as specified by the Object Management Group (OMG). Fourth, with regard to *accessibility*, we assume that architects have access to the in-house architectural documents of their own software units. Meanwhile, third-party clients are assumed to have access to some kind of shared documents about the black-box units, like API documentation, but not to architectural documents or code.

## 1.5 Overview of Contributions

In this section, we state the key contributions (C) of this Ph.D. thesis in four categories as described below.

**Foundation contribution.** This category includes the theoretical scientific contributions that offer a basis comprised of concepts, properties, and relations upon which the rest of the contributions can be built. In this category, we have the following contribution:

- **C1: Conceptual model and a classification.** Our “**COIN Model**” provides the foundation for the entire thesis work and explains how the conceptual interoperability constraints are notionally related to

interoperable software elements, software types, and mismatch types. By defining these relationships, we can determine which constraints need to be identified and mapped between two software units that are intended to interoperate in order to achieve an effective and efficient conceptual interoperability analysis. Based on it, we also provide a classification for the “**Conceptual Interoperability Mismatch**”.

**Methodical contributions.** This category includes the contributions to software engineering methods that will assist specific roles in performing specific engineering activities. In this category, we have a number of contributions that are the building blocks of our “**COINA Framework**”, which aids conceptual interoperability analysis as follows:

- **C2: Proactive preparation approach for interoperable software units.** This contribution aims at supporting software architects in the *COINs extraction and documentation* activities that are performed in-house for their own software units. This preparation serves both providers of interoperable software units and establishers of interoperable systems during the software development life-cycle.
  - o **C2.1: Documentation template for conceptual constraints.** Our “**COIN Portfolio**” is a documentation template designed to support architects in maintaining the COINs of their system in a standard and structured way. It allows consistency in documenting the constraints among different interoperable elements of the same software unit and among the different software units. Providers of interoperable units can use this portfolio as a means for explicitly sharing conceptual information about their software units with clients, which in turn will allow the clients to perform effective and efficient interoperability analyses.
  - o **C2.2: Extraction method for structured conceptual constraints through formal templates for UML diagrams.** Our “**COIN Templates**” are formal rules for identifying specific types of conceptual interoperability constraints from UML diagrams of interoperable software units. Whenever a rule is satisfied, a COIN instance is detected and can be documented in the COIN Portfolio of its software unit. This is facilitated by means of extraction algorithms.
  - o **C2.3: Extraction method for unstructured conceptual constraints by building a corpus and a machine learning classification model.** Our contribution “**COIN Corpus**” is a repository for natural language sentences that we fetched from a number of real API documents and manually labeled with one of the COIN types that we defined in the COIN Model. Based

on the corpus, we created our “**Machine Learning COIN Classification Model**”, which can automatically detect COIN instances in natural language text of API documents.

- **C2.4: Guidelines for improving API documentations.** Our proposed guidelines aim at enhancing the shared API documentation with regard to the content and presentation of conceptual interoperability constraints. These guidelines increase the usability and usefulness of API documentation from the point of view of the architects or analysts who perform the conceptual interoperability analysis.
- **C3: Systematic mapping approach for conceptual constraints to detect mismatches.** This contribution aims at guiding software architects in their manual *conceptual mismatches detection activity* between two software units. It starts by guiding the manual extraction of COINs for the two software units from their shared documentation (if their COIN Portfolios are not created proactively). Then it guides the manual mapping of the two units’ constraints to detect their conceptual mismatches. It mainly serves the interested party in building the interoperation during the design time and the development life-cycle of the integration project.
  - **C3.1: Guidance for manual extraction of conceptual constraints through cheat sheets.** Our “**COIN Cheat Sheets**” offer guidance for the manual extraction of the COINs of interoperable software units as a perspective-based analysis (if COIN Portfolios have not been created already). The sheets describe the different types of COINs and their categories, from different perspectives, along with examples and directions on their locations in the software artifacts (i.e., software requirements specification, UML diagram, and API documents).
  - **C3.2: Algorithm-based guidance for manual detection of conceptual mismatches through a cheat sheet.** Our mapping algorithm defines the process for comparing the conceptual constraints (or COIN Portfolios) of two units in order to find any conceptual mismatches between them. We support architects with a “**Mismatches Cheat Sheet**”, which describes the different types of conceptual mismatches and the potential COINs causing them with examples.
  - **C3.3: Documentation template for conceptual mismatches.** Our “**Mismatches List Template**” is a documentation template designed to support architects in maintaining traceability between the detected mismatches and the COINs causing them. It also facilitates consistency in documenting the mismatches, which supports trade-off and comparison among different analyzed interoperable software unit candidates.



**Technical contributions.** This category includes contributions that aim at realizing the methodical contributions to enable their efficient use by practitioners. In this category, we have contributions that demonstrate the feasibility of our COINA preparation through software tools for extracting the COINs as follows:

- **C4: Architecture add-in tool for semi-automatic extraction of conceptual constraints.** We implemented our “*CoinsExtractor Tool*” as an add-in for Enterprise Architect. It implements our formal COIN Templates for extracting constraints from different UML diagrams, then documents them in our proposed COIN Portfolio documents. To enable this, an architect needs to determine the interoperable software elements that he seeks by finding and documenting their constraints.
- **C5: Web browser add-in tool for automatic extraction of conceptual constraints.** This tool “*COINer*” is implemented as an add-in for the Chrome web browser and embeds our contribution “Machine Learning COIN Classification Model”. The tool allows an architect to select natural language sentences in an API document and to choose the COIN types he is looking for, and it automatically determines whether the sentences have such COINs.

**Empirical evaluation contributions.** This category includes empirical studies that aim at evaluating the hypotheses regarding the benefits of the research ideas. Our contributions in this category are as follows:

- **C6: A survey with practitioners**, where we have confirmed at a statistically significant level the practical problems addressed in this thesis (i.e., the lack of guidance and the insufficient input for conceptual interoperability analysis). From another angle, the survey also shows the relevance and importance of the conceptual constraint types, which we capture in our COIN Model, for a successful conceptual analysis.
- **C7: A multiple-case study**, where we collected evidence on the comprehensiveness of our COIN Model regarding coverage of the conceptual interoperability constraints that can be described in the current publicly shared documentation of black-box software units. In particular, this study investigated a number of real API documentations.
- **C8: A multiple-run controlled experiment**, where we confirmed our hypotheses that our systematic analysis method significantly increases the architects’ effectiveness (in terms of recall and precision) and efficiency (in terms of expended time) in detecting the conceptual constraints and mismatches of two software units.
- **C9: A survey with practitioners**, where we confirmed at a statistically significant level our hypotheses that our guidelines for improving the API documentation are perceived by practitioners as

being capable of improving the usefulness and ease of use of API documentations.

- **C10: A small initial experiment**, where we got an indication that our guidelines for improving the API documentation can actually improve the usefulness and ease of use of API documentations.

In Figure 6, we map the foundation/methodical contributions to their related evaluation contributions.

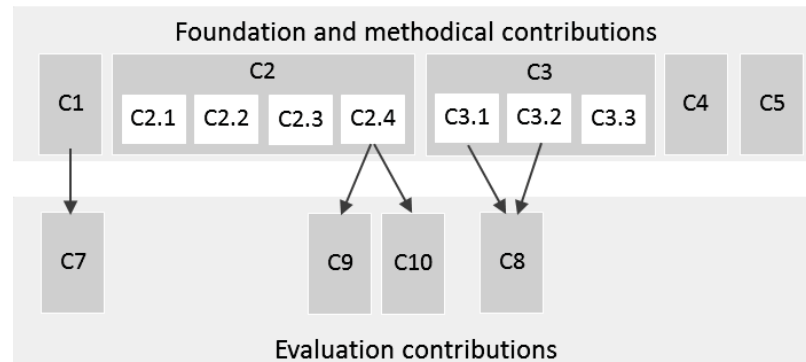


Figure 6 Relations among foundation/methodical contributions and evaluation contributions

## 1.6 Research Method

In this section, we will describe the method employed in this Ph.D. research, which is summarized in Figure 7.

**Identifying the practical problem (exploring the state of the practice).** To characterize the practical problems of conceptual interoperability analysis, we designed and performed an “Online survey on the difficulties of interoperability analysis practices and its input artifacts”. The goal of this study was to identify the problems faced by software architects and analysts when they perform interoperability analysis. The investigation explored problems such as experience with the conceptual constraints, insufficient conceptual information in shared documents, and lack of guidance and tool support for this task. The survey results confirmed the criticality of the explored problems that we aim at tackling in this thesis work.

**Identifying existing solutions and research gaps (exploring the state of the art).** To identify and characterize the research works on solving conceptual interoperability difficulties, we designed and performed a “Scoping study on conceptual interoperability problems and solutions”. The study revealed that studies have been performed mostly on technical interoperability, while few have targeted conceptual interoperability. The identified solutions for conceptual interoperability problems were mainly reactive rather than proactive. These results

helped us in directing our research ideas and formulating our hypotheses.

**Developing the solution idea.** After stating our research goals, we started by designing the research components we needed in order to achieve our solution ideas. Then we incrementally elaborated the abstract components with our concrete specification for the input, the processes, and the output. Our solution development approach included a “Multiple-case study” with experiments. One of its goals was to explore the potential advantages of utilizing machine learning (ML) techniques in automating the extraction of conceptual interoperability constraints from natural language text of API documents. The results of this study included our COIN Corpus, our ML COIN Classification Model, and guidelines for improving API documentation. In addition, we received regular feedback from senior software architects and software engineering researchers at Fraunhofer IESE throughout our Ph.D. work, which we used to enhance and refine our research ideas and components. By presenting our ideas and results to the Software Architecture community at several scientific conferences we got confirmation and support from experts on the value of our contributions and our research direction.

**Evaluating the solution idea.** With regard to evaluating our solution ideas, we performed this evaluation on two levels:

- On the *research level*, where we partially tested our internal hypotheses in controlled experiments. We designed and performed a multi-run experiment to evaluate our hypotheses regarding the systematic approach. Furthermore, we designed and performed an initial small experiment to evaluate our hypotheses regarding our guidelines for improving API documentation.
- On the *practical level*, where we partially tested our external hypotheses in a survey and a multiple-case study. We designed a confirmative survey to evaluate our hypotheses regarding the perceived value of our guidelines for improving API documentation as perceived by practitioners. Besides, we used the collected data from our previously mentioned multiple-case study to check the comprehensiveness of our COIN Model in covering all types of existing constraints in current documentations in practice.

## 1.7 Thesis Outline

In Chapter 2, we will introduce the background of this thesis work. This will start with a description of the sources of conceptual interoperability information, conceptual interoperability mismatches, and the machine learning and natural language processing techniques we used.



In Chapter 3, we will describe the state of the practice for conceptual interoperability analysis problems that we characterized through our survey with practitioners. The survey revealed problems related to practices and input documentation of interoperable software systems. We will describe the study design, the data analysis and its results, and the threats to validity.

In Chapter 4, we will characterize the state of the art regarding conceptual interoperability constraints and problems that we obtained through our scoping study. We will describe the study design, the data analysis and its results, and the threats to validity. Furthermore, we will present important related work on the topic of identifying conceptual interoperability constraints and existing interoperability analysis approaches.

In Chapter 5, we will present our COIN Model, which defines the different types of conceptual interoperability constraints. We will relate the types to the interoperable software elements and to the type of interoperating software unit. We will also mention the different types of conceptual mismatches that can be caused by the constraints. Furthermore, we will describe the standard templates that are our contribution for documenting conceptual interoperability constraints and mismatches.

In Chapter 6, we will introduce our engineering contributions that we propose as the COINA Framework for supporting software architects in performing conceptual interoperability analysis. We will present each of the framework components, their methods, and the supporting tools in detail with an illustrative example.

In Chapter 7, we will present the empirical evaluation studies that we used to test our hypotheses. We will start by refining our hypotheses, then we will describe how we partially evaluated them on the research level using controlled experiments and on the practical level using a multiple-case study. For each evaluation study, we will describe its design, the data analysis and its results, and the threats to validity.

In Chapter 8, we will summarize our contributions and results, discuss benefits and limitations, and finally provide ideas for future work.

Note that some of the chapters of this thesis have been published earlier by the author in a number of papers (i.e., [ATR14], [Abu14a], [ANR15], [AAR15], [AR16], [AAHR16], [AAR16], and [AANR17]). Some material from these papers has also been incorporated into this introductory chapter.

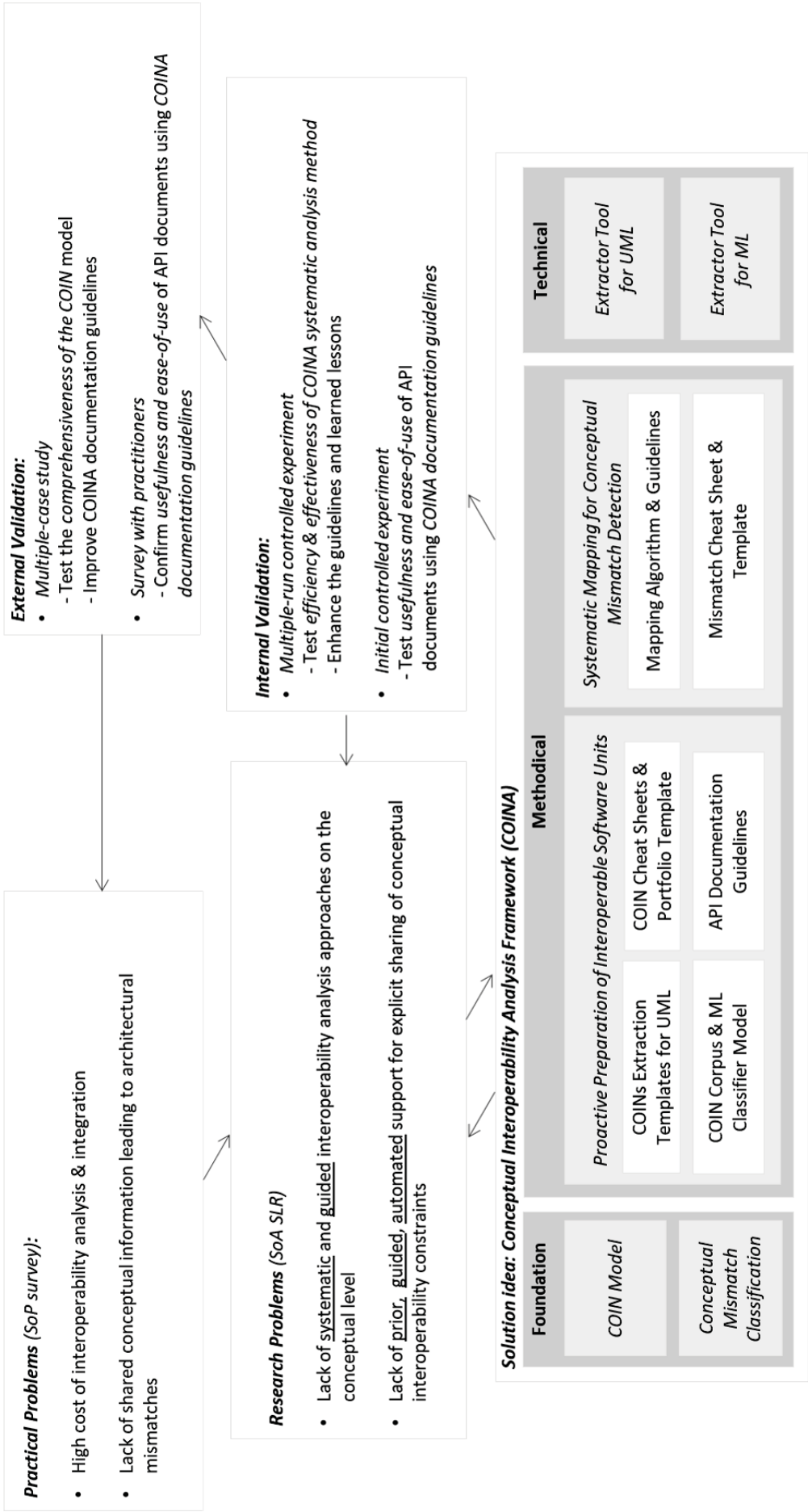


Figure 7 Summary of Research Method and Contributions



---

## 2 Background

### 2.1 Introduction

This thesis contributes a framework for analyzing the conceptual interoperability of software units. In this chapter, we will briefly explain the foundations on which we base our contributed framework as follows:

- In Section 2.2, we will introduce the basic concepts of the software artifacts that we consider as the input for our conceptual interoperability analysis framework in the context described in Section 1.4.
- In Section 2.3, we will clarify the fundamental principles of interoperability mismatches, which we consider as the output of performing our proposed conceptual interoperability analysis.
- In Section 2.4, we will present the natural language processing and machine learning techniques that we utilize in our framework to support some activities of the conceptual interoperability analysis.
- In Section 2.5, we will summarize the content of this chapter.

Some of the content presented in this chapter has been published in background sections of the author's publications [AAR15, AR16, AAR16].

### 2.2 Sources of Conceptual Interoperability Information

#### Software Architecture

The core artifact that preserve conceptual information about a software system is its architecture. Many definitions of software architecture can be found in the literature. One widely adopted definition of software architecture is given below [BCK03]:

#### **Definition 4 – Software Architecture**

The structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [BCK03].

This software artifact abstracts the details of the software unit on both the level of small building blocks and the level of the whole composed

system. It encompasses the design decisions about the software's structural and behavioral aspects in more detail than just natural language text. In fact, according to the Architecture Decomposition Framework, architecture captures information about data (e.g., data flows), functions (e.g., interfaces and connectors), deployment (e.g., communication paths), activities (e.g., operation processes), and technologies (technology usage) of the software system and its context both at runtime and at the development time of the software system [FI]. As the focus of our research is on runtime conceptual interoperability, we pay attention to the runtime information of the software system and its context in this framework. For example, on the function level we care about runtime layering information but not about development packaging or modules.

Therefore, software architecture and low-level design documents are rich sources of information for architects and analysts regarding the software's conceptual constraints and assumptions. In particular, within the context of this thesis, such documents are used by architects to determine the conceptual interoperability constraints that need to be shared with third-party clients to allow the conceptual analysis.

A widely used modeling language for software architecture is the Unified Modeling Language (UML) [Boo05], which allows describing the structure and behavior of a software with standard notations. For example, software structure can be described using class/object diagrams (which describe the data objects and their relations), component diagrams (which show how components are wired to form larger components or systems), deployment diagrams (which model the allocation of artifacts to physical nodes), etc. Similarly, software behavior can be described using sequence diagrams (which show the interactions and their order among actors and software objects), use case diagrams (which give an overview of the system functionalities), activity diagrams (which capture the business processes or workflows), etc.

## Application Programming Interface (API) Documentation

The term API stands for "Application Programming Interface", which provides access to software functionalities at a very high level of abstraction level. That is, APIs allow software developers to reuse an existing software solution without knowing its implementation and without the ability to edit it (i.e., black-box interoperation). In its simplest form, an API can be defined as:

### **Definition 5 – API**

The standard contract provided by a piece of code (i.e., software unit) to another to enable their black-box interoperation.

There are different types of APIs in different application domains. These include, for example, *platform APIs*, which offer import of libraries to use their functions (e.g., Java SDK, AppleWatch APIs, Eclipse APIs, etc.) and *web APIs*, which offer services that can be accessed via their address on the World Wide Web (e.g., Google Maps APIs, Twitter APIs, SoundCloud APIs, etc.). These APIs cover a wide range of applications, such as social blogging, audio, software development, etc.

To facilitate successful black-box interoperation with a software unit through an API, providers of the API share a public document specifying the offered services and how to use them. In particular, the API documentation describes the structure of exchanged input and output data (e.g., classes and variables), the interaction procedures, and sometimes illustrative usage scenarios and code snippets. Accordingly, this documentation is considered as an essential source of information for its users to learn how to use it [HBH+10]. API documentation can exist as files, websites, wikis, blogs, etc.

## Software Requirements Specification (SRS)

In the software development lifecycle, the software requirements specification is the artifact that states unambiguously and completely what needs to be developed within the project. Based on the IEEE description of the SRS [IEE98], it can be defined as follows:

### Definition 6 – SRS

The standard specification document for a particular software product, which declares the functional and non-functional requirements along with usage context, interaction interfaces, and design constraints [IEE98].

Often, the SRS document is written using free natural language text (e.g., documented in a MS Word file or a tabular MS Excel sheet) and may include scenarios and use cases (e.g., documented according to a structured or tabular template). For organizing the structure of SRS documents, there are a number of widely used templates, such as the IEEE standard 830 [IEE98], which is used for communicating developer requirements, and the Volere Template [RR07], which is used for communicating both users' and developers' requirements.

An SRS for a software integration project is written to specify the desired interoperation requirements, constraints, and qualities. Hence, it provides a baseline for third-party clients against which they select an external software unit and determine if it meets their needs. Thus, we consider the SRS of an integration project as an important source of conceptual interoperability constraints expected by the clients of interoperable software units.

## 2.3 Interoperability Mismatches of Software Units

In the following, we will explain software mismatches in general and architectural mismatches in particular as they are the core of the conceptual mismatches we target in our analysis framework (i.e., the output of the conceptual interoperability analysis).

### Interoperability Mismatches

Analyzing the features of two separately developed software units can reveal different types of interoperability mismatches between them that can prevent their interoperation. These software mismatches may be due to incompatible or *conflicting features* of the software units. For example, a software unit may exchange textual data in a specific file format (e.g., .txt file) that is different than the format used by another unit with which it is supposed to interoperate with (e.g., a .pdf file). Also, software mismatches may happen as a consequence of some *influential features* of a software unit that are inappropriate for other interoperating units. For example, the programming language of a software unit can affect a desired quality (e.g., portability is better supported by units written in Java compared to those written in C) or the expected input/output data (e.g., a unit written in C may exchange *void*, which is not a supported type in another unit written in Java).

Interoperability mismatches stretch over different perspectives depending on the type of the features of the software units intended to interoperate. That is, mismatches may exist due to conflicts among influential features, including those of technical (e.g., different technologies and protocols), functional (e.g., interoperation needs not being met by offered units), conceptual (e.g., unaligned data models), developmental (e.g., software units developed with no built-in quality, like testability, can affect the integration productivity), organizational nature (e.g., different policies for data privacy).

### Architectural Interoperability Mismatches

The previous description of software interoperability mismatches is too broad for the focus of this thesis. Therefore, we will here give a deeper explanation of the architectural mismatches that our research focuses on for the reasons discussed in Chapter 1.

Two of the critical reasons causing interoperability mismatches between software units are *architectural constraints* and *design decisions* (e.g., a software unit designed to be secure may influence the performance and usability, which are of higher interest for another software unit). Architectural constraints may cause mismatches between the structures of different units (e.g., control model and data manipulation), the relationships between units (e.g., interaction protocol



and exchanged data), or the global architectural structure (e.g., assumptions about the topology of the system communication) [GAO95]. Architectural mismatches vary considerably with regard to the impact they have on the desired interoperation [EMG00] and consequently on the required effort to achieve it. Some can be easily resolved (e.g., inconsistent interfaces require mapping with the help of adapters or wrappers), whereas others may be more complex and expensive to handle (e.g., inconsistent data semantics may require a semantic-based middleware to enable meaningful interoperation among different units). This emphasizes the importance of performing conceptual interoperability analysis, for potentially interoperable software units, early in an integration project.

## 2.4 Natural Language Processing and Machine Learning Techniques

### NLP Techniques

Communication using natural language (NL) is well suited for humans; however, it is difficult for machines to interpret and learn. Natural Language Processing (NLP) has emerged as a field in computer science that combines the usage of both Artificial Intelligence (AI) and Computational Linguistics (CL) to enable interaction between machines and human natural languages. Below we will briefly explain the main NLP techniques we have used in our research.

*Part-of-speech (POS) tagging* [MS99, KM03] is also known also as “grammatical tagging” or “word tagging”. It is a technique that identifies the part of speech to which a term in a sentence belongs (e.g., noun, verb, pronoun, adjective, etc.).

*Phrase and Clause Parsing* [Gro99] is also known as “chunking”. It is a technique that divides sentences into sets of words that are logically related (e.g., verb phrase and noun phrase). On top of POS tagging, this technique improves the syntax of an NL sentence.

*Typed Dependencies* [DMMM+, DMM08] is a technique that represents dependencies between individual words through labels for grammatical relations (e.g., subject, indirect object, etc.). It provides a simple description of relations and does not require linguistic expertise to extract them from the text.

*Named Entity Recognition* [FGM05] is also known as “entity identification”. It is a technique that identifies specific words and categorizes them based on predefined classes. These classes often have a higher level of abstraction and depend fundamentally on the semantic meaning of the words.



## ML Techniques

Machine learning is a branch of computer science that enables machines to do what they have not been explicitly programmed to do [SSBD14]. It includes a wide range of techniques including, but not limited to, “text classification” techniques. Text classification refers to automating the detection of patterns in the text of specific problem domains through algorithms that learn from offered training data. Below, we briefly introduce the ML text classification techniques that we utilized in this thesis work.

*Text Classification (TC)* [Seb02] is the process of labeling natural language sentences in textual documents with one or more predefined classes or themes. In this thesis, we utilize supervised ML, which relies on the exposure of text classifier algorithms to a training data set. We prepared this dataset by manually labeling sentences with one of our predefined classification classes. Our research includes use of the most effective text classifier algorithms, such as Naïve Bayes [Mur06], Support Vector Machine [Joa98], [TK01], Random Forest Tree [LW02], K-Nearest Neighbor (KNN) [CH67], and others. The accuracy results of such algorithms depend on the quality and size of the training dataset [BB01].

The *ML Text classification process* consists of:

- Building the classification model, where all features of the sentences in the dataset are identified and modeled mathematically. In our research, we used popular techniques for building our model: (1) Bag of Words (BOWs) [CL05] considers each word in a sentence as a feature; accordingly, a document is represented as a matrix of weighted values; (2) N-Grams [OJ12] considers each N adjacent words in a sentence as a feature, where ( $N > 0$ ); (3) Term Frequency- Inverse Document Frequency (TF-IDF) [Rob04] is often used in text classification as a weighting factor to illustrate the importance of a word for a document in a dataset.
- Evaluating the classification model, where the manually labeled dataset is divided into a training set and a testing set. The training set is used for training the ML classification algorithm on the features captured in the model, while the testing set is used for evaluating the accuracy of the classification. For our research, we used k-fold cross-validation [K+95], dividing our ground truth dataset into k folds, meaning that, ( $k - 1$ ) folds were used for training and one fold was used for testing. Finally, an average of k evaluation rounds was computed. The ML binary (or two-class) classification algorithms are all about finding the best classifier model that represents the distribution from which the data comes and that separates the two classes effectively. Finding this most appropriate model not only requires finding the most proper ML algorithm, but also it requires a computationally expensive tuning for

the algorithm's parameters. This is called optimization, which helps to get better classification results by choosing the best combination of values for the different algorithm parameters. This can be implemented using, for example, the Grid Search method [HCL+03], which is an exhaustive search process to find the best values from all possible combinations of the tuned parameters. Another option to do the parameter tuning is to adopt a randomized search [BB12].

## 2.5 Summary

In this chapter, we have explained the fundamental concepts of input artifacts for conceptual interoperability analysis in the context of black-box integration, software interoperability mismatches, and both the NLP and ML techniques used in this thesis work. In the next chapters, we will frequently refer to the fundamental concepts presented here.



---

## 3 State of the Practice

### 3.1 Introduction

It is important to collect practitioners' experience and confirmation regarding current state of the practice problems related to software interoperability analysis. This shows us the relevance and significance of the practical problems we address in this thesis. This also directs us in shaping solution ideas that will have a valuable impact. Hence, we decided to perform a systematic explorative study in the form of an online survey to identify the practical challenges related to interoperability analysis. It has been proven that this type of studies is appropriate for approaching a large number of participants and that it allows collecting a wide range of needed data (e.g., actual experiences or personal opinions) [DSM11] [Fow95].

### 3.2 Survey on the State of the Practice

In this section, we will start by describing the research methodology of our survey study (Subsection 3.2.1), then present the results and discuss them (Subsection 3.2.2). Finally, we will present the threats of validity to the survey study (Subsection 3.2.3).

#### 3.2.1 Research Methodology

##### Goal and Research Questions

We formulate our survey goal in terms of the GQM goal template [VSBCR02] as follows:

**Goal:** *to explore the state of the practice of interoperability analysis for external software units for the purpose of characterizing its current state and identifying its difficulties with respect to its practices and input artifacts in the context of a survey from the viewpoint of architects and analysts as the basis for developing practically applicable enhancements towards efficient and effective interoperability analysis.*

We translate this goal into the following research questions (RQs):

**RQ1:** How is interoperability analysis *currently* being performed *by practitioners*?

**RQ2:** What are the difficulties experienced when performing interoperability analysis?

**RQ2.1:** with respect to its *current practices*?

**RQ2.2:** with respect to its *input* (i.e., currently shared sources of interoperability-relevant information about external software units)?

## Survey Design and Execution

We designed this survey according to the guidelines proposed by [Dan11], [DSM11] [Fow95], and [Kva08].

**Target Group.** The target group of our survey consists of architects and software engineers with practical experience in interoperability analysis in software integration projects (including projects in which integration is a part). Accordingly, we invited 115 practitioners known for their experience in software integration directly via emails and asked them to distribute the call to whoever they deemed appropriate. We also posted the call for experts' participation on websites of professional groups (e.g., LinkedIn special interest groups for architecture, software interoperability, and software integration).

**Questionnaire Structure.** Based on our stated goal and research questions, we designed a questionnaire-based survey, which we supported with a *2-minute video* [Abu16c] to introduce the concepts and terminologies of interoperability analysis that we use in order to ensure correct understanding of questions and answers. The questionnaire was designed to collect information about the following aspects as depicted in Figure 8:

- Current state of interoperability analysis practices and input artifacts (RQ1), i.e., the respondents' practical feedback on performing the interoperability analysis task with regards to the process, the responsible roles, the methods or tools used , and typically available sources of information used by them.
- Perceived difficulties (RQ2 including RQ2.1 and RQ2.2), i.e., the main difficulties that practitioners experience when analyzing the interoperability of external software units especially on the conceptual level. This includes the cost of the analysis and of unexpected conceptual mismatches, the lack of knowledge and guidance, the quality and availability of sources of information, the need for tooling support, standard templates for documenting analysis results, etc.
- Demographic information, i.e., the respondents' experience in performing interoperability analysis, the type of applications they build and integrate, their position, organization type, sector, size, and location.

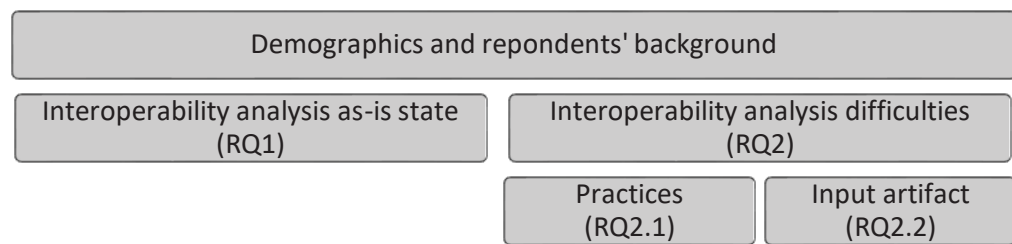


Figure 8 Questionnaire Design

**Type of questions.** The questionnaire consists of fixed-set answer questions, including single-choice and of multi-choice questions. To avoid restricting the range of potential answers that respondents may provide, some questions offer the choice to enter a free text answer if none of the provided selections applies. Also, some questions have the option of answering “I don’t know” to avoid forcing an answer if there is a lack of knowledge.

**Questionnaire length.** The designed questionnaire includes up to 26 questions as we used filters to avoid overwhelming the participants with questions that may be irrelevant for their experience. Answering the questionnaire required between 10 and 15 minutes. The final version of the questionnaire is provided Appendix A.

**Pre-execution evaluation.** To assess the quality of our survey, we had it peer-reviewed by two senior software engineers with expertise in interoperability and architecture to check the relevance and clarity of the questionnaire and its supporting video. After revising the survey according to the results of the peer review, one expert in empirical software engineering research assessed the quality of the questions with respect to the principles defined in [Kva08] and the ethical criteria stated in [Ass13]. As a result, some questions were re-categorized and shortened to improve understandability.

**Implementation.** We implemented the survey using Limesurvey [Tea15] and deactivated it after six weeks. The final dataset is stored in a repository of the AGSE group of the University of Kaiserslautern. If the reader is interested in further anonymized analysis results, please contact the author.

**Pilot study.** We performed a pilot study with four software engineering researchers (with a background in interoperability) and two computer science researchers (with no background in interoperability) to assess the understandability of the questions and to estimate the time required to answer them. We encouraged the six researchers to take notes on any ambiguous words, uncertainties regarding the meaning of the questions or their answers, and to track the time they spent on filling out the questionnaire. As a result, two questions were classified as very complex, so we reduced their complexity by splitting them into logical

subgroups. Note that we configured our online survey such as to record the time spent by each participant on the questions, which we found to be as reported by the participants themselves.

### Data Analysis

We analyzed the data using MS Excel and IBM SPSS Statistics 23 [Cor10]. Our descriptive analysis includes the median, mean, max, min, standard deviation, and frequency. As some of the questions were presented to the respondents conditionally, we explicitly report the total number of subjects ( $N$ ) who answered each question. We further present statistical analysis that explores how significantly different the ordinal answers are from a specific point (e.g., a neutral 3 for questions with a 5-point rating scale) using the one-sample Wilcoxon signed-rank test [Woo08]. Also, we statistically analyzed the difference in the answers between two groups of respondents based on their interoperability analysis experience. For this, we used Pearson's chi-square ( $\chi^2$ ) test for binary data and the Mann-Whitney (U) test for ordinal data. In addition, we ran a Spearman rho ( $\rho$ ) test to check the correlation between the ordinal variables.

### 3.2.2 Results and Discussion

In total, we got 73 complete responses for our survey questionnaire from the targeted group. However, we excluded nine responses as their demographic information showed that they neither had any year of experience in integration nor worked on any integration projects. This ensures that our findings are credible and based on actual experiences rather than inexperienced opinions. Accordingly, the final number of included respondents was  $N = 64$ .

Next, we will first provide a brief overview of the demographic characterization of the respondents, and then answer the research questions by presenting the results we found based on the included responses.

#### Background Overview

**Software integration experience.** The respondents' experience in software integration varied in terms of their years of experience and the number of projects they had participated in (see Figure 9). The largest shares (35.9% and 30.2%) of years of experience were for "2-5 years" and "> 8 years", respectively. Most respondents (25.4%) had participated in more than five projects. Across the different integration projects, the respondents reported playing different roles. These roles were frequently reported to be programmers (71.4%), architects (54%), system analysts (39.7%), project managers (34.9%), testers (23.8%), requirements engineers (19%), and technical writers (10%). Single



cases reported playing other roles (e.g., DevOps engineer, technical project manager, system expert, and risk assessor).

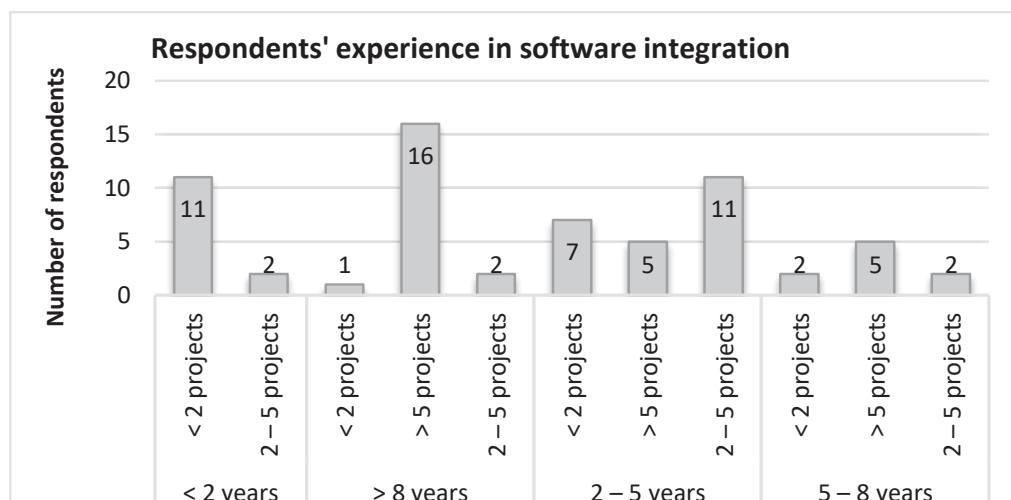


Figure 9

Respondents' experience in software integration

**Nature of integration projects.** The respondents worked mainly in the domains of Information Systems (80.8%), Mobile Systems (30.1%), and Embedded Systems (26%). The respondents reported experiences in integrating open source software (OSS), commercial-of-the-shelf (COTS) software, web service APIs, and platform APIs, all with almost equal shares (54.5% on average for each type). The projects' size was mainly (40.6%) medium (i.e., 3 - 6 months, \$250-750K, 4 - 10 team members) and the remaining share was divided almost equally between large projects (i.e., > 6 months, > \$750K, > 10 team members) and small projects (i.e., < 3 months, < \$250K, 3 - 4 team members).

**Nature of work organizations.** While, the respondents were mostly (37.1%) employed by enterprises (i.e., with > 1000 employees), very few (only 8.1%) were employed by large organizations (i.e., 251 – 999 employees). The others were distributed equally (27.4%) between small (i.e., < 50 employees) and medium (i.e., 51 – 250 employees) organizations. Among the many reported sectors (e.g., agriculture, health, finance, military, automotive, etc.), the software development sector was dominant. The data was collected internationally from several locations (e.g., USA, UK, Switzerland, Belgium, China, etc.), but the majority (32.8%) was from Germany.

### Interoperability Analysis: As-Is State in Practice (Answering RQ1)

In this part, we will characterize the current state of the practice with regard to the interoperability analyses actually performed by the respondents to our survey.

**Neglected interoperability analysis.** To our surprise, we found that a large group (42.2%) of the respondents stated that interoperability

analysis did not take place at all in their integration projects. These respondents ( $N = 27$ ; 42.2%) attributed this behavior to different reasons, including:

**R1:** There is not enough knowledge and experience about it (37%).

**R2:** Priority is given to other tasks, e.g., coding and testing (37%).

**R3:** Tight schedule and limited resources (29.6%).

**R4:** It is hard to perform (18.5%).

**R5:** Unit and integration testing are performed instead (18.5%).

**R6:** It is not that necessary (14.8%).

Furthermore, some respondents independently provided their reasons for not performing interoperability analysis. For example, one respondent stated that, in the context of integrating software units into ecosystems, they could replace this interoperability analysis with some negotiation on business rules and software interfaces. This shows a business-oriented analysis regarding interoperability, which cannot reveal conceptual software mismatches by itself. Another respondent, who also selected *R6*, reported that in his unit they had decided to proceed with integrating units only if they were implemented using the same implementation technologies. This indicates a probable awareness problem regarding the consequences of ignoring early interoperability analysis.

Looking at the demographic characteristics of these 27 participants, we found that the majority were working in small organizations with fewer than 50 employees (35%) on medium-size integration projects (40%) with less than 5 years of integration experience (61.53%). Note that the only statistically significant factor correlated to this group of participants (i.e., Spearman's  $\rho = 0.307$ ,  $p\text{-value} = 0.015$ ) was their reported estimation regarding the rather expensive cost of the analysis. This indicates that it could be the cost of the analysis that discourages practitioners from performing it.

**Immature, unstandardized, unsystematic interoperability analysis.**

The rest of the respondents ( $N = 37$ ; 57.8%) stated that interoperability analysis actually took place in their integration projects.

*When?* The responses show that the majority (67.6%) performed it at the beginning of the integration projects and before starting the technical implementation. However, many other respondents (24.3%) stated that interoperability analysis happens during the technical implementation. Such an approach would obviously require reworking the implemented parts of the integration if the analysis shows the existence of more mismatches. Through statistical analysis, we found that the analysis time was significantly correlated to integration

experience in terms of number of years (i.e., Spearman's  $\rho = -0.608$ ,  $p\text{-value} = 0.000$ ) and number of projects (i.e., Spearman's  $\rho = -0.679$ ,  $p\text{-value} = 0.000$ ). That is, experienced analysts recognized the importance of early analysis. Very few respondents (only 2.7%) reported that analysis takes place after the technical implementation, which is, in fact, the worst time (in terms of rework consequences) to detect conceptual mismatches.

*Who?* The roles responsible for performing the interoperability analysis task varied. However, architects were most prominent (see Figure 10) among the roles. We observed that unlike the other roles, testers were never reported to assume responsibility for the analysis on their own. Some of the respondents chose one role, while most (72.9%) selected at least two roles, which indicates a form of collaboration on the analysis task. In fact, it was explicitly reported that collaboration might happen between technical managers and the engineering or DevOps team. Also, collaboration between domain experts was reported by one respondent, which would be of great value for analyzing the conceptual level of interoperability. Regarding the size of the analysis team, only three respondents stated that this depended on the project type and size. However, 62.2% said it was performed by a small team (i.e., < 5 members) although the project size ranged from small to large. Unexpectedly, 66.6% of the respondents who stated that it was performed by exactly one person worked on large projects.

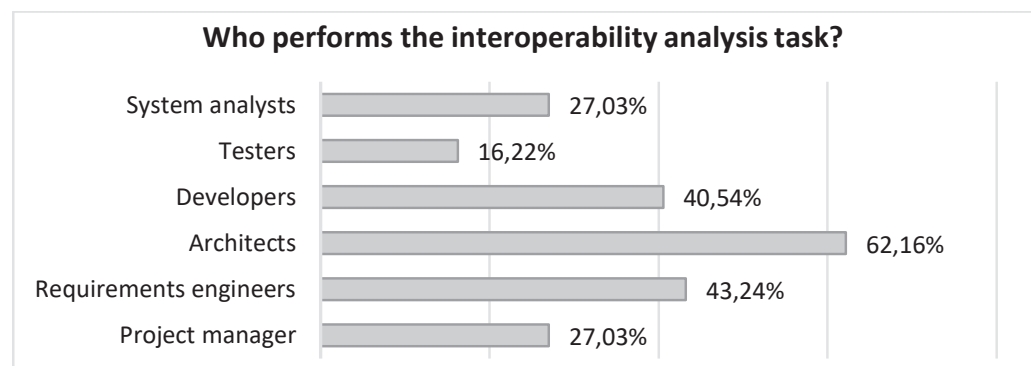


Figure 10

The roles responsible for performing interoperability analysis

*What?* Digging deeper, we asked the practitioners to specify the types of information they targeted during the interoperability analysis task. The answers showed that technical aspects were dominant. Specifically, 75.7% targeted communication constraints (e.g., networking protocols, message formats, etc.) and 64.9% targeted the technical syntax (e.g., argument order, data types, etc.). Fewer respondents (58.1% on average) reported being interested in semantic constraints (e.g., terminologies, goals, rationale, etc.) and behavior constraints (e.g., pre-/post-conditions, invariants, interaction protocols, control flow, etc.). Minor shares of attention (48.6% on average) were given to context, structure, and quality. As expected, the covered

information aspects were less for those who reported that interoperability analysis was performed by one person rather than by a team. For example, one respondent indicated that the analysis was performed by one developer and the only information targeted was the technical communication. Statistically, the significantly correlated factor to the information targeted during analysis was the responsible role (i.e., Spearman's  $\rho = 0.493$ ,  $p\text{-value} = 0.002$ ). For example, architects and developers were the main roles who targeted semantic and behavior information. Furthermore, we found a statistical significance in the correlation between the years of integration experience and the targeting of semantic information (i.e., Spearman's  $\rho = 0.333$ ,  $p\text{-value} = 0.008$ ).

*How?* To better assess the current situation, we asked about the input, process, and output documentation of the interoperability analysis. The survey confirmed our expectation that *API documentation was the main available input artifact* and source of information as stated by 78.4% of the respondents. Other available sources of information included high-level architecture, requirements specification, and source code. However, these apply to white-box integration (e.g., open source software projects) rather than black-box integration (e.g., commercial COTS). One respondent reported that contacting the team members of the external software unit was his source of information due to a lack of shared artifacts.

With respect to the *support used for performing the interoperability analysis*, we found that about 30% of the practitioners with knowledge about this issue had no support. Two respondents declared that they did the analysis in an ad-hoc manner, focusing on identifying the gaps between the integration requirements and the capabilities offered by the external software units. On the other hand, 13.5% used analysis models and frameworks, 10.8% followed guidelines, 8.1% performed systematic analysis, 8.1% used a template, and 5.4% had tool support. However, the respondents did not give any further information, details, or references for the reported support, with two exceptions. One added that the guidelines they followed had been developed internally and the other one added that they also had an internally defined process to follow. With regard to *documenting the analysis results*, we found that about 30% of the practitioners did not document the results of their interoperability analysis. This documentation status had a statistically significant correlation with integration experience in terms of the number of projects that the participants had worked on (i.e., Spearman's  $\rho = -0.438$ ,  $p\text{-value} = 0.009$ ). Consequently, there was a loss of information that would support decision traceability within a project as well as a loss of knowledge that would allow learning from experiences and cases across integration projects.

## Problems in Performing Conceptual Interoperability Analysis (Answering RQ2)

Here we present the evidence collected from practitioners on the relevance of the practical problems that are related to interoperability analysis. Then, we will shed light specifically on those problems that are related to the conceptual level of interoperability analysis. We collected data about perceived problems from the entire sample size ( $N = 64$ ). However, for the questions that depend on actual experience with interoperability analysis (i.e., RQ2.1 and RQ2.2), we differentiate between the results obtained from those who claimed to perform interoperability analysis (actual experiences) and those who did not (opinions).

**High cost of interoperability analysis.** In Figure 11, we depict the cost of interoperability analysis reported by the respondents who had knowledge about it ( $N = 43$ ). The largest share of the responses shows that interoperability analysis accounts for 10% to 30% of the total cost of integration projects. Moreover, a portion of the respondents ( $N = 9$ ) stated that the analysis would range between 30% and 50%. Obviously, such cost ranges are relatively high taking into account the other development activities included in an integration project (e.g., requirements analysis, design, implementation, testing, etc.). In one case, a respondent stated that it would even be worse and would reach 51% to 70% of the total cost. Remarkably, this respondent was one of those who reported performing the analysis during the implementation of the integration.

Few responses ( $N = 12$ ) indicated that the cost of interoperability analysis would be less than 10% of the total cost of integration projects. Most of these respondents had rather low integration experience (i.e., they had worked on 2 - 5 projects). Some respondents ( $N = 21$ ) did not provide an answer due to a lack of knowledge or traceability information about the cost of interoperability analysis. For example, one respondent stated in the comment box that they did not track the cost of the analysis independently, but rather accumulated it within the whole project cost. Another respondent reported that, in the context of software projects that were not dedicated to integration only, the costs for interoperability analysis varied depending on the size of the integration requirements compared to the total project requirements. No significant correlations were found between the cost of the analysis and the respondents' demographic features.

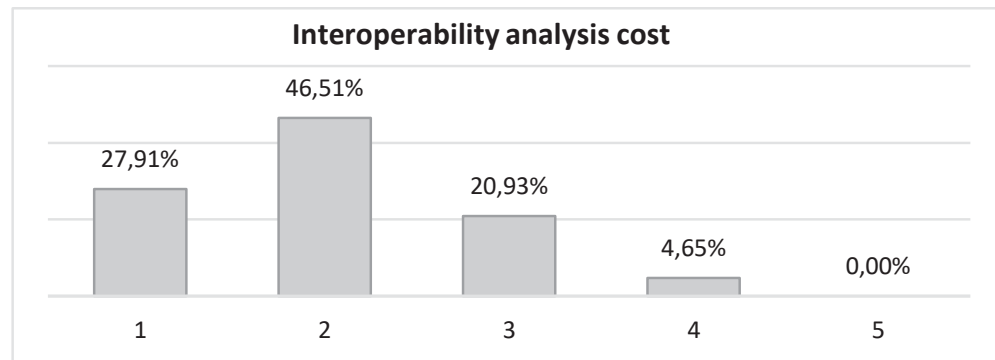


Figure 11

Cost for interoperability analysis reported by practitioners with knowledge

### Frequently undetected conceptual interoperability mismatches.

The survey respondents who had knowledge about the frequency of integration problems related to undetected conceptual mismatches ( $N = 59$ ) stated that this problem was most likely to happen. The majority (52.54%) stated it happened sometimes, while 30.51% said that it was a common issue. Only three respondents stated that unexpected conceptual mismatches were always happening. Very few ( $N = 7$ ) claimed that this problem was rare and none said that it was never a problem. Statistically, the responses show significant agreement regarding the existence of the problem of undetected conceptual mismatches (see Table 1).

Table 1

Frequency of problems related to undetected conceptual mismatches

	Median	Mean	Standard deviation	Test statistics <sup>b</sup>	
				Z	P
Conceptual mismatch frequency <sup>a</sup>	3	3.57	1.145	470	0.000***
<sup>a</sup> Response scale for frequency from 1 (Never) to 5 (Always) <sup>b</sup> One-sample Wilcoxon signed-rank test $H_0$ : Median (all respondents) = 3 (sometimes); * $p < 0.05$ ; ** $p < 0.01$ , *** $p < 0.001$					

### Expensive resolution for unexpected conceptual interoperability mismatches.

Those respondents who had knowledge about the resolution cost ( $N = 47$ ) reported rather high additional costs for resolving undetected mismatches. As seen in Figure 12, the majority of the respondents agreed that the total integration project cost increased by 10% to 30%. Furthermore, a considerable portion of them (more than a third) agreed that the added cost would even be 31% to 50% of the project cost. Note that the few respondents who said that conceptual mismatches were a rare cause of problems still agreed that they were expensive (e.g., one of them stated that it would cost > 70%).



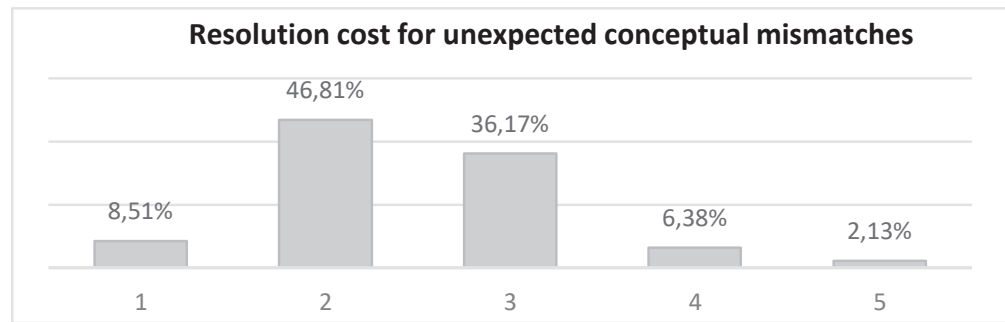


Figure 12 Additional project cost for resolving unexpected conceptual mismatches

### Problems Related to Practices of Interoperability Analysis (Answering RQ2.1)

The aim of this question was to reveal any significant difficulties faced in performing interoperability analysis with respect to current *practices*.

**Perceived need for better practical support for performing interoperability analysis.** We offered the survey participants a list of practice-related difficulties (D) that would impede performing interoperability analysis and asked them to select which ones they considered to be the main ones. This list included the following:

- D1:** Lack of focus on detecting "conceptual" mismatches compared to "technical" mismatches
- D2:** Lack of support for traceability between interoperability analysis activities and results (i.e., within a project and among projects)
- D3:** Lack of standard templates for consistent documentation of interoperability analysis results
- D4:** Lack of interoperability analysis guidelines and best practices for practitioners
- D5:** Undirected collection of information about external software units (i.e., no plan or predefined data elements)
- D6:** Posterior collection of information about the external software unit (i.e., reactive collection based on rising problems along the project)
- D7:** Manual effort in analyzing the description of external software units and in documenting the analysis results

According to the responses ( $N = 64$ ), D4, D7, and D1 had the highest agreement among practitioners as seen in Table 2. This provides evidence on the need for helping interoperability analysts and architects identify conceptual mismatches by providing practical guidelines and automation tools. Furthermore, D2 and D3 also had a considerable amount of agreement. Obviously, these two essential difficulties are related, as the ability to trace interoperability analysis results requires



documenting them. Accordingly, supporting practitioners with standard documentation templates would serve the aforementioned traceability need along with other benefits such as consistency and readability. Although D5 and D6 got the least shares of agreement, there were still practitioners who agreed on the importance to overcome them. This indicates that directed analysis with proactive preparation can enhance the analysis experience and the results for some analysts and architects.

Table 2 Perceived difficulties of interoperability analysis practices

Difficulty (D)		D1	D2	D3	D4	D5	D6	D7
Total agreement		25	24	23	26	17	9	26
Agreement percentage %		39.06	37.50	35.94	40.63	26.56	14.06	40.63
Test statistics <sup>a</sup>	$\chi^2$	.080	.961	3.025	6.723	.225	.337	.249
	p	.777	.327	.082	.010*	.635	.562	.618
Test statistics <sup>b</sup>	$\rho$	-.035	.123	.217	.324	.059	-.073	-.062
	p	.781	.335	.084	.009**	.641	.569	.624
<sup>a</sup> Pearson's chi-square ( $\chi^2$ ) test $H_0$ : Agreement percentage (respondents who performed interoperability analysis) = Agreement percentage (respondents who did not perform interoperability analysis) <sup>b</sup> Spearman's rho ( $\rho$ ) test $H_0$ : There is a correlation between the agreement percentage and the respondents' group (performed interoperability analysis or not); * $p < 0.05$ ; ** $p < 0.01$ , *** $p < 0.001$								

After a more thorough investigation, we found one statistically significant difference between the agreement percentages on D4 of the two groups of surveyed practitioners (i.e., those who performed interoperability analysis in their software integration projects and those who did not). In fact, this difference was also justified by the statistical significance of the correlation between the group type and the agreement on D4.

As visualized in Figure 13, there were some other percentage differences; however, they were not statistically significant. For example, D2 and D3 had more votes by practitioners inexperienced in performing the analysis task (21.12%, and 12.01%, respectively). Thus, we conclude that all reported difficulties are important for both groups. However, overcoming D4 would be of higher value for practitioners with no or low experience in interoperability analysis.

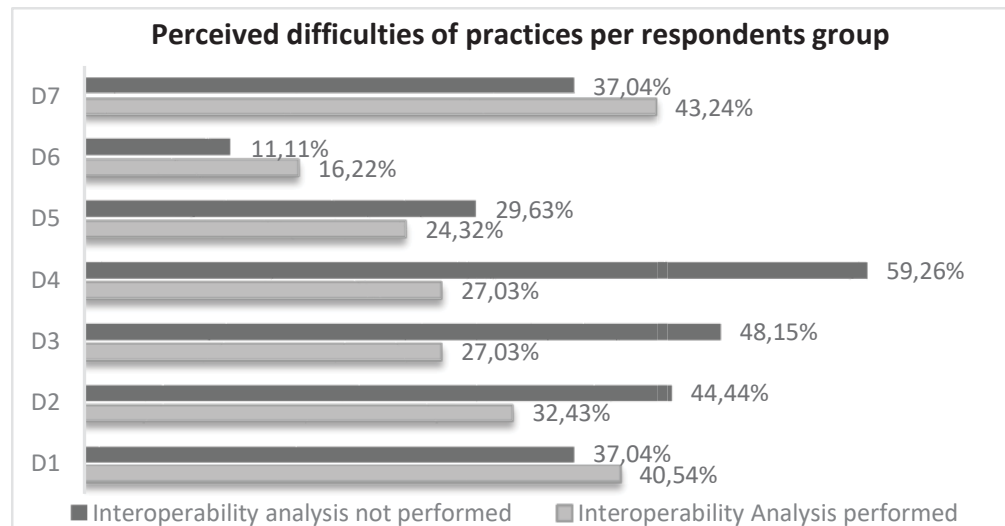


Figure 13

Distribution agreement on the difficulties of interoperability analysis practices for each group of respondents

### Problems related to the Input Artifacts of Interoperability Analysis (Answering RQ2.2)

The aim of this question was to reveal any significant difficulties faced in performing interoperability analysis with respect to current *input artifacts*. This includes the *content* and *presentation* of the input.

**Perceived insufficiency of shared information.** According to the respondents with knowledge about the current input artifacts of interoperability analysis ( $N = 59$ ), they mostly (37.50%) reported it to be “3: Not sufficient”. In other words, on the 5-point Likert scale, the main rating was 2: insufficient. Table 3 shows the median, mean, and standard deviation. It also presents the significance of the insufficiency problem of input artifacts. Furthermore, there was statistically significant agreement from both groups of respondents (i.e., those who performed interoperability analysis and those who did not) on the input insufficiency problem.

Table 3

Perceived sufficiency of the input artifacts of the interoperability analysis task

	Median	Mean	Std. Deviation	Test statistics <sup>b</sup>		Test statistics <sup>c</sup>	
				Z	P	U	P
Perceived sufficiency of input artifacts <sup>a</sup>	3	2.98	1.25	-4.76	0.000***	452	.497

<sup>a</sup> Response scale from 1 (not sufficient at all) to 5 (very sufficient)

<sup>b</sup> One-sample Wilcoxon signed-rank test  $H_0$ : median (all respondents) = 4

<sup>c</sup> Mann-Whitney (U) test  $H_0$ : median (respondents who performed interoperability analysis) = median (respondents who did not perform interoperability analysis); \*  $p < 0.05$ ; \*\*  $p < 0.01$ , \*\*\*  $p < 0.001$

**Perceived need for enhancing conceptual information content.**

The respondents ( $N = 64$ ) voted for what they perceived as required enhancements for the content of input artifacts used in the interoperability analysis task. We offered a list of interoperability-related content (C) and asked the respondents to select what they considered to be important for enhancing the content related to them. This list included the following:

**C1:** Communication constraints (e.g., networking protocols, message formats, etc.)

**C2:** Syntax constraints (e.g., argument order, data types, etc.)

**C3:** Semantic constraints (e.g., glossaries, goals, rationale, etc.)

**C4:** High-level architecture view (e.g., architecture style, patterns, etc.)

**C5:** Low-level design decisions (e.g., inheritance, synchronicity, concurrency, etc.)

**C6:** Behavior constraints (e.g., pre-/post-conditions, interaction protocols, control flows, etc.)

**C7:** Context constraints (e.g., stakeholders, use cases, etc.)

**C8:** Quality constraints (e.g., data precision, service performance, etc.)

Based on the responses ( $N = 64$ ), C4 got the biggest share of the practitioners' interest (see Table 4). This evidently indicates the serious need to enrich shared documents about interoperable software units with high-level architecture to improve the analysis. Next, C1 and C6 got substantial agreement, which gives them high priority, too. Note that C1 is a technical type of content, while C6 is conceptual. Afterwards, C3, C8, C2, and C5 got convergent large shares of the respondents' agreement. This shows there is awareness of the need to improve the quality, semantics, syntax, and low-level design information of interoperable units. Although C7 got the lowest share of agreement, 24 practitioners still agreed that it is important to be enhanced. These results denote potential improvement for interoperability analysis results when the content issues are resolved.

A more thorough investigation showed us that there was one statistically significant difference between the agreement percentages on C7 from the two respondent groups (i.e., those who performed interoperability analysis in their software integration projects and those who did not). In addition, there was a statistical significance of the correlation between the group type and the agreement on C7.

Table 4

Perceived need for enhancing the content of input artifacts for interoperability analysis

Content problem (C)		C1	C2	C3	C4	C5	C6	C7	C8
Total agreement		37	29	31	39	27	36	24	31
Percentage%		57.81	45.31	48.44	60.94	42.19	56.25	37.50	48.44
Test statistics <sup>a</sup>	$\chi^2$	.098	2.705	2.430	.568	.098	1.246	4.651	1.108
	p	.755	.100	.119	.451	.755	.264	.031*	.293
Test statistics <sup>b</sup>	$\rho$	-.039	-.206	-.195	-.094	.039	-.140	-.270	-.132
	p	.759	.103	.123	.459	.759	.272	.031*	.300
<sup>a</sup> Pearson's chi-square ( $\chi^2$ ) test $H_0$ : agreement percentage (respondents who performed interoperability analysis) = agreement percentage (respondents who did not perform interoperability analysis) <sup>b</sup> Spearman's rho ( $\rho$ ) test $H_0$ : There is a correlation between agreement and respondents' group (performed interoperability analysis or not); * $p < 0.05$ ; ** $p < 0.01$ , *** $p < 0.001$									

Figure 14 visualizes the percentage differences among more content types; however, they are not statistically significant. For example, C2 and C3 had more votes by practitioners experienced in performing the analysis task (almost 20% each). We conclude that all content items are important for both groups, but interoperability analysis experts perceived them to be more important (especially C7) compared to inexperienced respondents.

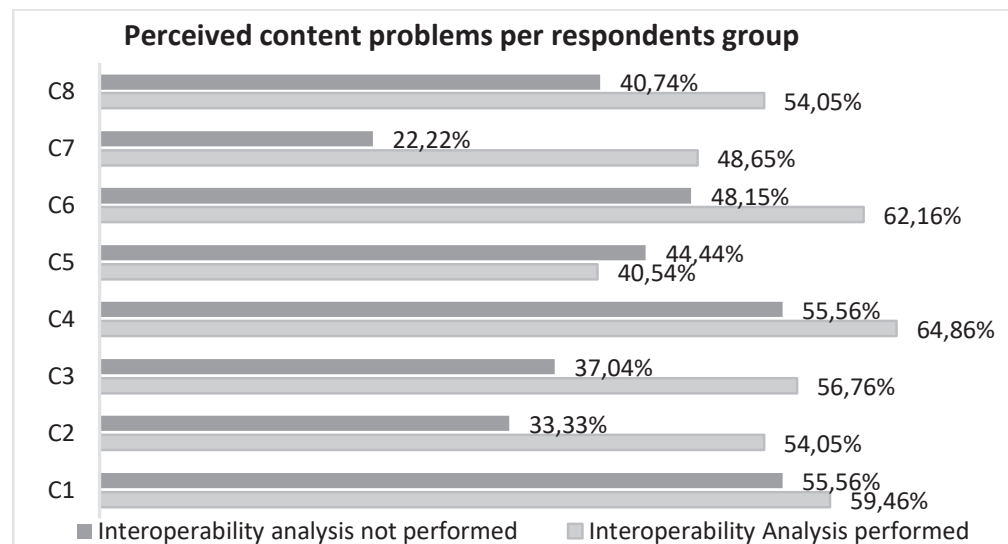


Figure 14

Distribution of agreement on perceived content problems of input artifacts for interoperability analysis per respondent group

**Perceived need for enhancing conceptual information presentation.** According to the respondents ( $N = 64$ ), some

enhancements are required for the presentation of the input artifacts used in the interoperability analysis task. Out of the list of presentation enhancements (P) that we suggested in the survey, the respondents voted for those they perceived as being the most important ones. This list included the following:

**P1:** Mixing conceptual and technical constraints without clear borders between them

**P2:** Unstructured verbose text

**P3:** Lack of easy-to-read process diagrams (e.g., flowcharts)

**P4:** Inconsistency in reporting constraints for the different data items and services

**P5:** Level of formality too low, which prevents potential automation of the analysis

Based on the responses ( $N = 64$ ), P1 and P3 got the highest agreement in equal amounts of the practitioners' interest (see Table 5). This points out a critical need to improve the structure of the information shared about interoperable software units in order to clearly differentiate between conceptual and technical information. Moreover, an abstract process view could enhance these documents. Afterwards, the respondents voiced considerable agreement on P4 and P2, which shows that structure and consistency in presenting content among equal elements improve the usefulness of shared artifacts. Although P5 got the lowest agreement by a large number of practitioners, 20 practitioners still agreed that it is an important presentation issue. Apparently, resolving all the presentation problems for interoperability analysis input artifacts on which the respondents agreed would add value to their users.

Table 5 Perceived need for enhancing the presentation of input artifacts for interoperability analysis

Presentation problem (P)		P1	P2	P3	P4	P5
Total agreement		30	25	30	27	20
Agreement percentage%		46.88	39.06	46.88	42.19	31.25
Test statistics <sup>a</sup>	$\chi^2$	.706	.055	4.854	.040	.616
	p	.401	.814	.028*	.841	.432
Test statistics <sup>b</sup>	$\rho$	-.105	.029	.275	-.025	-.098
	p	.409	.818	.028*	.844	.440
<sup>a</sup> Pearson's chi-square ( $\chi^2$ ) test $H_0$ : agreement percentage (respondents who performed interoperability analysis) = agreement percentage (respondents who did not perform interoperability analysis) <sup>b</sup> Spearman's rho ( $\rho$ ) test $H_0$ : There is a correlation between agreement and respondents' group (performed interoperability analysis or not); * $p < 0.05$ ; ** $p < 0.01$ , *** $p < 0.001$						

The statistical test results showed that there was one statistically significant difference between the agreement percentages on P3 of the two respondent groups (i.e., those who performed interoperability analysis in their software integration projects and those who did not). Moreover, a statistical significance of the correlation between the group type and the agreement on P3 was found. Figure 15 offers a visualization of all percentage differences for all presentation issues, but only P3 has statistical significance. Hence, we conclude that enhancing the presentation of the content of input artifacts for interoperability analysis would be of value for both experienced and inexperienced analysts. However, inexperienced ones would appreciate it more if the processes were abstracted in diagrams rather than in unstructured text.

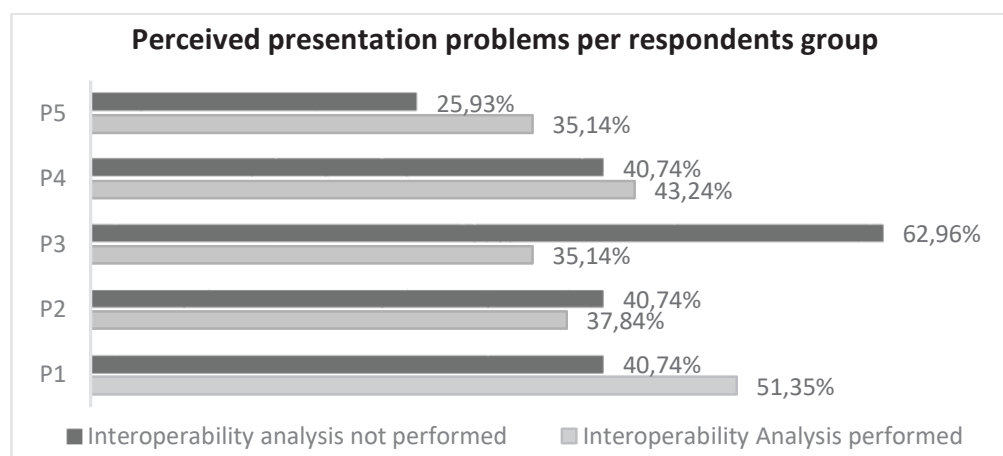


Figure 15 Distribution of agreement on perceived presentation problems of input artifacts for interoperability analysis per respondent group

### 3.2.3 Threats to Validity

In this section, we will present the internal and external threats to the validity of our survey study.

**Internal validity (content validity).** As described earlier in the survey design (see Section 3.2.1), we did multiple peer reviews with experts in software architecture, software engineering, and empirical research. Furthermore, we evaluated the survey in pilot studies to assess the understandability of the questionnaire.

**External validity (representative sample).** The final number of included responses was ( $N = 64$ ). These included software architects and engineers with integration experience from different organizations, industrial domains, and locations. Thus, we assume that our results are very likely representative for the state of the practice of interoperability analysis as of June 2016. However, for better generalization and observations over time, further surveys with a larger sample size are required.

**External Validity (completion rate).** As peer reviews considered the questionnaire to be too long, we shortened it to increase the completion rate for our questions. Also, to ensure that reliable responses would be collected that are based on genuine experience, we had conditional appearance of questions and we offered the option to answer “I don’t know” to questions referring to knowledge rather than opinions. As a result, we got completely answered questionnaires from all respondents.

### 3.3 Summary and Conclusion

In this chapter, we presented a consolidated description of the current state of the practice of interoperability analysis. We started by describing the as-is situation, which revealed that 30% of practitioners do not perform interoperability analysis in their integration projects. The main reasons behind this were found to be a *lack of knowledge* about how to perform it and a lack of awareness regarding its importance, which leads to prioritizing other tasks over it. This consequently increases the risk of unexpectedly facing conceptual mismatches late in an integration project. On the other hand, practitioners who performed interoperability analysis showed us that the current state of interoperability analysis was immature. More specifically, there was *no standard or systematic activities being followed* (e.g., some performed interoperability analysis during the implementation rather than before it) and there was *no comprehensive investigation to find interoperability information* during the analysis (e.g., very few of the practitioners with little experience targeted conceptual information).

Afterwards, we presented the evidence collected on the practical relevance of the problems addressed by this Ph.D. thesis. In particular, there was agreement on the *high cost of performing interoperability analysis*. Besides, there was significant agreement on the *frequency of unexpected conceptual interoperability mismatches*. This kind of mismatches was also reported to be *expensive to resolve*. Furthermore, we identified the exact difficulties related to both the practices and the input artifacts of the conceptual interoperability analysis task. For example, a significant difficulty reported in the context of practices was the *lack of guidelines and best practices for interoperability analysis* that practitioners could use. Also, *input artifacts were considered significantly insufficient for the analysis task, especially on the conceptual level* (e.g., context constraints).

To sum up, this chapter confirms the practical problems we stated in Section 1.2 and paves the way towards presenting our solution ideas and improvement hypotheses in the following chapters.



---

## 4 State of the Art

### 4.1 Introduction

As we described in the introduction (Chapter 1) and as confirmed by practitioners in the state-of-the-practice study (Chapter 1), interoperability analysis is not a trivial task and conceptual interoperability mismatches are frequent and expensive to resolve. This chapter presents related work in the literature as follows:

- In Section 4.2, we will present our *state-of-the-art scoping study*, which we used to gain in-depth information about the types of interoperability-related architectural mismatches and their proposed solutions in the literature. Note that this section has been published by the author of this thesis in [ATR14].
- In Section 4.3 and Section 4.4, we will describe the work related to the focus of this thesis, which includes conceptual interoperability foundations, respectively analysis approaches for software units.
- In Section 4.5, we will summarize the limitations of existing work on conceptual interoperability foundations and analysis approaches, and describe our research directions.

### 4.2 Scoping Study on the State of the Art

In this section, we start by describing the research methodology of our scoping study (Subsection 4.2.1), then we will present the results and discuss them (Subsection 4.2.2). Finally, we will present the threats of the validity of the survey study (Subsection 4.2.3).

#### 4.2.1 Research Methodology

In this study, our aim was to systematically investigate the nature and extent of software architecture research regarding interoperability problems and before-release solutions in information systems. The purpose was to collate, summarize, and disseminate research findings, and identify research gaps. Therefore, we performed a scoping study following the process proposed by Petersen et al. [PFMM08] along with a data extraction form. Unlike systematic literature reviews [Kit04], we aimed at a broad analysis of the literature rather than an in-depth analysis with quality assessment for selected papers. All materials of this study are available at the scoping study web page [Abu14c].

## Goal and Research Questions

The **goal** of this scoping study was *to identify architectural problems and before-release solutions of interoperability in the context of ISs from the viewpoint of researchers and software engineers*. This goal was translated into the following research questions:

**RQ1:** Which levels of interoperability are handled in the literature with architectural solutions?

This question intends to determine the extent to which architecture research addresses interoperability in terms of the levels of the LCIM model.

**RQ2:** What are the architectural problems faced when building interoperability among ISs?

This question intends to identify the issues and key drivers that need to be considered when designing ISs to support the desired interoperability property.

**RQ3:** What are the architectural solutions for handling the identified problems?

This question intends to identify the architectural design decisions and activities proposed in the literature for handling the identified interoperability issues.

**RQ4:** How are architectural solutions for interoperability evaluated?

This question intends to explore the evidence provided about the quality of the identified solutions in terms of the evaluation method used.

**RQ4.1:** What interoperability measures are used to evaluate the architectural solutions?

This question intends to investigate interoperability metrics used as part of the evaluation.

## Data Sources and Search Strategy

In accordance with the recommendations of Dybå et al. [DKJ05], we looked for published papers in journals and conference proceedings of the following databases: IEEE Xplore, ACM Digital Library, Springer Digital Library, Google Scholar, and Science Direct. Having the data sources selected, we performed trial searches using various combination of search terms derived from our research questions. Based on the results we defined our search terms as: (T1) Interoperability AND Architecture, (T2) Interoperation AND Architecture, (T3) Interoperability AND Architectural Design, and (T4) Interoperation AND Architectural Design. The search process was carried as follows:

**Stage 1:** Pilot search the databases using the defined terms T1 to T4 separately and then combined with the “OR” operation to remove duplicates. It was applied to the titles and abstracts (4128 studies).

**Stage 2:** As abstracts from stage 1 showed irrelevance to the research questions, the database search was refined to be applied on titles only (246 studies).

**Stage 3:** Inclusion/exclusion criteria, described next, were applied to the 246 studies based on keywords, abstracts, and conclusions (22 studies).

### Inclusion and Exclusion Criteria

A study got included if it met all the inclusion criteria (I) and none of the exclusion criteria (E); otherwise it got excluded. These criteria were:

**I1:** Studies with the main focus on interoperability problems and architectural solutions in ISs.

**I2:** Studies with architectural solutions to support interoperability before release.

**E1:** Studies written in languages other than English.

**E2:** Gray studies with an unclear peer-review process (e.g., technical reports, short papers, keynotes, abstracts, etc.).

**E3:** Secondary studies about interoperability problems and solutions (i.e., work related to this research).

**E4:** Studies with minor interest in architectural aspects regarding interoperability.

**E5:** Studies proposing solutions for specific projects under restricted settings and conditions that cannot be generalized to ISs.

Two researchers separately applied the criteria to the studies. If discrepancies were found between the results, discussion sessions were held and consensus-based decisions were made. The search was conducted in November 2013 and had no timeframe limitations in order to allow getting a broader coverage of studies related to our research questions. Note that we did not contact the authors of included studies to seek unpublished evaluation or other related research.

### Data Extraction Strategy

One researcher extracted the data from the 22 included studies and another researcher checked it against the studies to ensure completeness and correctness of the extraction (see data extraction form in Appendix B).

## Data Analysis

Qualitative data analysis was performed using an initial coding scheme in a tabular form including interoperability problems, interoperability levels, architectural solutions, architectural components, and evaluation types. The coding scheme provided a definition of concepts, categories, and criteria, which guided the translation of raw data into descriptions that answer the research questions.

### 4.2.2 Results and Discussion

#### Demographic Overview

The identified 22 primary studies were from a diversity of application domains (e.g., eGovernment, eCommerce, eLearning, geography, military, and biomedical systems). As seen in Figure 16, a small increase in the number of studies on interoperability can be observed after 2004.

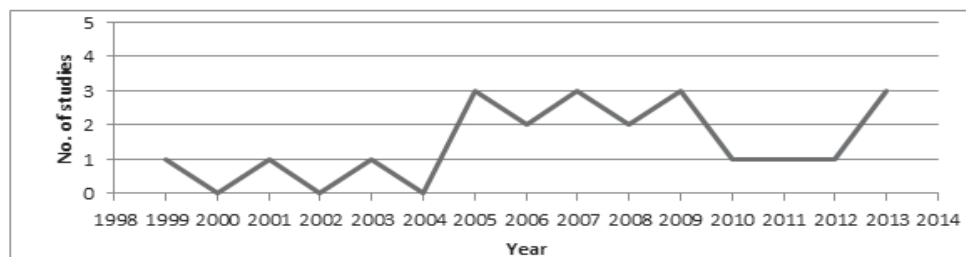


Figure 16 Year-wise distribution of selected studies

Studies were conducted in academic and industrial environments with 10 of 22, 45%, of the studies were performed in collaboration between the two. Almost all studies (21 of 22, 95%) were published at conferences, while one study appeared in a journal. Remarkably, there is no dominating conference publishing many studies on interoperability-related architectural problems and solutions, i.e., each conference published one study, except for one which published two studies. One conference named “Distributed Applications and Interoperable Systems” was dedicated to software interoperability.

#### Interoperability-related Architectural Problems and Solutions

RQ1: Which levels of interoperability are handled in the literature with architectural solutions?

To determine the interoperability concerns of each study, we analyzed its keywords F5, its objectives F6, its problem description F8, and its solution advantages F15. Afterwards, we compared these concerns to the description of interoperability levels according to the LCIM model

[Tur05]. Figure 17 illustrates the distribution of the handled levels of interoperability over the included studies. Some studies addressed more than one level, e.g., S3 addressed both the semantic and pragmatic levels. Note that the semantic level has the largest share of the studies' focus with a growing interest over the years, while the pragmatic level has a low share and disappeared after 2007. Syntactic and technical levels have convergent shares. In recent years, especially in 2012 and 2013, the technical level has caught the attention of inter-Cloud systems researchers (S18 and S22). Both the dynamic and conceptual levels got no share in the studies at all.

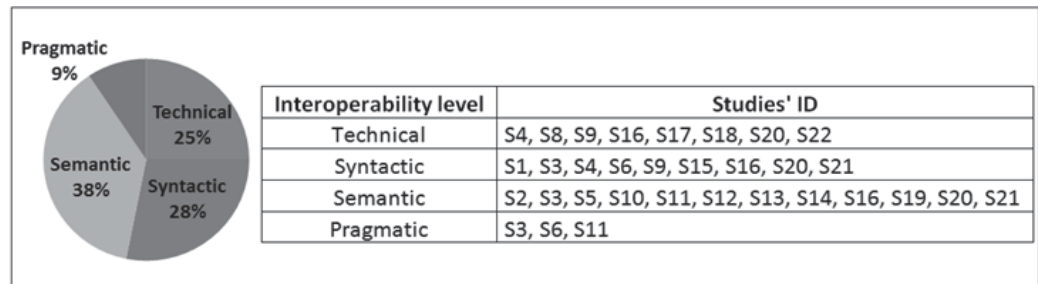


Figure 17 Interoperability-level distribution over selected studies

RQ2: What are the architectural problems faced when building interoperability among ISs?

For each study, we examined the interoperability problem it addresses from the problem description F8. Then we mapped each problem to the corresponding level of the LCIM model that shares and includes its concerns. Synthesizing the problems of all studies, we identified eight distinct architectural issues, with seven of them being related to LCIM levels as seen in Table 6.

Table 6 Overview of LCIM levels with the problems and solutions identified in the studies

Interoperability Level	Problem ID	Solution ID	Study ID
Technical	P3	Sol5	S4
		Sol7	S9, S16, S17, S18, S20, S22
		Sol10	S8
Syntactical	P2	Sol5	S6
		Sol7	S15, S16, S20
		Sol8	S3
		Sol9	S9
	P5	Sol6	S4
Semantic	P1	Sol1	S14, S21
		Sol2	S3, S5, S10, S12, S13
		Sol4	S11
		Sol3	S16, S19, S20
	P6	Sol11	S2
Pragmatic	P4	Sol2	S3, S11
		Sol5	S6

n/a	P8	Sol12	S7
-----	----	-------	----

**P1: Semantic heterogeneity of data** is the most common problem (occurrence number (N) = 11). It concerns architects designing interoperable systems that correctly interpret the meaning of data elements being exchanged among them. For example, the authors of (S11) investigated designing interoperability among different GIS systems and stated that it was a challenge due to the growing number of heterogeneous spatial data sources with semantic differences.

**P2: Syntactical heterogeneity of data** has been reported frequently (N = 7). It requires architects to take into account the differences in data types, formats, and modeling languages of interoperating systems. For instance, in (S6), Carvalho et al. stated that exchanging geographic data among different layers of GIS required resolving its different representations first.

**P3: Heterogeneity of communication protocols, platforms, and technical standards** is considered a serious architectural problem (N = 7). It is essential for interoperability to make design decisions that enable the system to establish communication with systems having different technical properties. In (S9), Rabhi observed that developing cooperation among financial market systems required enormous effort due to their variant technologies, communication interfaces, and network protocols.

**P4: Heterogeneity of data context** has been reported as a problem in the context of financial and GIS systems (N=3). It is important for architects to reflect on the context in which the designed system's functionalities and data can be used to assure meaningful interoperability. For example, (S11) describe possible context heterogeneity in interpreting a domain value of a CropType attribute in the designed system. While in one country it could be "Wheat", in another one it might be "Corn".

Other stated problems include **P5: Heterogeneity of method signatures**; **P6: Misunderstanding of the semantic meaning of interoperability**; **P7: Redundancy of data**; and **P8: Inadequacy of architecture framework for supporting interoperability**.

RQ3: What are the architectural solutions for handling the identified problems?

For each study, we studied the interoperability solution it proposed from the architectural solution F10, its components F11, and the technology used F12. Then we mapped the solutions to the identified problems in RQ2 (see Table 6).

**Sol1: Standards** address semantic interoperability problems, e.g., (S21) unambiguous semantic metadata is achieved through a standard-based metadata repository, which provides a formal description of the



meaning of data types used in classes and attributes of data systems. Also, (S14) proposes standard-based modeling for processes and data between collaborating organizations.

**Sol2: Ontologies** solve semantic and context-related interoperability problems. For example, (S13) proposes an ontology-based blackboard architecture to facilitate user retrieval of the correct service offered by an eGovernment system based on the user's needs with less effort by modeling the basic concepts of services from a user perspective.

**Sol3: Semantic mediators** align semantically related concepts. We identified three identified forms of mediators: formal-methods-based mediators align the behavior of systems using their LTS models (S16), thesaurus-based mediators mediate concepts using knowledge structures simpler than ontologies (S19), and standard-based mediators facilitate standardized information exchange and orchestration (S20).

**Sol4: Wrappers** encapsulate local data sources in an export schema comprising the main concepts of the real-world entities. As described in (S11), a wrapper receives queries from interoperating systems and translates them into a local form to enable processing them and to retrieve the required information from the local system.

**Sol5: Adaptors** embed the connection state and the logic into one or more external systems, e.g., they can encapsulate a telnet-based connection into a remote Unix host (S4). Also, (S6) proposes using adaptor components to transform data among the interfaces of different GIS devices.

**Sol6: Facets** provide different implementations for a standard interface of an action. Hence, the action can be invoked by different system types through its corresponding facet. In (S4), these facets are automatically generated by specialized tools.

**Sol7: Middleware** handles heterogeneities in communication protocols and data formats. In (S16), Bennaceur et al. present how an on-the-fly middleware component dynamically resolved the heterogeneity of data formats in messages being exchanged between distributed systems.

**Sol8: External data models** are concerned with representing all sources of data that the system may exchange with other interoperating systems. In (S3), the authors give examples of external data, including relational database sources, XML sources, HTML web wrapper sources, and computational procedures modeled as relations.

**Sol9: Internet data formats** are proposed for use on the data level of distributed systems to ensure wide applicability of the associated components (S9), i.e., using XML and its variants like FIXML with CORBA for handling the communication.

**Sol10: Technical reference models** provide guidance for expeditiously selecting technical standards using a common



vocabulary. According to (S8), this fosters interoperability by providing appropriate system standard profiles.

**Sol11: Semantic reference models** provide guidance for developing semantic interoperability capabilities in systems by fulfilling a set of semantic requirements. In (S2), these requirements are categorized as policy and governance, organization, and technology.

**Sol12: Enterprise architecture frameworks** provide a systematic blueprint to build interoperability among enterprise IS. In (S7), the identified framework resolves weaknesses determined comparatively in legacy enterprise architecture frameworks.

**Sol13: Central repositories** allow cooperative sharing of information among systems. For example, (S1) proposes using a central repository for applications installed on a phone device to enable sharing of resources and context data among them.

A recurring theme we observed in the findings is basing the identified solutions on the service-oriented architecture style (SOA) and implementing it with web service technology. This theme is reported in nine studies (S5, S6, S10, S12, S13, S14, S17, S18, and S22). Also, we found that the different solutions are not associated with any particular application domain or research field, i.e., they are applicable in general ISs.

## Evidence on the quality of existing solutions

RQ4: How are architectural solutions for interoperability evaluated?

As seen in Figure 18, 8 of the 22 identified studies did not provide any evaluation of their proposed solutions. Because of the lack of empirical evidence regarding the quality of the identified solutions, it was not possible to determine their effectiveness.

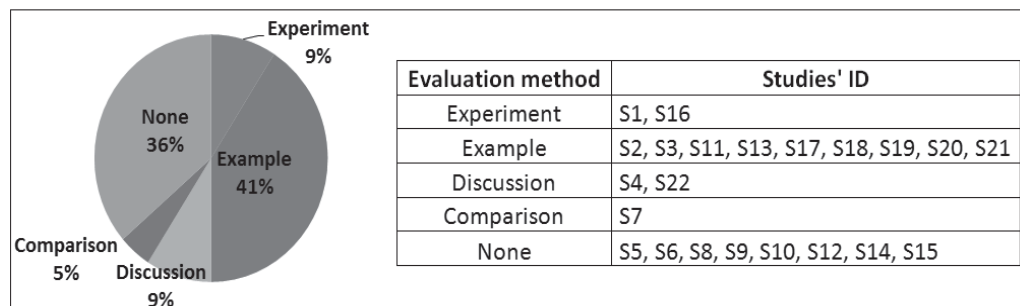


Figure 18 Distribution of evaluation method over selected studies

#### RQ4.1: What interoperability measures are used to evaluate the architectural solutions?

None of the studies included in this scoping study used interoperability metrics to appraise whether it has been achieved in the systems. Studies with empirical evaluation focused only on assessing performance in terms of query execution time (S1), feasibility in terms of understandability and ease of development of the concepts (S7), and validity in terms of overcoming the interaction and application heterogeneity (S16). It is noteworthy that neither (S7) nor (S16) was accompanied by quantitative data.

Studies with toy examples described their solutions' benefit in light of different interests: (S2) argues that their solution provides a good basis for evaluating the maturity level of the semantic interoperability capability of agencies; (S3) shows that their solution allows context mediation without the rigidity imposed by changing the original context models; (S13) explains how end-users are provided with appropriate interfaces for published services; (S17) illustrates how groupware requirements diversity could be more easily fulfilled by controlling concurrency access to shared documents; (S19) clarifies the feasibility of achieving semantic interoperability with simpler structures rather than ontologies; (S20) claims gains in adaptivity, flexibility, and security; and (S21) presents the feasibility of making data semantically interoperable using ontologies and standards.

Studies with no evidence claim to achieve autonomy, flexibility, and extensibility (S11) and to allow optimized provisioning of computing, storage, and networking resources (S18). No reflection of such claims was found in the given examples.

## Discussion

The study results reveal that architectural problems and solutions related to software interoperability have been studied especially on the syntactic and semantic levels over the last fifteen years. However, only a few studies proposing solutions to the higher LCIM levels have been published. Also, the results demonstrate the low evidence level of the studies as the quality of the proposed solutions was not properly evaluated in the papers included in our scoping study. Consequently, we want to draw attention to the following issues that should be overcome to advance the research area:

**Architectural basis for higher levels of interoperability.** This scoping study demonstrates that research efforts have not addressed the dynamic and conceptual levels of interoperability yet. In fact, standalone architectural solutions are not adequate by themselves to comprehensively solve the aforementioned high levels. That is, a broader interdisciplinary view is needed, involving organizational,

managerial, and advanced technical decisions made, e.g., with the help of artificial intelligence methods and technologies. Accomplishing this interdisciplinary solution effectively requires the support of a mature architectural basis. For example, unaligned models of business processes would be handled better if constraints such as the ambiguity of dynamically exchanged business data had already been handled using mature architectural solutions. Accordingly, we emphasize the importance of reaching a reasonable degree of architectural maturity in backing interoperability on its higher levels. As indicated by [GCN09], achieving a clear interoperability maturity level determines the strengths and weaknesses of systems in terms of their likelihood to interoperate, and hence defines the improvement priorities on the path towards successful interoperability.

**Prior architectural solutions to support interoperability before release.** The results show that researchers tend to deal with interoperability problems after facing them, i.e., they propose expensive posterior solutions [GMM07]. In contrast, adopting prior architectural solutions can save time and effort; e.g., designing and implementing an interface adaptor for a system under construction is less expensive than modifying a released system and integrating it with new components [GMM07]. Therefore, it is necessary to push the wheels of research in the direction of prior architectural solutions for interoperability.

**Architectural practices to support software interoperability.** In this study, only architectural design decisions were found in the area of software architecture. However, software architecture includes other activities that affect system characteristics, like architectural analysis, synthesis, evaluation, and documentation [HKN+05]. It is thus of significant importance to direct such activities towards improving the interoperability potential of ISs and facilitating its tasks. For instance, it would be useful to have studies about best practices for evaluating design patterns with regard to interoperability. Also, studies about architecture documentation activities that introduce specialized interoperability views could be helpful in analysis phases. Hence, research on architectural activities supporting interoperability is required.

**Empirical evidence on the quality of proposed solutions.** Based on our collected data, the majority of the identified architectural solutions have not been associated with reliable validation. This can lead to difficulties for practitioners to properly adopt interoperability solutions and to systematically enhance them in future works. Thus, it is important to provide trustworthy evidence such as empirical evaluations to increase the reliability of a solution and encourage its adoption. Such evaluation should analyze a solution with respect to its achieved interoperability level, its costs, and any other claimed benefits. The experience reported in the field of evidence-based software engineering explains the necessity of empirical evaluation to enable fast

adoption of good practices, improve product quality, and minimize project failures [DKJ05].

**Comparisons among interoperability-related architectural solutions.** The results show that the identified interoperability-related architectural solutions had not been compared to the solutions already existing in the literature. This is absolutely acceptable if solutions aim at solving interoperability problems that have not been addressed before. However, proper justification for the preference of adopting a new solution over others addressing the same problem would be needed. Specifically, we call for comparing the experimental results of new solutions with results obtained from previous ones. A similar recommendation has been proposed by Aleti et al. [ABG+13] in the context of building new software architecture optimization methods. Moreover, it would be of additional help if trade-offs of the solutions were declared, too.

**Interoperability metrics for assessing solutions.** The included studies are inconsistent in estimating the benefits of their solutions, i.e., they differ in both the qualities they assessed and the metrics they used. This lack of consistency impedes comparing the solutions and thus we were unable to infer the architectural characteristics that influence the interoperability property of systems. Another issue is that some studies measured interoperability using indirect metrics that have an unclear relationship with interoperability, e.g., autonomy, resource provisioning, security, and concurrency. Hence, the reporting bias represented in both inconsistency and indirectness should be overcome by using valid and reliable measures of interoperability. These measures include interoperability models like the Levels of Information Systems Interoperability (LISI) model [oDCIWG98], the Operational Interoperability Model (OIM) [CJ99], the LCIM [Tur05], the System of Systems Interoperability (SOSI) model [MLM+04], and others. Using these interoperability models would be a good basis for reporting the results of previously discussed empirical evidence and for making comparisons on the quality of interoperability solutions. However, it would be of even greater benefit to come up with metrics that can precisely quantify system interoperability and clearly draw the lines between the semantic, pragmatic, and conceptual levels.

By combining empirical evaluation, consistency in reporting the results, and directness in assessing the achieved interoperability in interoperability solutions, the strength of evidence for these solutions would definitely improve. Thus, estimations of the effectiveness and interoperability achieved when adopting these solutions would be more certain and trustworthy.

**Reference rules for selecting appropriate interoperability-related architectural solutions.** Currently, various interoperability-related architectural solutions have been identified, some of which address

similar problems. Therefore, it is important to provide guiding rules that define interoperability problems and assign them to their most suitable architectural solutions. For example, it would be a valuable assistance for junior interoperability architects facing a semantic data heterogeneity problem to have precise directions on how to choose from alternative solutions such as ontology-based, standards-based, and thesaurus-based mediations. Designers of such rules certainly need to carefully take into account the different factors that may influence the effectiveness of adopting a specific solution. These factors include available resources, modularity and dependency of system components, targeted interoperability level, system domain, project size, developers' experience, etc.

**Tool support for interoperability.** Another useful support for practitioners designing and building interoperability would be to aid them with software tools that can automatically identify potential interoperability problems between two systems from their architectural models. Such tools would be even more helpful if they were to also suggest plausible architectural solutions for the detected problems using the aforesaid guidelines. For example, this could be implemented as a plug-in to an existing software architecture modeling language (e.g., UML) that provides an interoperability view, reports architectural mismatches, and supports resolving these mismatches.

#### 4.2.3 Threats to Validity

**Researcher bias.** (1) To prevent bias in the conducting of this study, the selection criteria and the data extraction protocol were derived from the research questions and reviewed by an independent researcher. For the same purpose, the study selection was performed by two researchers. (2) To ensure correct inference in extracting data from studies with poor or insufficient description, data extraction was performed by one researcher and reviewed by another with discussions as needed. (3) To increase the confidence about the outcome of interpreting the qualitative data, the analysis results were reviewed and discussed until agreement was achieved between the two researchers. This was important in cases where interoperability was described using different or no models. (4) To ensure transparency and replicability of the study, the data and the results of each step were documented.

**Publication bias.** Although we performed our search in large electronic databases, we did not contact any authors to identify unpublished evaluation or other related research. Also, even though the search terms were derived from the research questions, software engineering keywords are not standardized. Consequently, relevant studies might be missed due to our choice of search terms. For these reasons, we do not claim to generalize the results for the whole research field.



However, this research covered a significant part of the literature and provided valid results.

### 4.3 Conceptual Interoperability Foundations

In this section, we will describe the related work that lays out the current basis and characterizations for conceptual software interoperability, from which interoperability-related activities and artifacts are derived. Although interoperability is a software property that enables actual reuse, these two terms (i.e., interoperability and reuse) have been used interchangeably in practice. Hence, we cover related work using both terms as keywords.

**Interoperability standards.** There is a wide range of standards for software interoperability and integration. A given standard only specifies some aspects of interoperability, which are mainly technical rather than semantic. For example, to allow exchange of data between information systems in the healthcare domain, standards have been proposed for communication, such as Health Level Seven (HL7) and Digital Imaging and Communications in Medicine (DICOM); for terminology, such as ICPC-2 and SNOMED CT; for documentation, such as IEEE 1073 Point of Care Medical Device Communication; for management, such as HL7 Clinical Context Management (CCOW); and many others [B+10]. However, all these standards lack the ability to achieve semantic interoperability. Also, standards may require domain-specific knowledge to use them. For example, in health information systems, many integration standards are based on generic technology standards, but they still require some healthcare know-how [MT08].

**Interoperability and reuse models.** Multiple classification models have been built for defining and organizing interoperability levels in software systems. These models help to define the compatibility level between systems and the amount of effort required to enable them to work together. For example, Vallecillo et al. [VTH99] present a classification for the interoperability of object-oriented software components into signature, configurations, semantics, interaction constraints, and quality. Also, Putman and Hybertson [VHT00] propose an interaction framework for object interoperability based on software architecture views. This framework supports interoperability between distributed systems and includes views for object relationships, interfaces, binding connectors, cross-domain interceptors, and behavioral semantics of the interaction. Independent of the programming type, interoperability models exist for information systems, such as the Levels of ISs Interoperability (LISI) [oDCIWG98] and the NC3TA Reference Model for Interoperability (NMI) [Pow08]. While, these two models are more focused on the technical aspects, a well-known and widely used model of interoperability is the Levels of Conceptual Interoperability Model (LCIM) [Tur05]. The LCIM focuses

on conceptual interoperability of information systems from the perspective of their data sharing capabilities. It encompasses seven levels of interoperability (i.e., No Interoperability, Technical, Syntactic, Semantic, Pragmatic, Dynamic, and Conceptual) with no specific attributes described under them. Another important model, which we consider the main model related to our work in this thesis, is the Reuse Model proposed by Basili and Rombach [BR91]. This model allows capturing reuse candidates and requirements with a set of characteristics that include dimensions for the reuse object (i.e., name, function, use, type, granularity, and representation), its interface (i.e., input, output, and dependencies), and its context (i.e., application domain, solution domain, and object quality). Other dynamic aspects that describe the behavior of reusable objects are not covered by this model.

**Interoperability mismatch classifications.** Researchers have proposed various classifications for component interoperability errors and mismatches. For example, a classification schema structuring interaction incompatibilities has been proposed by [YTB99], which includes a dimension for syntactic and semantic aspects and a dimension for the system and the environment. Also, [BOR04] provides four models. The first is based on a linguistic classification scheme for errors (i.e., including syntax, static semantic, and dynamic semantic). The second is based on a Hierarchical Interface Model [BJPW99] (i.e., contracts are classified into syntactic, behavior, synchronization, and quality). The third is based on enhancing the interface classification in two dimensions (i.e., functional and nonfunctional with interface granularity). The fourth model is based on the Unified Software Component Specification Framework (UnSCom) [Ove04]. This UnSCom was proposed for specifying components from different development perspectives with an orthogonal distinction among three design views (i.e., static, operational, and dynamic). In addition, some works deal with classifying architectural mismatches, such as [GAO95], [Sha95], and [GAACB95].

Other proposed models and classifications of interoperability cover broad aspects that are beyond the focus of this Ph.D. work. They include models and classifications for enterprise interoperability, such as [Che06] (which proposed a three-dimensional enterprise interoperability model that inspired us to build a dedicated dimensional model for software interoperability) and [GCN09], heterogeneity classifications that include middleware and network layers [BPGG11], and classification of components based on enumerated and faceted classification schemes for library search purposes [PD91].



## 4.4 Conceptual Interoperability Analysis

In this section, we will explore related work in the literature concerning current methods and approaches for analyzing interoperability between two software units. As in the previous section, we will cover related work using both interoperability and reuse as keywords.

### 4.4.1 Identification of Conceptual Interoperability Constraints

Extracting conceptual constraints in black-box interoperation

**Mining API documentation.** A number of static approaches have been proposed to mine limited types of interoperability constraints from API documentation using NLP and rule-based techniques. For example, Wu et al. [WWL+13] infer dependency constraints of parameters, Pandita et al. [PXZ+12] infer pre- and post-conditions of methods, and Zhong et al. [ZZXM09] infer resource specifications. Dekel and Herbsleb [DH09] extracted method constraints and pushed them into programming editors.

**Mining software executions.** There are also dynamic approaches for extracting software constraints from the execution. By running test suites, Nimmer and Ernst [NE02] infer software invariants, while Gabel and Su [GS12] infer temporal constraints. Gao et al. [GWZH14] infer data preconditions from API signatures, error messages, and testing results. Betrolino et al. [BIPT09] learn a behavioral service protocol by observing its execution. The accuracy of the results in these approaches depends in part on the quality and completeness of the test cases. The analyzed executions must also fully characterize all possible executions of the service in order to be credible.

**In-house mining of UML diagrams.** There are a few studies that propose tools to support interoperability analysis. Integration Studio (iStudio) [Bhu07] automates the interoperability assessment of COTS-based architectures and recommends possible resolutions for mismatches, while iStudio depends on a completely manual specification for the architectural interfaces. Ullberg et al. [UFBJ10] propose a tool for enterprise architecture models that supports specifying assessment theories in Pi-OCL to be used in interoperability analysis. Buschle's tool [BJS13] focuses on supporting decision-making regarding information technology in enterprise architecture models by analyzing many properties including interoperability.

On a broader scope, other works propose retrieving information using machine intelligence to assist software architects in different tasks. Anvaari and Zimmermann [AZ14] retrieved architectural knowledge from documents for architectural guidance purposes. Figueiredo et al. [FDRR12] and Lopez et al. [LCAC12] searched for architectural

knowledge in emails, meeting notes, and wikis for proper documentation purposes. Although, these are important achievements, they do not meet our goal of assisting architects in interoperability analysis tasks.

#### 4.4.2 Identification of Software Mismatches in Black-Box Contexts

**Conformance checking.** This method has an active research field with techniques for enforcing design-by-contract [Mey92]. Many studies, such as [CD00], [DW98], [Kin99], and [Han99], propose preparing technical contracts and interfaces for software units to assess their conformance with units. Conformance checking techniques allow function authors to formally specify their methods' pre- and post-conditions, and object invariants for performing static [RB10] or dynamic [HBH+10] analysis. Some works like Bastide's [VHT00] propose formal specifications for the behavior of interoperating objects by using Petri nets as an example. On the same path Canal et al. [VTH99] propose using an extension for interface description languages (IDL) that uses a subset of  $\pi$  calculus for describing objects' service protocols. [TN99] proposes presenting software features using special component description languages (CDL) instead of IDL, which specify technical contracts (i.e., invariants, pre- and post-conditions) along with context dependency and component relationships. Similarly, Gaspari et al. [VTH99], propose using a Unified Problem-solving Method description language (UPML) for specifying reusable components in a way that an intelligent broker can semi-automatically select a proper reusable library. Also, Ruiz et al. [VHT00] propose tackling object interoperability by using certificates that were granted to components after passing specific tests and then saved in a repository of certificates. Other technology-specific works focus on creating dynamic behavioral interfaces of software units. For example, [Mik99] proposes mathematical representations for object-oriented client-server interactions. Another type of common technical interfaces are network interfaces, such as the WSDL documents published for web services. Although these works are useful for detecting conformance violations automatically, most of them are manual-based solutions and focus on technical constraints rather than conceptual ones. For instance, the contracts proposed in [RB10] were designed to check only null values, value range, and object size.

**Reuse and COTS analysis.** Several approaches have been proposed for finding component mismatches, typically with the help of general rules. For example, Bhuta [Bhu07] proposes creating component definitions for technical and architectural assumptions manually, then applying mismatch detection rules to these definitions. Abdullah [AA96], Gacek [Gac98], and Egyed et al. [EMG00] propose detecting architectural mismatches based on formalizations of architectural styles and their underlying features. In fact, other studies such as [PKG99]

state that characteristics based solely on style properties might not expose conflicts with enough detail to aid resolution. Further studies, such as [DGP02] and [UY00], define architectural features and assumptions that need to be investigated to detect component mismatches. From a different perspective, Kontino et al. [KCB96] suggest evaluating COTS based on integration goals and their factor refinements. Similarly, Alves et al. [AFCF05] define types of COTS mismatches based on the degree to which they fulfill goals and functional requirements. Moreover, Franch et al. [FC03] and Carvallo et al. [CFGQ04] propose using quality models to evaluate COTS.

A lot of work has been performed with a focus on the technical interoperability level, which is not within the scope of our Ph.D. thesis research. For example, model-based approaches have been proposed to overcome heterogeneous middleware solutions, e.g. [BRLM09] and [BGRB11]. Also, technical interoperability platforms with APIs for developing interoperable software has been proposed, e.g. by [GBS03]. Beyond these, dynamic synthesis for emergent connectors [SG03] [AINT07] [IBB11] [BPGG11] have been proposed to mediate the interaction protocols executed by network systems.

**Testing-based analysis.** These techniques, such as [HBH+10] are useful for detecting software mismatches; however, they require the creation of complete, high-quality test suites and the launch of interoperation for each test. Besides, intensive testing is not always feasible due to invocation costs. Some methods consider prototyping for component analysis by simulating its usage within other systems [LBCC08]. As this requires component acquisition, learning, and evaluation, it is expensive and limits the number of analyzed components. Similarly, model-based interoperability testing and frameworks [BP] [GBPS14] have been proposed for the web service technology. However, this solution is designed for a single technology and requires detailed models (i.e., technical interface using WSDL and behavior using BPEL) to generate automated tests.

## 4.5 Summary and Conclusion

In this chapter, we have presented related work found in the literature about conceptual interoperability problems, solutions, models, and analysis approaches that is close to our research ideas described in the introduction (Section 1.3). Below, we will summarize the analysis of these related works and discuss their research gaps in insufficiently addressing the problems we described earlier in Section 1.2 and in the findings of our state of the practice survey in Subsection 3.2.2.

We found that there is no single approach or framework to which we can compare our intended contributions. Instead, we found many works related to each of our ideas, which partially lay the foundations for this

thesis work. The main contribution of this Ph.D. thesis is in the area of software interoperability analysis. That is, we introduce a framework for supporting providers and clients of interoperable software units in performing conceptual interoperability analysis.

The basis for our research is a model for the conceptual interoperability constraints (COIN Model). As described in Section 4.3, existing interoperability standards are domain-specific (i.e., they focus on limited business or technical levels), unsustainable, and usually need to be complemented with additional standards or project-specific conventions [MT08]. On the other hand, current interoperability models and classifications have established a strong basis for this property, but they are abstract classifications for the concept and do not support the purposes and activities of practical analysis. That is, none of them specifies precisely what each classification level would include in terms of constraints that restrict the software units that are intended to interoperate. Besides, the existing models do not relate the classifications to the types of mismatches they cause. Hence, these models have not found their way into practical approaches for conceptual interoperability analysis. This is why we extend this work into a comprehensive model for conceptual interoperability constraints with fine-grained attributes. In addition, we provide detailed guidance on how to practically benefit from this model as a reference for supporting different conceptual interoperability analysis activities (including documenting relevant information about interoperable software units and checking specific constraints of external software units).

The core of this thesis is the conceptual interoperability analysis (COINA) framework. As described in Section 4.2, the majority of current research works focus on the technical level of interoperability with a reactive kind of architectural solutions with limited automation support. We introduce a proactive, tool-supported framework for supporting interoperability analysis with a focus on the conceptual level. Both our COIN Model and the COINA Framework are domain- and technology-independent.

A key component of our COINA Framework is its support for architects and interoperability analysts in extracting conceptual interoperability constraints from the in-house architecture documents of their own software units and from the shared documentations about external software units. All works related to this idea follow the same trend of assigning the third-party clients the responsibility for extracting the conceptual constraints for interoperable units. We propose proactively preparation of such information by the owners of the interoperable software in order to reduce the cost expended by each third-party system for inferring these constraints and to raise the value and competitiveness of interoperable software. We extend and stretch the related works to extract a comprehensive set of conceptual

interoperability constraints based on our COIN Model. Furthermore, we semi-automate the process and utilize machine learning capabilities to decrease the manual effort and make our approach practical and usable.

Another key component of our COINA Framework is to support architects and analysts in detecting conceptual interoperability mismatches between two software units early in a project. Most of the presented related works regarding this idea (see Subsection 4.4.2) propose manual specifications for interfaces, which mainly focus on method contracts (i.e., invariants, pre- and post-conditions). Also, some of the related works are domain- or technology-specific and some entail high cost (e.g., for prototyping and running test suites). These works inspired us to propose our systematic, guided approach for detecting a comprehensive set of conceptual interoperability mismatches. Our algorithm-based approach directs practitioners in mapping the detected conceptual interoperability constraints in order to identify any existing conceptual mismatches.

In summary, we provide a comprehensive model for conceptual interoperability constraints, which in turn lays the foundation for our analysis framework. This framework provides guidance for architects and analysts to help them perform simple, efficient, and effective conceptual interoperability analysis with reusable output.



---

## 5 The Conceptual Interoperability Constraint (COIN) Model

### 5.1 Introduction

In this chapter, we introduce our Conceptual Interoperability Constraints (COIN) Model, which builds the notional relations among the conceptual interoperability constraints and the interoperable software elements, software types, and mismatch types. The model addresses the theoretical research problem described in Section 1.2 (i.e., R.P1: Lack of theoretical foundation that defines the conceptual interoperability constraints and their related mismatches for software units). The model extends the Reuse Model of Basili & Rombach [BR91] with a particular focus on the conceptual, non-technical characteristics of interoperable software units.

The COIN Model is the foundation for the remainder of this thesis work. In particular, it directs the related activities of conceptual interoperability analysis including (1) the search for the conceptual constraints that need to be identified for two software units that are intended to interoperate, (2) the identification of conceptual mismatches and their impact based on the constraints causing them, and (3) the documentation of the results of (1) in a standard, structured document, which we call the COIN Portfolio, for each system. *Note* that some parts of this chapter have been published by the author of this thesis in [ANR15], [AAR15], and [AR16].

- In Section 5.2, we will start by defining the term “COIN”, then introduce the COIN Model and its relations with various aspects.
- In Section 5.3, we will characterize the COIN Model in its three dimensions and specify its attributes in detail. Afterwards, we will connect the COINs with the conceptual mismatches they cause.
- In Section 5.4, we will present our proposed “*COIN Portfolio*” and “*Mismatches List Template*”, which are documentation templates for conceptual constraints and mismatches that maintain information in a standard and structured way.
- In Section 5.5, we will summarize the presented model and its value for the subsequent chapter.



## 5.2 Model Overview

In this section, we will give a brief overview of the COIN Model, its aspects, and relations, which offers a proper reference that can be used for the detailed explanation in the next section.

What are COINs? First of all, we define the conceptual interoperability constraints (COINs) of a software unit as follows:

**Definition 7 – Conceptual Interoperability Constraints (COINs)**

These are the conceptual, non-technical characteristics of a software unit that, if mis-assumed, may lead to conceptually wrong, meaningless, or improper interoperation results.

Directly conflicting or indirectly influential COINs between two software units that are intended to interoperate lead to conceptual mismatches as described in detail in Section 1.1. In other words, successful integration requires identifying and satisfying the units' COINs.

Why a COIN Model? From a model theory point of view, the COIN Model bridges the gap between the abstract classifications of conceptual interoperability (see Section 4.3) and concrete real-world elements of interoperable software units in specific interoperation contexts. Thus, it offers a solid basis for our methodical ideas of supporting the conceptual interoperability analysis tasks presented in Chapter 6. Moreover, the model is general and can be used to support interoperability analysis in the context of different integration projects. That is, it supports analyzing the interoperability of either black-box or white-box software units.

Who benefits from the model? For all roles involved in software integration projects, the COIN Model directs the understanding of the conceptual interoperability constraints and their impact on integration projects. Hence, the model can be used to guide the tasks of identifying the COINs, analyzing their impact, resolving their associated conceptual mismatches, estimating the cost of satisfying them, or testing the implemented integration. It can also be used as a reference for researchers interested in classifying constraints found in different artifacts of an interoperable software unit. The simplicity of the model enables its users to remember it and use it as a mental guide.

What does the model cover? The COIN Model relates the most important aspects regarding COINs. It defines the categories of conceptual interoperability constraints (i.e., syntax, semantic, structure, dynamic, context, and quality). It also relates the constraints to elements of the interoperable software (i.e., data, function, and system) and applies them to specific types of software units (e.g., IS, ES, MS, etc.). The model also shows that a constraint has a value (i.e., qualitative and/or quantitative) and a

significance weight (i.e., high, medium, or low). Furthermore, the model states that a constraint can be associated with different categories of the conceptual mismatches that it can cause (i.e., direct, indirect, or potential). Figure 19 summarizes the main aspects of the COIN Model and its relations.

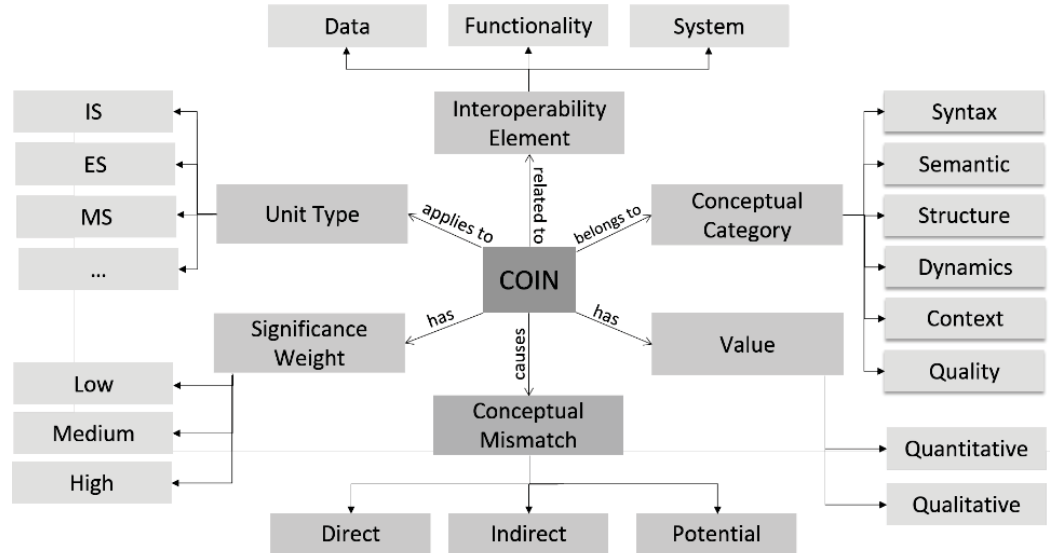


Figure 19 The main aspects of the COIN Model

Although we claim to build a comprehensive model, we do not claim that it is complete. Accordingly, we introduce not only fine-grained types of COINs and mismatches, but also coarse-grained COIN and mismatch categories to allow extending the model with whatever may emerge to be a conceptual interoperability constraint or mismatch for future software units.

## 5.3 Model in Detail

In this section, we will introduce in detail each of the aspects of the COIN Model. We will explain the dimensions of a COIN (Subsection 5.3.1), its detailed attributes (Subsection 5.3.2), and its associated types of conceptual mismatches (Subsection 5.3.3).

### 5.3.1 Principal Dimensions

Defining the dimensions of the COIN concept helps us categorize its fine-grained types and grouping them according to the integration context. Thus, the users of the COIN Model can focus on the necessary conceptual constraints that are relevant for the context of their integration project. Our proposed COIN Model has three main dimensions that extend and enhance the Reuse Model [BR91] and the Model for Enterprise Interoperability [Che06] to fit the purposes and tasks of software interoperability.

Chen et al. [Che06] state that enterprise interoperability is defined in terms of interoperability barrier categories, enterprise levels, and approaches for developing interoperability. However, this categorization is too broad and abstract for our interest in software interoperability. We rather need consideration of the interoperable elements of a software system or unit, the type of external interoperating software units, and detailed categories for the conceptual interoperability constraints. Basili and Rombach [BR91] defined a characterization scheme for reuse candidates, reuse requirements, and reuse processes, which we extend with our three dimensions.

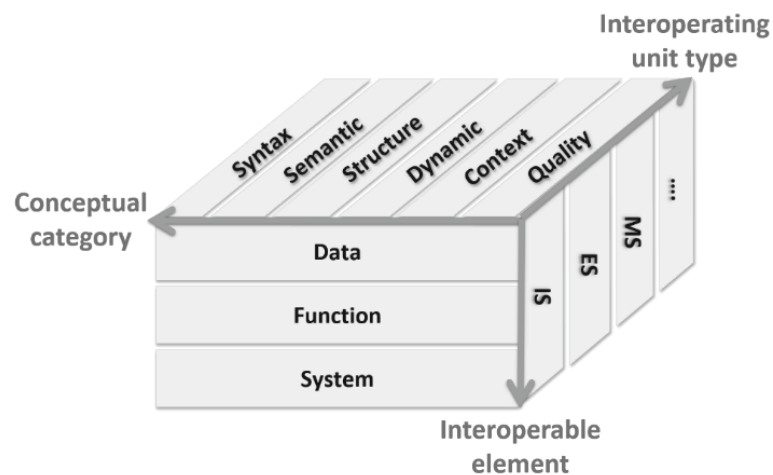


Figure 20 The three dimensions of our COIN Model

As shown in Figure 20, the three dimensions of the conceptual interoperability constraints (namely, interoperable element, conceptual category, and interoperating unit type) are independent. For example, a COIN can belong to the context category and be about a data element. The intersection of the dimensions defines the subset of conceptual interoperability constraints that users of the model should pay attention to in their integration projects. An example is the intersection of the quality category with the function element exchanged with a software unit of the type embedded system. In such a scenario, special quality constraints would be of interest to the user (e.g., safety, power consumption, etc.). On the other hand, if the user is interoperating with an information system, other quality constraints would be of higher interest (e.g., security, response time, etc.). Next, we will discuss each of the dimensions in more detail.

### First Dimension (Interoperable Software Element)

As software interoperation can take place at different elements of a software unit, for our COIN Model we define the following three categorizations of COINs based on the interoperable elements:

- **Data COINs:** These refer to the conceptual constraints regarding the data that need to be satisfied in order to have a successful and meaningful data exchange between two software units. For example, these constraints may declare the meaning of the data, define relations between data elements, or even specify the qualities of exchanged data in terms of resolution, security, etc.
- **Function COINs:** These refer to the conceptual constraints regarding the functions or services that need to be satisfied in order to allow successful and meaningful exchange of functions between two software units. For example, these constraints may declare the goal of a function, state its pre- or post-conditions, announce its interaction protocol, or even specify functional qualities like performance, availability, etc.
- **System COINs:** These refer to the conceptual constraints imposed by the overall software unit (i.e., the software system) that offers the exchange of data or functionalities. These constraints must be satisfied to allow overall success in building interoperation between software units. For example, these constraints may specify the system goals and the end users intended to benefit from the offered functions, define usage and development contexts and environments, announce general qualities that apply to all offered functionalities and data, etc.

Such clear and direct relations between the conceptual constraints and the interoperable elements can improve software integration projects. They will, for instance, help interoperability analysts structure the identification of the conceptual constraints of a software unit. They also help to trace mismatches back to their source elements.

## Second Dimension (Interoperability Category)

Conceptual interoperability constraints are not about syntax and semantic constraints as categorized by some models. They rather include a number of categories that we derived from various literature sources and problem statements. The categories of conceptual constraints that we propose are as follows:

- **Syntax COINs:** These state the concept-packaging methods and the lexical references of exchanged data, functions, and the system. For example, they state the terminology used or the conceptual modeling language that interoperating units need to agree on. Examining the syntactic match paves the way towards investigating the semantic one.
- **Semantic COINs:** These express the meaning-related constraints of the syntax used. For example, they may state that the measuring unit for calculating a distance service is kilometers, not miles. They also state the semantic meaning of input, output, and service goals.

Moreover, semantic COINs could include domain references (e.g., reference ontologies) that encode the meaning of exchanged data and service goals. However, as no reference ontology has been widely adopted yet, we consider such references as theoretical constraints that are left for future advances in ontology research.

- **Structure COINs:** These depict the software unit's elements, its relations, and the arrangements that affect the interoperation results. For example, interoperating with a software system without being aware of its data distribution may introduce a security threat if network links between remote sites are not encrypted. In this case, the distribution of the system is a structural COIN, and so are function dependency, system layering, etc.
- **Dynamic COINs:** These restrict the behavior of interoperating elements during interoperations. If such details are missed, they can introduce conceptual interaction flaws. For example, interoperating with a software system with regularly changing data may lead to synchronization issues if this property is not declared and addressed properly. Also, dynamic COINs specify interaction protocols, timing constraints, etc.
- **Context COINs:** These pertain to external aspects forming the interoperation settings of a software unit. For example, software systems that are designed to interoperate with software systems on desktop devices may cause display and memory issues on mobile devices. Hence, context COINs specify the user and usage properties for data and functions or for the overall software system.
- **Quality COINs:** These capture both the required and the provided quality characteristics related to exchanged data and functions. For example, inaccurate results may occur when interoperating with a face detection service that requires images with a specific degree of resolution.

Categorizing the conceptual constraints according to this dimension helps to enhance the structured investigation of these constraints and to better understand their impact. It aids different tasks in an integration project, including analysis, resolution planning, and cost estimation.

### Third Dimension (Interoperating Software Unit Type)

The third dimension allows further categorization of the conceptual interoperability constraints according to the type of the interoperating software unit. Hence, users of the COIN Model include/exclude conceptual interoperability constraints in their focus during software interoperability related tasks (e.g., analysis) based on their importance and relevance for the interoperating system. For example, in the case of interoperating with an IS, ES, or MS, the following COINs would be used:

- **IS COINs:** These refer to conceptual constraints that are important especially when interoperating with an information system. For example, when interoperating with a farm management application, conceptual constraints of interest would include data structure, image data resolution, performance and usability of query functions, etc.
- **ES COINs:** These refer to conceptual constraints that are important especially when interoperating with an embedded system. For example, when interoperating with field mowers, conceptual constraints of interest would include functional complexity, data transfer speed and frequency, reliability and safety of functionalities, etc.
- **MS COINs:** These refer to conceptual constraints that are important especially when interoperating with mobile systems. For example, when interoperating with a farmer app, conceptual constraints of interest would include some IS COINs, like data structure and functionality performance and reusability, and also some ES COINs, like power consumption and reliability.

This distinction between the COINs on the basis of the interoperating unit type allows avoiding unnecessary or even overwhelming conceptual constraints that would not affect the desired interoperation. For example, it would be irrelevant to declare user interface qualities for embedded systems. Note that these types of systems can be extended with further instances like systems-of-systems, cyber-physical systems, ecosystems, etc.

### 5.3.2 Detailed COIN Attributes

In this subsection, we will present detailed COIN attributes under each category of our model's second dimension (i.e., the conceptual category). To allow practical and easy use of our COIN Model and make it useful for conceptual interoperability related tasks, we define a set of COIN attributes that need to be investigated after determining their importance by checking their intersection with the other two dimensions (i.e., the interoperable element and the interoperating unit type).

We have derived these attributes by building upon and extending the existing literature works (including [BR91], [Sha95], [GAO95], [Gac98], [DGP02], [Bhu07], and more). Our selection criterion for an attribute was its potential for causing any conceptual interoperability mismatch between two software units regardless of the integration context (e.g., black-box or white-box integration). Remember, we claim that our proposed attributes are comprehensive, but not complete. The conceptual categories can always be extended with further attributes as imposed by future characteristics of software units.



Table 7 Detailed attributes of COINs

COIN Category	Attributes	Brief Description/Example
<b>Syntax</b>	Lexical references	Dictionary, thesaurus, glossary, etc.
	Modeling languages	XML, UML, ADL, WSDL, etc.
<b>Semantic</b>	Semantic references	Reference ontologies
	Semantic constraints	Data units, scale ratio, ordering style, etc.
	Goals	Explanation of why to interoperate with this interoperable element
	Input and output	Conceptual description of expected input and output
<b>Structure</b>	Data structural constraints	Inherited constraints, multiplicity constraints, structure size, etc.
	Data storage	Relational database, flat files, etc.
	Dependency	Underlying dependencies (i.e., additional required functions or data)
	Distribution	Distributed, centralized
	Encapsulation	Encapsulated, not encapsulated
	Concurrency	Single-threaded, multi-threaded
	Redundancy	Single or multiple units or interfaces dedicated to a specific element
	Layering	Layered, not layered
<b>Dynamic</b>	Dynamicity	Static, periodic change, irregular change, continuous change, etc.
	Service conditions	Invariants, pre-, and post-conditions
	Interaction protocols	Data/control flow, acknowledgement protocol, error handling protocol, etc.
	Interaction properties	State(ful/less), (a)synchronous, etc.
	Interaction time constraints	Session timeline, acknowledgment timeline, response timeline, etc.
	Communication style	Messaging, procedure call, blackboard, streaming, etc.
<b>Context</b>	Usage context	Device type, wired/wireless, access rate, time, location, etc.
	Intended users	Human/machine, gender, age, etc.
<b>Quality</b>	Data quality	Security, trust, accuracy, etc.
	Service quality	Safety, availability, efficiency, etc.

As seen in Table 7, each COIN category has a number of different attributes that are briefly explained or illustrated with explanatory examples. Some of the presented attributes can have multiple values and some can have exactly one value. For example, dynamic service conditions can be filled with multiple pre- and post-conditions, while the structure concurrency of a unit can either be single-threaded or multi-threaded. Next, we will describe each attribute in more detail.

**Syntax – Lexical references.** This attribute is concerned with the terminology used by the software unit. It determines if the software units intended to interoperate are using the same terminology for the same concepts. This attribute applies to domains in which such information



is preserved using common glossaries, catalogs, dictionaries, or protocol data units. For example, in the health domain, if a clinic software is built using the ICPC-2 standard terminology and a hospital software is built using the SNOMED CT terminology, a resolution for the syntax difference is needed in order to allow their successful interoperation. However, if a system is not using any data reference, then it will be a challenging task to check the terminologies and determine the system's readiness to interoperate meaningfully with other systems.

**Syntax – Modeling languages.** This attribute is concerned with the conceptual modeling languages used in representing the concepts of the interoperable elements. It is not about the technical data types or formats. The techniques used primarily for software modeling include Entity-Relationship (ER) diagramming, data flow diagramming, systems flowcharting, and workflow modeling [DGR+06]. Accordingly, different conventions can be used, such as UML notations, BPMN, SysML, EXPRESS, IDEF1X, etc. To explain the importance of such an attribute, imagine that the GPS tracking system uses UML for modeling the ER diagram of exchanged data. Meanwhile, the farm management system uses the IDEF1X notation. If both systems provide formal conceptual models, then this will not be of any concern and the mapping will be on the formal level. However, if the analysis is to be based on these models, then one of them should be transformed into the other notation to enable analysis and mapping in order to find any mismatches between the systems.

**Semantic – Semantic references.** This attribute is concerned with stating any references aimed at defining the meaning of the terminology and syntax used, such as domain-specific ontologies. This helps to determine whether both the software provider and the client have a common understanding of the meaning of the requested data or services. Ideally, this would be defined using ontological references, but as we mentioned earlier, there are no references yet that have been widely adopted in any domain. Hence, we consider this attribute to be theoretical and leave it for future advances in ontology research

Generally, semantic interoperability of software units is not a trivial issue. Accordingly, traditional studies on semantic interoperability focus merely on the behavioral specifications of software units, as embracing all semantic aspects of software interoperation is almost impossible [VTH99]. However, we here try to cover more aspects than just behavior, as explained next.

**Semantic – Semantic constraints.** This attribute is concerned with the data units, measurement systems, scale ratios, ordering style (e.g., ascending and descending) used. It helps to ensure that the data exchanged between two systems are interpreted in the right way. For example, the GPS tracking system uses the latest version of the World

Geodetic System [Tru04], which is WGS84, as its coordinate system. However, a farm management system may use an earlier version, such as WGS72. The architect needs to decide whether the difference between the two coordinate systems introduces a semantic mismatch between the systems or not.

**Semantic – Goals and Input/Output.** These attributes are concerned with stating the goals, input, and output of methods or services in a conceptual way. That is, they do not aim at declaring the technical face of these aspects, but rather their notion in order to ensure that interoperation with the service happens for the right reason. Regarding the input and output, this attribute makes it clear whether these consist of data streams, events, function calls, triggers, etc. Typically, these aspects would be expressed using natural language text, as their formalization using mathematical languages would not be easy to write or to understand. For example, the GPS tracking system produces the location of a vehicle as coordinates, while the farm management system requires the location output as an address. This explicit conceptual description of the required and provided output makes it much easier for analysts to detect mismatches than if they had to read example codes explaining the input and output of a service.

**Structure – Structural data constraints.** These are concerned with the restricting properties of the exchanged data entities and their relationships. They represent business rules or policies imposed on the exchanged data, such as cardinality constraints (e.g., one-to-one, one-to-many, many-to-one, or many-to-many), structure size and ranges (e.g., list length or value range), participation constraints (e.g., total or partial), relationship constraints (e.g., functional, is-a, OR, XOR, inheritance, aggregation, composition, etc.), and attribute constraints (e.g., key, derived, composite, component, single, or multivalued). To illustrate the impact of such constraints on interoperation, imagine that the GPS tracking system has the data entity “Coordinates” composed of the two entities “Longitude” and “Latitude”, whereas the farm management system has the data entity “Coordinates” with the two attributes “Longitude” and “Latitude” (see Figure 21).

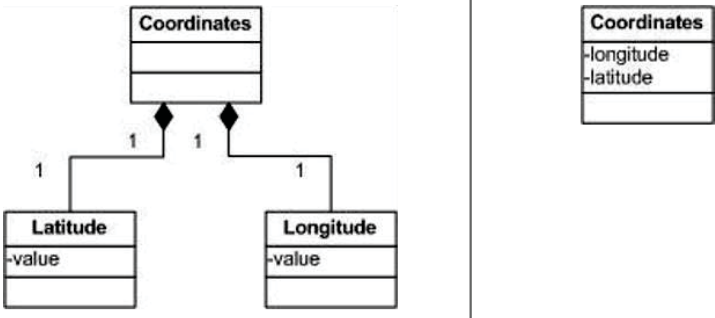


Figure 21 Structural differences of data between two systems

The example in the figure shows a similarity in concept between two systems with two different structural representations, which would also be implemented differently. Hence, it should be indicated in the interoperability analysis results to ensure resolving it correctly.

**Structure – Data storage.** This kind of constraints is concerned with the type of data storage that a specific system uses. Possible types of data storage are relational databases, XML files, full-text files, graphical databases, etc. Hence, it would be important to know the type of data storage used by another system to ensure the feasibility of exchanging data between them. To understand the value of stating such constraints, imagine a white-box integration between a hospital system that uses relational data and a clinical system that uses XML files. Knowing such information in advance would be of great value for estimating the effort required to overcome such a mismatch.

**Structure – Dependency.** This attribute is concerned with announcing the underlying dependencies, i.e., the additionally required functions or data to enable the exchange of data and services. For example, to enable the farm management system to launch the AutoSteering service of a smart tractor in the field, it is required that the ObstacleDetecting service of the smart tractor is enabled. However, the farm management system disables the ObstacleDetecting service of the smart tractor, which requires accessing the farm database (which is basically allowed for farmers' applications only and not for machinery) to record found obstacles. Hence, such a mismatch needs to be resolved to enable the desired interoperation.

**Structure – Distribution.** This attribute is concerned with announcing the distribution status of interoperable software elements, which may be either distributed or centralized. Such information can be of value for integration scenarios with a specific interest in security. For example, if a software unit that requires security interoperates with another software system without being aware of the latter's distribution, a security threat could be introduced if network links between remote sites are not encrypted. Also, it is necessary to announce the distribution status as some software systems need to know the complete system state in order to make decisions, whereas no single machine in a distributed system has complete information about the system state and makes decisions based on local information.

**Structure – Encapsulation.** This attribute is concerned with announcing the encapsulation status (i.e., has an interface or not) of the offered data and services in white-box integration context. This would be of value for interoperability analysts in determining how to access the interoperable elements properly without breaking their access rules for internal data and implementation. For example, if the

smart tractor provider announces that each of its services, such as AutoSteering, has an interface, the analysts could directly study this interface and the requirements to build the interoperation with the farm management system. However, if the provider announces a white-box service with no interfaces, the analyst's job would be harder as it would require investigating the technical code and finding the starting point to establish the interoperation. In other words, the encapsulation status plays a role in deciding the effort required to build the interoperation between the systems.

**Structure – Concurrency.** This attribute is concerned with stating the concurrency status (i.e., single-threaded or multi-threaded) of the interoperable software unit in the context of white-box integration. This information would be important for interoperability analysts to determine the amount of re-architecting work required to enable interoperation between single-threaded units and multi-threaded ones. For example, imagine that the farm management system is a multi-threaded system that allows receiving multiple requests from several smart tractors at a time, while the smart tractor is a single-threaded system. To overcome this mismatch and enable successful interoperation between these systems, it is necessary to replace all the function calls issued by the smart tractor that are not thread-safe.

**Structure – Redundancy.** This attribute is concerned with the existence of duplicates for an interoperable software unit or its interface. Announcing this information is important as it has a potential impact on the decision regarding interoperation with a software unit, especially in safety-critical systems. For example, a farm management system may have safety as a priority requirement, but the smart tractor might have no duplicated functions or controlling components. Such a mismatch is a threat to the realization of a highly reliable and safe system, and to overcome it may come at high cost for the integration project.

**Structure – Layering.** This attribute is concerned with the layered architecture of an interoperable unit, which is important information in case of white-box integration. This attribute helps interoperability analysts in determining how to access each layer of the unit properly without breaking its hierarchical access rules. For example, if the provider of a farm management system allows white-box integration and declares its exact layers (i.e., user interface, business logic, data access, and database), the analyst's job would be structured, as he would obviously investigate the constraints of each level against the overall farm management system. In other words, the layering status allows analysts to focus on the right access interface for each layer to find its constraints and to plan the resolution of any mismatches.

Note that there are further attributes that can be added to the structure category of COINs, like those stated by [Bhu07] , which include

backtracking to the old state of the system; preemption, which allows interrupting a running task and suspending tasks; and reentrance, which allows for non-interfering nested invocations. They also propose some technical types that are beyond the scope of this thesis work.

**Dynamic – Dynamicity.** This attribute is concerned with the exchanged data or service behavior and their state of changes. The behavior type of data can be either static/persistent or dynamic. For dynamic data, other properties of interest for interoperating systems include the data change frequency (e.g., periodic, continuous, or irregular) and the data growth rate (e.g., static, logarithmic, linear, or exponential). For both dynamic and static data, the data size is also an important aspect to be documented. For service dynamicity, an important aspect are runtime changes that happen to the behavior due to specific conditions (for example, the priority of serving customers changes according to quota changes). To illustrate this attribute in an integration scenario, imagine that the GPS tracking system updates the location information of registered dynamic objects every 30 seconds, while the farm management system reads the location of its registered dynamic objects every 10 seconds. Such a mismatch needs to be resolved before implementing the integration.

**Dynamic – Service conditions.** This attribute captures the contracts of interoperable functions, which state their invariants as well as their pre- and post-conditions. These contracts are critical for allowing the desired interoperation and should, accordingly, be investigated by interoperability analysts. For example, to enable interoperation between a farm management system and the AutoSteering service of a smart tractor, certain preconditions need to be satisfied (e.g., the user must be logged in with authorization to activate this service).

**Dynamic – Interaction protocols.** This attribute is concerned with clearly stating the different interaction protocols of interoperating software units. These protocols include data and control flows, acknowledgment protocols, error handling protocols, etc. They describe steps and activities included in the interoperation. For example, the farm management system expects notifications on failed tasks, while the smart tractor sends no notifications, but backtracks the system to a previous state. Such a mismatch needs to be resolved to ensure meaningful results.

**Dynamic – Interaction properties.** This attribute focuses on the interaction properties between two software units. These properties include, for example, the statefulness of the interaction (i.e., stateful or stateless) and their synchronicity (i.e., synchronous or asynchronous). To understand this in the context of a desired interoperation, imagine that the farm management system requires no blocking for its components when waiting for a response (i.e., it allows asynchronous interaction), while the smart tractor blocks components until it sends a



response to them. Such a behavioral mismatch needs to be resolved for successful integration.

**Dynamic – Interaction time constraints.** This attribute is concerned with time constraints restricting the interoperation between software units. This includes session timeline, acknowledgment timeline, response timeline, etc. It is important to have these constraints satisfied to ensure successful communication. For example, the farm management system may drop an interaction session if no response is sent within a minute, while the smart tractor may take longer to respond depending on different variables. Such a mismatch can impede the desired interaction and must be resolved.

**Dynamic – Communication style.** This attribute is interested in the style of communication that a software unit follows (e.g., messaging, procedure call, blackboard, streaming, etc.). For example, the farm management system may communicate using message passing, so it does not need to know the exact name of a method. The smart tractor, on the other hand, may expect its procedures to be called directly by their name. Such a mismatch prevents the desired interoperation and needs a resolution.

**Context – Usage context and intended users.** These attributes are concerned with the user and the usage properties of interoperable data and functions or of the overall software system. Regarding the usage context, the attributes include device type (e.g., mobile system or desktop system), connection state (e.g., wired or wireless), application domain, access rate, usage time, supported geographical locations, etc. Regarding the intended users, the attributes include user type (e.g., human or machine), gender, age, technology experience, etc. To explain the effect of these attributes, imagine that the smart tractor is designed to be used by humans with medium to high experience in using advanced software technologies. However, farmers in developing countries might have lower experience than required to use such advanced technology. Such a mismatch will not be detected in the analysis if the context is not clearly stated, and it is difficult to overcome even if the smart tractor can be integrated with the other software systems of the farm.

**Quality – Data quality.** This attribute is concerned with a set of data quality characteristics that have an impact on the interoperation between two software units. There are very many of these data qualities, so we only describe some of them here, such as availability (e.g. instantly accessed, mean time to access, lock frequency, etc.), accuracy (e.g., approximate, precise, correct, reliable, certified, etc.), completeness (e.g., sufficient, inadequate, non-null values, missing values, etc.), timeliness (e.g., up-to-date, out-of-date, valid until, validity period, etc.), and structure state (e.g., structured, unstructured, semi-structured). Note that the availability of data is not independent as it is

affected by the availability of its system in general and more specifically by the availability of the functions that allow accessing it. Some of a system's availability factors are the redundancy of the network on which the data is being transferred, storage architecture redundancy, and others. An example of their effect on interoperation is that the GPS tracking system sends object coordinates with an average inaccuracy of 5 meters, which increases to 30 meters during space storms. Meanwhile, the farm management system assumes that the inaccuracy of the coordinates does not exceed 6 meters at any time. Such a mismatch should be reported and used to decide whether to proceed with the integration or to search for an alternative.

**Quality – Function quality.** This is concerned with a set of function/service quality characteristics that affect the interoperation between two software units. There exist many such function qualities, so we mention only some of them here, including availability (e.g., mean time to failure, meant time to recover, downtime frequency, etc.), performance (e.g., responsiveness and stability under workloads), usability (e.g., understandability and learnability), and others. Like data availability, function/service availability and performance are also affected by a number of system availability factors. Some of these factors are infrastructure redundancy, resilient client/server solutions, technical backup solutions, etc. An example that can serve to explain the effect of this attribute on interoperation is that the GPS tracking system sends object coordinates within 5 to 10 seconds, while the farm management system assumes coordinates to be ready within a maximum of 3 seconds. Such a mismatch influences the decision about whether integration is suitable.

### 5.3.3 Conceptual Interoperability Mismatches and Their Types

In this subsection, we will introduce our definition of conceptual interoperability mismatches and the different possible types that can be caused by COINs.

#### **Definition 8 – Conceptual Interoperability Mismatch**

The inconsistency due to conflicting or influential conceptual constraints between two software units that are intended to interoperate.

Like the general interoperability mismatches we introduced in Section 2.3, conceptual mismatches between software units can be caused by either conflicting or influential features or constraints. However, these constraints are the non-technical ones that particularly state the software units' notions and abstract aspects (i.e., constraints regarding the software units' syntax, semantics, context, structure, behavior, and qualities). For example, a conceptual interoperability mismatch can be caused by conflicting contexts (e.g., different usage



modes between a software unit that works only online, requiring an Internet connection, and another unit working offline in airplane mode).

**Direct conceptual mismatches:** This type of mismatches is caused by COINs of similar categories and attributes when they have explicitly contradicting values for the corresponding interoperable software units. For example, the farm management system may have a structure COIN that states that its lists have a maximum capacity of 100 items, while the smart tractor may have a Structure COIN that states that the maximum size of its lists is 50 items. This leads to a direct mismatch on the structure level. This type of mismatches can be associated with any of the COIN types.

**Indirect conceptual mismatches:** This type of mismatches is caused by COINs with values that do not directly contradict the restrictions of other COINs in the corresponding interoperable software unit, but rather influence them implicitly. For example, the smart tractor may have a Dynamic COIN stating that the interaction is synchronous and any interacting software unit is blocked until a response is received and the task is accomplished, while the farm management system may have a Quality COIN stating that the system requires high response time. This leads to an indirect mismatch on the quality level. Mostly, structure constraints and dynamic constraints are the reasons for indirect mismatches and mainly affect Quality constraints of the other interoperable software unit.

**Potential conceptual mismatches:** This type of mismatches is caused by COINs that have no corresponding or conflicting constraints in the other system/service, neither directly nor indirectly. However, they have requirements depending on the following subtypes:

- **Adherence-type conceptual mismatches** demand work for satisfying them. For example, the farm management system may have a Structure COIN to have redundancy for safety-critical services and controlling units to ensure availability of a service, while the smart tractor has no constraints regarding redundancy. This leads to a potential adherence mismatch if the integration developers do not duplicate the critical units in the smart tractor. Hence, it has to be reported to ensure that the constraint is satisfied. Any type of COINs can be a reason for an adherence mismatch if it has no corresponding COINs and is not influenced by COINs in the other software unit.

**Consensus-type conceptual mismatches** demand a common understanding or agreement. For example, a fertilizer supplier system located in Finland offers a service to automatically deliver fertilizer to farms in the growing season. This system has a Semantic COIN that states the meaning of “Growing season” to be the period of time from June to September only. On the other hand, the farm management

system, which is located in Spain, has no corresponding constraint to define the aforementioned term, but it is commonly known that the growing season in Spain is almost year-round. This leads to a potential consensus mismatch if users of the latter system misunderstand the definition from the former one. It is obvious that Syntax, Semantic, and Context constraints are the main causes of consensus mismatches.

Later on, in Chapter 6, we will explain how to detect these types of mismatches following an algorithm-based method and how to document the results in a standard result template.

## 5.4 Standard Documentation Templates

In this section, we will show the first practical benefit from the COIN Model by introducing our proposed standard templates for documenting the COINs of an interoperable software unit (Subsection 5.4.1) and for documenting detected mismatches (Subsection 5.4.2). We will explain the role of these templates, their structure, benefits, and limitations. In Chapter 6, we will place these templates in the big picture of serving the tasks of conceptual interoperability analysis.

### 5.4.1 Conceptual Interoperability Constraints Template (COIN Portfolio)

What is a COIN Portfolio?	As mentioned in the introduction chapter, we call our proposed documentation template for conceptual interoperability constraints the “ <i>COIN Portfolio</i> ”. It is a standard and structured document used to explicitly and comprehensively declare the COINs of an interoperable software unit or system. This portfolio gathers all conceptual interoperability related aspects of a software system in a single coherent place to allow clients and interested parties to investigate desired meaningful interoperations. The COIN Portfolio is written in a human-readable format (i.e., natural language text) to allow all users with different levels of experiences to use it easily. However, it can be partially formalized using a formal-based description language (e.g., special DSL) for the COINs with quantitative values, but not for those with qualitative values (e.g., Quality COINs for response time can be formalized, but Semantic COINs for goals cannot). Such formalization can open the door for automating the comparison between portfolios, which is beyond the scope of this thesis work.
Why is it needed?	This template maintains the COINs of interoperable software units in a comprehensive and organized way. The goal of this consolidation of conceptual constraints is to support architects and interoperability analysts by using it as input for the conceptual interoperability analysis task. When the COIN Portfolio is prepared for each of the software units intended to interoperate, it facilitates comparing the units’ COINs and detecting their conceptual mismatches. This consequently allows project managers to make early informed decisions about the feasibility

of integration, to make trade-off between alternatives, and to assess the required effort.

**Who prepares it?** In the context of black-box integration, the COIN Portfolio is prepared by the providers of the interoperable software unit. That is, software architects and analysts create it so it can be shared with interested clients. This explicit announcement of conceptual information about their software units allows the clients to perform effective and efficient conceptual interoperability analysis and increases the competitiveness of the software unit. On the other hand, in the context of white-box integration, the COIN Portfolio can be prepared by the interested clients if the provider does not share it. That is, the architects and analysts of the third-party client create this document for their own units as well as for the external one. Adding work for each interested client can obviously decrease the competitiveness of the offered software unit, especially if alternative units do provide COIN Portfolios.

**When to create it?** For software units under construction, it is possible to create the COIN Portfolio gradually along the development lifecycle. In this case, the role responsible for creating the portfolio has to be careful and update its content as needed. It can also be prepared once the software unit is ready for interoperation with other units and no further changes are expected. Note that, for evolving software units, versions of the COIN Portfolio should be created and associated with the interoperable software versions to ensure sharing correct and up-to-date information.

### Structure of the COIN Portfolio

The COIN Portfolio is a structured document that reflects the dimensions of the COIN Model. Accordingly, it captures the COINs for the interoperable elements of a software unit (i.e., system, data, and services). For each of these elements, their set of COINs is declared in groups based on the COIN category (i.e., syntax, semantic, structure, dynamic, context, and quality). For each COIN instance, the portfolio includes a dedicated sheet in which all the COIN's details are reported (e.g., related interoperable element, interoperating unit type, COIN category, value, weight, etc.).

Figure 22 presents an example of a COIN Portfolio for a GPS tracking system ( $S_1$ ) in which only COINs that are relevant for its interoperable elements are announced explicitly. Such a portfolio benefits a software architect of an interoperating farm management system ( $S_2$ ) in detecting his system's conceptual mismatches with  $S_1$ . For example,  $S_2$  may have security concerns and need to exchange location data with  $S_1$ . The first station for the architect of  $S_2$  using the portfolio of  $S_1$  to assess the general suitability for the interoperation goals will be the COINs associated with the overall system. As the system COINs show no conflicts with  $S_2$  interests, the second station for the architect of  $S_2$  will be to review only the COINs of the Location data element. The

detailed COIN sheets then reveal the distribution characteristics of the Location data, which may introduce a security threat to S2 (conceptual interoperability mismatch).

Interoperable element(s)			COIN category	COIN(s)
Overall software system			Context	<u>Intended user</u>
			Quality	<u>Data resolution</u>
Data	Dimensions		Semantic	<u>Semantic constraint (Data unit)</u>
			Quality	<u>Data quality (precision)</u>
	Location		Structure	<u>Structural constraint (Inherited)</u>
				<u>Distribution</u>
Service	getLocation		Dynamics	<u>Interaction time constraint (Session timeline)</u>
			Context	<u>Usage context (Age)</u>
			Quality	<u>Service quality (Response time)</u>
	getMap		Structure	<u>Distribution</u>
	...		...	...

COIN Elements	Description	
Software system	GPS Tracking System (S1)	
Interoperable element	Location	
Category	Structure	
Name	Distribution	
Value	Qualitative: distributed	Quantitative: NA
Weight	Constraint	
Consequence(s)	Security threats can result in as network links between remote sites are not encrypted.	
Comment(s)	Technically there is no problems in receiving requests and sending results about Location.	

Figure 22 An example of a COIN Portfolio (left) and one of its sheets (right)

Figure 23 represents a meta-model for the COIN Portfolio concept and its captures its relation with some other concepts, such as the COIN, interoperable elements, conceptual interoperability mismatches, etc.).

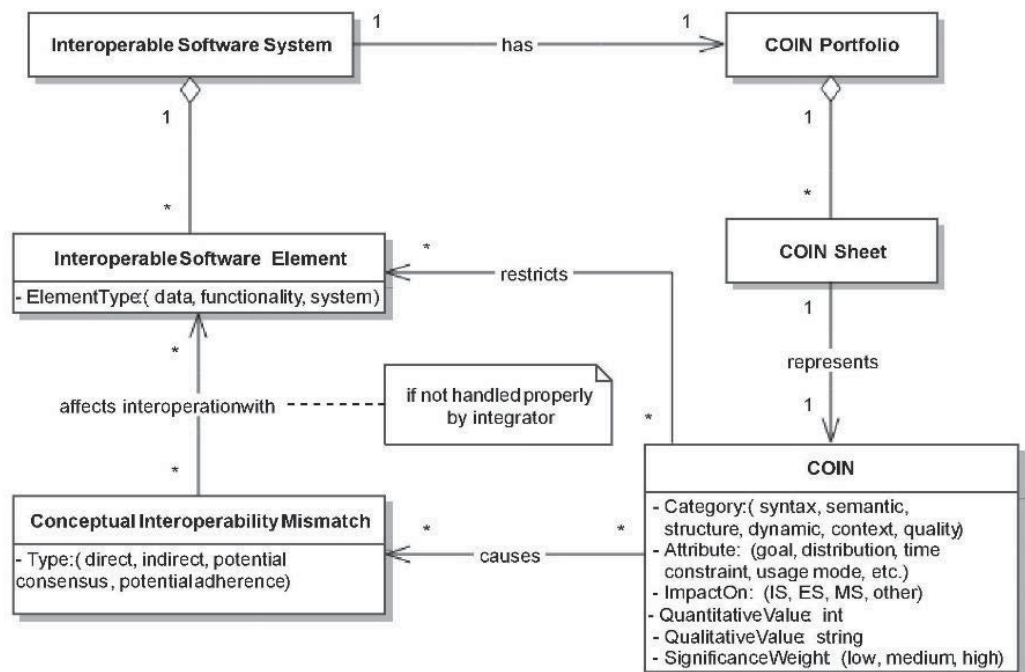


Figure 23 Meta-model for the COIN Portfolio of Interoperable Software System

Appendix C.1, contains an empty template of the COIN Portfolio document that can be used for saving the COINs of any interoperable software unit. In Chapter 6, we will introduce our software tool support for creating such a COIN Portfolio semi-automatically.

## Benefits and Limitations of the COIN Portfolio

The COIN Portfolio offers a number of benefits for its providers and targeted users, which are described in the following characteristics:

**Comprehensive.** The COIN Portfolio endeavors to deliver a comprehensive aggregation of the conceptual interoperability related information of an interoperable software unit. Based on this, using it as input enables performing a thorough and effective conceptual interoperability analysis. Hence, it lowers the risk of undetected conceptual mismatches between two units.

**Customized.** The COIN Portfolio focuses on presenting only the conceptual interoperability relevant information about an interoperable software unit that is really needed. In other words, it reduces the amount and complexity of the information by indicating what is enough for the conceptual analysis without overwhelming the reader with irrelevant technical information. Narrowing down the investigated artifacts and information consequently saves time and effort for the user.

**Well-structured.** The design of our proposed portfolio arranges interoperability information according to the categories of the conceptual concerns. Therefore, using it guides interoperability analysts in determining the different types of conceptual mismatches between two software units in a systematic way. Hence, it increases confidence and trust regarding the produced analysis results.

**Consistent.** The definite structure of the COIN Portfolio allows documentation consistency to be achieved among the different interoperable elements of the same software system and among different software systems. Such consistency is desired to help the user understand the content and to increase the efficiency in locating the desired information (this skill improves from one project to the next). It also helps to avoid the insufficiency of shared information (which can be mistakenly missed or ignored by the software provider).

**Reusable.** The COIN Portfolio of a software unit can be reused in its different integration projects with different systems. This saves time and effort for clients in every potential interoperation. It is important to mention that this applies only if the COIN Portfolio is kept up to date and maintained whenever a change happens to its system. In the long run, this improves the competitiveness of the interoperable software units' providers and grows their business impact as it increases the rate



of interoperation success. So, it is a one-time effort that saves cost in each future interoperation.

In addition to these beneficial characteristics of the COIN Portfolio, it can also be used as a basis for interoperability testing and exception handling. That is, it can be used for designing interoperability-related test cases at design time (expected scenarios). It can also be used as a basis for an exception handling mechanism at runtime (emergent scenarios). We will not go deeper into this discussion as these potential usages of the COIN Portfolio are beyond the scope of this thesis.

With regard to the limitations of the COIN Portfolio, it focuses only on the conceptual level of interoperability constraints and does not cover other levels (e.g., technical, organizational, etc.). Hence, it cannot be depended on as the only input for comprehensive interoperability analysis. However, its structure allows extending it in a flexible manner with further categories and attributes as needed. Thus, analysts interested in adding other levels of interoperability to the portfolio can do so easily. Also, creating the COIN Portfolio introduces some difficulties as it requires understanding of the COIN categories and attributes, spending effort on writing it, and continuously maintaining it. Thus, manual creation of the COIN Portfolio and its sheets is a cumbersome, expensive task that requires sifting through the software documentation to identify its conceptual interoperability constraints. To overcome these difficulties, in Chapter 6 we will present our proposed support for software architects in creating the portfolio (i.e., guidelines to support manual creation and software tools to allow semi-automated creation).

#### **5.4.2 Mismatches List Template**

What is a Mismatches Template?	We also propose a standard and structured template for documenting the detected conceptual mismatches between two interoperable software units. This “Mismatches List Template” gathers all aspects of conceptual mismatches in a single coherent place. It is written in a human-readable format and is the result of comparing the COIN Portfolios of two software units.
Why is it needed?	This template lists the conceptual mismatches between two software units in a comprehensive and organized way. The goal of this template is to support architects, interoperability analysts, and project managers in determining the feasibility of resolving these mismatches and going forward in building the interoperation. It also enables traceability between a conceptual mismatch and the COIN(s) causing it. Moreover, this standard document enables trade-offs between multiple candidates of external software units. This consequently allows decision makers to make informed decisions about the selected integration candidates.

Who  
creates it?  
When?

The Mismatches List is created by third-party clients interested in integrating their system and an external software unit. That is, software architects and analysts create it by comparing the COIN Portfolios of the two software units intended to interoperate. At a later point in time, this comprehensive documentation of the conceptual mismatches between software units allows making confident decisions about pursuing integration with an external software unit.

### Structure of the Mismatches List Template

The Mismatches List template is a structured document that maintains information about the conceptual mismatches in the two software units and the COINs causing them. Accordingly, for each mismatch instance, the template captures the mismatch aspects including the related interoperable element (e.g., system mismatch, data mismatch, etc.), its type (i.e., direct, indirect, and potential), and its detailed description.

Figure 24 presents an example of a Mismatches List between a farm management system ( $S_1$ ) and a smart tractor system ( $S_2$ ), where mismatches are recorded in terms of their related interoperability element. Such a list benefits the decision makers in an integration project by helping them decide whether or not integrating  $S_2$  within their  $S_1$  is feasible. The first station for the reader of the Mismatches List will be the conceptual mismatches on the level of the two overall systems to assess their general suitability for the interoperation goals. Then, the second station will be the conceptual mismatches of the data and service elements. For example, the detailed description of the location data mismatch reveals a direct conflict with the structural length of the list.

Interoperable element(s)		Mismatch type	Description
Overall software system		Direct	<u>Technology expert users vs. inexperienced users</u>
		Direct	<u>High response time vs. Medium response time</u>
Data	Dimensions	Direct	<u>Distance in km vs. in miles</u>
		Direct	<u>High precision data vs. low precision</u>
	Location	Direct	<u>Different length of List of locations data structures</u>
Service	getLocation	Direct	<u>Session timeline ends within 30 seconds of idle state vs. limitless session time</u>
	getMap	Indirect	<u>Distributed service vs. security requirements</u>
	...	...	...

Mismatch Elements		Description
Software system (S1)		Farm Management System (S1)
Software system (S2)		Smart Tractor (S2)
Interoperable element		Location data element
Category		Direct
Name		Different data structures
Description		S1 expects a list of locations for the tractor over the whole week, while S2 provides a list of the last 100 locations.
Reference COINs	COIN ID of S1	C13
	COIN ID of S2	C22

Figure 24 An example of a Mismatches List (left) and one detailed mismatch description (right)



Appendix C.2 contains the empty template of the Mismatches List document, which can be used for saving the conceptual mismatches between two interoperable software units. In Chapter 6, we will introduce our systematic analysis approach that makes use of this template.

## Benefits and Limitations of the Mismatches List Template

With regard to the benefits of this Mismatches List template, they resemble the ones of the COIN Portfolio. That is, it is similarly:

**Comprehensive.** The Mismatches List delivers comprehensive information about the conceptual mismatches between two software units from their system, data, and service perspectives. Based on this, using it as input for project managers enables making effective and evidence-based decisions and tradeoffs in their integration projects.

**Customized.** It focuses on presenting only the conceptual mismatches between two software units, which reduces the amount and complexity of the information presented to decision makers. That is, it reports what is enough for the conceptual analysis level without providing overwhelming technical information, so it saves time and effort.

**Well-structured.** The design of our template guides the reader in determining the impact of the different types of conceptual mismatches between two software units in a systematic way. Therefore, it increases the effectiveness of the decisions.

**Consistent.** The definite structure of the Mismatches List supports documentation consistency among the different external software units candidates investigated. Such consistency is favorable as it paves the way towards selecting the most appropriate candidate. This consistency also helps users to understand the content and locate the desired information efficiently (this skill improves from one project to the next).

In addition to these beneficial characteristics of the Mismatches List, it is currently offering a qualitative metric for estimating the required effort to enable integration between software units. It also offers a basis for developing a quantitative metric that would be derived from the weights given to the quantity, type, and perspective of the mismatches. Such quantitative weights could be used to build formulas for estimating integration cost in order to support decision-making and trading off between candidate units. A radar or spider chart could be used to visualize the metric. It is obvious that developing such a metric requires reported experiences in the cost of resolving the different kinds of mismatches. We will not go deeper into this discussion as this potential benefit of our Mismatches List is beyond the scope of this thesis.

With regard to the limitations of the Mismatches List, it focuses only on the conceptual level of mismatches, just like the COIN Portfolio. Hence, it cannot be used as a standalone for comprehensive decision-making, which needs to cover further aspects (e.g., organizational, technical, etc.). However, its structure allows extending it with further categories as long as the COIN Portfolio also gets extended with corresponding constraint categories. Also, creating the Mismatches List document requires understanding of the mismatch types and the COINs causing them. Thus, creating it could be a cumbersome task for inexperienced analysts; in Chapter 6, we will therefore present our proposed support for software architects to help them create it (i.e., guidelines for mapping COIN Portfolios).

## 5.5 Summary

In this chapter, a model for conceptual interoperability constraints (COIN) has been introduced. The three-dimension model describes the relevant aspects for the COINs and shows their relations to it. In particular, we described the constraints that reflect the conceptual characteristics restricting the interoperable software unit and its data and services. We discussed the differences in their importance based on the type of the software units that are intended to interoperate with the unit. Afterwards, we defined categories for the conceptual interoperability mismatches that can be caused by COINs.

Our model provides a solid basis for the practical benefits of interoperability analysis. It also offers a structured basis for standard documentation templates, the COIN Portfolio and the Mismatches List, which comprehensively hold all related COINs and conceptual mismatches of two software units. The model also serves as a basis for other practical activities, which we will describe in the context of our interoperability analysis framework on the conceptual level in Chapter 6.

---

## 6 The Conceptual Interoperability Analysis (COINA) Framework

### 6.1 Introduction

In this chapter, we introduce our Conceptual Interoperability Analysis (COINA) Framework, which we built upon the COIN Model introduced in the previous chapter. As described earlier in the solution idea (Section 1.3), the framework includes our methodological and engineering contributions, which support practitioners in performing the conceptual interoperability analysis. It helps software architects and analysts in identifying the conceptual interoperability constraints and mismatches of software units more effectively and efficiently. *Remember*, although the COIN Model can be used as a foundation for activities related to either black-box or white-box integration, the current version of the COINA Framework focuses on black-box integration only.

The COINA Framework comprises two methodical components aimed at assisting providers and clients of interoperable software units. The first component of COINA addresses the methodological research problem R.P2: Lack of proactive approaches and automated solutions for guiding providers of interoperable black-box software units in identifying and sharing the conceptual interoperability constraints for their units. The second component of COINA addresses the methodological research problem R.P3: Lack of systematic analysis approaches for guiding interoperability analysts in identifying the conceptual constraints of two software units and detecting their mismatches). *Note* that most of this chapter has been published by the author of this thesis in [ANR15], [AR16], [AAR15], [AAHR16], and [AAR16].

- In Section 6.2, we start by presenting a big picture of the COINA Framework and its components. Then we will discuss the methodological overview of the framework and its contextual scenario.
- In Section 6.3, we will describe in detail the first component of the framework, which comprises proactive, in-house preparation for interoperable software units. This section includes a method for extracting COINs from UML diagrams with tool support (Subsection 6.3.1). We will also present our multiple-case study and the experiments that helped us propose our next method, which uses ML techniques for automating the extraction of COINs from NL text of API documents (Subsection 6.3.20). The multiple-case study

also resulted in a set of guidelines for enhancing the API documentation for conceptual interoperability analysis purposes (Subsection 6.3.3).

- In Section 6.4, we will present the second component of our COINA Framework, which is a systematic approach for detecting conceptual interoperability mismatches between two software units. We will present our algorithmic approach in detail and describe the associated guidelines and cheat sheets for its activities.
- In Section 6.5, we will summarize the presented framework, its benefits, and its limitations.

## 6.2 Framework Overview

In this section, we will give a brief overview of the COINA Framework, its components, and its context. This overview offers a proper reference that can be used for the detailed explanation in the next sections.

Why the  
COINA  
Framework?

While the COIN Model provides the foundation knowledge for the conceptual constraints and their types, the Conceptual Interoperability Analysis (COINA) Framework provides methods and guidelines for detecting the COINs and their related mismatches between two software units. Unlike existing analysis approaches, our framework does not focus merely on detecting conceptual mismatches. It rather offers comprehensive support for both the providers of interoperable units and third-party clients, as their efforts are closely intertwined. This means that the conceptual information about an interoperable unit shared by its providers affects the conceptual analysis and mismatch detection performed by the clients. Thus, the goal of the first component of the framework is to assist the providers of interoperable units in proactively publishing the COINs of their units while keeping the associated effort as low as possible. In addition, the goal of the second component of the framework is to support third-party clients who are interested in building interoperation with an external software unit in detecting conceptual interoperability mismatches effectively and efficiently.

### 6.2.1 Methodical Overview: Input, Output, Activities

What is the  
first COINA  
component?

Based on the COIN Model introduced in Chapter 0, the first component of the COINA Framework calls for **proactive, in-house preparation for software units** that are intended to interoperate with other units. It assists software providers in explicitly sharing the conceptual constraints with interested clients with the least effort. ***Prerequisite input?*** We assume the *availability* of the software unit's internally shared architectural and low-level design documentation (e.g., UML diagrams) and its externally shared API documentation. All documentations are also assumed to be *stable* (i.e., not undergoing

frequent major changes), *consistent* (e.g., consistency between the different UML diagrams and consistency between UML diagrams and API documentation), and *up-to-date* (i.e., representing the current state of the software system). In addition, the UML notations are expected to be used *correctly* as specified by OMG for Version 2.5<sup>1</sup>. **Output?** The resulting output is a standard document that explicitly states the conceptual interoperability constraints for the interoperable elements of the software unit (i.e., its COIN Portfolio). **Activities?** The process at this component starts with the definition of the list of interoperable elements of the software unit. That is, the software architect of the unit determines what data items and functions are supposed to be part of future interoperations with other software units. Then the framework semi-automatically identifies the COINs for the previously determined interoperable elements. Finally, the framework helps to document the extracted COINs in the standard documentation template called the “COIN Portfolio”. This portfolio then gets proactively published, making black-box units ready for proper analysis by potential third-party clients. In Section 6.3, we will explain in details and with examples how this component works.

What is the second COINA component?

The second component of the COINA Framework is also based on the COIN Model introduced in Chapter 0 and the defined types of conceptual mismatches. It proposes **a systematic, algorithmic method for detecting the conceptual mismatches between two units**. It assists third-party clients in comparing the COINs of two units in order to effectively and efficiently detect their conceptual mismatches. **Input?** The input to this component are the two COIN Portfolios, one for each of the software units that are intended to interoperate. If the first component of the framework has already been applied, then the input should be ready to use by the second component. Otherwise, the second component offers guidance on how to manually prepare the portfolios systematically. **Output?** The resulting output of this component is a list of the existing conceptual mismatches between the software units that are intended to interoperate. We propose documenting these results using a standard template. Thus, the results can be used for determining the requirements in order to enable meaningful interoperation and design the resolution (this activity is beyond of the scope for this thesis work). **Activities?** The process at this component starts with the mapping of the COIN Portfolios of the two software units in a structured way using the offered guidance. This detailed guidance is based on an algorithm that we designed. Then we determine the conceptual mismatches based on their criticality; either they cause direct contradiction or they require conceptual adherence. In Section 6.4, we will explain in detail with examples how this component works, what guidance it offers for manual analysis, and what the mismatch template looks like.

---

<sup>1</sup> <http://www.omg.org/spec/UML/2.5/>



### 6.2.2 Contextual Scenario: Who, When, How

The current version of the methodical contributions of the framework supports performing the conceptual interoperability analysis in *black-box integration projects*. This focus does not mean that white-box integration is less important, but with the limited time and resources for this Ph.D. work, we prioritized black-box integration due to the serious lack of shared conceptual information and support offered in the literature.

Context of the proactive preparation component?

**Who?** The proactive preparation component is particularly appropriate for software companies interested in building software units that are offered for clients (e.g., web services) or those companies building units with the intention of extending their capabilities through interoperation (e.g., initiators of ecosystems). The software engineering roles who are expected to apply this preparation are software architects and domain experts. Experience in interoperability and integration is not a prerequisite, as the framework offers detailed guidance and automation support. However, a high level of architectural knowledge and expertise would be the prerequisite if no automated support was used in analyzing low-level design documents (i.e., UML diagrams). **When?** As we mentioned earlier, this preparation is aimed to be proactive, which means having the COIN Portfolio of an interoperable software unit ready before it is needed by a client interested in the unit. Hence, this preparation is a design-time activity (not a runtime activity), which can be performed progressively throughout the development lifecycle of the unit or once it is ready at the end of the development. In both ways, the provider has to maintain this portfolio up to date. **Automation status?** This proactive preparation can be semi-automated with our tool support for COIN extraction from UML diagrams using our predefined templates and from NL text of API documents using our ML classification model.

Context of the systematic analysis component?

**Who?** The second component of COINA is designed in particular to aid third-party clients interested in building interoperation between their own software units and external black-box ones. As in the first component, the software engineering roles expected to perform the proposed systematic analysis are software architects or interoperability analysts. Also, experience in interoperability and integration is not required due to the detailed guidance offered by the framework. Thus, other roles such as software developers can perform the systematic analysis. **When?** Once an external software unit is considered for achieving some business requirements through interoperation, it has to be analyzed regarding conceptual interoperability with the existing software system. Hence, our systematic analysis takes place at the design time of integration projects (not at runtime). It is the first activity, preceding resolution design, integration implementation, and testing. **Automation status?** The current version of the systematic analysis only supports manual application based on detailed guidance and standard templates to save the results at each step. However, if the

COIN Portfolios for two software units are formalized, then automation possibilities can be introduced.

In Figure 25, we summarize the COINA Framework components and the included activities (in the middle), the expected input and output from two software units intended to interoperate (on the left), and the engineering context (on the right).

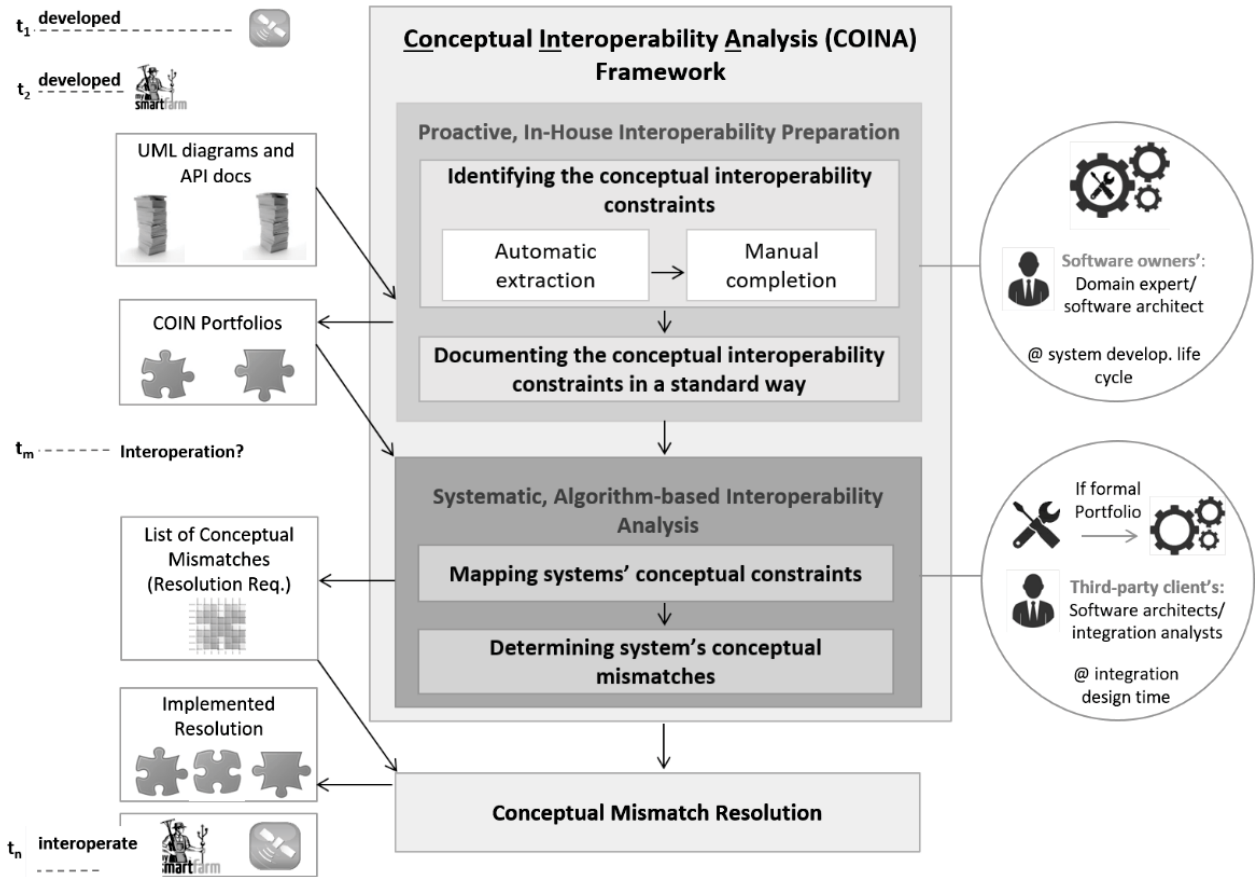


Figure 25 Overview of the COINA Framework

### 6.3 Proactive, In-House Preparation for Interoperable Software Units

In this section, we will describe in detail what we contribute to better support providers of interoperable software. In particular, we will explain our methodical contributions for extracting the conceptual interoperability constraints from already existing documents of the software unit. Remember that we have provided an overview of the method (i.e., input, output, and process) and the context (i.e., who, when, and how) in Subsections 6.2.1 and 6.2.2, respectively.

Next, in Subsection 6.3.1, we will describe how to extract the COINs from the internally shared architecture and low-level design documents using predefined templates. Afterward, in Subsection 6.3.2, we will describe how to extract the COINs from NL text of externally shared API



documents using machine learning. This subsection is based on a multiple-case study and on experiments that we will present in detail. Finally, in Subsection 06.3.3, we will present our guidelines for improving API documentation with regard to its COIN-related content and presentation based on our observations from the multiple-case study.

### 6.3.1 Extracting COINs from UML Diagrams

**Method goal** The foremost goal of this method is to support providers of interoperable software units in identifying the interoperability-relevant conceptual constraints of the unit. More specifically, the method helps to extract the structured conceptual constraints from UML diagrams. As described earlier in Subsection 3.2.2, extracting this information manually is a challenging task, as it is an undirected task that requires experience with software architecture documents and COIN types. It is also a tedious and time-consuming task that requires sifting through the UML documentation of the whole software system and then extracting only the useful pieces of information for the interoperability analysis. By facilitating the extraction of COINs, we pave the way towards proactively sharing useful documents with interested third-party clients in order to detect conceptual mismatches.

**Contribution** The key contribution of our method is an increase in the effectiveness of conceptual interoperability analysis as a result of making the conceptual interoperability constraints shared explicitly and comprehensively. Additionally, we enable the extraction of COINs from UML diagrams through our contributed “COIN Extraction Templates”. This template-based extraction is performed by means of extraction algorithms. We aid this extraction by implementing an add-in for the Enterprise Architect modeling tool.

Next, we will describe the COIN Extraction Templates for UML diagrams, which are the basis of our method. Then we will describe the semi-automatic extraction method in detail. Finally, we will present the supporting tool along with examples from the Smart Farming scenario.

*Note* that most of this subsection has been published by the author of this thesis in [ANR15], [AR16], and [AAR15].

### COIN Extraction Templates

**What are the templates?** Our predefined set of templates covers certain conceptual constraints of a software unit from its structural and behavioral UML diagrams. We define these COIN Extraction Templates as:

**Definition 9 – COIN Extraction Templates**

A set of rules that identifies specific types of conceptual interoperability constraint from the UML diagrams of interoperable software units.

Our predefined set of templates covers different types of COINs from different UML diagrams. More specifically, these templates target COINs from component diagrams, deployment diagrams, class diagrams, use case diagrams, and sequence diagrams. Remember, we assume that the UML notations are used correctly as specified by the OMG for Version 2.5. In Table 8, we show the specific COIN types that our templates target from each of the aforementioned included UML diagrams.

Table 8

Predefined COIN Extraction Templates

Template ID	COIN source diagram	COIN category	COIN type
t1	Component	Structure	Layering
t2	Component	Structure	Component distribution
t3	Component	Structure	DB distribution
t4	Deployment	Structure	Component distribution
t5	Deployment	Structure	DB distribution
t6	Class	Structure	Structural multiplicity
t7	Class	Structure	Inherited constraints
t8	Use case	Context	Allowed users
t9	Use case	Context	Usage multiplicity
t10	Use case	Structure	Inherited constraints
t11	Sequence	Dynamic	Interaction synchronicity
t12	All	NA	Natural language constraints

We represent each of our COIN Extraction Templates formally. For example, the formal representation of template t7 is as follows:

**Template (t7): Inherited constraints of class diagrams**

$t7 \in \text{Structure COINs Category}$

$\forall \text{ element } (e) \in \{ \text{interoperable elements (E)} \cap \text{class diagram elements (CDE)} \},$

$t7(e) = \text{True} \leftrightarrow e.\text{Parent} \neq \emptyset \wedge e.\text{Parent}.\text{Constraints} \neq \emptyset$

How to use the templates?

These templates can be used in the manual search for COINs in UML diagrams. However, the formality of the templates makes them a suitable basis for the desired automatic extraction of COINs for interoperable software units from their already existing UML diagrams.

This is done by checking the UML diagrams against these predefined rules using algorithms. Whenever a rule of a certain template is satisfied, it indicates the existence of a COIN instance. For example, template t7 can be checked using the following identification algorithm:

#### Algorithm (t7 identification)

**Input:** Class diagram (CD), interoperable elements (E)

**Process:**

```

1  coinCandidates  $\leftarrow \emptyset$ 
2  coin  $\leftarrow \text{nil}$ 
3  For each  $e \in \{\text{CD.elements}\}$ 
4    If  $(e \in E) \wedge (e.\text{Parent}, e.\text{Parent}.\text{getConstraints} \neq \emptyset)$ 
5      For each  $c$  in  $e.\text{Parent}.\text{getConstraints}$ 
6         $\text{coin} \leftarrow (e, \text{"Structure"}, \text{"inherited constraint"}, \text{value}(c))$ 
7         $\text{coinCandidates} \leftarrow \{\text{coinCandidates} \cup \text{coin}\}$ 
8      End For
9    End If
10 End For

```

**Output:** *coinCandidates*

As described in the algorithm, it takes the class diagram and the list of defined interoperable elements as input, then returns the COIN candidates found in this class diagram. The process starts by initializing the candidate list to null and so for a COIN variable. As mentioned in line 4 of the algorithm, each element in the class diagram that is an interoperable element gets checked against the rules of t7. As seen in lines 6 and 7, if the rules are satisfied, then the COIN variable gets initialized with concrete values for its attributes (i.e., interoperable element, COIN category, COIN type, COIN value). This is repeated for each constraint found in the parent of the interoperable element.

To see the full set of our formal COIN Extraction Templates and their identification algorithms, please refer to Appendix D. Note that due to the limited time and resources available for this Ph.D. work, we provide only an initial basic set of templates, which we do not claim to be complete. However, this set is a component of our framework that can be easily extended with further templates. For example, other templates can be introduced to cover further UML diagrams such as state machine diagrams, composite structure diagrams, and others. Also, our templates focus on extracting the COINs from the low-level design, which can be extended with further templates that target COINs of the

high-level architecture (e.g., templates for identifying the architectural decomposition of a software system and dependencies on the component level).

Next, we will show how we use our COIN Extraction Templates within a method to semi-automate the extraction of COINs from UML diagrams.

### Semi-Automatic COIN Extraction Method

#### Input & Output

The input for our template-based COIN extraction method is a consistent, complete, and up-to-date UML document about the interoperable software unit. This UML document includes structural diagrams (e.g., component diagram, deployment diagram, class diagram, etc.) and behavioral diagrams (e.g., use case diagram, sequence diagram, etc.). Note that it is also possible to create such input through systematic abstraction and analysis of available source code (reverse engineering), but this is beyond the scope of our work. Hence, we assume that the input already exists. With regard to the output, as we mentioned earlier in Subsection 6.2.1, it is the standard ready-to-share document, called the COIN Portfolio.

#### Process

As seen in Figure 26, the input goes through the following four activities in order to result in the desired COIN Portfolio for a software unit:

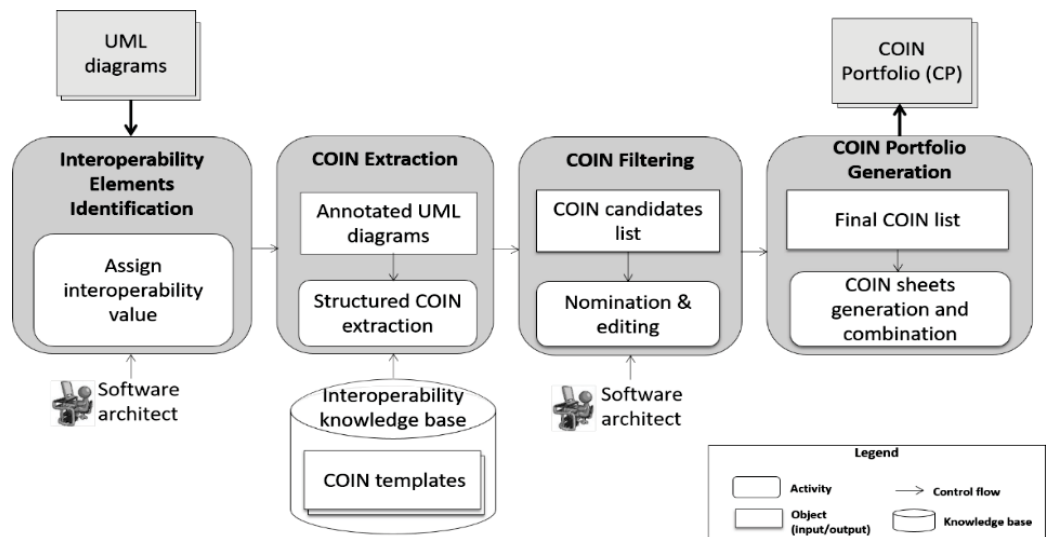


Figure 26

COINs extraction process from UML diagrams

**Identification of interoperable elements.** In this activity, the software architect identifies the UML elements (i.e., components, classes, use cases, or actors) of the software system that are intended to be involved in interoperations with other software systems. This identification happens by assigning an "Interoperability Type" property for the element. This property declares whether the element plays a role in interoperations with other software systems. For example, in the smart

tractor example  $S_1$  described in Section 1.2, the "RemoteSteering" use case in the use case diagram would have the interoperability property declared, and, similarly, this would be declared for the "Direction" data element in the class diagram. Declaring this element property directs the subsequent COIN extraction activities.

**Automatic template-based extraction of COINs.** To enable the proposed automation in this activity, we create an Interoperability Knowledge Base (IKB) and charge it with our predefined COIN Templates. In a software company, the IKB can be maintained and updated by the software architects and domain experts. Thus, the UML diagrams are checked against the predefined templates in the IKB. If the set of rules for a template is met by an interoperable element within a UML diagram, then a COIN candidate for the element is created and added to the list of COIN candidates. Note that the extraction activity starts with checking each UML diagram of the systems to determine whether it contains any element with the interoperability property declared. Then, only the interoperable elements are checked against the predefined COIN templates saved in the IKB. For example, on a Structure COIN template this is done as follows:

- If there is a use case within the use case diagram that is identified as an interoperable element (e.g., the "RemoteSteering" use case within the use case diagram of  $S_1$ ),
- and if it is inherited from another use case (e.g., the "RemoteSteering" use case is one kind of "Steering" use case),
- and if the use case from which it is inherited has a constraint (e.g., the "Steering" use case has a constraint that only one authorized farmer at a time can enable it),
- then Structure COIN will be added to the list of COIN candidates for the use case element (e.g., the "RemoteSteeringI" use case has a constraint that only one authorized farmer at a time can enable it).

**Manual completion (filtering and editing) for the COIN candidates.**

This activity is performed manually by the system architects who have the final word to approve or disapprove the automatically extracted COINs within the final published COIN Portfolio. For example, an architect can detect a redundancy in COINs extracted from a duplicated constraint (e.g., the use case diagram determines that "RemoteSteering" has a 1:1 relation with "Farmer" and the use case inherited from "Steering" has it also, which leads to a duplicate in the multiplicity constraints of the "RemoteSteering" use case in the COIN candidates list). Furthermore, architects can manually add more COINs to the Portfolio if they consider them important and useful to share with clients. Further guidance for this manual adding of COINs to the list can be derived from the COIN Model and its three dimensions (see Sections 5.2 and 5.30). The specific COIN attributes described in the

model directs the architects in checking what could be missing in the automatically generated portfolio. Furthermore, in Subsection 6.4.1, we will propose detailed guidelines and a cheat sheet to direct the manual extraction of COINs from UML diagrams.

**Automatic generation of the COIN Portfolio.** Finally, the approved and manually added COINs are bundled together and categorized according to the elements to which they are related. Then they are documented in a standard defined form that is ready to be shared with clients (i.e., a structured COIN Portfolio with a detailed sheet for each COIN as described in Section 5.4). Thus, clients can use these for their interoperability analysis task if they are interested in interoperating with the software system. The COIN Portfolio can always be updated by the architect as needed. In practice, COIN Portfolios would be shared online or saved in a Portfolio Repository that the software provider could maintain and for which it could provide licensed access.

### Add-In Tool for Enterprise Architect

Tool goal	This subsection presents a technical contribution (i.e., a software tool called CoinsExtractor) that we developed in the context of this thesis in order to facilitate our template-based method for the automatic extraction of COINs from architecture UML documents (depicted in Figure 26). It brings our template-based method to life and makes it applicable in practice. It takes some burden off the architects' shoulders through its easy-to-use interfaces, which effectively identify and share the conceptual interoperability constraints of the software systems.
Feature overview	The CoinsExtractor tool is an add-in for the Enterprise Architect modeling tool and implements our predefined COIN Extraction Templates. It enables the extraction of relevant COINs about determined interoperable elements from existing UML diagrams. Also, it supports architects in efficiently reviewing, updating, approving, or deleting the extracted COINs. Finally, the tool creates the COIN Portfolio, which arranges the COINs according to their categories to allow the conceptual interoperability analysis task. A tool demo is available at [Abu17].

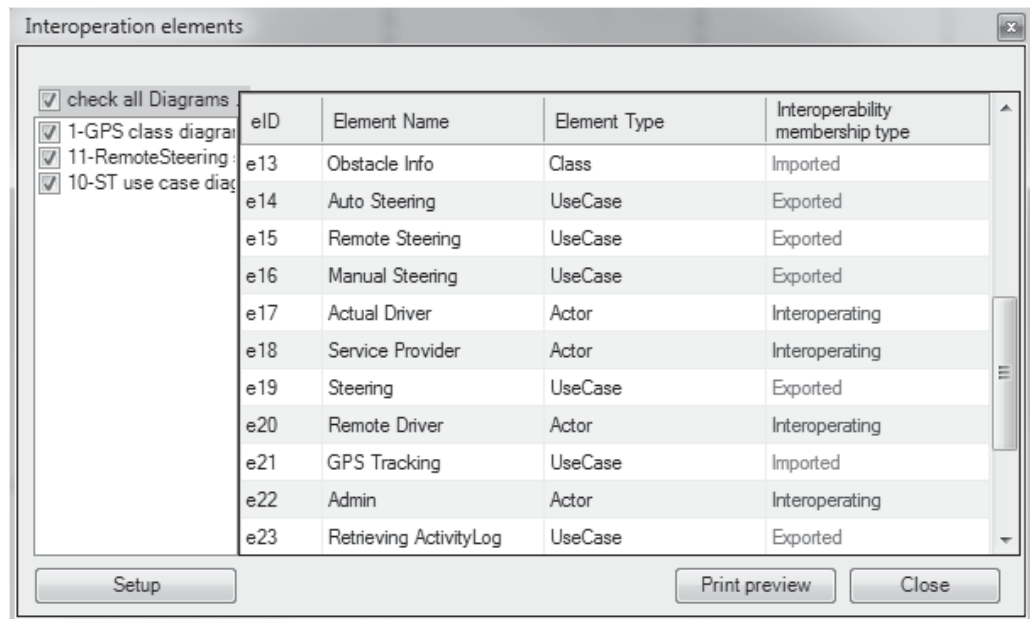
### Tool Features and Example Results

Below we will describe the functional features of our tool and explain them with example results from the smart tractor example that we have been using throughout the chapters.

**F1: Interoperable element annotation.** This feature allows architects to annotate any UML component, class, use case, or actor with an "Interoperability Type" property. Once an element gets annotated with this property, all its instances in all diagrams get it automatically. Figure 27 shows a list of all annotated elements of  $S_1$ , where there are three



different possible values for the interoperability property: (1) **interoperating**, for external software units that interoperate with the system (e.g., the Remote Driver mobile app  $S_3$  that is intended to interoperate with the smart tractor  $S_1$ ), (2) **exported**, for elements that the system provides to interoperating parties (e.g., the RemoteSteering function provided by  $S_1$ ), or (3) **imported**, for elements that the system acquires from interoperating parties (e.g., the GPS tracking service used by  $S_1$ ). This step requires one-time manual effort to capture the architects' knowledge about the software system, which will be reused in all future interoperations.



eID	Element Name	Element Type	Interoperability membership type
e13	Obstacle Info	Class	Imported
e14	Auto Steering	UseCase	Exported
e15	Remote Steering	UseCase	Exported
e16	Manual Steering	UseCase	Exported
e17	Actual Driver	Actor	Interoperating
e18	Service Provider	Actor	Interoperating
e19	Steering	UseCase	Exported
e20	Remote Driver	Actor	Interoperating
e21	GPS Tracking	UseCase	Imported
e22	Admin	Actor	Interoperating
e23	Retrieving ActivityLog	UseCase	Exported

Figure 27 CoinsExtractor example of a list of interoperable elements

**F2: Automatic COIN extraction from UML diagrams.** The tool saves the architects effort by automatically extracting only relevant pieces of information, the COINs, about the interoperable elements from the whole UML document. It also preserves consistency in how a COIN is documented (across diagrams), and what is being extracted for sharing and how it is documented (across projects). The tool parses the UML input and checks it against our implemented predefined COIN Extraction Templates, which we explained earlier in this subsection (see Table 8). Once a COIN instance is found, the tool adds it to a list of candidates, each of which is detailed in a COIN sheet. Figure 28 shows a sheet for a context COIN for the AutoSteering function, which states that this function is allowed for “Actual Driver” only. This COIN has been extracted from the use case diagram of  $S_1$  (see the back side of Figure 28). The architect can determine the weight of the COIN (e.g., high, medium, and low) and can attach other files to it (e.g., a formal notations file). She can also enrich the COIN sheet by adding information about the expected consequences of not satisfying the COIN for interested third-party clients.

**F3: Automatic categorization for extracted COINs.** The tool provides two views for the extracted COINs. The first is a diagram-based view, where architects can navigate the COINs based on the diagram which they have been retrieved. This helps architects trace the source of the interoperability constraints in their systems (see the pane tree on the left side of Figure 28). The second view is an element-based view, where COINs can be navigated according to the interoperable element to which they are related. This view is also used in the final shared COIN Portfolio to help third-party clients focus on the COINs of desired elements only.

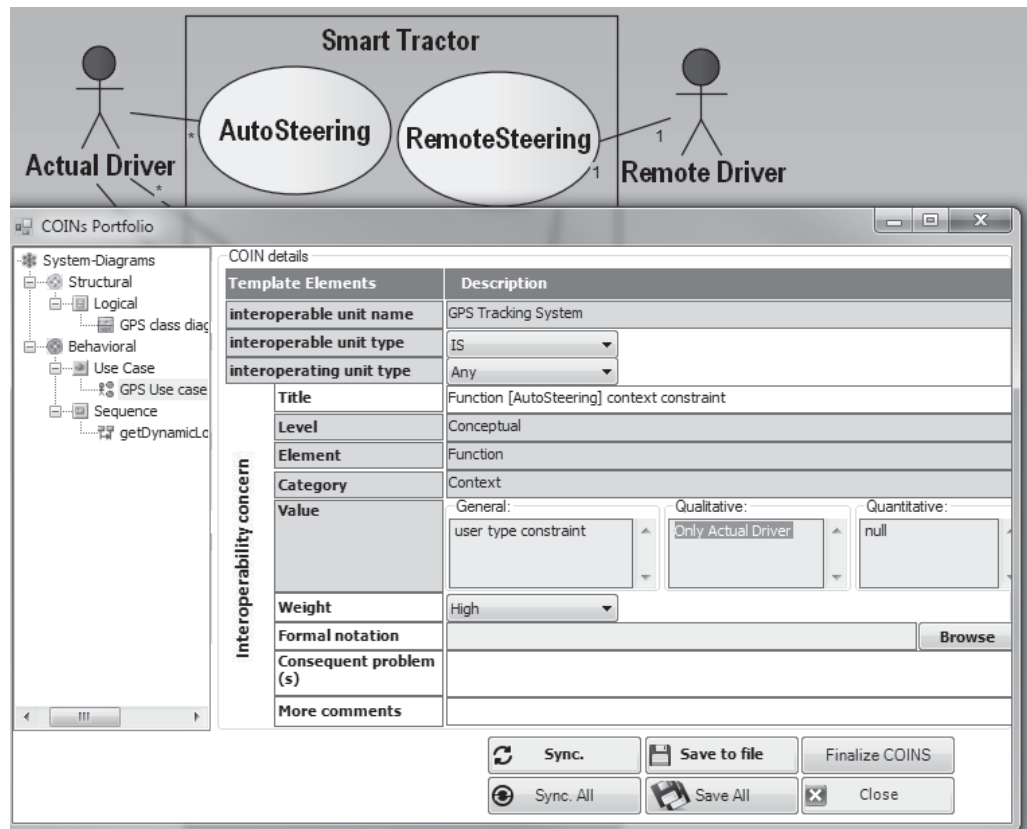


Figure 28

CoinsExtractor example of an extracted Context COIN from a use case diagram

**F4: Manual COIN Reviewing.** The CoinsExtractor tool realizes the manual completion and filtering of COINs by enabling the architects to efficiently approve or delete the automatically extracted COINs. Furthermore, they can review the COIN sheets and edit them as needed (see the right side of the portfolio finalization screen in Figure 29). In addition, if constraints are missing in the automatically extracted list, the tool enables architects to manually add more COIN instances to any of the interoperable elements (see the bottom side of the portfolio finalization screen in Figure 29).

As can be seen in Figure 29, the COIN Portfolio has two parts: One part is dedicated to the actual COINs of exported elements that third-party clients need to pay attention to when aiming to interoperate with the

software system. The other part is about expected COINs from imported elements that architects need to look for when selecting a software system to interoperate with. The first part is shared with third-party clients, whereas the second part is kept internal for the software provider's own interest and use when extending his system. Both parts share the same goal of supporting conceptual interoperability analysis to detect conceptual mismatches early in integration projects.

Interoperability membership type	Interoperability element(s)		COINs				Decision	
	Element	Object Name	COIN category	COIN title	COIN sheet (details)		Approve	Remove
Exported	Data	Dimension	Structure	Data [Dimension] structure constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
			Structure NL	Data [Dimension] structure constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
	Function	Get Dynamic Object Location	Context	Function [Get Dynamic Object Location] context constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
			Structure	Function [Get Dynamic Object Location] structure constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
Imported	Data	Dimension Request	Structure NL	Data [Dimension Request] structure constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
		Object & Location	Structure NL	Data [Object] relation constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>
		User Request	Structure	Data [User Request] structure constraint	<a href="#">Click here</a>		<input type="checkbox"/>	<input type="checkbox"/>

Buttons: Close, Add COIN, Save changes, Create COINs Portfolio, Undo

Figure 29 CoinsExtractor example of a system's COIN Portfolio

**F5: Automatic generation for the COIN Portfolio.** The CoinsExtractor relieves architects from having to manually gather and classify all extracted and manually added COINs. It generates the final, ready-to-share, web-based COIN Portfolio (see Figure 30), which can be easily integrated into the API documentation.

Interoperability element(s)		COIN Category	COIN Title	COIN sheet (details)
Element	Object			
Data	Activity Log	Structure	Data [Activity Log] structure constraint	<a href="#">click here</a>
		Structure NL	Data [Activity Log] dynamic constraint	<a href="#">click here</a>
	Night Driving	Dynamic NL	Data [Night Driving ] structure constraint	<a href="#">click here</a>
	Request Response	Structure NL	Data [Request Response] dynamic constraint	<a href="#">click here</a>
	Steering	Dynamic NL	Data [Steering] dynamic constraint	<a href="#">click here</a>
Function	Auto Steering	Context	Function [Auto Steering] context constraint	<a href="#">click here</a>
		Structure	Function [Auto Steering] structure constraint	<a href="#">click here</a>
	Remote Steering	Context	Function [Remote Steering] context constraint	<a href="#">click here</a>
		Structure	Function [Remote Steering] structure constraint	<a href="#">click here</a>

Figure 30 CoinsExtractor example of generated COIN Portfolio

## Tool Implementation

**Development technology.** We developed the CoinsExtractor using C#.NET. We implemented it as an add-in for Enterprise Architect (EA) [Spa], which is one of the most powerful, widely used architecture tools. EA offers an extension API that we used.

**Architectural overview.** The CoinsExtractor has a multi-layered architecture (see Figure 31). The *Presentation Layer* offers the user interaction between the tool and architects through windows for identifying the interoperable elements, filtering the extracted COINs, and adding additional ones. It also visualizes the results of such actions in a table of interoperable elements and a table of COINs. The second layer is the *Business Layer*, which includes: (1) The *Business Logic* is responsible for processing the UML diagrams and for extracting the COINs from them. It contains the working units of the tool, such as the COINs Extractor class and the Portfolio Generator class. (2) The *Business Entities* are the predefined data structures needed for the tool's functionalities (e.g., the Node position, which we use to determine the distribution) and our COIN templates. The *Data Access Layer* reads input from the database and writes results to the output file. The tool's modularity allows developers to extend it with more COIN templates to cover further constraints. A template can be defined as a rule with condition(s) to check and values to assign for each COIN aspect.

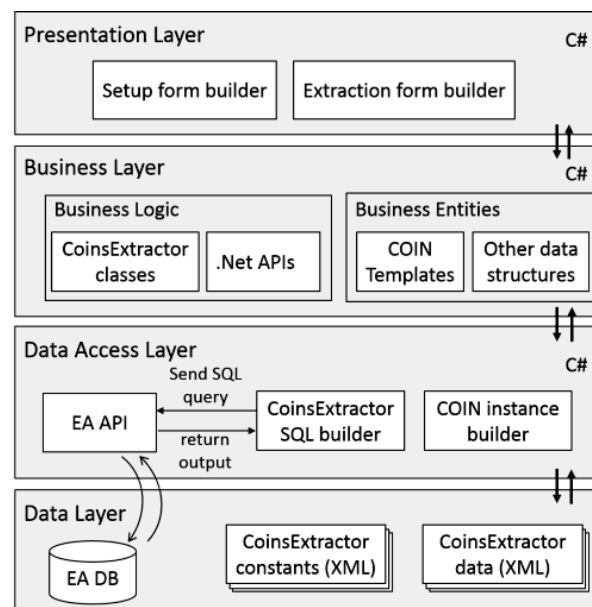


Figure 31

CoinsExtractor architectural overview

**Extraction process.** We have implemented the COIN extraction as a sequential series of algorithms each serving a different scenario. In the first step, all diagrams are read using SQL Query and saved to an XML output. Then the interoperable elements' constraints are inspected by checking the conditions regarding each element's properties, its

relations, and its containing diagram. Specifically, each type of constraint has a special extracting algorithm. Finally, the extracted COINs get fetched from the XML file to a user-friendly table for architects to review and edit as needed.

**Challenges.** Some of the challenges we faced in building an add-in for EA were: (1) The inability to extend existing windows of EA (e.g., we could not add the Interoperability Type of the elements in their already existing properties window, but had to define it as a tagged value). (2) The EA documentation provides good directions on how to use its APIs, but it lacks the what (i.e., a conceptual representation of the elements, their features, and constraints). This required us to reverse engineer the EA database to find the properties of elements distributed among tables. (3) EA APIs do not help to determine the distribution feature of components. Hence, we implemented a method for calculating the physical position of elements using the coordinate plane, then checking if it is within the boundaries of multiple nodes. In addition, the quality of the extracted COINs and the produced portfolio depend on the quality of the UML input in terms of consistency, completeness, and stability.

**Scalability analysis.** The complexity of the algorithms we implemented in CoinsExtractor varies between  $O(1)$ ,  $O(n)$ , and  $O(n^2)$ . For example, reading a block of data from the EA Repository has a complexity of  $O(1)$ , while pre-processing the UML project has a complexity of  $O(n^2)$ : one  $n$  for iterating models, and the second for retrieving each model's elements. However, the COIN extraction algorithms have a complexity of  $O(n)$ . In an attempt to improve the tool's performance, we decided to use completely pure SQL statements to access the models' elements instead of depending on EA objects (e.g., ObjectType, Connector, Diagram, etc.). These EA objects offer indirect and sequential access to element information. Hence, using SQL queries retrieves the needed constraints and properties of the elements directly.

## Summary

In this subsection, we described our COIN Extraction Templates from UML diagrams, then explained the template-based extraction method. This method is part of the proactive preparation component of our framework that serves providers of black-box software units. It helps to identify interoperability-relevant conceptual constraints from in-house architecture documents and supports sharing them in a standard document. We support our semi-automatic method with a software tool that we implemented as an add-in for the Enterprise Architect modeling tool.

### 6.3.2 Extracting COINs from the Text of API Documents

In the previous subsection, we presented the first COIN extraction method (for UML diagrams) in our preparation component of the COINA Framework. In this subsection, we present the second COIN extraction method (for natural language text in API documents) in this preparation component.

**Method goal** The goal of this method is to support extraction of unstructured conceptual constraints from natural language text in API documents of interoperable software units. This helps the providers of interoperable software units identify the interoperability-relevant conceptual constraints of their units to merge them in the COIN Portfolio. However, it can also be used by third-party clients to identify the conceptual constraints of software units from their API documents if the providers do not offer the portfolio. In either way, by facilitating the extraction of COINs from the text of API documents, we pave the way towards effective detection of conceptual mismatches. As we previously described in Section 1.2, extracting such conceptual information manually from the verbose text is a challenging task. It requires knowledge and experience about the COIN types and linguistic skills to sift the text and document the information about the extracted constraints. Accordingly, analyzing the text of API documents manually is a tedious and time-consuming task that critically depends on the experience of the person performing it.

**Contribution** The key contribution of our method is support for effectively and efficiently identifying conceptual interoperability constraints from natural language sentences using machine learning techniques. This contribution includes building a “COIN Corpus” (i.e., a repository of manually labeled sentences that is used for training the machine learning algorithms) and a “Machine Learning COIN Classification Model” (i.e., a model that can automatically predict the existence of a COIN type in a natural language sentence of an API document). We aid this extraction by implementing an add-in for the Chrome web browser.

In the next part, we will start with the research methodology we followed in developing our method, and then we will describe the resulting COIN Corpus, the COIN Classification Model, and the results of the fully automated extraction. Finally, we will present the supporting tool along with examples.

Note that the author of this Ph.D. thesis has contributed the whole design of the multiple-case study and the related experiments, the execution of each case, the analysis of the experimentation results, the design of the add-in tool’s goal, functionalities, architecture, technologies used, and its evaluation study. However, the author also supervised two master theses: The first thesis work [Abu16a] repeated the execution of the studies, implemented experiments using the



different ML algorithms, and provided a descriptive analysis of the results. The second thesis work [Nai17] implemented the designed ML-based tool and executed the designed evaluation study for its acceptance. Also, note that the author of this Ph.D. thesis has published most of this subsection in [AAHR16], [AAR16], and [AANR17].

## Research Methodology

In this study, we wanted to systematically reveal the potential of automating the extraction of COINs from natural language text in API documents using ML techniques. Therefore, we performed a multiple-case study and followed it up with experiments to investigate the potential of ML techniques. The websites of the original API documentations of the selected cases in this multiple-case study are available on the web page of this study [Abu14b].

### Goal and Research Questions

Our research **goal** formulated in terms of the GQM goal template [VSBCR02] was: *to support the conceptual interoperability analysis task for the purpose of improvement with respect to effectiveness and efficiency in identifying the COINs from the viewpoint of software architects and analysts in the context of analyzing the text of API documentations within software integration projects.* We translated this goal into the following research questions:

**RQ1:** What are the existing conceptual interoperability constraints (COINs) in the text of API documentation?

This question explores the current state of COINs in real API documents. It also aims at building the ground truth dataset (i.e., the COIN Corpus representing a repository of sentences labeled with their COIN class). This forms a main building block towards the envisioned automatic extraction idea.

**RQ2:** How *effective* and *efficient* would it be to use ML techniques in automating the extraction of COINs from the text in API documentations?

This question explores the actual benefits of utilizing ML in supporting software architects and analysts in analyzing the text. It aims at building a classification model that will be evaluated through well-known ML classification algorithms.

This question explores the actual benefits of utilizing ML in supporting software architects and analysts in analyzing the text. It aims at building a classification model that will be evaluated with the help of well-known ML classification algorithms.

In order to achieve the stated goal and answer the aforementioned questions, we performed our research in two main parts as follows:

**Research Part 1 (Multiple-case study).** In this part, we systematically explored the state of COINs in six cases of API documentations. The result of this part is a ground truth dataset (i.e., the COIN Corpus).

**Research Part 2 (Experiments).** In this part, we started by using the ground truth dataset, which resulted from the previous part, in building the ML COIN Classification Model. Afterwards, we investigated the accuracy of different ML classification algorithms in identifying the COINs in the text using our model.

Next, we will detail the design and execution process for both research parts. Afterwards, we will present their results.

### Multiple-case study: COIN Corpus (Ground Truth Dataset)

#### Study Design

**Study goal.** We intend to answer the first research question RQ1 stated in Subsection 06.3.2. In order to do so, we needed to examine real-world API documentations to discover the state of conceptual interoperability constraints in them.

**Research method.** We decided to perform a multiple-case study with literal replication of cases from different domains. Such a method helps to collect significant evidence and draw generalizable results. We followed the process proposed by Runeson et al. [RH09]. Our case study is also considered action research as we, the researchers, participated in it.

**Case selection.** To ensure systematic selection of cases of API documentations, we considered the following selection criteria:

**SC1:** Mashup Score. This is a published statistical value<sup>1</sup> for the popularity of a web service API in terms of its integration frequency into new bigger APIs.

**SC2:** API Type. This can be either a web service API or a platform API.

**SC3:** API Domain. This is the application domain for the considered API document (e.g., social blogging, audio, software development, etc.).

**Analysis unit.** Our case study has a holistic design, which means that we have a single unit of analysis. This unit is “the sentences in API documents that include COIN instances”. To document and maintain the analyzed sentences, we designed a data extraction sheet, which we implemented as an MS Excel sheet. This sheet consists of demographic fields (i.e., API name, date of retrieval, mashup score, API

type, API domain, and number of sentences) and analysis fields (i.e., case id, sentence id, sentence textual value, and the COIN class).

**Study protocol.** Our multiple-case study protocol includes three main activities that are adapted from the process proposed by Runeson. The study activities are case selection, case execution, and cross-case analysis, as summarized in Figure 32 and described in detail below.

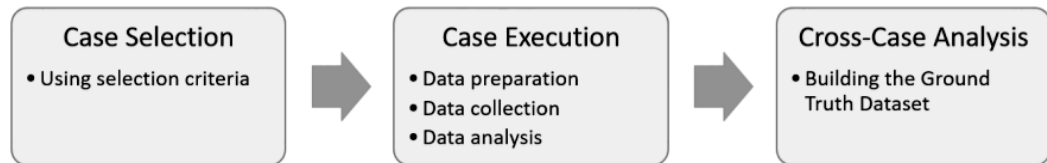


Figure 32 Multiple-case study process

## Execution and Results

Based on our predefined case selection criteria, we chose six API documents in August 2015; four documents of the web services type (i.e., SoundCloud, GoogleMaps, Skype, and Instagram) and two of the platform type (i.e., AppleWatch and Eclipse-Plugin Developer Guide). These cases cover different application domains (i.e., social micro-blogging, geographical location, telecommunication, social audio, and software development environment). Regarding the mashup criteria, our four cases of web service APIs were chosen to cover a wide range of scores starting from 30 for Skype and ending with 2582 for GoogleMaps. After selecting the cases, we executed each case as follows:

**Data Preparation.** We started this step by fetching the API documentation for the selected case from its online website. Then we read the documents and determined the web pages that had textual content offering conceptual software description and constraints (e.g., Overview, Introduction, Developer Guide, API Reference, Summary, etc.). Subsequently, we started processing the text on the selected web pages by performing manual and automatic filtering. For more details, please refer to our published paper [AAR16].

**Data Collection.** In this step, we cut the content of the text file resulting from the previous step into single sentences within our designed data extraction sheet (.xsl file). We completed all the fields of the data sheet for each sentence except for the “COIN class” field, which we did in the next step. Note that we maintained data storage, where we stored the original HTML web pages of the selected API documentations, their text file, and their Excel sheet. This enables later replication of our work as documentations get changed frequently.

**Data analysis.** We manually analyzed each sentence in the extraction sheet and carefully assigned it a COIN class. This classification was based on interpretation criteria consisting of the COIN Model with its six classes (i.e., Syntax, Semantic, Structure, Dynamic, Context, and Quality). We added a seventh class for sentences with no COIN instance (i.e., Not-COIN class). For example, a sentence like “A user is encapsulated by a read-only Person object.” was classified as “Structure”. On the other hand, “You can also use our Sharing Kits for Windows, OS X, Android or iOS applications” was classified as “Not-COIN” as it does not express a conceptual constraint, but rather a piece of technical information.

The result of this step was a very critical point towards our envisioned automatic COIN extraction idea. Hence, the data analysis was performed by two researchers. It was first performed by the author of this Ph.D. thesis and then repeated by the master student she supervised. They independently classified all sentences for each case, then compared their classification decisions in multiple discussion sessions and resolved conflicts based on consensus. Obviously, the case execution process consumed time and mental effort, especially in the data analysis step.

Table 9 summarizes the distribution of the collected sentences among the cases along with the effort we spent executing them (in hours).

Table 9

Case share of sentences and execution time

API Document	Total number of sentences	Total execution efforts (Hours)
Sound Cloud	219	7.7
GoogleMaps	473	6.5
AppleWatch	360	8.0
Eclipse Plugin	651	12.0
Skype	325	4.5
Instagram	255	4.8
<b>Total</b>	<b>2283</b>	<b>43.5</b>

**Cross-Case Analysis** (*Answering RQ1: What are the types of existing COINs in the text of current API documentations?*). After executing all cases, we arranged the incrementally classified sets of sentences from all cases (i.e., 2283 sentences) into one repository, which we call the **ground truth dataset** or the **COIN Corpus**, in ML terminology. We developed two versions of this dataset as follows:

- **Seven-COIN Corpus**, where each sentence belongs to one of the seven classes (i.e., Not-COIN, Dynamic, Semantic, Syntax, Structure, Context, or Quality).
- **Two-COIN Corpus**, where each sentence belongs to one of two classes rather than seven (i.e., COIN or Not-COIN). In fact, the

Two-COIN Corpus is derived from the Seven-COIN Corpus by abstracting the six COIN classes into one class.

In Table 10, we present the differences between the two corpora with example sentences.

Table 10 Example of content in the Seven-COIN and Two-COIN Corpus

Sentence ID	Sentence	Seven-COIN Class	Two-COIN Class
s1	You can also use our Sharing Kits for Windows, OS X, Android or iOS applications.	Not-COIN	Not-COIN
s2	When it is finished manipulating the object, it releases the lock.	Dynamic	COIN
s3	A user is encapsulated by a read-only Person object.	Structure	COIN
s4	A user's presence is a collection of information about the users' availability, their current activity, and their personal note.	Syntax	COIN
s5	A dynamic notification interface lets you provide a more enriched notification experience for the user	Semantic	COIN
s6	This service is not designed to respond in real time to user input	Context	COIN
s7	Your interfaces need to display information quickly and facilitate fast navigation and interactions.	Quality	COIN

The aim of building these two versions of the corpus was to better investigate the ML performance results when using the different versions of the ground truth dataset later in our research. We will explain this in more detail in the experiments subsection.

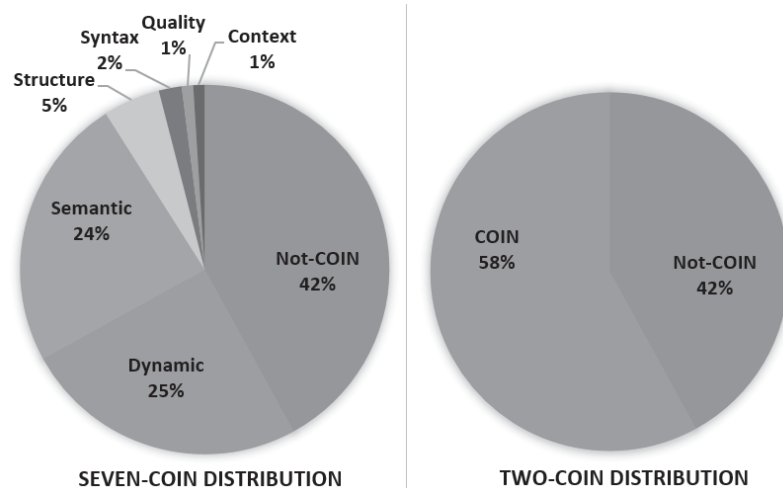


Figure 33 COIN share in the Ground Truth Dataset

*COIN-share in the contributed ground truth dataset.* In Figure 33, we illustrate the distribution of sentences among the COIN classes within the Seven-COIN Corpus (on the left). It can be observed that the Not-COIN class, which expresses technical constraints rather than conceptual ones, is dominant among the other six COIN classes (i.e.,

42%). The Dynamic and Semantic classes have the second and third biggest shares, respectively. Remarkably, the Structure, Syntax, Quality, and Context shares are very low, with convergent shares ranging between 1% and 5%. An aggregated share for all COIN classes is shown in the Two-COIN Corpus (on the right of Figure 33).

*COIN share in the cases.* On a finer level, we investigated the state of the COINs in each case rather than in the whole ground truth dataset. We found that the content of each API document was focused on the Not-COIN, Dynamic and Semantic classes, similarly as in the aggregated findings on the complete dataset seen in Figure 33. For example, in the case of the AppleWatch documentation, 40.8% of the content is for Not-COIN, 26.1% for Dynamic, and 25% for Semantic. Add to this that all cases had less than 10% of their content in the Structure, Syntax, Quality, and Context classes (e.g., Eclipse-Plugin gave them 8.5%).

*Observed patterns.* For the dominant classes in the ground truth dataset, we observed in a considerable number of sentences for the Not-COIN, Semantic, and Dynamic classes a number of patterns in terms of frequently occurring terms and sentences. We envision that using the patterns in combination with the Bag-of-Words (BOWs) technique in future experiments would enhance the results of automatic COIN identification. For more details about these patterns, please refer to [AAR16] and [Abu16a].

## Threats to Validity

**Case bias.** To obtain significant results and draw generalizable conclusions, we included multiple cases for building the ground truth, which plays a prominent role in our research. We literally replicated six API documents (i.e., SoundCloud, GoogleMaps, Skype, Instagram, AppleWatch, and Eclipse-Plugin Developer Guide) from two different types (web service APIs and platform APIs). Thus, the results are very likely to be representative of current API documentations. However, further cases with larger number of sentences and constraints are required to generalize the results and observe the changes over time.

**Case size bias.** Due to resource limitations (i.e., time and manpower), we were unable to analyze the large API documents completely. However, we were careful with respect to selecting inclusive parts of these large documents. For example, out of the huge document of Eclipse APIs, we covered the Plugin part.

**Researcher bias.** To build our ground truth dataset in a way that guarantees accuracy and impartiality of the results, we had two researchers separately replicate the manual classification of the cases' sentences based on the COIN Model as interpretation criteria. In



multiple discussion sessions, the researchers compared their classification decisions and resolved conflicts based on consensus.

## Experiments: Machine Learning COIN Classification Model

### Experiments' Design

**Goal of the experiments.** This part of our research aims at answering the second research question RQ2, which we stated earlier in Subsection 06.3.2. In order to do so, we needed to examine ML techniques to discover their potentials in supporting architects and analysts in automatically identifying the COINs in the text of API documents.

**Research method.** We built a classification model and ran multiple experiments employing different ML text classification algorithms. This method enables comparing the results of the algorithms and drawing solid conclusions about the ML advantages in addressing the challenges of manual interoperability analysis.

**Evaluation method and metrics.** We used k-fold cross-validation, which we explained in the background (Section 2.4), with  $k = 10$ . Regarding the evaluation metrics used for classification accuracy, we used the following commonly used measures [Pow11]:

- *Precision*: the ratio of sentences classified correctly by the classification algorithm to the total number of sentences it classified either correctly or incorrectly.
- *Recall*: the ratio of sentences classified correctly by the classification algorithm to the total number of sentences in the corpus.
- *F-measure*: the harmonic mean of precision and recall, which is calculated as:  $(2 * \text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$ .

**Experiments' protocol.** Our experiments protocol included three main activities: feature selection, feature modeling, and evaluation of the ML algorithms. We illustrate this protocol in Figure 34, and describe it in detail below. We ran this protocol twice, once for the Seven-COIN Corpus and once for the Two-COIN Corpus. Also, we performed follow-up experiments, in which we ran the same protocol with automatic tuning for the parameters of the evaluated algorithms to see if we would get better results.

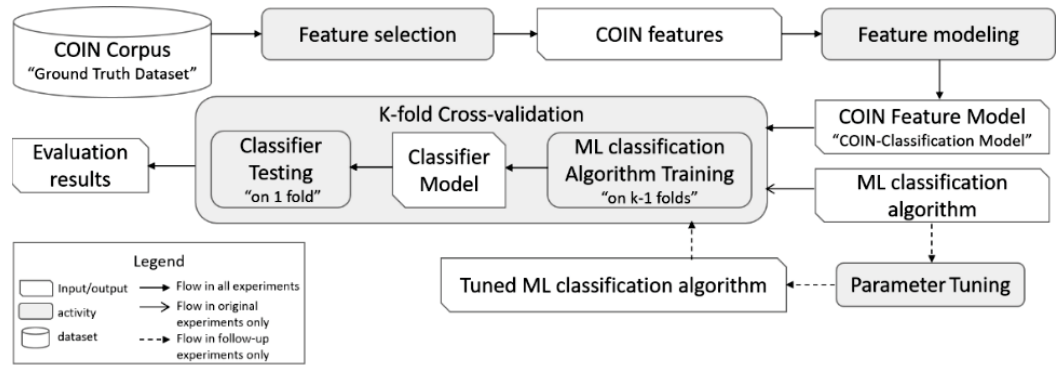


Figure 34 Process of the ML experiments

## Execution and Results

The experiments were executed under the supervision of this Ph.D. author by the master student [Abu16a]. In a nutshell, the experiment configuration and runs were executed on Weka v3.7.11<sup>2</sup>, which is a suite of ML algorithms written in Java with result visualization capabilities. The execution started with the processing of the textual sentences in our contributed dataset (i.e., the COIN Corpus) using natural language processing (NLP) techniques. The processing included tokenizing sentences into words, lowering cases, eliminating noise words (e.g., is, are, in, of, this, etc.), and stemming words into their root format (e.g., encapsulating and encapsulated are returned as encapsulate).

**Feature selection.** After processing the text, we identified the most representative features or keywords for the COIN classes within the COIN Corpus using the BOW and N-Gram approaches, which we explained in the background section. That is, each sentence was represented as a collection of words. Then each single word and each n-combination of words in the sentence were considered as features, where N was between 1 and 3. For example, in a sentence like “A user is encapsulated by a read-only Person object”, the word “encapsulate” and the combination “read-only” were considered as two of its features. The output of this step was a set of features for the COIN Corpus.

**Feature modeling (Building the ML COIN Classification Model).** In this stage, the whole COIN Corpus was transformed into a mathematical model. That is, it was represented as a matrix where the headers contained all features extracted from the previous phase, while each row represented a sentence of the corpus. Then we weighted the matrix, with each cell [row, column] holding the weight of a feature in a specific sentence. For weighting, we used the Term Frequency-Inverse Document Frequency (TF-IDF), which is often used for text retrieval. The result of this was the COIN Feature Model (or the COIN

<sup>2</sup> Weka: <http://www.cs.waikato.ac.nz/ml/weka>

Classification Model), which is a reusable asset preserving knowledge about conceptual interoperability constraints in API documents.

**Evaluating the COIN Classification Model (K-fold cross-validation).** We selected a number of well-known ML text classification algorithms (e.g., Naïve Bayes versions, Support Vector Machine, Random Forest Tree, K-Nearest Neighbor KNN, and others). Then, we put each algorithm through the cross-validation, which included two steps. The first step was supervised *training*, where the ML algorithm got to learn k-1 folds of our COIN Classification Model with their COIN classes for k rounds. The result of this training was a classifier model. The second was *testing*, where the resulting classifier got tested for predicting the COIN class for 1 fold for k rounds.

**Evaluation results (Answering RQ2: How effective and efficient would it be to use ML techniques for automating the extraction of COINs from the text in API documentations?).** The results are as follows:

*Effectiveness of identifying COINs using ML algorithms.* We report the effectiveness results in terms of accuracy metrics for two cases:

- Seven-COIN Corpus Case. The evaluation results showed that the best accuracy in automatically identifying seven classes of interoperability constraints in the text was achieved by the ComplementNaïveBayes algorithm (see Table 11). It achieved 70.4% precision, 70.2% recall, and 70% F-measure. Second place went to the NaïveBayesMultinomialupdateable algorithm with about 5% less accuracy than the former algorithm. For the other algorithms, the values for accuracy and F-measure were between 62.8% and 59.0%. The worst results were obtained with the KNN algorithm.
- Two-COIN Corpus Case. When applying the same algorithms on the Two-COIN Corpus, we obtained better results. In particular, accuracy increased by almost 11% compared to the results in the Seven-COIN case with the ComplementNaïveBayes algorithm. That is, precision increased to 81.9%, recall to 82%, and F-measure to 81.9%. Similar to the previous case, NaïveBayesMultinomialupdateable came in second and the 2-Nearest Neighbor algorithm had the worst results (see Table 11).

*Efficiency of identifying COINs using ML algorithms.* Obviously, the machine beats human performance in terms of the time spent for analyzing the text. As we mentioned earlier, analyzing the documents cost us about 44 working hours, while it took the machine much less time. (For example, training and testing the NaïveBayesMultinomialupdateable algorithm took about 5 seconds on our complete corpus with 2283 sentences). This efficiency would improve even more when using a faster and more powerful CPU (we ran the

experiments on a machine with an Intel core i5 460 M CPU with 2.5 GHZ speed).

Table 11

COINs identification results using different ML algorithms

ML Algorithm	Seven-COIN Corpus		Two-COIN Corpus	
	Precision	Recall	Precision	Recall
Complement NaïveBayes	70.4%	70.2%	81.9%	82.0%
NaïveBayes Multinomialupdateable	66.0%	65.1%	81.9%	82.0%
Support Vector Machine	59.3%	60.0%	75.7%	75.7%
Random Forest Tree	60.4%	56.3%	73.7%	73.9%
Simple Logistic	52.5%	54.4%	68.2%	68.4%
KNN K=1	54.8%	45.5%	64.2%	52.3%
KNN K=2	49.8%	36.1%	64.4%	48.7%

### Follow-up Experiments with Parameter Tuning for ML Algorithms

We further investigated the effectiveness of our COIN Classification Model through our follow-up experiments. In these experiments, we performed an optimization by tuning the parameters of the ML algorithms. In particular, we applied the Grid Search method because our dataset is relatively small. An example of the parameters that can be tuned is the degree  $d$  of the polynomial kernel of SVM algorithm [TK01]. With regards to the implementation, we used the scikit-learn python library [PVG+11].

Table 12

The results of classification after parameter tuning

ML Classification Algorithm	Two-COIN Corpus	
	Precision	Recall
SVM (Polynomial, $d = 3$ )	87.0%	87.0%
SVM (Linear)	81.0%	81.0%
LogisticRegression (L2)	81.7%	82.0%
LogisticRegression (SGD)	80.0%	79.0%

The results of the follow-up experiments showed no improvement for the classification effectiveness (i.e., accuracy) in the case of the Seven-COIN Corpus. However, we got noticeably higher effectiveness in the case of the Two-COIN Corpus compared to the results reported in the previous section. Table 12 shows the best performing algorithms with their accuracy improvements achieved by performing the parameter tuning. As seen, the highest accuracy result (i.e., F-measure = 87%) was obtained from the Polynomial SVM algorithm [TK01] with kernel degree = 3. This result is 5% higher than the best result achieved without parameter tuning. The other best performing tuned algorithms had achieved almost the same accuracy as the best achieved without

parameter tuning. That is, the Linear SVM [TK01], Logistic Regression with L2 regularization level [Ng04], and Logistic Regression with Stochastic Gradient Descent (SGD) [Zha04] achieved F-measure of 81%, 81.8%, and 79.5% respectively.

While recall, precision, and F-measure inform us about the classification accuracy of the algorithms using our model, these measures do not take into account the true negatives [DG06]. Hence, we further investigated the False Positive Rate - True Positive Rate (FPR-TPR) curves for the different binary ML classification algorithms that we tuned their parameters (see Figure 35). The curve of each algorithm is a plot of the trade-off between the algorithm ability to correctly detect the COINs (i.e., TPR or recall) and the number of incorrect alarms for COINs (i.e., FPR). Thus, the area under curve (AUC) [HM82], which has a value range from 0 to 1, measures how each algorithm is effective in segregating the two classes (i.e., COIN and Not-COIN). Therefore, the larger the AUC (or the closer the curve to the upper left corner), the higher the algorithm's probability in correctly classifying the sentences. Note that, the dashed diagonal line in the figure represents the curve for a random classification algorithm that has AUC of 0.5. It is commonly used as a baseline to see if the other algorithms are useful. That is, an algorithm with AUC larger than 0.5 is considered as nonrandom binary classification algorithm. Hence, the depicted algorithms in the figure show good classification effectiveness when compared to the random algorithm, which supports the validity of our COIN Classification Model and the robustness of our automation idea.

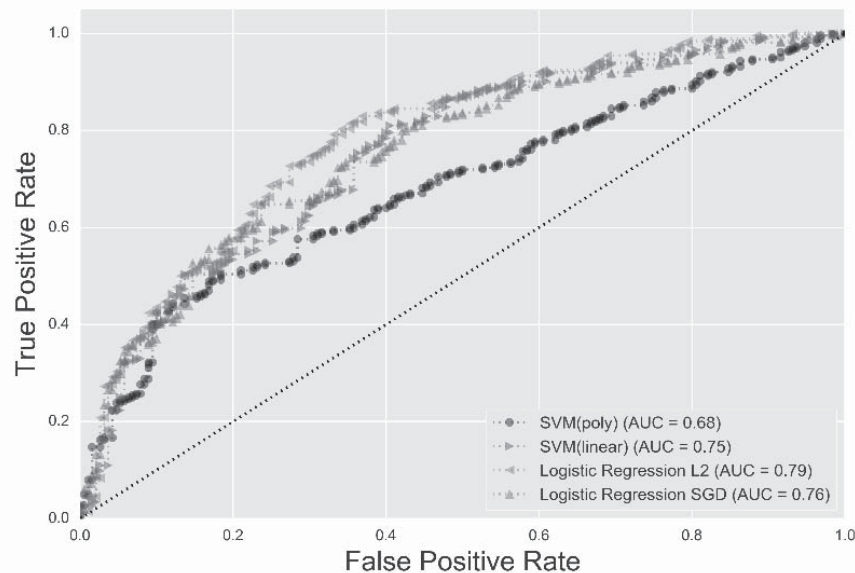


Figure 35

The area under FPR-TPR curve for classification algorithms after parameter tuning

## Automatic COIN Extraction Method

Potential	<p>The effectiveness achieved by ML COIN Classification Model in automatically identifying the seven COIN classes from NL text of API documents (e.g., 81.9% F-measure) is promising. It shows the potential of utilizing our model to support architects in their interoperability analysis tasks. We consider this accuracy high, as we compared the algorithms' results with our complete sentence-by-sentence manual analysis for the API documents, which we performed for the sake of building a robust corpus. However, in practice, sentences are not examined in such a heavyweight way, especially when projects are limited in terms of available time and manpower. Hence, our model and its provided results are a step towards achieving a good level of automation intelligence for classic software engineering practices that are both error-prone and resource-consuming.</p>
Current context	<p>The current version of our classification model cannot be used to identify COINs from any API document. This is due to the fact that our corpus is relatively small (i.e., ~3k sentences) and is built from six cases only (which have specific characteristics with regard to company size, mother language of the document writer, role of the document writer, maturity of the APIs, etc.). Hence, it will not be appropriate to generalize the features of the sentences in our small corpus to all existing sentences of all API documents. Thus, the main effort that we need to focus on in the future is to enlarge the corpus (e.g., hundreds of thousands sentences) covering a wider range of API documents with different variations of the characteristics and to update the classification model with further features based on the new sentences. Accordingly, our proposed method and its supporting tool are currently reliable in the context of our six cases and similar cases.</p>
Input & Output	<p>The input to our ML-based COIN extraction method are API documents of good quality (e.g., up-to-date, meaningful, correct, etc.). These API documents include NL text that expresses conceptual interoperability constraints. With regard to the output, it is a list of all existing COINs in the API document along with their exact category. Such an output can be merged by providers of interoperable units in the COIN Portfolio they offer to third-party clients. Or the method and its output can be used directly by clients analyzing the API documents.</p>

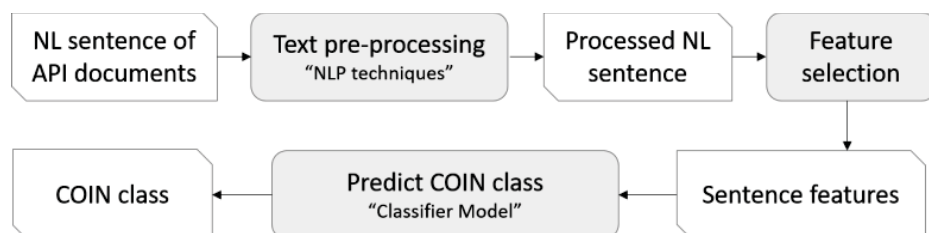


Figure 36 COIN extraction process from an NL sentence of an API document



**Process** As seen in Figure 36, each NL sentence in the input API document goes through the following three activities in order to identify its COIN class:

**Text pre-processing.** The same NLP techniques that we used during the development of the COIN Corpus are automatically applied to each NL sentence in the API document. In particular, the processing tokenizes the sentence into words, lowers cases, eliminates noise words, and stems words into their root format.

**Feature selection.** The processed sentence has its most representative features identified automatically using the BOW and N-Gram approaches. Thus, the sentence features are a collection of combinations of 1 to N of its words.

**Predict COIN class.** The features of the sentence get checked by the best performing classifier model. According to the results seen in Table 11, this classifier is the ComplementNaïveBayes algorithm after being trained on our ML COIN Classification Model (i.e., the COIN feature model). Then the output of this activity is the COIN class of the input sentence, which can be one of our predefined seven classes (i.e., Not-COIN, Syntax, Semantics, Structure, Dynamic, Context, or Quality).

### Add-In Tool for the Chrome Web Browser

**Tool goal** This subsection presents the second technical contribution (i.e., a software tool called COINer), which we developed in the context of this thesis in order to facilitate our ML-based method for the automatic extraction of COINs from NL text in API documents (depicted in Figure 36). It brings our ML-based method to life and makes it applicable in practice. It alleviates some of the challenges that architects and interoperability analysts face during the conceptual analysis of API documents. This is facilitated through its easy-to-use interfaces that automatically identify the conceptual interoperability constraints in the verbose text. Thus, the tool helps analysts understand the impact of the identified constraints based on their class. Hence, the tool has the potential to improve the effectiveness of the interoperability analysis, especially for inexperienced analysts.

**Feature overview** The COINer tool is an add-in for the Chrome web browser and embeds our contributed ML COIN Classification Model within the best-performing classifier model. The tool locates the COIN instances within the text of its API document web page and offers their class within seconds. It also generates a separate report with all sentences that have COINs. In addition, it allows architects to edit the automatically determined COIN class for a sentence and send the feedback to the tool provider. A tool demo is available at [AN17].

## Tool Features and Example Results

Below we will describe the functional features of our tool and explain them with example results from the SoundCloud API document case, which we included in the multiple-case study described earlier in this subsection.

**F1: Highlighting COINs within the text of the API document web page.** This feature takes natural language sentences from the API documents as input and highlights the sentences that have COINs. By hovering over the highlighted sentence, the user can see the COIN category (e.g., semantic, structure, etc.). The tool allows architects to select via computer mouse either *all text* in the web page of an API document or *some text* (i.e., a sentence or more). It also allows the user to determine which COIN types should be highlighted (i.e., all COIN categories or only some of them). Figure 37 shows the options for the highlighting functionality. Note, each sent sentence from the client side get processed by the server side using our implemented NLP techniques (e.g., tokenizing and stemming) before it is investigated by the classification model that responds with the COIN type.

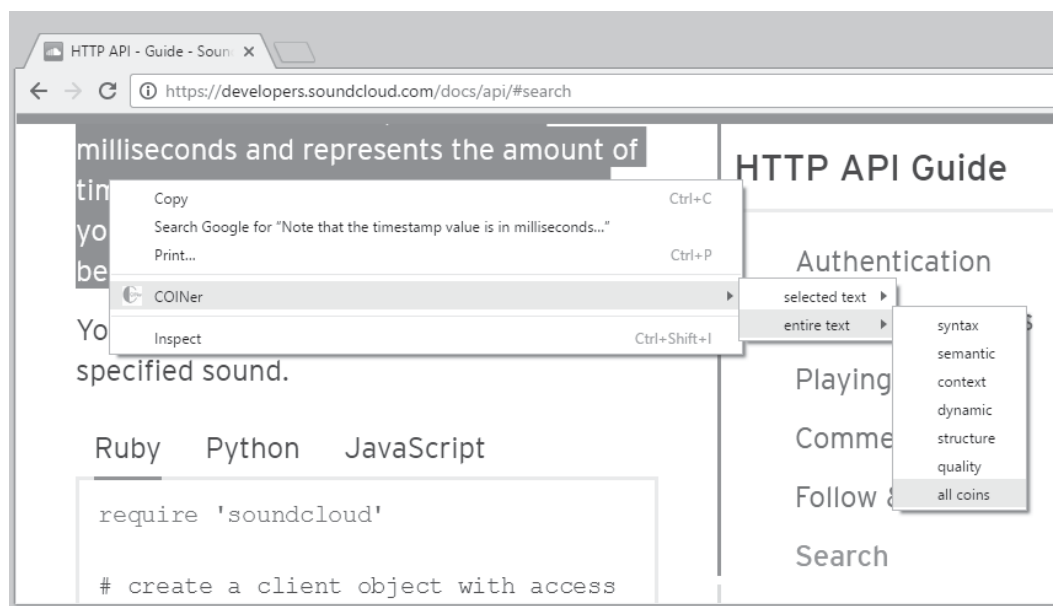


Figure 37

The selection options of the COINer tool for the COIN highlighting feature

Figure 38 shows an example of a Dynamic COIN highlighted within the API document. Note that the highlight color differs for the different COIN categories (for a better look see the tool demo [AN17]).

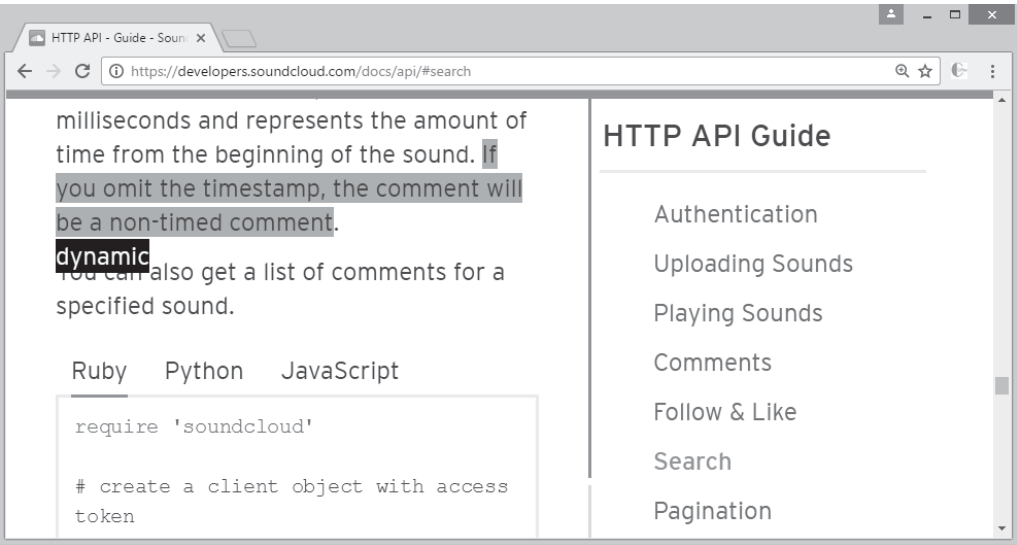


Figure 38 A COINer tool example for an automatically highlighted Dynamic COIN

**F2: Generating a separate COIN report.** This feature takes natural language sentences from the API documents as input and generates a separate report including the found COINs and their categories. As in F1, the tool allows architects to select generating the report either for the whole text or for some of it. It also allows determining which COIN types are to be displayed in the report. The report can be saved as an electronic file and can be printed, too. Figure 39 shows an example of a generated COIN report for Structure COINs in the SoundCloud document. Note, this report paths the way towards identifying the conceptual mismatches between the API and the software system that would interoperate with it.

A screenshot of a web browser displaying a generated COIN report. The report is presented as a table with four columns: Sentence, Coin category, Edit, and Remove. The 'Coin category' column has a dropdown menu currently set to 'structure'. The table contains six rows of sentences from the SoundCloud API documentation, each with its corresponding category and edit/remove buttons. Below the table, there is a note and three buttons: reset, save, and submit.

Sentence	Coin category	Edit	Remove
The returned object will have an access_token property and a refresh_token property as well as expires_in and scope.	all coins syntax semantic	update coin category	remove
Most results from our API are returned as a collection.	context dynamic	update coin category	remove
The number of items in the collection returned is limited to 50 by default with a maximum value of 200.	structure quality	update coin category	remove
When this parameter is passed, the response will contain a next_href property if there are additional results.	structure	update coin category	remove
To fetch the next page of results, simply follow that URI.	structure	update coin category	remove
If the response does not contain a next_href property, you have reached the end of the results.	structure	update coin category	remove

Note: If any selected sentences are not shown in the table, then they do not have a COIN category

reset save submit

Figure 39 A COINer tool example for a COIN report generated for Structure COINs

**F3: Editing the COIN class for sentences and sending feedback.**

As the highest achieved accuracy for our classifier was 81.9%, we give the users the possibility to update the COIN class for a sentence (using a drop-down list for the seven classes) for their own local report copy. The tool also offers the user the option to share his opinion with the tool providers through the “submit” button. If this button is pressed, the new update is saved in a special table for our later use in maintaining the tool’s performance and updating the COIN Corpus and the classification model. In addition, the users are given the option to edit their report copy by removing a COIN instance from the report based on their needs or opinions. The editing “update” and “remove” buttons are shown on the right side of Figure 39 too. Also, the tool allows the user to reset the generated report to its original state through the “reset” button. The “submit” and “reset” buttons are shown at the bottom of Figure 39.

**Tool Implementation**

As mentioned earlier, the tool and its evaluation study were completely designed by the author of this Ph.D. thesis and its implementation was performed under her supervision by a master student. Below, we offer a brief overview of the implementation, but for a closer look at the details, challenges, and empirical evaluation study of the tool, please refer to this master thesis [Nai17].

**Development technology.** The COINer tool was developed mainly using the Python and Java programming languages. It is implemented as an add-in for the Chrome web browser. The tool encapsulates our contributed COIN Classification Model and mirrors its efficiency and accuracy described earlier in this subsection.

**Architectural overview.** The COINer tool has a client-server architecture, in which the server side has the greater workload of the tool. That is, the server takes care of processing and classifying the text sent from the client, while the client side takes care of sending the user requests to the server and representing the received output.

**Extraction process.** The client sends the text to be classified to the server, which starts pre-processing it using the NLP techniques. Then the server sends sentence by sentence to the classifier model, which was built using Java. The classifier responds with the COIN class for each sentence. The server aggregates the sentences with their classes and sends them back to the client, who represents them to the user.

**Summary**

In this subsection, we described our multiple-case study and the experiments we used to investigate the potential of using ML techniques for automatically extracting COINs from NL text of API documents. Based on the promising results, we proposed our ML-

based COIN extraction method, which is the second part of the proactive preparation component of our framework. This method can serve either the providers or the clients of black-box software units, as it helps to locate the conceptual constraints in the overwhelming amount of text. We support our automatic method with a software tool that we implemented as an add-in for the Chrome web browser.

### 6.3.3 Guidelines for Improving API Documentations

In the previous subsection, we presented the second extraction method (for natural language text in API documents) in our preparation component of the COINA Framework. In this subsection, we will present our guidelines for improving API documentation with respect to its value for conceptual interoperability analysis.

Development  
strategy of  
the  
guidelines

We propose the guidelines based on our observations and the user experience we got from our multiple-case study, where we analyzed six API documents in detail (see Subsection 06.3.2). That is, our observations on the six cases indicated that improvements are needed for two aspects of the API documents, namely presentation and content. In Chapter 7, we will present evaluation studies for our guidelines (i.e., a survey with practitioners and an initial controlled experiment).

A widely adopted strategy for interoperability analysis performed by software clients is to read externally shared Application Programming Interface (API) documentation for the software system of interest in order to find its constraints [RB10]. Hence, we propose guidelines that aim at increasing the usability and usefulness of the API documentation from the point of view of the architects or analysts who are responsible for performing the conceptual interoperability analysis. Providers of API documents can benefit from these guidelines to increase the value and competitiveness of their interoperable systems.

Next, we will describe our observations on the current API documents and then present our guidelines for improvement.

#### Observations from the Six Cases of API Documentation

**Technical-oriented API documentations.** The Not-COIN class accounted for 42% of the total sentences in the investigated parts of the API documents that were supposed to be conceptual (i.e., overview and introduction sections). A noteworthy example is the GoogleMaps case, which took it to an extreme level of focus on the technical information (i.e., 63% of its content was in the Not-COIN class, 11.2% in the Dynamic class, 13.1% in the Semantic class, and the rest was shared by the other classes). Accordingly, it is important to raise a flag about the lack of sufficient information about the conceptual aspects of

interoperable software units or APIs (e.g., usage context, terminology definitions, quality attributes, etc.). This obviously has a direct influence on the effectiveness of architects and analysts regarding activities related to conceptual interoperability analysis.

**Blended presence of Dynamic and Semantic constraints.** Our study findings reveal that the Dynamic and Semantic COIN classes have considerable shares in the current API documents (i.e., 25% and 24% of the dataset, respectively). This reflects a favorable awareness of the importance of proper and explicit documenting of the API semantics (e.g., data meaning, service goal, conceptual input and output, etc.) and dynamics (e.g., interaction protocol, flow of data, pre- and post-conditions, etc.). Nevertheless, based on the tedious work we went through to perform our manual analysis for the six cases, we believe that it would be of great help for architects and analysts to have clear borders between these two classes of constraints within the verbose text. For example, it would be easier to skim the text if the API goal were separated from its interaction protocol rather than blending it into long paragraphs. This would offer architects and analysts a better experience and would consequently enhance their analysis results.

**COIN deficiency in platform and web service API documents.** Our findings from the six investigated cases revealed a convention on assigning insignificant shares for the Structure, Syntax, Quality, and Context COIN classes. Interestingly, the cases varied with regard to what they chose to slightly cover out of these four classes. On the one hand, the cases of the web service APIs were the main contributors to the Context, Quality, and Syntax classes in the ground truth dataset. That is, the documents from GoogleMaps, SoundCloud, Skype, and Instagram provided 82.5% of the Syntax COINs, 70.4% of the Quality COINs, and 92% of the Context COINs. Such a contribution cannot be related to the nature of web service APIs, as platform APIs also need to share these COINs explicitly. For example, it is critical for a FarmerWatch application to know the response time offered by the Notification service of AppleWatch APIs.

On the other hand, the platform API documents accounted for 56.1% of the Structure COINs in the ground truth dataset, while the web service API documents accounted for 43.9%. Note that this is not related to the larger number of sentences that these two documents contributed to the dataset, but is rather due to the internal case share of Structure COINs. On average, the platform API documents allocated about 6% of their content to structural constraints, while the web service API documents allocated about 3.6% to these constraints.

## **Content Guidelines for Improving the API Documentation**

This set of guidelines aims at improving the information provided about the conceptual interoperability constraints of the interoperable software



unit. This, in turn, will help the users of the documents perform comprehensive and effective conceptual interoperability analysis. Our content guidelines are as follows:

- **CG1: Provide a view for the high-level architecture of the software unit.** The goal of this guideline is to provide an overview of the entire system and its components. This can be implemented by offering an architecture diagram depicting the structures, layers, distribution, encapsulation, etc.
- **CG2: Provide a conceptual view for complex input and output.** The goal of this guideline is to make it easier to understand the structure of complex data exchanged between software units. It is very important for the reader to know such information without forcing him to start reading technical code examples and information about data formats. This can be implemented via basic data model diagrams (e.g., an ER diagram showing entities and relations).
- **CG3: Provide a conceptual view of data flow and control flow.** The goal of this guideline is to make it easy for the document reader to recognize the dynamic behavior of the interoperable software unit. That is, the data and control flow should be abstracted into conceptual information before overwhelming the reader with code examples and error codes. This can be implemented by offering a simple sequence diagram or flowcharts.
- **CG4: Explicitly specify the interaction properties.** The goal of this guideline is to enrich the API documents with direct information about the interaction properties (e.g., stateless or stateful, synchronous or asynchronous, etc.). This can be implemented by explicitly stating such information in a way that allows it to be easily located while scanning the page. Avoid burying it into verbose text or code examples.
- **CG5: Explicitly specify the runtime qualities.** The goal of this guideline is to enrich the API documents with the quality information that affects the interoperation between software units (e.g., availability, response time, security, etc.). Such information may be a critical directive for clients in making integration decisions. Hence, this information needs to be clearly specified for each offered service and should be easy to find in the document without the need to read it line by line.
- **CG6: Explicitly specify the main usage scenario first, then point out special or exceptional cases.** The goal of this guideline is to avoid confusing the reader as to what fits his

interoperation needs. This means that the context of using a specific service should be clearly stated and distinguished from its subservices and exception handling details. To implement this, avoid blending context information into textual paragraphs (e.g., Android vs. desktop users) and offer simplified use case diagrams or usage scenarios.

- **CG7: Explicitly define special terminologies.** The goal of this guideline is to ensure correct mutual understanding of the terminology used in the API documents. This applies to both invented terms and domain-specific ones. This can be implemented by describing such terms clearly and early in the document prior to using them (e.g., glossaries).
- **CG8: Explicitly specify the expected output of offered services.** The goal of this guideline is to inform the reader directly and explicitly about the nature of the service output. For example, the output could be a returned data item with a specific structure and format, a behavioral action with no returned data, a confirmation message, or a mix of these. Thus, this guideline can be implemented by organizing the information of a service in an obvious way rather than embedding it into code examples or overwhelming text.
- **CG9: Explicitly specify measurement systems.** The goal of this guideline is to ensure an agreed-on system for describing the data (e.g., data units, scale ratio, ordering styles, etc.). For example, if the input or output of an offered functionality is a sorted group of values, specify its sorting criteria and ordering style explicitly (e.g., dates in a transaction log have a descendant or ascendant order).

## Presentation Guidelines for Improving the API Documentation

This set of guidelines aims at improving the way information about conceptual interoperability constraints is displayed and presented in the API documents. This, in turn, will help the readers perform efficient and effective conceptual interoperability analysis. Our presentation guidelines are as follows:

- **PG1: Create a clear border between conceptual and technical information.** The goal of this guideline is to satisfy readers with different needs and analysis perspectives. For example, conceptual information is the main target for architects and conceptual interoperability analysts, while technical information is of high importance for developers and integration implementers. This separation of information saves the reader time in locating the needed information. This

guideline can be achieved by creating different views or distinct sections for the offered information in the API document.

- **PG2: Provide a graphical presentation of conceptual information whenever possible.** The goal of this guideline is to minimize the amount of overwhelming text, which can be simplified through graphical diagrams. For example, interaction protocols are easier to read and remember using representative diagrams compared to hard-to-trace text.
- **PG3: Provide a consistent description of the conceptual constraints for all offered data or services.** The goal of this guideline is to help the reader learn the API document faster and easier. For example, specifying the COINs for different services in equal amounts and formats allows the user to know what to expect and where to find needed information.
- **PG4: Structure the conceptual information according to importance.** The goal of this guideline is to avoid distracting the reader and to keep him focused on high-priority information. This can be implemented by describing the conceptual constraints and clearly distinguishing them from optional recommendations.
- **PG5: Specify the conceptual constraints precisely.** The goal of this guideline is to avoid confusing the reader about the meaning or criticality of sentences. This includes using the correct words (e.g., must vs. should) and keeping the sentences simple (e.g., not specifying two constraints in one long sentence).

## Summary

In this subsection, we described our observations on the six cases of API documents we studied with regard to their limitations in supporting conceptual interoperability analysis. These observations inspired us to propose improvement guidelines for the API documents regarding both their content and presentation aspects. These guidelines are part of the proactive preparation component of our framework. They serve the providers of black-box software units in producing useful and usable API documents from the point of view of conceptual interoperability analysts. These guidelines will be evaluated in Chapter 7.

## 6.4 Approach for the Systematic Detection of Conceptual Mismatches

In Section 6.3, we described the first component of our COINA Framework, which supports proactive preparation for interoperable software units. In this section, we present the second component of our framework, which is a systematic approach for detecting conceptual mismatches between software units. Remember that we have provided an overview of the method (i.e., input, output, and process) and the context (i.e., who, when, and how) in Subsections 6.2.1 and 6.2.2 respectively.

Approach goal	The foremost goal of this approach is to support third-party clients of interoperable software units in detecting conceptual mismatches between the software units intended to interoperate.
Input & Output	As we described earlier in Subsection 6.2.1, the input to our systematic mismatch detection approach is a COIN Portfolio for each interoperating system. However, if these portfolios are not prepared proactively, our approach provides detailed guidance on how to prepare the input. In this case, the expected input is the available software documentation. Such input is different for in-house software units (i.e., SRS, UML diagrams, and API documentation) and external software units (i.e., API documentation only). With regard to the output, this is, as we also mentioned earlier, the list of conceptual mismatches between the two interoperating software units.
Process overview	In Figure 40, we give an overview of the manual systematic approach with its two main activities. These activities are the <i>COIN Extraction</i> for the two software units intended to interoperate into COIN Portfolios (if they are not already prepared); and the <i>Mapping of the Portfolios</i> , which results in the list of mismatches between the two units. The figure also shows that we support the first activity with a documentation template for the extracted COINs (which we described earlier in Section 5.4), a COIN Cheat Sheet, and guidelines. The second activity is supported with a documentation template for detected mismatches, a Mismatches Cheat Sheet, and guidelines. We define the supporting cheat sheets as follows:

### Definition 10 – COINA Cheat Sheets

These are reference tools that help conceptual interoperability analysts to accomplish their tasks manually. They provide brief descriptions of concepts (i.e., types of COINs and mismatches) with simple examples and guidelines.

In Subsection 6.4.1 and Subsection 6.4.2, we will explain these supporting materials within their related activities in detail.

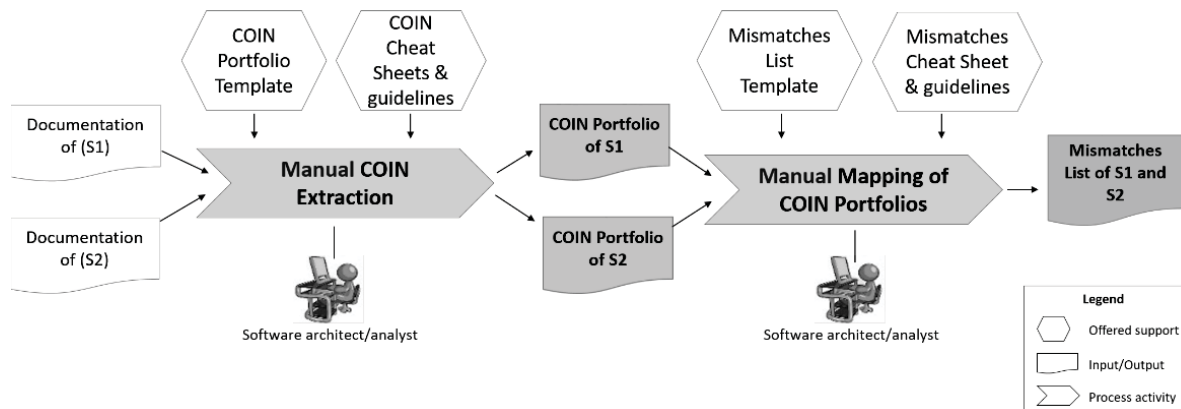


Figure 40 Process overview of the systematic approach for detecting conceptual mismatches

### 6.4.1 Perspective-based Extraction of COINs

In this subsection, we will describe how third-party clients can manually and systematically identify the COINs for two software units.

**Method goal** The goal of this perspective-based, systematic method is to support third-party clients in manually identifying the COINs for the software units intended to interoperate if COIN Portfolios have not been prepared proactively. As described earlier in Subsection 3.2.2, extracting such information manually is a challenge, especially for inexperienced analysts. By offering detailed guidance, we pave the way towards effectively detecting conceptual mismatches.

**Contribution** The key contribution of our method is an increase in the effectiveness of manual extraction of COINs. This is facilitated through our contributed COIN Portfolio Template, “COIN Cheat Sheets”, and guidelines, which direct the analysts in performing perspective-based analysis for interoperable software units.

Next, we will describe the COIN Cheat Sheets, which are the basis for our method. Then we will describe the guidelines for manual application of our perspective-based method.



### COIN Cheat Sheets

The COIN Cheat Sheets are a derivation of the perspective-based reading (PBR) technique [SRB00]. The PBR has been proven to have better effectiveness results when used for defect inspection in software requirements documents, software code, or UML diagrams [BGL+96, LA99, LASEE00, LD00]. Uniquely, our approach uses PBR for the purpose of conceptual interoperability analysis.

Our COIN Cheat Sheets provide guidance for extracting the different types of COINs and their categories from the system, data, and service perspectives, along with directions on their locations. An example of a record in the table of our COIN Cheat Sheet for UML diagrams looks like this:

Table 13

Example of a record in the COIN Cheat Sheet for UML diagrams

Perspective	COIN Category	COIN Type	Description	Where to find (probably)?
Service	Dynamic	(IP) Interaction property	State(ful/less), (a)synchronous, etc.	In the type of messages in sequence diagrams (i.e., a synchronous message is denoted by a solid arrowhead  ; an asynchronous message by a line arrowhead  ).

This record guides the user in finding the interaction property constraint (i.e., synchronicity) for a service or method in the UML diagram.

Although this thesis strongly focuses on the architecture and low-level design documents as the in-house input for conceptual interoperability constraints, it also considers the software requirements specification SRS (see background Section 2.2) as input for COINs in integration projects. Hence, we developed cheat sheets to help extract the COINs from different software documents (i.e., SRS, UML diagram, and API documents). For the full version of the COIN Cheat Sheets, see Appendix E.

## Guidelines for Applying the Perspective-based Extraction Method

### Activity 1

Here we state our guidelines associated with using the COIN Cheat Sheet for manually extracting the COINs for the first software unit:

- Read the in-house architecture documentation of your software unit (e.g., UML diagrams of the Smart Farm) and read the integration requirements stated in the SRS (e.g., need for a smart machine that offers RemoteSteering functionality in the field).
- Abstract your software system and build its input/output model.
  - Overall system, e.g., Smart Farm
  - Input, e.g., steering directions
  - Interoperable functionality required, e.g., RemoteSteering
  - Output, e.g., machine moving and confirmation message
- Read the COIN Cheat Sheets and learn about the different categories and types of COINs.



- Search for the COINs related to the elements of the abstraction that you have built.
- Use the hints provided in the Cheat Sheets to direct your search.
- If you find a COIN instance, record it in the documentation template “COIN Portfolio Template”.

**Activity 2** Similarly, we provide guidelines associated with using the COIN Cheat Sheet for extracting the COINs manually for the second software unit:

- Read the available documentation of the external software unit (e.g., smart tractor API documentation)
- Abstract the unit into its input-output model.
  - Overall system, e.g., smart tractor
  - Input, e.g., steering directions
  - Interoperable functionality offered, e.g., RemoteSteering
  - Output, e.g., movement and confirmation
- Read the COIN Cheat Sheets and learn about the different categories and types of COINs.
- Search for the COINs related to the elements of the abstraction that you have built.
- Use the hints provided in the Cheat Sheets to direct your search.
- If you find a COIN instance, record it in the documentation template “COIN Portfolio Template”.

In Table 14, we show an example of a COIN Portfolio created using our perspective-based extraction method for the smart farm system. Table 15 shows another portfolio created for the smart tractor system.

Table 14 Example of a snippet of a COIN Portfolio for the Smart Farm System

Interoperable element	COIN Sheet				
	ID	Category	Type	Value	
				Qualitative	Quantitative
Overall system	C1	Context	Intended user	Users with experience in smart technology	
Function RemoteSteering	C2	Quality	Function quality		Response time <= 2 ms
Function GetLog	C3	Semantic	Function output	List of last week's activities only	
Data Location	C4	Syntax	Definition	Location is a position on the farm field that is measured via GPS coordinates	
...	...	...	...	...	...

Table 15 Example of a snippet of a COIN Portfolio for the Smart Tractor System

Interoperable element	COIN Sheet				
	ID	Category	Type	Value	
				Qualitative	Quantitative
Function RemoteSteering	C1	Quality	Function quality		Response time between 3 <= and <= 5 ms
Function GetLog	C2	Semantic	Function output	All tractor activities for its lifetime	
Function GetLog	C3	Dynamic	Synchronicity	Synchronous	
Overall system	C4	Context	Intended user	Are in the age group between 20 and 60 years	
...	...	...	...	...	...

### 6.4.2 Checklist-based, Algorithmic Mapping of Portfolios

In this subsection, we will describe how to systematically compare two COIN Portfolios in order to detect the different conceptual mismatches between their software units.

**Method goal** The goal of this checklist-based, algorithmic method is to support third-party clients in manually detecting the conceptual mismatches between the software units intended to interoperate. Manual detection of the different types of such mismatches and understanding of their impact is a challenge, especially for inexperienced analysts. By offering detailed guidance, we support effective detection of conceptual mismatches.

**Contribution** The key contribution of our method is an increase in the effectiveness of the manual mapping of COIN Portfolios. This is facilitated through our contributed mapping algorithm, the “Mismatches Cheat Sheet”, the Mismatches List Template, and guidelines. These direct the analysts in performing checklist-based analysis of the portfolios of interoperable software units.

Next, we will describe the mapping algorithm, the Mismatches Cheat Sheet, and the Mismatches Template, which are the basis for our method. Then we will describe the guidelines for manual application of our systematic checklist-based method.

### Mapping Algorithm

Our mapping algorithm defines the process for comparing two lists of conceptual constraints (or COIN Portfolios) for two software units, in order to find the conceptual mismatches between them. The algorithm is as follows:

**Algorithm (Mapping COIN Portfolios)****Input:** COIN Portfolio of  $S_x$  ( $CP_x$ ), COIN Portfolio of  $S_y$  ( $CP_y$ )**Process:**

```

1  mismatchesList  $\leftarrow \emptyset$ 
2  mismatch  $\leftarrow \text{nil}$ 
3  c1.hasCorrespondent  $\leftarrow \text{false}$ 
4  c2.hasCorrespondent  $\leftarrow \text{false}$ 
5  For each c1  $\in \{CP_x.\text{coins}\}$ 
6    For each c2  $\in \{CP_y.\text{coins}\}$ 
7      If (c1.element. = c2.element)
8        c1.hasCorrespondent  $\leftarrow \text{true}$ 
9        c2.hasCorrespondent  $\leftarrow \text{true}$ 
10       If ((c1.category = c2.category)  $\wedge$  (c1.type = c2.type)  $\wedge$ 
            (c1.value  $\neq$  c2.value))
11         mismatch  $\leftarrow$  (c1.element, "direct mismatch", c1, c2)
12         mismatchesList  $\leftarrow \{\text{mismatchesList} \cup \text{mismatch}\}$ 
13       End IF
14       If (((c1.category  $\neq$  c2.category)  $\mid$  (c1.type  $\neq$  c2.type))  $\wedge$ 
            (c1.value.influence(c2.value) = true))
15         mismatch  $\leftarrow$  (c1.element, "indirect mismatch", c1, c2)
16         mismatchesList  $\leftarrow \{\text{mismatchesList} \cup \text{mismatch}\}$ 
17       End IF
18     End IF
19   End For
20 End For
21 For each c1  $\in \{CP_x.\text{coins}\}$ 
22   If (c1.hasCorrespondent. = false)
23     mismatch  $\leftarrow$  (c1.element, "potential mismatch", c1)
24     mismatchesList  $\leftarrow \{\text{mismatchesList} \cup \text{mismatch}\}$ 
25   End If
26 For each c2  $\in \{CP_y.\text{coins}\}$ 
27   If (c2.hasCorrespondent. = false)

```

```

28      mismatch ← (c2.element, "potential mismatch", c2)
29      mismatchesList ← {mismatchesList ∪ mismatch}
30    End If
10 End For

```

**Output:** mismatchesList

In this dissertation, we focus on supporting the application of this algorithm manually through our cheat sheet and guidelines. However, it is also possible to run it automatically, but only if the input COIN Portfolios are formalized. Although such formalization can open the door for automation benefits, it cannot be complete and its associated effort is not trivial. That is, the formalization task for portfolios is time-consuming and requires experience in using formal-based specification languages. In addition, formalizing a COIN Portfolio can be done partially for COINs with quantitative values only. For example, a Quality COIN capturing that the response time for  $S_1$  should be within 5 seconds can be formalized, but it is not possible to formalize a Semantic COIN that specifies the goal of adding a RemoteSteering functionality for boosting the performance of farmers in the field.

Example of  
mapping  
automation  
based on  
COIN  
formalization

Despite the fact that the potential of formalization is not within the scope of this dissertation, we show in the following a trivial example of how it would look like to map two formalized COINs. In our example, we built a trivial modeling language for the COINs based on the architectural meta-model of the Flexible Modeling Framework (FMF), which is similar to modeling frameworks found in model-driven language workbenches [Fow05]. Then we created COIN instances using the language syntax. Figure 41 illustrates how two COINs for two units  $S_1$  and  $S_2$  are directly contradictory and can be detected automatically using our algorithm.

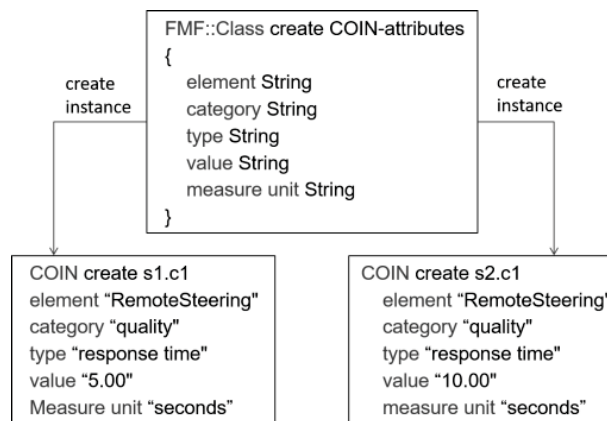


Figure 41

Example of a formal modeling language and two COIN instances

## Mismatches Cheat Sheet

The Mismatches Cheat Sheet is a derivation of the checklist-based reading (CBR) technique [SMKI02], which is typically used for inspecting and testing purposes. The CBR has been proven to have better effectiveness results when used for defect inspection in software code and UML diagrams [LEE01], [SMKI02]. Uniquely, our approach uses CBR for the purpose of conceptual interoperability analysis.

In our Mismatches Cheat Sheet, we provide guidance for extracting different types of conceptual mismatches based on the COINs causing them, along with examples. That is, this sheet uniquely directs the identification of mismatches according to the relationships between the COINs in the two portfolios. Table 16 shows one record of our Mismatches Cheat Sheet.

Table 16 Example of a record in the Mismatches Cheat Sheet

Mismatch Type	How to find?	Causing COINs	Examples
Direct	COINs of similar category and type with explicitly contradictory values for corresponding elements.	All types of COINs can be the cause of direct mismatches.	<ul style="list-style-type: none"> <li>▪ <math>S_1</math> has a “size of lists” constraint that the returned object has a maximum capacity of 100 items.</li> <li>▪ <math>S_2</math> has a “size of lists” constraint that the maximum size of the lists used in the system is 50 items.</li> </ul> <p>This leads to a “direct mismatch” on the structure level.</p>

The record depicted above guides the user in finding direct mismatches between two COINs from two software units over a structural constraint (i.e., the size of the list) for a data object. For the full version of the Mismatches Cheat Sheet, see Appendix F.

## Guidelines for Applying the Checklist-based Mapping Method

Activity 3 Here we state our guidelines associated with using the COIN Cheat Sheet for manually extracting the COINs for the first software unit:

- Read the Mismatches Cheat Sheet and learn about the different categories of conceptual mismatches and the COIN types typically causing them.
- Compare the COINs in the portfolios of the two software units intended to interoperate in two main steps as follows:
  - *Step1*: Compare each  $COIN_x$  in the portfolio of the first software unit  $S_1$  with each  $COIN_y$  in the portfolio of the other unit  $S_2$ .
    - If  $COIN_x$  and  $COIN_y$  are about a correspondent interoperable element (e.g., both are about the same function) of the same category and type, and if they contradict each other, document a *direct mismatch* in the

“Mismatches List Template”. Mark the two COINs with a sign (e.g., ✓) to denote that they have been checked.

- If  $\text{COIN}_x$  and  $\text{COIN}_y$  are about a correspondent interoperable element, but of a different category or type, and if they influence each other, document an *indirect mismatch* in the “Mismatches List Template”. Mark the two COINs with a sign (e.g., ✓) to denote that they have been checked.
- If  $\text{COIN}_x$  and  $\text{COIN}_y$  are not contradicting or influencing each other, move to the next COIN in the portfolio of  $S_1$  (if not finished) and compare it with each COIN in the portfolio of  $S_2$ .
- *Step2*: For each unmarked  $\text{COIN}_z$  in both portfolios of  $S_1$  and  $S_2$ , check if it introduces a potential mismatch (either consensus or adherence).
  - If  $\text{COIN}_z$  introduces a potential mismatch by requiring work in order to be satisfied, document an *adherence mismatch* in the “Mismatches List Template”. Mark the COIN with a sign (e.g., ✓) to denote that it has been checked.
  - If  $\text{COIN}_z$  introduces a potential mismatch by requiring conceptual agreement only, document a *consensus mismatch* in the “Mismatches List Template”. Mark the COIN with a sign (e.g., ✓) to denote that it has been checked.
  - If  $\text{COIN}_z$  does not introduce a potential mismatch, move to the next unmarked COIN until both portfolios are completely checked.

An example of the detected conceptual mismatches between the COIN Portfolios of the smart farm and the smart tractor (see Table 14 and Table 15) using our checklist-based method is depicted in Table 17.

Table 17 Example of a snippet of the Mismatches List for the Smart Farm and the Smart Tractor

Interoperable element	Mismatch				Reference COINs	
	ID	Category	Type	Description	COIN ID from $S_1$	COIN ID from $S_2$
Overall system	M3	Potential consensus	Context	S1 and S2 have different (not necessarily contradictory) user characteristics	C1	C4
Function RemoteSteering	M1	Direct	Quality	S1 and S2 contradict over the quality of the RemoteSteering function	C2	C1
Function GetLog	M2	Direct	Semantic	S1 and S2 contradict over the output of the GetLog function	C3	C2



Data Location	M4	Potential consensus	Syntax	S1 has a definition that, if misunderstood, might lead to a mismatch	C4	-
Function GetLog	M5	Potential adherence	Dynamic	S2 has synchronous communication style and S1 has to satisfy it	-	C3
...	...	...	...	...	...	...

## 6.5 Summary

In this chapter, we presented our methodical and technical contributions within the Conceptual Interoperability Analysis Framework (COINA). Our framework is based on the model of conceptual constraints introduced in Chapter 0, which extends and refines the existing models of reuse and interoperability. The overall goal of COINA is to support software architects and analysts in performing effective and efficient conceptual interoperability analysis.

The framework has two supporting components: the first component tells providers to proactively share the conceptual interoperability constraints about the software units offered by them, while the second component guides clients interested in systematically analyzing the conceptual interoperability between their own software units and external ones. In this regard, we contributed methods to help providers extract the COINs from in-house architectural documents (using templates and supported by an add-in tool for the Enterprise Architect tool) and from shared API documents (using machine learning and supported by an add-in tool for the Chrome web browser). Furthermore, we supported clients with methods for comparing the list of COINs for two software units and detecting their conceptual mismatches (using perspective-based and checklist-based methods supported by cheat sheets and standard documentation templates).

---

## 7 Evaluation

### 7.1 Introduction

In this chapter, we will describe the empirical evaluation studies and their results.

- In Section 7.2, we start by stating our evaluation objectives and the derived hypotheses.
- In Section 7.3, we will describe our multi-run controlled experiment, which we conducted to show the effect of using our systematic conceptual interoperability analysis approach on the produced analysis results.
- In Section 7.4, we will describe a survey and an initial controlled experiment, which we used to show the perceived value of our guidelines and the actual effect on the results of conceptual interoperability analysis.
- In Section 7.5, we will describe the comprehensiveness results of our COIN Model from the collected data of our multiple-case study described in Subsection 6.3.2.
- In Section 7.6, we will summarize the presented evaluation studies and their findings.

### 7.2 Objectives and Hypotheses

For each of our proposed solution ideas, we expect to have some practical improvements or benefits, which we translated into a number of scientific hypotheses. From the theoretical perspective, we have hypotheses about our conceptual foundation idea (i.e., S.11: the COIN Model) regarding its validity and comprehensiveness as follows:

#### **Hypotheses regarding the COIN Model**

**H<sub>1</sub>:** The model of conceptual interoperability constraints is **valid** in defining the relationships among types of constraints, interoperable elements, types of software units, and the type of relevant conceptual mismatches.

**H<sub>2</sub>:** The model of conceptual interoperability constraints is **comprehensive** in covering the different types of conceptual

interoperability constraints that can restrict the interoperation of software units.

From the engineering perspective, we have different hypotheses about the first methodical idea of the COINA Framework (i.e., S.I2.1: Proactive preparation method) with respect to its effectiveness, efficiency, and acceptance. As stated below, these improvements are from the point of view of providers of interoperable software units (H<sub>3</sub> to H<sub>5</sub>) as well as third-party clients (H<sub>6</sub> to H<sub>8</sub>).

### Hypotheses regarding the COINA Framework – Proactive Preparation

**H<sub>3</sub>:** Using the COINA extraction method **increases** the **effectiveness** of architects in extracting and sharing the relevant conceptual interoperability constraints of their software units when compared to ad-hoc approaches.

**H<sub>4</sub>:** Using the COINA extraction method **increases** the **efficiency** of architects in extracting and sharing the relevant conceptual interoperability constraints of their software units when compared to ad-hoc approaches.

**H<sub>5</sub>:** Using the COINA extraction method as proposed is **accepted** by architects for extracting and sharing the relevant conceptual interoperability constraints of their software units.

**H<sub>6</sub>:** Implementing the COINA guidelines for improving documentation **increases** the **effectiveness** of third-party client architects in identifying the conceptual interoperability constraints of external software units when compared to not applying them.

**H<sub>7</sub>:** Implementing the COINA guidelines for improving documentation **increases** the **efficiency** of third-party client architects in identifying the conceptual interoperability constraints of external software units when compared to not applying them.

**H<sub>8</sub>:** Implementing the COINA guidelines for improving documentation is **accepted** by third-party client architects as valuable for identifying the conceptual interoperability constraints of external software units.

From another engineering perspective, we have hypotheses about the second part of our methodological idea (i.e., S.I2.2: Systematic detection of conceptual mismatches) with respect to its improvement effects on effectiveness, efficiency, and acceptance. The hypothesized improvements are from the point of view of third-party clients who are responsible for identifying mismatches between software units.

### Hypotheses regarding the COINA Framework – Systematic Analysis

**H<sub>9</sub>:** Using the COINA systematic analysis *increases* the *effectiveness* of third-party client architects in identifying the conceptual constraints (from structured and unstructured documents) and the mismatches of two software units when compared to ad-hoc analysis approaches.

**H<sub>10</sub>:** Using the COINA systematic analysis *increases* the *efficiency* of third-party client architects in identifying the conceptual constraints (from structured and unstructured documents) and the mismatches of two software units when compared to ad-hoc analysis approaches.

**H<sub>11</sub>:** Using the COINA systematic analysis as proposed is **accepted** by third-party client architects for identifying the conceptual constraints and mismatches of two software units.

Overall, we also have hypotheses for applying the COINA Framework in general. Validating these hypotheses is beyond the scope of our thesis work as this would require further activities that take place after the analysis task.

**H<sub>12</sub>:** Using the COINA Framework *increases* the *effectiveness* of third-party client architects in designing the resolution for conceptual mismatches compared to results from ad-hoc analysis approaches.

**H<sub>13</sub>:** Using the COINA Framework *increases* the *efficiency* of third-party client architects in designing the resolution of conceptual mismatches compared to results from ad-hoc analysis approaches.

In Figure 42, we summarize our hypotheses regarding each main contribution. The shaded hypotheses are the ones we evaluated within our thesis work. In particular, we evaluated:

- H<sub>2</sub> through our multiple-case study (Section 07.5);
- H<sub>6</sub>, H<sub>7</sub>, and H<sub>8</sub> through a survey and an initial controlled experiment (Section 7.4);
- H<sub>9</sub>, H<sub>10</sub>, and H<sub>11</sub> through a multi-run controlled experiment (Section 7.3).

In the following figure, the unshaded hypotheses represent those that were not evaluated and have been left for future work and studies.

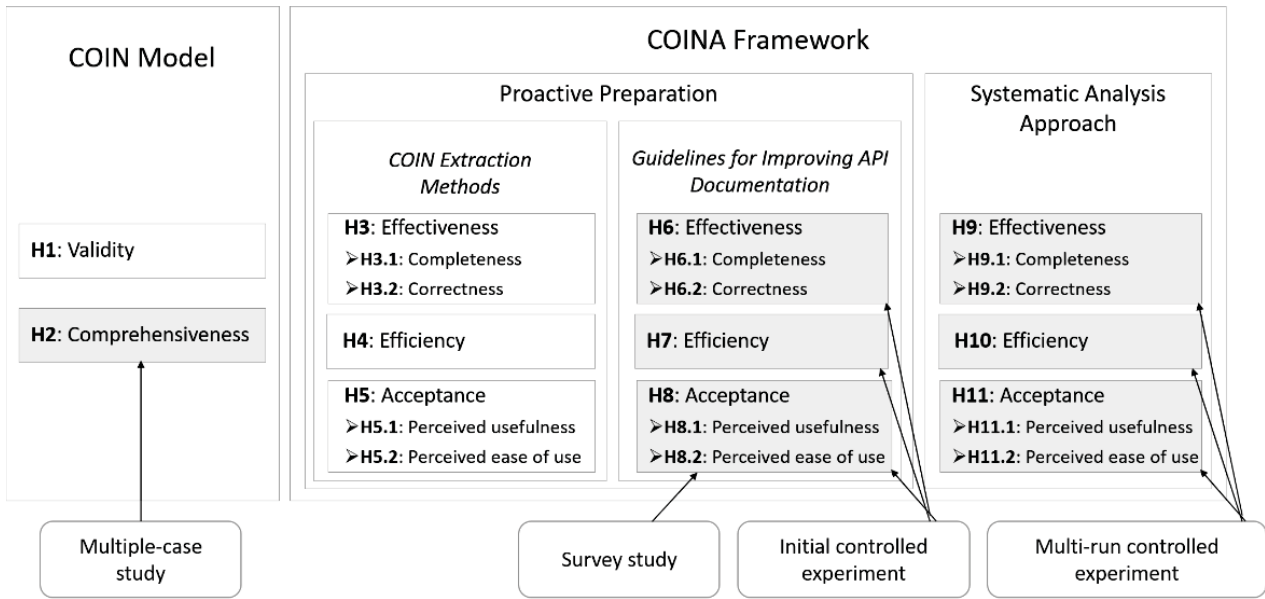


Figure 42 Hypotheses for the contributions

## 7.3 Multi-Run Controlled Experiment

In this section, we will present a multi-run controlled experiment for evaluating the effectiveness (H<sub>9</sub>), efficiency (H<sub>10</sub>), and acceptance (H<sub>11</sub>) of our systematic analysis approach. Thus, we will describe the study goal and research question, the experimental context and setup, the analysis of the experiment results, a cross-run discussion, and the threats to validity.

### 7.3.1 Objectives and Research Questions

The main goal of this study, formulated by means of the GQM goal template, was *to analyze the systematic conceptual interoperability analysis approach for the purpose of evaluation with a focus on effectiveness, efficiency, and acceptance from the perspective of software architects and analysts in the context of a controlled experiment with students*. That is, we wanted to know if our proposed analysis approach allows performing a more effective and efficient conceptual interoperability analysis compared to ad-hoc interoperability analysis. Also, we wanted to know if our approach is accepted in practice. In line with the goal, our research questions were as follows:

**RQ9** (Effectiveness): *Does adopting the systematic conceptual interoperability analysis approach of COINA enable software architects to analyze interoperable software units and identify their conceptual constraints and mismatches more effectively compared to performing an ad-hoc analysis?*

**RQ10** (Efficiency): *Does adopting the systematic conceptual interoperability analysis approach of COINA enable software architects*

*to analyze interoperable software units and identify their conceptual constraints and mismatches more efficiently compared to performing an ad-hoc analysis?*

**RQ11** (Acceptance): *Do practitioners perceive the systematic conceptual interoperability analysis approach of COINA as useful and easy to use when they follow it to analyze interoperating software units?*

### 7.3.2 Experimental Context

The experiment was conducted in two runs (Run I and Run II). Run I was performed in a practical course entitled “Team-based Software Development” for master students at the University of Kaiserslautern, Germany (TU KL). The practical course was co-supervised together with the Fraunhofer Institute for Experimental Software Engineering IESE in the winter semester 2015/2016. Run II was performed in the German-language “Grundlagen des Software Engineering (GSE)” course for bachelor and master students at TU KL. The GSE course was supervised by Prof. Dieter Rombach in the winter semester 2015/2016.

### 7.3.3 Experimental Setup

To explain the setup of the experiment, we will first define the study variables and formulate our statistical hypotheses. Then we will provide more information about the participants, the experimental design, the procedures, tasks, and the materials to be provided.

**Study variables.** The main concept behind performing experiments is to examine some variables (dependent variables) while manipulating some other variables (independent variables). For the experiment we designed, we defined these variables as follows:

*Dependent variables:* effectiveness and efficiency for the conceptual interoperability analysis results, and the participants’ acceptance of the approach.

*Independent variables:* the approaches applied for the conceptual interoperability analysis.

It is important to mention that we distinguish between: (1) *full COINA* support provided for the systematic analysis approach (where both the COIN extraction and the mapping is supported by cheat sheets); and (2) *half COINA* support provided for the systematic analysis approach (where only the COIN extraction is supported by cheat sheets). Hence, our independent variables are called “full COINA”, “half COINA”, and ad-hoc approach.

**Statistical hypotheses.** In this experiment, we intended to answer the previously stated research questions from Subsection 7.3.1, namely



RQ9, RQ10, and RQ11. In order to do this, we derived the statistic null hypotheses ( $H_0$ ) and the corresponding alternative hypotheses ( $H_1$ ) from the stated research questions. Note that the arithmetic mean of effectiveness and efficiency, and the median of acceptance of the full COINA approach are denoted by  $\mu_c$ . Meanwhile,  $\mu_b$  denotes the mean of effectiveness and efficiency of the half COINA approach and  $\mu_a$  denotes the effectiveness and efficiency of the ad-hoc analysis approach. In Table 18, we describe the hypotheses related to each research question of this controlled experiment.

Table 18

Hypotheses of the Multi-run Controlled Experiment

Research Question	Hypothesis	Quantified Hypothesis
RQ9: Effectiveness	H <sub>9.1</sub> : Full COINA <b>increases</b> the <i>completeness</i> in extracting the COINs of interoperating systems manually from <i>structured</i> documents compared to ad-hoc analysis.	H <sub>9.1, 0</sub> : $\mu_c \leq \mu_a$
		H <sub>9.1, 1</sub> : $\mu_c > \mu_a$
	H <sub>9.2</sub> : Full COINA <b>increases</b> the <i>correctness</i> in extracting the COINs of interoperating systems manually from <i>structured</i> documents compared to ad-hoc analysis.	H <sub>9.2, 0</sub> : $\mu_c \leq \mu_a$
		H <sub>9.2, 1</sub> : $\mu_c > \mu_a$
	H <sub>9.3</sub> : Full COINA <b>increases</b> the <i>completeness</i> in extracting the COINs of interoperating systems manually from <i>unstructured</i> documents compared to ad-hoc analysis.	H <sub>9.3, 0</sub> : $\mu_c \leq \mu_a$
		H <sub>9.3, 1</sub> : $\mu_c > \mu_a$
	H <sub>9.4</sub> : Full COINA <b>increases</b> the <i>correctness</i> in extracting the COINs of interoperating systems manually from <i>unstructured</i> documents compared to ad-hoc analysis.	H <sub>9.4, 0</sub> : $\mu_c \leq \mu_a$
		H <sub>9.4, 1</sub> : $\mu_c > \mu_a$
	H <sub>9.5</sub> : Full COINA <b>increases</b> the <i>completeness</i> in finding the conceptual mismatches compared to ad-hoc analysis and half COINA.	H <sub>9.5, 0</sub> : $\mu_c \leq \mu_a, \mu_b$
		H <sub>9.5, 1</sub> : $\mu_c > \mu_a, \mu_b$
H <sub>9.6</sub> : Full COINA <b>increases</b> the <i>correctness</i> in finding the conceptual mismatches compared to ad-hoc analysis and half COINA.	H <sub>9.6, 0</sub> : $\mu_c \leq \mu_a, \mu_b$	
	H <sub>9.6, 1</sub> : $\mu_c > \mu_a, \mu_b$	
RQ10: Efficiency	H <sub>10</sub> : Full COINA <b>decreases</b> the <i>time</i> when used for interoperability analysis compared to ad-hoc analysis and half COINA.	H <sub>10, 0</sub> : $\mu_c \leq \mu_a, \mu_b$
		H <sub>10, 1</sub> : $\mu_c > \mu_a, \mu_b$
RQ11: Acceptance	H <sub>11.1</sub> : Practitioners perceive full COINA as <i>useful</i> when they use it for interoperability analysis.	H <sub>11.1, 0</sub> : $\mu_c \leq 3^*$
		H <sub>11.1, 1</sub> : $\mu_c > 3^*$
	H <sub>11.2</sub> : Practitioners perceive full COINA as <i>easy-to-use</i> when they use it for interoperability analysis.	H <sub>11.2, 0</sub> : $\mu_c \leq 3^*$
		H <sub>11.2, 1</sub> : $\mu_c > 3^*$
*The value 3 in H <sub>11.1</sub> and H <sub>11.2</sub> is compared to a defined scale of values from 1 to 5 (more details below). All hypotheses in the table were tested at a confidence level of $\alpha = 0.05$ .		

**Operationalization (Evaluation metrics).** As stated previously, the comparison in H<sub>9</sub> and H<sub>10</sub> is between an ad-hoc conceptual interoperability analysis and analysis using the support of the COIN Cheat Sheets or the Mismatches Cheat Sheets. In order to perform the comparison, we need to have specific metrics that can be calculated from the results produced by the participants in the experiment. In order to evaluate the *effectiveness* in H<sub>9</sub>, we use the following basic metrics:

- True positive (TP): a COIN/mismatch instance that is correctly identified as a COIN/mismatch instance

- False positive (FP): a non-COIN/non-mismatch that is incorrectly identified as a COIN/mismatch instance
- True negative (TN): a non-COIN/non-mismatch that is correctly not identified as a COIN/mismatch instance
- False negative (FN): A COIN/mismatch instance that is incorrectly not identified as a COIN/mismatch instance

These basic metrics are used for calculating the following two derived metrics, which we use in our evaluation:

- For completeness:  $\text{Recall} = \frac{TP}{(TP + FN)}$
- For correctness:  $\text{Precision} = \frac{TP}{(TP + FP)}$

Meanwhile, we evaluate *efficiency* in H<sub>10</sub> by directly using a basic cost metric, which is Time in minutes spent on the tasks. With regard to *Acceptance*, we evaluate it in H<sub>11</sub> using the following derived metrics:

- Perceived usefulness, which is the degree to which software architects believe that applying the full COINA approach will help them perform better COIN/mismatch analysis and achieve better results; and
- Perceived ease of use, which is the degree to which software architects believe that applying the full COINA approach would be free of effort.

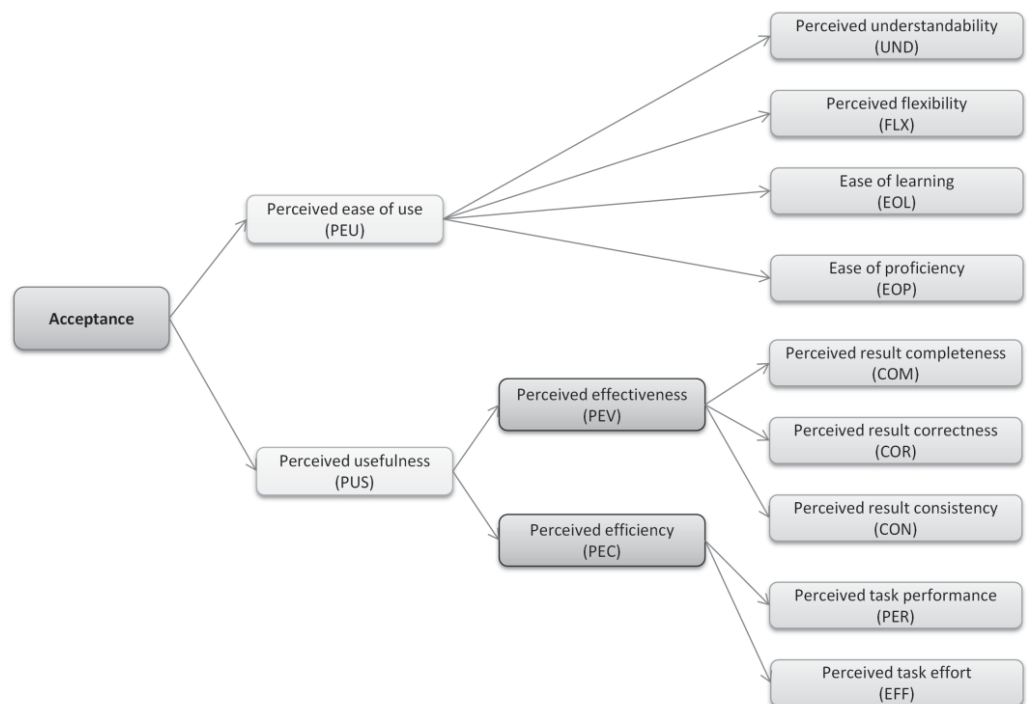


Figure 43

Acceptance evaluation metrics

To calculate these two metrics, we use basic metrics from the Technology Acceptance Model (TAM) [DBW89] as seen in Figure 43. All the basic metrics of acceptance were measured by means of a debriefing questionnaire at the end of the experiment (more details to come in this subsection). The questionnaire included questions with a 5-level Likert scale [Lik32] for each basic metric. The five levels were: strongly disagree, disagree, neither, agree, and strongly disagree. To quantify the Likert scale data and consequently test  $H_3$ , the scale levels were weighted as 1, 2, 3, 4, and 5 accordingly.

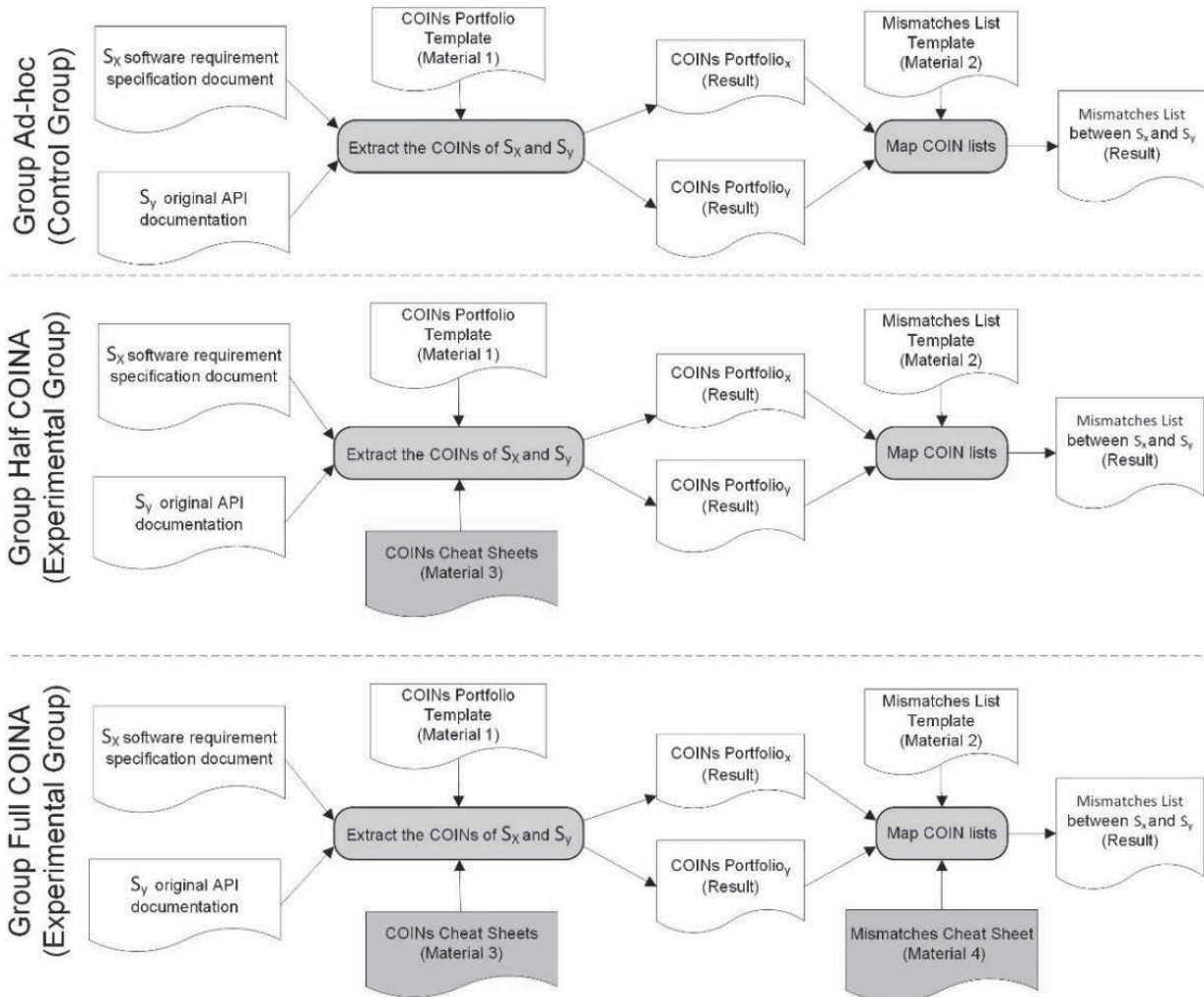


Figure 44

## Experimental Design

**Experimental Design.** In accordance with our independent variables, we have three different analysis approaches (i.e., ad-hoc, half COINA, and full COINA). Hence, we designed the experiment accordingly with three groups. **Group Ad-hoc (control group)** applied the ad-hoc approach to perform the conceptual interoperability tasks on the available documentations of two software units. The group was not given any supporting sheet; only documentation templates were provided. **Group Half COINA (experimental group)** performed the same analysis tasks, but was supported by the COIN Cheat Sheet and

documentation templates. **Group Full COINA (experimental group)** performed the same tasks, but was supported by both the COIN Cheat Sheet and the Mismatches Cheat Sheet with the documentation templates. Figure 44 captures the experimental design and shows the exact differences between the groups.

Note that, the participants were randomly assigned to the groups and none of them was known to the experimenter before. The participants did not know about the differences between the groups either.

For ethical learning purposes, we ran the experiment in three sessions with three input examples (e1, e2, and e3). Each input example included an SRS document for a required software unit  $S_x$  and an API document for an offered software unit  $S_y$ . Thus, each of the GSE students got the opportunity to experience each of the three approaches on different days and with different input examples to avoid the bias of results at the last session (see Figure 45).

Run I				Run II			
	Group A	Group B	Group C		Group A	Group B	Group C
<b>Approach A: Ad-hoc</b> Session 1 on Nov. 2 <sup>nd</sup> , 2015	e1			<b>Approach A: Ad-hoc</b> Session 1 on Dec. 14 <sup>th</sup> , 2015	e1	e2	e3
<b>Approach B: Partial COINA</b> Session 2 on Nov. 4 <sup>th</sup> , 2015		e1		<b>Approach B: Partial COINA</b> Session 2 on Dec. 15 <sup>th</sup> , 2015	e2	e3	e1
<b>Approach C: Full COINA</b> Session 3 on Nov. 6 <sup>th</sup> , 2015			e1	<b>Approach C: Full COINA</b> Session 3 on Dec. 18 <sup>th</sup> , 2015	e3	e1	e2

Figure 45 Design of Experimental Sessions

**Participants (Subjects).** The participants of Run I were 27 master-level students from the Team-based Software Development course, who participated on a voluntary basis. All participants had a background in computer science, their average age was 26, and most of them had participated in software engineering projects ( $N = 21$ ). In Run II, 60 bachelor-level students from the GSE course participated, with the motivation to get bonus grades. Not all participants had a background in computer science (e.g., some of them were from electrical engineering, economics, sociology, etc.), their average age was 23, and most of them had participated in software engineering projects ( $N = 51$ ). In order to monitor the influence of the participants on the experimental results, more information was gathered about the participants by asking them to fill out a briefing questionnaire before the experiment was conducted. The questionnaire included questions regarding the participants' background.

**Experimental procedure.** The time allocated to the experiment session was 90 minutes, which included: (1) **Preparation** (25 minutes), which started with an *introduction* tutorial presented to the students about the concepts of interoperability, COINs, and conceptual

mismatches with examples. Also, the participants got a clear statement on the role of the experiment in the course and on their role in the experiment. Then the briefing questionnaire was filled out by the participants. (2) **Execution** (55 minutes), which started with task assignment, where the participants received the materials explaining their task and the procedures in detail. Throughout the execution, the experimenter recorded observations on plausible disturbing factors, such as participants' emotions and events. The data collection instrument used was a pre-defined observation template (see Appendix G). These observations were to be used to analyze confounding variables and to explain extreme values, e.g., outliers. (3) **Finalization** (10 minutes), where the debriefing questionnaire was filled out by the participants to get their feedback on the assigned tasks and approach.

**Experimental tasks.** During the experiment, the participants played the role of software architects/analysts and performed the conceptual interoperability analysis tasks. All groups got the same input example of documents for two software units (for  $S_X$  and  $S_Y$ ) that were supposed to interoperate with each other, but in a different order in the three sessions for Run II. The tasks assigned to the participants were:

- **Task 1:** All groups were to *analyze* the SRS document of  $S_X$  (which specified the characteristics of the required unit) and *extract* the list of COINs (COIN Portfolio) from it.
- **Task 2:** All groups were to *analyze* the API document of  $S_Y$  (which specified the characteristics of the offered unit) and *extract* the list of COINs (COIN Portfolio) from it.

**Task 3:** All groups were to *compare* the two COIN lists (portfolios) and *detect* the list of conceptual mismatches between  $S_X$  and  $S_Y$ .

In other words, the participants were expected to produce two types of results as presented in Figure 44. The first was a COIN List for each analyzed software unit. The second was a Mismatch List between the two software units. To ensure format consistency and comparability, Group A (which applied the ad-hoc approach) was provided with the documentation template for the COINs and the mismatches. In this way, we were able to get the same result artifacts from all groups.

**Materials.** Different artifacts were used to execute the experiment and collect the data. These included the following:

- *A briefing questionnaire*, which was the first input filled out by the participants to collect information about their characteristics and backgrounds.
- *A debriefing questionnaire*, which was the last artifact filled out by the participants after the experiment session to collect their feedback and experience regarding the analysis approach.



- *The input examples* that were analyzed by the participants of the three groups were:
  - (e1), which included a made-up SRS document for a My Job Notification app and a part of the real-world, publicly shared AppleWatch API documentation;
  - (e2), which included a made-up SRS document for a music app and a part of the real-world, publicly shared SoundCloud API documentation;
  - (e3), which included a made-up SRS document for a postman app and a part of the real-world, publicly shared Google Map Directions API documentation.

Note that the API document in e1 is from the platform API domain, while the API documents of e2 and e3 are from the web service API domain.

- *The Documentation Templates for the COINs and the Mismatches*, which guided all groups in documenting their results in a structured format.
- *The COIN Cheat Sheets*, which were provided to the participants of Group Half COINA and Full COINA only.
- *The Mismatches Cheat Sheet*, which was provided to the participants of Group Full COINA only.

Along with these materials, the participants were given a non-disclosure agreement and an informed consent form, an explanation of the experimental procedure, and the task description. See the experimental materials in Appendix H.1.

**Pilot study.** We performed a pilot study with five computer science students (with no previous background in interoperability analysis) with different study levels (i.e., one bachelor student, two master students, and two Ph.D. students). The goal was to assess the understandability of the experimental materials and the sufficiency of the allocated time to read and perform all tasks. We encouraged the participants to ask about any confusing or uncertain words and to write their comments on the space provided on the debriefing questionnaire. Accordingly, the SRS documents and the API documents were reported to be too long to finish analyzing them within the given time. Hence, we reduced their length by deleting paragraphs without affecting the understandability of the input.



7.3.4 Analysis Results

Data Analysis Procedure

**Measuring** One of the most important steps in the data analysis is to evaluate the dependent variables of the participants' analysis results. Therefore, we developed a reference solution for the existing COINs and mismatches for each of the analyzed input examples. The author of this thesis evaluated the results produced by the participants using this reference solution, and a second researcher repeated the evaluation for a portion of the produced results. The observed agreement percentage between the two researchers was 88% and Cohen's Kappa was 0.6 (interpreted as substantial agreement).

**Cases' selection** Run I was applied by 27 students on the e1 example. Thus, the size of the produced data was reasonable to analyze it *all* (i.e., 27 cases each containing two COIN Portfolios and a Mismatches List). However, the results produced in Run II were very large, as we had three sessions with three examples each applied by 60 students (i.e., 180 cases, each containing two COIN Portfolios and a Mismatches List). Because of the time and resource constraints of this Ph.D. work, we *randomly* selected a *portion* of the results of Run II. For comparability reasons with Run I, we selected 36 cases from e1 (i.e., 12 cases per approach).

Then we analyzed further cases from e2 to explore the generalizability of the results among different input examples. Hence, we selected 12 cases from e2 (i.e., 4 cases per applied analysis approach) via the *stratified sampling* technique, using the participants' demographic characteristics and their correlations. That is, we statistically tested the correlation between the four characteristics, which showed some differences with regard to the results (i.e., reuse experience, study program, study level, and English proficiency). We tested the correlation using the Spearman RHO test, as we could not use the Pearson test due to the categorical and nominal nature of the data. Figure 46 shows the results of the Spearman RHO test (denoted as  $\rho$ ). These results mean that Study Program, Study Level, and English Proficiency are correlated and the null hypotheses are rejected with a significant p-value ranging between 0.00 and 0.03. However, with regard to Reuse Experience, there is no evidence on its correlation with any other characteristic and the null hypotheses are not rejected.

**Building the sampling strata** Hence, we chose the Study Level (i.e., one of the three correlated characteristics) along with Reuse Experience for building the sampling strata as follows: Master student with Reuse Experience (Stratum 1), Master student without Reuse Experience (Stratum 2), Bachelor student with Reuse Experience (Stratum 3), and Bachelor student without Reuse Experience (Stratum 4).

**Stratified sampling** Finally, we chose a sampling fraction of 1/3 of the sample size of Run II on e1 (i.e., 36 cases). Thus, we randomly selected 12 cases (i.e., 4 participants per approach, one from each stratum).

Correlations			Reuse_Exp	Study_program	Study_level	English_Prof
Spearman's rho	Reuse_Exp	Correlation Coefficient	1.000	-.302	-.239	-.275
		Sig. (2-tailed)	.	.073	.160	.104
		N	36	36	36	36
	Study_program	Correlation Coefficient	-.302	1.000	.632**	.522**
		Sig. (2-tailed)	.073	.	.000	.001
		N	36	36	36	36
	Study_level	Correlation Coefficient	-.239	.632**	1.000	.347*
		Sig. (2-tailed)	.160	.000	.	.038
		N	36	36	36	36
	English_Prof	Correlation Coefficient	-.275	.522**	.347*	1.000
		Sig. (2-tailed)	.104	.001	.038	.
		N	36	36	36	36

\*\* . Correlation is significant at the 0.01 level (2-tailed).

\* . Correlation is significant at the 0.05 level (2-tailed).

Figure 46 Correlation results from SPSS for the demographic characteristics of the participants

We left e3 for future work as it has the same domain as e2 (i.e., web service API) and the same type of COINs and mismatches as e2.

## Statistical Analysis Results

### Results of Effectiveness ( $H_9$ ) and Efficiency ( $H_{10}$ )

**Analysis methods** After evaluating the selected results cases, the data was entered into SPSS version 23 [Cor10]. Using this tool, we checked the data normality for each measure using the Shapiro-Wilk test and cross-checked it with the Lilliefors test. To compare the results of the three groups when data was not normally distributed, we used the Kruskal-Wallis test. Then, if the null hypothesis was rejected, we ran a post-hoc analysis using Dunn's test, else we performed a pair-wise comparison using the Mann-Whitney (U) test. However, when data was normally distributed, we used the one-way ANOVA test. Then, if the null hypothesis was rejected, we ran the post-hoc Tukey's Honestly Significant Difference (HSD) method, else we performed a pair-wise comparison using an independent t-test. Our one-tailed hypotheses were tested at a p-value < 0.5. Cohen's effect size ( $d$ ) = (mean difference/standard deviation) and its classifications are: small = 0.2, medium = 0.5, and large  $\geq$  0.8.

**Run I on e1** By analyzing the results of the 27 cases in (Run I \* e1), we found that, on average, the results of the full COINA approach were better than the

ad-hoc approach results. That is, with full COINA, COIN extraction from the SRS document improved by 12.5% for precision and by 11.7% for recall. Also, full COINA improved COIN extraction from the API document by 13.4% for precision, but only by 3.4% for recall. Additionally, full COINA improved the mismatch detection by 17.4% for precision and by 3.4% only for recall. Regarding time spent on analysis, full COINA had an improvement of 16.3%. With regard to the difference between half COINA and ad-hoc analysis, we found that the former improved COIN extraction from both SRS and API documents, except for extraction recall from API documents (i.e., half COINA scored 1.6% less). Half COINA also had lower precision (14.7% less) and recall (4.3% less) in detecting mismatches compared to the ad-hoc approach, but the time spent was less (16.8%). Obviously, the full COINA had a better improvement effect compared to half COINA.

- Run II on e1 While analyzing the results of the 36 cases in (Run II \* e1), we found that, on average, the results of the full COINA approach were better than the ad-hoc approach results. That is, with full COINA, COIN extraction from the SRS document improved by 11.9% for precision and by 17.3% for recall. Also, full COINA improved COIN extraction from the API document by 6.6% for precision and by 5.5% for recall. Additionally, full COINA improved the mismatch detection by 33.9% for precision and by 22.4% only for recall. Regarding time spent on analysis, full COINA had an improvement of 12.5%. With regard to the difference between half COINA and ad-hoc analysis, we found that the former improved the results as the full COINA did. Thus, half COINA improved COIN extraction from both SRS (by 8.2% for precision and by 14.4% for recall) and API documents (by 7.4% for precision and by 6.8% for recall). Half COINA also increased precision (36.7% higher) and recall (15.6% higher) in detecting mismatches compared to the ad-hoc approach and the time spent was less (7.2%). Clearly, the full COINA had a better improvement effect compared to half COINA.
- Run II on e2 With regard to the results of the 12 cases in (Run II \* e2), we found that they confirm the improvement results that the full COINA approach showed earlier in Run II on e1. In other words, this extended analysis showed conformance between the effect of full COINA on different input examples (e1 and e2). That is, with full COINA, COIN extraction from the SRS document improved by 9.5% for precision and by 25% for recall. Also, full COINA improved COIN extraction from the API document by 10.8% for precision and by 16.3% for recall. Additionally, full COINA improved the mismatch detection by 14.9% for precision and by 17% only for recall. Regarding time spent on analysis, full COINA had an improvement of 10%. With regard to the difference between half COINA and ad-hoc analysis, we found that the former improved the results as the full COINA did. Thus, half COINA improved COIN extraction from both SRS (by 7.9% for precision and by 25% for recall) and API documents (by 7.5% for precision and by 9.6% for recall). Half COINA also increased precision (15.4% higher) and recall (12% higher)

in detecting mismatches compared to the approach ad-hoc and the time spent was less (3.1%). Thus, the full COINA had a better improvement effect compared to half COINA.

In Table 19, we show all the statistical analysis results for comparing the differences between the different approaches applied on the different runs and examples. The statistically significant improvements (p-value < 0.05) are denoted with a (✓) mark. We also include the effect size for each improvement (Cohen's  $d$ ). For example, the null hypothesis of  $H_{9.1}$  was rejected with statistical significance on both Run I and Run II on e1, which means that the recall results for extracting COINs from structured SRS documents were improved by both half and full COINA compared to the ad-hoc analysis approach. However, in Run II on e2, there were not enough participants to calculate the statistical significance of the improvement. Still, in all the runs and examples, the effect size ranges from medium to large (i.e.,  $d$  between 0.6 and 1.1).

### Results of Acceptance ( $H_{11}$ )

Analysis method	Using the SPSS tool, we ran the one-sample Wilcoxon signed-rank ( $Z$ ) test to check $H_{11.1}$ and $H_{11.2}$ . The test finds out if the responses given in the debriefing questionnaire (in which answers were on a scale from 1: strongly disagree to 5: strongly agree) on perceived usefulness and ease of use are rather in agreement (more than 3).
For all runs and examples	For both Run I on e1 and Run II on e1 and e2, the participants agreed on the ease of use of the full COINA approach in terms of learnability, proficiency, etc. The statistical significance of this agreement could be shown in Run I and Run II on e1, but Run II on e2 could not be calculated due to the small sample size. The participants also agreed on the usefulness of using our approach for achieving better results in terms of completeness, correctness, etc. Although the statistical significance could be shown in Run I, a larger sample size would have been needed in Run II. In Table 20, we show the median value and the statistical significance of the agreements. These results indicate that practitioners show acceptance for our proposed COINA approach and its associated cheat sheets and templates.

Table 19 Analysis results regarding Effectiveness ( $H_9$ ) and Efficiency ( $H_{10}$ ) for Run I and Run II

Effectiveness (Task1: extracting COINs from structured doc.)				Effectiveness (Task2: extracting COINs from unstructured doc.)				Effectiveness (Task3: finding mismatches)				Efficiency (for all analysis tasks)		
H <sub>9.1</sub> : Precision		H <sub>9.2</sub> : Recall		H <sub>9.3</sub> : Precision		H <sub>9.4</sub> : Recall		H <sub>9.5</sub> : Precision		H <sub>9.6</sub> : Recall		H <sub>10</sub> : Time		
	P-value	Cohen's d		P-value	Cohen's d	P-value	Cohen's d	P-value	Cohen's d	P-value	Cohen's d	P-value	Cohen's d	
Ad-hoc vs. half COINA	.01 ✓	.91	.01 ✓	1.43	.37 ?	.07	.27 ?	.13	.19 ✓?	.36	.12 ✓?	.96	.00 ✓	3.05
	.03 ✓	.64	.06 ✓?	.8	.22 ✓?	.6	.21 ✓?	.4	.15 ✓?	.61	.19 ✓?	.58	.00 ✓	2.63
	-	-	-	-	-	-	-	-	.07 ✓?	.89	.015 ✓	1.35	.49 ?	.07
Ad-hoc vs. half COINA	.045 ✓	.77	.047 ✓	.84	.023 ✓	.82	.036 ✓	.77	.0005 ✓	1.2	.002 ✓	1.2	.015 ✓	1.23
	.005 ✓	1.1	.019 ✓	.96	.033 ✓	.78	.062 ✓?	.66	.001 ✓	1.1	.00 ✓	1.5	.00 ✓	1.63
Half COINA vs. full COINA	-	-	-	-	-	-	-	-	.47 ✓?	.32	.404 ✓?	.52	.08 ✓?	.9
Run I on e1				Run II on e1										

Run II on e2												
Ad-hoc vs. half COINA	SNP ✓?	.80	SNP ✓?	1.22	SNP ✓?	.51	SNP ✓?	.81	SNP ✓?	1.09	SNP ✓?	.71
Ad-hoc vs. full COINA	SNP ✓?	1.17	SNP ✓?	1.69	SNP ✓?	1.08	SNP ✓?	1.3	SNP ✓?	1.15	SNP ✓?	2.34
Half COINA vs. full COINA	-	-	-	-	-	-	-	-	SNP ✓?	.03	SNP ✓?	1.88

✓: Improved and statistically significant ( $p < 0.05$ ); ✓?: Improved ( $d > 0.2$ ), but needs more participants to show statistical significance;  
?: No evidence of improvement;; No hypothesis assumed as these are identical parts of the half and full COINA; SNP: statistically not possible to calculate due to the small size of the sample

Table 20 Results of Acceptance ( $H_{11}$ ) for Run I and Run II

$H_{11}$ : Acceptance of COINA approach				
$H_{11.1}$ : Perceived usefulness			$H_{11.2}$ : Perceived ease-of-use	
Median		P-value	Median	P-value
Run I on e1		4	3.9	0.02
Run II on e1		4	3.6	0.001
Run II on e2		4.5	4.4	SNP



### 7.3.5 Discussion

Cross-run COIN extraction correctness	The findings presented in the previous subsection gave us statistically significant evidence about the improvement that our proposed systematic analysis approach COINA can bring to the correctness (measured as precision) of the manual extraction of COINs compared to ad-hoc analysis. This has been shown, in Run II on e1, for both structured artifacts (i.e., SRS documents) and unstructured artifacts (i.e., API documents). Across the runs, the correctness improvement for structured documents had an effect size $d$ ranging from 0.6 to 1.17 standard deviations (SD). This is interpreted according to Cohen's scale interpretation as a medium to large effect. Obviously, the more correctly detected COINs there are, the more mismatches are detected correctly.
Cross-run COIN extraction completeness	Moreover, COINA improved the completeness (measured as recall) of the manual extraction of COINs compared to ad-hoc analysis in structured documents with statistical significance. The completeness improvement has an effect size $d$ ranging from 0.8 – 1.69 SD, which is interpreted as a large effect size. This improvement has a direct influence on the detection of mismatches early in an integration project. Similarly, COINA improved the completeness of the manual detection of COINs in unstructured documents with an effect size $d$ ranging between 0.13 – 1.3 SD. This effect size widely varies from small to large improvement per analyzed service or method. To get statistical significance on this improvement, we would need to analyze more cases. One of the reasons that might be the cause of this variation is the fact that reading API documents is a challenging task, especially for inexperienced students from non-CS majors.
Cross-run mismatch detection correctness and completeness	It is noticeable that COINA improved the correctness (measured as precision) of the manual detection of mismatches compared to ad-hoc analysis, with an effect size $d$ ranging from 0.36 to 1.2 SD. This is considered a rather medium to large. Moreover, COINA improved the completeness (measured by recall) of the manual detection of mismatches compared to ad-hoc analysis. This improvement is of an effect size $d$ that ranges from 0.58 to 1.5 SD, which is interpreted as a rather medium to large effect size.
Cross-run efficiency	On top of precision and recall enhancements, COINA increased the efficiency of the analysis (measured by time in minutes) compared to ad-hoc analysis with very high statistical significance. The effect size $d$ ranges from 0.71 - 3.05 SD. This is interpreted as a rather large to very large effect, which is translated into 10% to 40% faster analysis with better results. Such improvements are of high value for the success of integration projects with limited time.
Observations	With regard to Run I, we notice that the results did not provide the same statistically significant evidence on our hypotheses as in Run II. This is

related to multiple influencing factors we noticed during the execution of Run I. First, there was the “*Randomization Effect*”, which unintentionally led to assigning the participants with the lowest experience score and the lowest English proficiency to the Full COINA Group (e.g., one participant was continuously using a digital translator during the experiment). Also, the Half COINA Group scored the lowest in the “ease of task” debriefing question. The Ad-hoc Group had the participants with the highest experiences. Second, there was the “*Learning Effort Effect*”, which means that the Half and Full COINA approaches required learning the COIN categories and using them within the experiment time, whereas in Run II, the participants were trained on them in a class session and in a special exercise session. Third, there was the “*Motivation Effect*”, which negatively affected the performance of the Full COINA Group. That is, it was observed that the participants were not motivated (e.g., distracted by texting on the phone, arriving late, asking to take the assignment home, attending only for the sake of a participation certificate required by another class, etc.). The Ad-hoc Group, on the other hand, had some highly motivated participants (e.g., two were seeking the experimenter’s supervision for their master theses). Add to this that participation in Run I was on a volunteering basis, while the participants of Run II were promised to get extra points in the GSE class.

This experiment included only one contribution of this Ph.D. work, namely the systematic approach for mismatch detection. It can be expected that the analysis results would be even better if the other components of COINA had been included in supporting the participants.

### 7.3.6 Threats to Validity

**Construct Validity.** This threat concerns the degree to which the experiment measures the stated goals and claimed hypotheses. It includes “*researcher bias*”, which was introduced by the experimenter (who was also the author of the COINA approach under experiment) and which may have influenced the design of the experiment unconsciously so she could prove her claimed hypotheses. In order to alleviate this effect, the experiment design was peer-reviewed by the supervisor of this thesis, Prof. Dieter Rombach, and three senior researchers from Fraunhofer IESE. We also used reliable test instruments (e.g., TAM) and the GQM approach.

**Internal Validity.** This threat concerns the degree to which independent variables influence the dependent variables (i.e., cause-effect relation). It includes “*learning bias*”, which is the influence of maturing skills during the experiment sessions of Run II. The learning effect is expected to be strong if the participants perform the tasks using first the ad-hoc approach, then the Half COINA approach, and finally the Full COINA approach if they perform them on the same input SRS

and API documents. Hence, we developed three different examples of input artifacts (i.e., e1, e2, and e3). Another threat to internal validity is “*evaluation bias*”, which is about influencing the findings through subjective evaluation of the collected data. Hence, we developed a reference solution and evaluation guidance to ensure objective evaluation of the data. Moreover, the evaluation was repeated partially by another researcher. The two researchers’ results had an observed agreement percentage of 88% and Cohen’s Kappa = 0.6, which is interpreted as substantial agreement. In addition, “*sample selection*” can threaten the internal validity if the participants were assigned to groups in a biased way. To avoid this bias, we randomly assigned the participants to the groups. We followed this for both Run I and Run II.

**External Validity.** This threat concerns the degree to which the results of the experiment can be generalized to different people and settings. It includes the concern about the “*representative sample*” as the experiment participants were students either at the master or bachelor level from either a computer science program or a non-computer science one. That is, they were not practitioner software engineers or architects. To alleviate this representation issue, we trained the participants on the concepts through introductory sessions using presentation slides and examples. Another threat to external validity is “*case bias*”, which is about affecting the generalizability of the achieved results by limiting them to specific input artifacts (in our study, these were the analyzed SRs and API documents) on the produced results. To overcome this issue, we included three different input artifacts from two different domains (i.e., platform APIs and web service APIs). We had the examples checked to ensure they were similar in length, difficulty, number, and kind of COINs they have, and the number and kind of mismatches they have. Despite the effort we put into including three examples, the “*case size*” in each of them is still quite small compared to the typical documentation for software systems in industry. An additional concern is the “*included COINs bias*”, which represents an issue related to the generalizability of the results due to the fact that the study did not cover all the different types of the COIN Cheat Sheets. The study included the cheat sheets for the SRS and the API documentation, and it covered most but not all the COIN types due to the time limits of the session. Hence, we do not claim that the results apply to the uncovered sheet and COIN types, but leave their evaluation to future studies.

**Conclusion Validity.** This threat concerns the degree to which the conclusions drawn from the experiment results are correct and proven with sufficient statistical tests. This includes the “*effectiveness metrics*” used for assessing the results of the interoperability analysis tasks. We based these metrics on the COIN Model, which the author proposed as the foundation for the analysis approach. This threat to validity holds especially true for the notion of “correctly extracted COIN” and “incorrectly extracted COIN”. Hence, the experiment definition of

“correctness” must be taken into account when interpreting the results. Another concern is the “*time measurement*”, as the reliability of this metric data affects the assessment of the efficiency of the interoperability analysis approaches and can also be a threat to validity. This is because the instrument used in recording the time stamps was pen and paper used by the participants themselves, even though they were allocated a specific timeframe for each activity throughout the whole experiment. Moreover, what can be counted as a threat to the reliability of measuring time is the fact that students tend to spend all the given time to work on assigned tasks or even on revising their answers, which may affect the preciseness of our conclusions regarding the efficiency of the approach.

## 7.4 Survey and Initial Controlled Experiment

In this section, we present a research study that included both a survey and an initial controlled experiment that we used for evaluating our proposed guidelines for improving API documentation. This study evaluated  $H_6$ ,  $H_7$ , and  $H_8$ . Thus, we will describe the study goal and research questions, its design, data analysis, and the threats to validity.

Note that these two studies have been completely designed by the author of this Ph.D. thesis; however, the execution was performed under her supervision by a master student [Jad16]. The master thesis included additional guidelines identified in the literature beside the ones proposed in this Ph.D. thesis. Hence, we will briefly describe the studies and their results only with respect to the proposed content guidelines in this thesis. For further information about the materials of the two studies, see their web page [Abu16b].

### 7.4.1 Objective and Research Questions

The main goal of this study, formulated by means of the GQM goal template, was *to analyze our guidelines for improving the content of API documentation for the purpose of evaluation with a focus on their perceived and actual effect on the effectiveness and efficiency of conceptual interoperability analysis from the perspective of software developers in the context of a survey with practitioners and an initial controlled experiment with students*. That is, we wanted to know if our proposed content guidelines (see Subsection 6.3.3) have the potential to produce more effective and efficient conceptual interoperability analysis results compared to not applying them. In line with the goal, our research questions are as follows:

**RQ6 (Effectiveness):** *Does implementing the guidelines for improving the content of API documentation enable software developers to detect conceptual constraints more effectively compared to not applying them?*

**RQ7** (Efficiency): *Does implementing the guidelines for improving the content of API documentation enable software developers to detect conceptual constraints more efficiently compared to not applying them?*

**RQ8** (Acceptance): *Do practitioners perceive the guidelines for improving the content of API documentation as enhancing the usefulness and ease of use of the analyzed API documentation?*

In order to achieve the stated goal and answer the aforementioned questions, we performed our research study in two parts as follows:

**Research Part 1 (Survey study).** In this part, we systematically explored the practitioners' acceptance of the value of the guidelines (RQ8). The result of this part directed us to choose a subset of the guidelines to further evaluate them in the next research part.

**Research Part 2 (Initial controlled experiment).** In this part, we performed a controlled experiment with a small number of students to get an indication of the guidelines' actual effect on the conceptual interoperability analysis results (RQ6 and RQ7).

Next, we will describe the design, execution process, and results for both research parts.

## 7.4.2 Survey Study

### Study Design

**Study goal.** We aimed at answering RQ8, which we stated earlier in Subsection 7.3.1. In order to do so, we needed to collect data from software developers in practice.

**Research method.** We performed a survey by means of structured interviews with a sample of software developers. Such a method is helpful for collecting evidence and drawing generalizable results about our guidelines. The survey started with the *preparation*, where a number of software developers in the industry with experience in using APIs were invited to participate in the study after we briefly explained it and its goals. Next, the *execution* was performed by the master student according to the structured interviews we had designed. He explained each guideline with an example, then asked the practitioners about their perceived value of it.

**Target population.** Our target population were software developers who used API and read API documentations in their daily work routine. We invited developers from different software companies. We had our invitation accepted by 20 practitioners whose job titles included intern, software developer, software architect, and software development lead.



Their experiences varied from 1 year to 10 years, and included experience in both software engineering and usage of different APIs.

**Evaluation metrics.** To measure the acceptance of the guidelines, we used a subset of the metrics defined in the TAM model for ease of use and usefulness. This subset is the same as the one we used in the multi-controlled experiment acceptance metrics (see Figure 43).

**Statistical hypotheses.** In this study, we intended to answer the previously stated research questions, namely RQ6, RQ7, and RQ8. In order to do this, we derived the statistic null hypotheses ( $H_0$ ) and the corresponding alternative hypotheses ( $H_1$ ) from the stated research questions. Note that the arithmetic median of acceptance is denoted by  $\mu$  and represents the answer on a response scale ranging from 1 (Strongly disagree) to 5 (Strongly agree). Thus, our hypotheses are as follows:

$H_{8.1,0}$  (Guideline effect on Ease of Use):  $\mu = 3$  and  $H_{8.1,1} : \mu > 3$

$H_{8.2,0}$  (Guideline effect on Usefulness):  $\mu = 3$  and  $H_{8.2,1} : \mu > 3$

**Data collection and questionnaire.** The answers of each interviewee were collected in a sheet, then all answers from all interviews were saved to an Excel sheet. The questionnaire included three parts (i.e., demographic questions, ease-of-use questions about the guidelines, and usefulness questions about the guidelines). For the questionnaire and the data collection sheet, see the study web page [Abu16b].

## Execution and Results

**Pilot study and execution.** We ran a pilot with four participants from industry to get their feedback on the questionnaire's understandability, length, and complexity. The results showed that there were no reported issues at all and the estimated time for the interview session was about 40 minutes. Following this pilot, the interviews were performed in person during July 2016.

**Results for acceptance of the content guidelines.** The collected data was analyzed descriptively and statistically as seen in Table 21. As the normality test showed that the collected data are not normally distributed, we used the one-sample Wilcoxon signed-rank test. This test aims at checking if the agreement answers on the guidelines' effect on ease of use and usefulness are statistically significant.



Table 21

Results of the survey on the guidelines for improving API documents

Guideline ID <sup>a</sup>	Acceptance	Median	H <sub>8</sub> test statistics <sup>b</sup>	
			Z	P-value
CG1	Ease of Use (H <sub>8.1</sub> )	5	190.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	190.0	0.000***
CG2	Ease of Use (H <sub>8.1</sub> )	5	186.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	167.0	0.000***
CG3	Ease of Use (H <sub>8.1</sub> )	5	171.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	171.0	0.000***
CG4	Ease of Use (H <sub>8.1</sub> )	4	102.0	0.001***
	Usefulness (H <sub>8.2</sub> )	4.5	132.0	0.001***
CG5	Ease of Use (H <sub>8.1</sub> )	4	153.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	171.0	0.000***
CG6	Ease of Use (H <sub>8.1</sub> )	5	210.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	210.0	0.000***
CG7	Ease of Use (H <sub>8.1</sub> )	4.5	153.0	0.000***
	Usefulness (H <sub>8.2</sub> )	4	105.0	0.001***
CG8	Ease of Use (H <sub>8.1</sub> )	5	210.0	0.000***
	Usefulness (H <sub>8.2</sub> )	5	210.0	0.000***
CG9	Ease of Use (H <sub>8.1</sub> )	4	74.0	0.004**
	Usefulness (H <sub>8.2</sub> )	4	87.0	0.003**
<sup>a</sup> See guidelines and their IDs in Subsection 6.3.3 <sup>b</sup> One-sample Wilcoxon signed-rank test H <sub>0</sub> : median (all respondents) = 3 (sometimes); * p < 0.05 ; ** p < 0.01, *** p < 0.001				

According to the presented results, all null hypotheses were rejected. That is, it was revealed that there is a consensus among practitioners on the value of our proposed content guidelines for enhancing the ease of use and usefulness of API documents. The agreement on accepting our guidelines is of high statistical significance, ranging from 0.00 to 0.003.

We decided to select a subset of these content guidelines to further evaluate their actual effect on the results of conceptual interoperability analysis. Our selection criterion was that the guidelines must be fully accepted with very high significance. That is, the guideline has to have the median value = 5 for both its perceived ease of use and effectiveness, and has to have its related null hypotheses rejected at p-value = 0.0. Thus, the guidelines we selected for the next step of the evaluation were CG1, CG2, CG3, CG6, and CG8. See [Jad16] for further descriptive analysis and charts.

## Threats to Validity

**Construct validity.** To avoid the risk of using improper measures (“*researcher bias*”), we used reliable metrics for acceptance (i.e., TAM) and followed the GQM approach. We also had the study design peer-reviewed by a second researcher with experience in empirical software engineering.

**Internal validity.** To avoid “*experimenter bias*”, we designed our interview to be structured to ensure that we would not get different results between the interviews. Furthermore, we evaluated the survey in pilot studies to assess the understandability of the questionnaire. Another concern is “*misinterpretation*” of our questions during the interview. To mitigate this issue, we allowed the participants to read the guidelines by themselves besides receiving our verbal explanation.

**External validity.** The number of participants in the study was 20 software developers with various levels of experience in using different types of APIs and reading their documentation. Although the results showed high statistical significance, we realize that performing further surveys with a larger sample size would increase confidence in the “sample representation” and would allow better generalizability of the results.

### 7.4.3 Initial Controlled Experiment

In this subsection, we present our further evaluation of the selected subset of guidelines (i.e., CG1, CG2, CG3, CG6, and CG8) using an initial controlled experiment. We implemented these selected guidelines in a new version of an already existing API document.

#### Experimental Context

**Study goal.** We aimed at answering RQ6 and RQ7, which we stated earlier in Subsection 7.3.1. In order to do so, we needed to experiment with the effect of applying the selected guidelines on the conceptual interoperability analysis results.

**Experimental context.** The experiment was performed with eight master students at the University of Kaiserslautern, Germany (TU KL), on a voluntary basis.

#### Experimental Setup

**Study variables.** For this designed controlled experiment, we defined the variables as follows:

*Dependent variables:* the effectiveness and efficiency of the conceptual analysis results.

*Independent variables:* the input of the API document analyzed for the conceptual analysis.

**Statistical hypotheses.** To answer RQ6, and RQ7, we derived the statistical null hypotheses ( $H_0$ ) and the corresponding alternative hypotheses ( $H_1$ ). Note that the arithmetic mean of the effectiveness and efficiency of the analysis using the modified input according to the selected guidelines is  $\mu_b$ , while  $\mu_a$  denotes the mean of the effectiveness and efficiency of the analysis using the original API document without applying the selected guidelines. Thus, our hypotheses are as follows:

$H_{6.1,0}$  (Guideline effect on Analysis Effectiveness):  $\mu_b \leq \mu_a$  and  $H_{6.1,1}$ :  $\mu_b > \mu_a$

$H_{6.2,0}$  (Guideline effect on Analysis Efficiency):  $\mu_b \leq \mu_a$  and  $H_{6.2,1}$ :  $\mu_b > \mu_a$

**Operationalization (Evaluation metrics).** As stated previously, the comparison in  $H_6$  and  $H_7$  is between analysis using modified API documents using our guidelines and analysis using the original API documentation without applying our guidelines. In order to perform the comparison, we needed to have specific metrics that can be calculated from the results produced by the participants in the experiment. In order to evaluate the *effectiveness* in  $H_6$ , we used the following two basic metrics for measuring the correctness of the analysis:

- True positive (TP): a COIN/mismatch instance that is correctly identified as a COIN/mismatch instance
- False positive (FP): a non-COIN/non-mismatch that is incorrectly identified as a COIN/mismatch instance

On the other hand, we evaluated *efficiency* in  $H_7$  by directly using a basic cost metric, namely Time in minutes spent on the task.

**Experimental design.** According to our independent variables, we have two different input artifacts for the conceptual analysis task. Hence, we designed the experiment with two groups accordingly. **Group A (control group)** applied the conceptual analysis tasks to the original API document (i.e., the original SoundCloud API document). **Group B (experimental group)** performed the same analysis tasks, but was given the modified API documents (i.e., a new version that we had created for the SoundCloud API document).

**Participants (Subjects).** The participants were eight master-level students from the CS Department of TU Kaiserslautern, who participated on a voluntary basis. All of them had participated in software engineering projects and all had experience in reading UML

diagrams. Some of them (3 students) had basic experience in software reuse (i.e., 1-2 years of experience) and all reported having a level of 4 out of 5 with regard to English proficiency. In order to monitor the influence of the participants on the experimental results, more information was gathered about the participants by asking them to fill out a briefing questionnaire before the experiment was conducted. Note that the 8 participants were randomly assigned to the groups and none of them had been known to the experimenter before. The participants did not know about the differences between the groups either.

**Experimental procedure.** The time allocated to the experiment session was 90 minutes, which included: (1) **Preparation** (20 minutes), which started with an *introduction* for the students about the concepts of interoperability analysis and clarification of their role in the experiment. Then the briefing questionnaire was filled out by the participants. (2) **Execution** (30 minutes), which started with task assignment, where the participants received materials explaining their task and procedures in detail. Throughout the execution, the experimenter recorded observations regarding plausible disturbing factors, such as the participants' emotions and events. (3) **Finalization** (10 minutes), where the debriefing questionnaire was filled out by the participants to get their feedback on the task and the input document.

**Experimental tasks.** During the experiment, the participants played the role of software architects/analysts and performed the conceptual analysis task. All participants brought their laptops as we had requested in the invitation. We gave each participant a copy of the digital .html files for the API document version assigned to their group. Then we handed the participants the printed material, which included the task description and an experiment sheet. After reading the question, the participants were to find the answer from the API document and fill in the answer in its dedicated space in the printed material.

- **Task 1:** All groups were to *read* the digital API document on their laptops.
- **Task 2:** All groups were to *read* the questions in the printed experiment sheet, which contained questions about some conceptual constraints of the SoundCloud API.
- **Task 3:** All groups were to look for the answer in the API document, then document it in the allocated space in the question sheet.

**Materials.** Different artifacts were used to execute the experiment and collect the data. These included:

- *A briefing questionnaire*, which was the first input filled out by the participants and which was aimed at collecting information about their characteristics and background.

- *A debriefing questionnaire*, which was the last artifact filled out by the participants after the experiment session to collect their feedback and experience on the analysis input.
- *The digital input examples*, which were analyzed by the participants of the groups on their laptops.
- *The Experiment Sheet*, which included 11 questions about conceptual constraints and space for answering the questions.

Along with these materials, there was a non-disclosure agreement and an informed consent form. For the experimental materials, see Appendix I.1.

## Analysis Results

**Data Analysis Procedure.** One of the most important steps in data analysis is to evaluate the dependent variables of the participants' analysis results. Therefore, we developed a reference solution for the questions on the experiment sheet (i.e., see Appendix I.2) and assigned a weight to each question (29 points in total). After evaluating the answers of the eight participants according to the reference solution, we ran some descriptive analyses for the data, but no statistical analysis, as the sample size is too small and the error rate would be very high.

**Effectiveness results (H<sub>6</sub>).** By analyzing the results of the eight cases, we found that, on average, the correctness results of Group B (using API documents modified according to our guidelines) were better than the results of Group A (using the original API document). That is, Group B scored an average of 20.5 out of 29 (70.7% correctness), while Group A scored 10.75 out of 29 (37.1% correctness). In other words, Group B produced results that were 33.6% more correct than Group A. This indicates that our evaluated content guidelines do have the potential to improve the effectiveness of conceptual interoperability analysis.

**Efficiency results (H<sub>7</sub>).** The data collected about the time spent on tasks indicates a minor improvement. On average, Group B (using API documents modified according to our guidelines) spent 26 minutes, while Group A (using the original API document) spent 27 minutes. This is a small 3.7% improvement on efficiency. Hence, there is not enough evidence on the ability of our evaluated content guidelines to introduce an improvement effect on the efficiency of the analysis task of API documents. This might be related to the typical tendency of students to spend all given time working on tasks even if they were actually done.

**Acceptance.** When analyzing the debriefing questionnaire data for the eight participants, we found that Group B had an increase (ranging from 10% to 12%) in the feedback provided on the perceived ease of use

and usefulness of the documents in producing the results (see Table 22).

Table 22

Acceptance results from the debriefing questionnaire

Acceptance Variable	Group A	Group B
Perceived ease of use of the API document	70%	80%
Perceived efficiency in analyzing the API document	55%	65%
Perceived effectiveness in analyzing the API document	60%	72.5%

### Threats to Validity

**Construct Validity.** This threat includes “*researcher bias*”, which was introduced by the experimenter (who was also the author of the guidelines under experiment), might have influenced the design of the experiment unconsciously so she could prove her claimed hypotheses. In order to alleviate this effect, the experiment design was peer-reviewed, and reliable test instruments (e.g., TAM) and the GQM approach were used.

**Internal Validity.** This threat includes “*evaluation bias*”, which is about influencing the findings by subjective evaluation for the collected data. Hence, we developed a reference solution to ensure objective evaluation of the data. In addition, “*sample selection*” can threaten the internal validity if the participants are assigned to groups in a biased way. Hence, we randomly assigned the participants to the groups.

**External Validity.** This threat includes concern about the “*representative sample*”, as the experiment participants were students and not practitioners. Also, there is “*case bias*”, which may affect the generalizability of the achieved results by limiting them to the specific API example used in this experiment (i.e., the SoundCloud API documentation). An additional concern is “*included guidelines bias*”, which represents an issue related to the generalizability of the results due to the fact that the study did not cover all the content and presentation guidelines proposed in Subsection 6.3.3. Hence, we do not claim that the results apply to the uncovered guidelines, but leave their evaluation to future studies.

**Conclusion Validity.** This threat includes concern about “*time measurement*”, as the reliability of this metric data affects the assessment of the efficiency of the interoperability analysis approaches and can also be a threat to validity. This is because the instrument used in recording the time stamps was pen and paper used by the participants themselves, even though they were allocated a specific time for each activity throughout the whole experiment. Moreover, what can be counted as a threat to the reliability of measuring time is the fact that students tend to spend all given time to work on assigned tasks or



even on revising their answers, which may affect the preciseness of our conclusions regarding the efficiency of the approach.

#### 7.4.4 Summary

In this section, we presented a research study that combined a survey with practitioners and an experiment with students in order to investigate the perceived and the actual value of our proposed content guidelines for improving API documentation. We included all nine of our content guidelines in the survey and got confirmation from 20 practitioners from industry on the potential of these guidelines in improving the results of the conceptual analysis. Although all the guidelines got statistically significant agreement on their improvement effect, we chose the top-rated ones for inclusion in the next part of the evaluation. The second evaluation part was a controlled experiment performed with a small sample size ( $N = 8$ ). The control group was given the original API documentation of SoundCloud, while the treatment group was given the modified version of the documentation in which we had implemented our selected content guidelines. Both groups were asked to answer the same set of questions about some conceptual constraints of the API. The treatment group results showed that our implemented guidelines did affect the correctness of the analysis results compared to the results of the control group. Also, the participants reported better perception of the ease of use and usefulness of the API documentation in the analysis task. However, we did not get enough evidence or indication of the effect of the guidelines on the efficiency of the analysis. Due to the limited number of participants, it was not possible to perform statistical tests on the collected data. Therefore, the experiment results cannot be generalized and should be interpreted as an indication. In the future, further experiments on the effect of the guidelines should be performed with a larger sample size to get generalizable evidence supported by statistical significance.

### 7.5 Multiple-Case Study

In this section, we present the evaluation results of  $H_2$ , which states that the COIN Model is comprehensive in covering the different types of conceptual interoperability constraints of software units. We performed the evaluation by means of a multiple-case study, where we investigated each sentence in six API documentations from different domains to identify the types of existing conceptual constraints in these real-world artifacts. We explained the design of the study and its threats to validity in detail in Subsection 6.3.2, as its first goal was to utilize it in developing our ML-based COIN extraction method. In this section, we will describe the second goal of this study and its related research questions, data analysis, and discussion.

### 7.5.1 Objectives and Research Questions

The goal of this evaluation, formulated by means of the GQM goal template, was *to analyze the COIN Model for the purpose of evaluation with a focus on its comprehensiveness in covering the different types of conceptual interoperability analysis from the perspective of interoperability researchers in the context of a multiple-case study*. That is, we wanted to know if our proposed model falls short in covering any conceptual interoperability constraint that actually existed in one of the API documentation cases that we included in the study. In line with the goal, our research question is as follows:

**RQ1** (Comprehensiveness): *Does every existing instance of a conceptual interoperability constraint in the API document have a matching category and attribute for it under the proposed COIN Model?*

**Measures.** To answer this question, we used a simple metric, which is “the percentage of conceptual constraints found that are not covered in the COIN Model”.

In order to achieve the stated goal and answer the aforementioned question, we performed our multiple-case study, where we analyzed each sentence in the six cases against the COIN Model. For more details on the study design, process, and execution, please refer to Subsection 6.3.2.

### 7.5.2 Data Analysis and Discussion

Overall, the multiple-case study showed very positive results with regards to the comprehensiveness of our proposed COIN Model. The results of the comprehensiveness analysis are summarized in Table 23. As seen in the presented results, for all six investigated cases of API documentation, the percentage of conceptual constraints uncovered by the proposed COIN Model is 0%. In other words, within the six cases, the COIN Model achieved 100% coverage for existing types of conceptual constraints on the systems level, data level, and service level. The only sentences not covered under the model relate to non-conceptual information (e.g., references, technical recommendations, development examples, etc.). This strongly indicates the capability of our COIN Model to offer a solid foundation for analyzing currently published documentations about interoperable software units.

When we compare our COIN Model to other existing models of conceptual interoperability (see Section 4.3), we find in addition that the COIN Model is more comprehensive. In other words, it covers more conceptual categories and attributes than any of the other models. As mentioned earlier, the existing models and classifications of interoperability found in the literature are rather abstract compared to

the detailed attributes and description of our model. Hence, the COIN Model paves the way for practitioners to benefit from its detailed categories and attributes for their analysis purposes in software integration projects.

Table 23 COIN Model comprehensiveness data analysis

API Document (Case)	No. of sentences	Percentage of conceptual constraints	Percentage of uncovered conceptual constraints
Sound Cloud	219	35.9%	0%
GoogleMaps	473	37.0%	0%
AppleWatch	360	59.2%	0%
Eclipse Plugin	651	71.0%	0%
Skype	325	63.4%	0%
Instagram	255	58.4%	0%
<b>Total</b>	<b>2283</b>	<b>58.0%</b>	<b>0%</b>

With regard to the generalizability of our results, we included six cases of API documents (i.e., SoundCloud, GoogleMaps, Skype, Instagram, AppleWatch, and Eclipse-Plugin Developer Guide) from two different domain types (web service APIs and platform APIs). Thus, the results are very likely to be representative of currently shared API documentation (which is a type of public documentation for interoperable software units). However, further case studies with larger numbers of sentences are needed to generalize the results and to observe changes over time.

## 7.6 Summary

In this chapter, we presented our empirical evaluation contributions for this Ph.D. work. We started by presenting our multi-run controlled experiment, which we used to evaluate the hypothesized benefits of our systematic approach for conceptual interoperability analysis. In particular, we tested our hypotheses about the effectiveness, efficiency, and acceptance of our approach. The results of the experiment show that software architects and analysts can produce more correct and complete analysis results in less time when they use the proposed systematic approach of COINA. The results have also shown that the approach is perceived as useful and easy to use.

The second study we presented in this chapter included a survey and a controlled experiment to evaluate both the perceived and the actual effect of applying our proposed content guidelines. The survey showed statistically significant agreement from practitioners about the value of applying the guidelines. The guidelines rated top by the survey participants were further evaluated through the controlled experiment. The results of the experiment show a strong indication of the actual effect of applying the guidelines on the effectiveness of conceptual

interoperability analysis. Due to the small number of experiment participants, it was not possible to show the statistical significance.

Finally, we presented a further analysis for our multiple-case study data in order to evaluate the comprehensiveness of our COIN Model in covering the existing conceptual constraints in current API documents. The results were very positive as in all six cases, the COIN Model showed 100% coverage of the cases' conceptual constraints.



---

## 8 Summary

In this chapter, we will summarize our thesis contributions and major results, discuss some ideas for potential future work, and finally provide concluding remarks.

### 8.1 Contributions and Results

**Thesis goal**      Successfully integrating a software system with an existing software system requires more than resolving any technical mismatches. It requires identifying and resolving the conceptual mismatches that might result in worthless technical integration and costly rework. The overall goal of this Ph.D. work was to support software architects and analysts in performing effective and efficient conceptual interoperability analysis in black-box integration projects.

#### Foundation Contribution

We have contributed a conceptual model for conceptual interoperability constraints, the “COIN Model”. It represents a basis for all the subsequent contributions of this thesis. The model captures the notion of a conceptual constraint and its relations with interoperable software elements, software types, and mismatch types. We evaluated the model’s comprehensiveness in covering existing conceptual constraints in current publicly shared documentations of interoperable software units.

#### Methodical Contributions

To achieve our goal, we proposed the COINA Framework for supporting conceptual interoperability analysis. This framework includes two components.

The first component of COINA supports providers of black-box interoperable software units in performing in-house preparation by proactively sharing comprehensive documents about the conceptual constraints of their units. This is achieved with the help of our contributed methods, which facilitate extracting the COINs of a software unit from its documentation. One contributed extraction method is based on our pre-defined templates, which identify COINs from internally shared UML diagrams. The other contributed extraction method is based on the ML COIN Classification Model we produced, which identifies COINs automatically from NL text of publicly shared API documents. This method showed very promising results as it achieved



an accuracy of 82% in automatically detecting seven COIN classes and 87% in detecting two COIN classes. We further support providers with guidelines to improve the API documentation with regard to the content and presentation of conceptual interoperability constraints.

The second component of COINA supports third-party clients in performing successful conceptual interoperability analysis that reveals the conceptual mismatches existing between two software units. This is achieved with the help of our proposed systematic algorithm-based analysis approach, which we support with cheat sheets and documentation templates for the COINs and the mismatches. It offers detailed guidance for inexperienced architects and analysts to manually extract the COINs of two software units and map them in order to find existing mismatches.

### Technical Contributions

To bring our methodical ideas to practical life, we implemented an add-in for the Enterprise Architect architecture modeling tool. This tool allows the semi-automatic extraction of conceptual constraints from UML diagrams. By implementing our contributed formal COIN Extraction Templates, the architects can determine the software interoperable elements that they are looking for by finding and documenting their constraints automatically.

In addition, we extended our technical contribution by implementing an add-in tool for the Chrome web browser to make our proposed ML-based COIN extraction method practical. This tool embeds our contributed Machine Learning COIN Classification Model. Thus, it allows architects to select natural language sentences in an API document and choose the COIN types they are looking for, and the tool then automatically reports if the selected sentences have such COINs.

### Empirical Evaluation Contributions

We evaluated parts of our contributions in a number of empirical studies. We performed a multi-run controlled experiment to evaluate our proposed systematic approach for detecting conceptual mismatches. The results of this experiment show that our approach has a statistically significant effect on enhancing both the effectiveness and efficiency of the analysis results.

In addition, we performed a survey to evaluate the perceived effect of our proposed content-related guidelines. The survey results provided us with statistically significant agreement from practitioners on the perceived usefulness and ease of use of API documents when applying our guidelines. We further evaluated the guidelines rated top by the practitioners through a controlled experiment with a small number of participants. The experiment results show a strong indication of the

actual effect of our selected content guidelines on the correctness of the results of conceptual interoperability analysis.

Moreover, we evaluated the comprehensiveness of our proposed COIN Model within a multiple-case study. The results that we derived from the six cases show that our model is solid in covering the different conceptual constraints (100% coverage).

## 8.2 Benefits and Limitations

In this subsection, we will shed light on the main benefits and limitations of our research work and contributions.

Overall, this Ph.D. research has great strength from the scientific research point of view. This is reflected in the fact that all of our included research studies (the SLR, the surveys, the multiple-case study, and the experiments) were methodologically rigorous and systematic studies. This minimizes bias and threats to the validity of our results and findings. It also supports traceability between the different activities and our reported results and derived conclusions. Thus, it enables future researchers to independently replicate our work and compare their results with ours.

With regard to the practical benefits of our work, we mainly expected and partially validated that the COINA Framework has the capability to increase software architects' effectiveness (in terms of completeness and correctness of the analysis results) and efficiency (in terms of time and effort spent) in identifying the conceptual constraints and mismatches of their software systems. Beside this main practical advantage, we believe that the added value of COINA from the practitioners' point of view includes the following aspects:

- *Extensible framework.* The components of the COINA Framework are designed to allow flexible extension. For example, the Interoperability Knowledge Base (IKB), which we introduced in Subsection 6.3.1, the COIN Extraction Templates and can be easily maintained and extended by architects and domain experts. That is, they can add further templates to cover extra input types (e.g., templates for automatically extracting COINs from structured SRS documents). Moreover, the COIN Corpus can also be extended with further manually labeled NL sentences to increase the prediction accuracy of the classifier model.
- *Technology-independent conceptual and methodological support.* The supporting methods, cheat sheets, and templates offered by the COINA Framework will work for any software unit regardless of its implementation technology. For example, the COIN Extraction Templates can be implemented for modeling languages other than the UML. Also, the ML COIN Classification Model and its tool can

be used to analyze Java API documents or .Net API documents. Add to this that the COIN Model is designed to cover all types of conceptual constraints in any software unit regardless of its domain or size.

- *Experience-independent analysis.* The COINA Framework provides detailed guidelines on how to apply it and use the supporting tools it offers. Hence, architects and analysts with limited experience can follow the directions and easily learn how to perform effective preparation for a software unit as well as effective analysis for mismatches between software units.
- *Ready-to-use shared documents.* The COIN Portfolio is the result of proactive preparation of interoperable software units, which takes place before the need for interoperation arises. Hence, having interoperability-related conceptual information all set and ready to use accelerates the analysis and the building of interoperation among systems. Accordingly, providers improve their business impact and competitiveness.
- *Traceable and repeatable analysis results.* Our proposed systematic analysis approach allows practitioners to trace the results between the analysis activities. For example, a mismatch recorded in the Mismatches List can be tracked down to the COIN(s) causing it, which in turn can be traced to their related interoperable elements. In addition, systematic analysis allows repeating it by following the exact same steps and comparing the achieved results to increase the trust and reliability of the decisions made. These well-documented results and replications are a form of saving interoperability analysis experiences, which can be used for training purposes, to reflect on performance, and to plan future improvements of analysis-related activities.
- *Consistent documentation.* The standard documentation templates proposed by the COINA Framework for recording the extracted COINs and the detected mismatches enable consistency. This consistency is achieved within the analysis results of an integration project and among different integration projects. Obviously, this makes it easier to learn how to read and use the documents and where to locate specific pieces of information in it. It also enables comparison between the results achieved by analyzing multiple candidate units. Accordingly, this supports making trade-offs and arguing reasons for selecting a specific candidate over others.

Despite the benefits of the COINA Framework, it does have some limitations that we could not resolve due to the time and resource constraints of this PhD work. We describe these limitations as follows:

- *Limited context of COINA.* The framework currently only offers support for conceptual interoperability analysis in the context of black-box integration. For example, the extraction methods do not cover input artifacts like source code or activity log, which would be

available in white-box integration. However, the extensibility of the framework allows extending it to include such artifacts. Accordingly, the supporting tools and guidelines would have to be extended, too.

- *Limited extraction templates for UML diagrams.* The proposed set of templates is initial and basic, which may lead to false negatives in the extraction results. That is, a COIN instance might not be addressed by our predefined COIN Extraction Templates, so it may get missed in the extraction process. To alleviate this issue, the framework's modularity allows extending its IKB with more templates as needed. For example, templates can be introduced to cover further UML diagrams (e.g., state machine diagrams, composite structure diagrams, etc.), further COIN types (e.g., technical, business, etc.), and different documents with further abstraction levels (e.g., high-level architecture documents).
- *Input-sensitive.* The results of the COIN extraction method from UML diagrams is subjective regarding the quality of the input assumed to exist (i.e., architectural documentation). For example, incomplete UML diagrams will lead to missing information in the created COIN Portfolio that is shared with clients. Although we hypothetically assumed that the input is complete and correct, we are still working on mitigating this challenge by further including other input artifacts (i.e., the API documentation and the SRS) in order to complement the extraction results.
- *Limited size of the COIN Corpus (ground truth).* It is known in the ML field that the larger the corpus, the better the accuracy results. Also, the more classification classes you want to train the machine classifier on identifying, the more training data it must be fed. This explains the higher accuracy we achieved using the Two-COIN Corpus compared to the Seven-COIN Corpus, even with the same number of sentences in both.
- *Unbalanced number of instances for each class in the corpus.* As noticed, the number of instances for the COIN classes is not balanced in the corpus. That is, dominant classes (i.e., Not-COIN, Dynamic, and Semantic) contribute the majority of the sentences in the data set (i.e., 91%), whereas the other classes (i.e., Structure, Syntax, Quality, and Context) are smaller and share the remaining 9% of the corpus. This affects the classification accuracy of classes with the fewer instances.
- *Limited automation capability for mapping the COIN Portfolios.* Although we presented an example for the potential of automating the mapping activity to detect conceptual mismatches, full automation for this activity is very difficult. On the one hand, formalizing the whole COIN Portfolio document for each interoperable software unit is a tedious and time-consuming task, which requires knowledge and experience in formalization languages. On the other hand, this requires semantic descriptions of all COINs with all their possible values. Nevertheless, we

consider our contributed mapping approach an important computer science contribution, as it formalizes the mapping process in an algorithm.

- *Limited context of the ML-based support.* The current version of our classification model cannot be used to identify COINs from any API document. This is due to the fact that our corpus is relatively small (i.e., 3k sentences) and is built from six cases only (which has specific characteristics with regards to company size, document-writer features like his language or role, the maturity of the API, etc.). Hence, it will not be appropriate to generalize the features of the sentences in our small corpus to all existing sentences of all API documents. Thus, we intend in the future to enlarge the corpus (e.g., hundreds of thousands of sentences) to cover a wider range of API documents with different characteristics and to update the classification model with further features based on the new sentences. Accordingly, our tool support is currently reliable in the context of our six cases and their similar cases.

### 8.3 Future Work

In this section, we will describe our ideas for potential future work and directions aligned with the contribution areas of this thesis.

In our foundation contribution, the COIN Model, we focused on building a solid basis for the analysis-related activities of the conceptual level of interoperability. The model could be extended to better support other activities (e.g., resolution design and implementation) and other levels of interoperability (e.g., technical and organizational).

With regard our methodical contributions within the COINA Framework, Section 8.2 described some limitations that open the door for many possible improvements, including the following:

- Extending the automatic COIN extraction methods (both the template-based method and the ML-based method) to cover artifacts other than software architecture diagrams and API documents. For example, they could be extended to include other input like the software requirements specification (which describes goals, data, interaction, structure, etc.) using different models like Business Process Modelling Notations (BPMN). It could also be extended to automatically extract COINs from source code and transaction logs.
- Extending the ML-based COIN extraction method to recognize the perspectives to which a COIN is related. This could be a tough task to train the machine on recognizing, due to the lack of standardization in the format and structure of API documents. However, it may be potentially possible to recognize them with the help of the special HTML tags used for formatting the heading that declares a new service of the system.

- Extending the systematic analysis approach to support the conceptual interoperability analysis in the context of white-box integration. Thus, further cheat sheets and guidelines could be provided to support the manual extraction of COINs in this context (e.g., reverse engineering COINs from code, execution logs, history reports, or even from StackExchange entries and Wikis).
- If the aforementioned extensions were performed, then our supporting tools could also be extended. For example, the add-in for the Chrome browser could be extended to search for COINs in source code pages, too.

On the empirical evaluation level of our contributions, we see room for the following extensions:

- Extending our multi-run experiment with further experiments that cover more elements of our systematic analysis approach. For example, include the Cheat Sheet for UML and cover the rest of the uncovered types of COINs in the analyzed input.
- Extending our experiment for evaluating the guidelines with further experiments that include larger numbers of participants. These further experiments could also include the presentation guidelines, which we did not evaluate. Moreover, it could be of benefit to run the experiments with experts from industry rather than with students to draw a clear border between the performance results of experienced and inexperienced participants.
- Extending the experiments to include our technical contributions (i.e., the add-ins for Enterprise Architect and for the Chrome web browser). That is, both tools still need to be evaluated with regard to their actual effect on enhancing the analysis results. Although the add-in for the Chrome web browser has been evaluated with regard to its acceptance and the results have shown significant agreement on its value (see the master thesis supervised by the author of this thesis [Nai17]), the acceptance of the add-in for the Enterprise Architect has not been evaluated yet.





---

## References

- [AA96] Ahmed Abd-El-Shafy Abd-Allah. *Composing heterogeneous software architectures*. PhD thesis, University of Southern California, 1996.
- [AAHR16] Hadil Abukwaik, Mohammed Abujayyab, Shah Rukh Humayoun, and Dieter Rombach. Extracting conceptual interoperability constraints from API documentation using machine learning. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 701–703. ACM, 2016.
- [AANR17] Hadil Abukwaik, Mohammed Abufouda, Thejashree Nair, and Dieter Rombach. How practical is it? Machine learning for identifying conceptual interoperability constraints in API documents, 2017. [Online: <http://abukwaik.com/site/wp-content/uploads/2017/04/Abukwaik-JSS-2017.pdf>; accessed 14-Feb-2017].
- [AAR15] Hadil Abukwaik, Mohammed Abujayyab, and Dieter Rombach. Coinsextractor: The architects’ buddy in identifying conceptual interoperability constraints. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, page 69. ACM, 2015.
- [AAR16] Hadil Abukwaik, Mohammed Abujayyab, and Dieter Rombach. Towards seamless analysis of software interoperability: Automatic identification of conceptual constraints in api documentation. In *Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings*, pages 67–83. Springer, 2016.
- [AB97] Christopher M Abts and Barry W Boehm. Cots software integration cost modeling study. 30602:1095, 1997.
- [ABG<sup>+</sup>13] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolk, and Indika Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.
- [Abu14a] Hadil Abukwaik. Empirical evaluation for the conceptual interoperability analysis approach: A controlled experiment design. Technical report, 2014.

- [Abu14b] Hadil Abukwaik. Materials of the multiple-case study, 2014. [Online: <http://abukwaik.com/site/multiple-case-study15>; accessed 14-Feb-2017].
- [Abu14c] Hadil Abukwaik. Materials of the scoping study, 2014. [Online: <http://abukwaik.com/site/scoping-study14>; accessed 14-Feb-2017].
- [Abu16a] Mohammed Abujayyab. Extracting conceptual interoperability constraints from API documentation, 2016.
- [Abu16b] Hadil Abukwaik. Materials of the related studies to the API documentation guidelines, 2016. [Online: <http://abukwaik.com/site/api-guidelines16>; accessed 14-Feb-2017].
- [Abu16c] Hadil Abukwaik. Software interoperability analysis - introduction, 2016. [Online: <https://www.youtube.com/watch?v=WUU8sQmn08>; accessed 14-Feb-2017].
- [Abu17] Hadil Abukwaik. Coinsextractor demo: Conceptual interoperability analysis tool for UML, 2017. [Online: <https://youtu.be/hBbKN71TBQw>; accessed 14-Feb-2017].
- [AFCF05] Carina Alves, Xavier Franch, Juan P Carvallo, and Anthony Finkelstein. Using goals and quality models to support the matching analysis during COTS selection. In *International Conference on COTS-Based Software Systems*, pages 146–156. Springer, 2005.
- [AINT07] Marco Autili, Paola Inverardi, Alfredo Navarra, and Massimo Tivoli. Synthesis: a tool for automatically assembling correct and distributed component-based systems. In *Proceedings of the 29th international conference on Software Engineering*, pages 784–787. IEEE Computer Society, 2007.
- [AN17] Hadil Abukwaik and Thejashree Nair. Coiner demo: A ML-based tool for extracting coins from NL text of API documents, 2017. [Online: <https://youtu.be/p9EwOnpidrA>; accessed 14-Feb-2017].
- [ANR15] Hadil Abukwaik, Matthias Naab, and Dieter Rombach. A proactive support for conceptual interoperability analysis in software systems. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pages 119–122. IEEE, 2015.
- [AR16] Hadil Abukwaik and Dieter Rombach. Tool-supported extraction of conceptual interoperability constraints of software units from UML diagrams. Fraunhofer IESE, 2016.
- [Ass13] British Sociological Association. Statement of ethical practice for the British sociological association, 2013.

- [ATR14] Hadil Abukwaik, Davide Taibi, and Dieter Rombach. *Interoperability-Related Architectural Problems and Solutions in Information Systems: A Scoping Study*, pages 308–323. Springer International Publishing, 2014.
- [AZ14] Mohsen Anvaari and Olaf Zimmermann. Semi-automated design guidance enhancer (sadge): a framework for architectural guidance development. In *European Conference on Software Architecture*, pages 41–49. Springer, 2014.
- [B<sup>+</sup>10] Patti Brooks et al. Standards and interoperability in healthcare information systems: Current status, problems, and research issues. In *In 5th Midwest Association for Information Systems Conference MWAIS*, 2010.
- [BA99] Barry Boehm and Chris Abts. Cots integration: Plug and pray? *Computer*, 32(1):135–138, 1999.
- [BB91] Bruce H. Barnes and Terry B. Bollinger. Making reuse cost-effective. 8:13–24, 1991.
- [BB01] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting on association for computational linguistics*, pages 26–33. Association for Computational Linguistics, 2001.
- [BB12] James Bergstra and Yoshua Bengio. Random search for hyperparameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.
- [BGL<sup>+</sup>96] Victor R Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sørungård, and Marvin V Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, 1(2):133–164, 1996.
- [BGRB11] Yérom-David Bromberg, Paul Grace, Laurent Réveillere, and Gordon S Blair. Bridging the interoperability gap: Overcoming combined application and middleware heterogeneity. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 390–409. Springer, 2011.
- [Bhu07] Jesal Bhuta. *A framework for intelligent assessment and resolution of commercial-off-the-shelf product incompatibilities*. ProQuest, 2007.

- [BIPT09] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 141–150. ACM, 2009.
- [BJPW99] Antoine Beugnard, J-M Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.
- [BJS13] Markus Buschle, Pontus Johnson, and Khurram Shahzad. The enterprise architecture analysis tool—support for the predictive, probabilistic architecture modeling framework. In *19th Americas Conference on Information Systems, AMCIS 2013; Chicago, IL; United States; 15 August 2013 through 17 August 2013*, pages 3350–3364. Association for Information Systems, 2013.
- [Boo05] Grady Booch. *The unified modeling language user guide*. Pearson Education India, 2005.
- [BOR04] Steffen Becker, Sven Overhage, and Ralf H Reussner. Classifying software component interoperability errors to support component adaption. In *International Symposium on Component-Based Software Engineering*, pages 68–83. Springer, 2004.
- [BP] Antonia Bertolino and Andrea Polini. The audition framework for testing web services interoperability. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 134–142. IEEE.
- [BPGG11] Gordon S Blair, Massimo Paolucci, Paul Grace, and Nikolaos Georgantas. Interoperability in complex distributed systems. In *Formal Methods for Eternal Networked Software Systems*, pages 1–26. Springer, 2011.
- [BPY<sup>+</sup>03] Barry Boehm, Dan Port, Ye Yang, Jesal Bhuta, and Chris Abts. Composable process elements for developing cots-based applications. *International Symposium on Empirical Software Engineering*, 2003.
- [BR91] Victor Basili and Dieter Rombach. Support for comprehensive reuse. 6:303–316, 1991.
- [BRLM09] Yérom-David Bromberg, Laurent Réveillère, Julia L Lawall, and Gilles Muller. Automatic generation of network protocol gateways. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 21–41. Springer, 2009.

- [CD00] John Cheesman and John Daniels. *UML Components: A Simple Process for Specifying Component-based Software*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [CFGQ04] Juan Pablo Carvallo, Xavier Franch, Gemma Grau, and Carme Quer. Costume: a method for building quality models for composite cots-based software systems. In *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*, pages 214–221. IEEE, 2004.
- [CGB<sup>+</sup>02] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting software architectures: views and beyond*. Pearson Education, 2002.
- [CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.
- [Che06] David Chen. Enterprise interoperability framework. In *EMOI-INTEROP*, 2006.
- [CJ99] Thea Clark and Richard Jones. Organisational interoperability maturity model for c2. In *Proceedings of the 1999 Command and Control Research and Technology Symposium*, 1999.
- [CL05] Wesley Chu and Tsau Young Lin. *Foundations and advances in data mining*, volume 180. Springer Science & Business Media, 2005.
- [Cor10] IBM Corp. IBM SPSS Statistics for windows, version 23.0, 2010.
- [CS12] Don Choi and Andrew Sage. A framework for interoperability assessments in systems of systems and families of systems. 11:275–295, 2012.
- [Dan11] Johnnie Daniel. *Sampling essentials: Practical guidelines for making sampling choices*. Sage, 2011.
- [DBW89] Fred D Davis, Richard P Bagozzi, and Paul R Warshaw. User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8):982–1003, 1989.
- [DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.
- [DGP02] L Davis, Rose F Gamble, and Jamie Payton. The impact of component architectures on interoperability. *Journal of Systems and Software*, 61(1):31–45, 2002.



- [DGR<sup>+</sup>06] Islay Davies, Peter Green, Michael Rosemann, Marta Indulska, and Stan Gallo. How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, 58(3):358–380, 2006.
- [DH09] Uri Dekel and James D Herbsleb. Improving api documentation usability with knowledge pushing. In *Proceedings of the 31st International Conference on Software Engineering*, pages 320–330. IEEE Computer Society, 2009.
- [DKJ05] Tore Dybå, Barbara A Kitchenham, and Magne Jorgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, 2005.
- [DMM08] Marie-Catherine De Marneffe and Christopher D Manning. Stanford typed dependencies manual. Technical report, 2008.
- [DMMM<sup>+</sup>] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses.
- [DSM11] Don A Dillman, Jolene D Smyth, and Leah Melani. *Internet, mail, and mixed-mode surveys: the tailored design method*. JSTOR, 2011.
- [DW98] Desmond F D’souza and Alan Cameron Wills. *Objects, components, and frameworks with UML: the catalysis approach*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- [EMG00] Alexander Egyed, Nenad Medvidovic, and Cristina Gacek. Component-based perspective on software-mismatch detection and resolution. *IEE Proceedings-Software*, 147(6):225–236, 2000.
- [FC03] Xavier Franch and Juan Pablo Carvallo. Using quality models in software package selection. *IEEE software*, 20(1):34–41, 2003.
- [FDRR12] Adriana Maria Figueiredo, Julio C Dos Reis, and Marcos A Rodrigues. Improving access to software architecture knowledge an ontology-based search approach. *International Journal Multimedia and Image Processing (IJMIP)*, 2(1/2), 2012.
- [FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

- [FI] Architecture Seminar Fraunhofer IESE. Aces-adf. [Online: <https://www.iese.fraunhofer.de/en/competencies/architecture/architecture-tools/aces-technologies.html>; accessed 18-April-2017].
- [For02] Andrew Forward. *Software documentation: Building and maintaining artefacts of communication*. University of Ottawa (Canada), 2002.
- [Fow95] Floyd J Fowler. *Improving survey questions: Design and evaluation*, volume 38. Sage, 1995.
- [Fow05] Martin Fowler. Language workbenches: The killer-app for domain specific languages. 2005.
- [GAACB95] Cristina Gacek, Ahmed Abd-Allah, Bradford Clark, and Barry Boehm. On the definition of software system architecture. In *Proceedings of the First International Workshop on Architectures for Software Systems*, pages 85–94, 1995.
- [Gac98] Cristina Gacek. *Detecting architectural mismatches during systems composition*. PhD thesis, University of Southern California, 1998.
- [GAO95] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. pages 179–185. *Proceedings of the 17th international conference on Software engineering*, ACM, 1995.
- [GBPS14] Paul Grace, Justan Barbosa, Brian Pickering, and Mike Surridge. Taming the interoperability challenges of complex iot systems. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, pages 1–6. ACM, 2014.
- [GBS03] Paul Grace, Gordon S Blair, and Sam Samuel. Remmoc: A reflective middleware to support mobile client interoperability. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1170–1187. Springer, 2003.
- [GC92] John Gaffney and RD Cruickshank. A general economics model of software reuse. pages 327–337. *14th international conference on Software engineering*, ACM, 1992.
- [GCN09] Wided Guédria, David Chen, and Yannick Naudet. A maturity model for enterprise interoperability. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 216–225. Springer, 2009.
- [GKM<sup>+</sup>91] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh

Porteous, and Fredrick Springsteel. Ieee standard computer dictionary,ieee std 610. 1991.

[GMM07] Ricardo J Gonçalves, Jörg P Müller, and Kai Mertins. *Enterprise Interoperability II: New Challenges and Approaches*. Springer, 2007.

[Gro99] The Stanford Natural Language Processing Group. The Stanford Parser: A statistical Parser. [Online: <https://nlp.stanford.edu/software/lex-parser.shtml>; accessed 18-April-2017].

[GS12] Mark Gabel and Zhendong Su. Testing mined specifications. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 4. ACM, 2012.

[GWZH14] Chushu Gao, Jun Wei, Hua Zhong, and Tao Huang. Inferring data contract for web-based api. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 65–72. IEEE, 2014.

[Han99] Jun Han. Semantic and usage packaging for software components. In *Proceedings of the Workshop on Object-Oriented Technology*. Springer-Verlag, 1999.

[HBH<sup>+</sup>10] Sylvain Hallé, Tevfik Bultan, Graham Hughes, Muath Alkhalaf, and Roger Villemaire. Runtime verification of web service interface contracts. *Computer*, 43(3):59–66, 2010.

[HCL<sup>+</sup>03] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[HKN<sup>+</sup>05] Christine Hofmeister, Philippe Kruchten, Robert L Nord, Henk Obbink, Alexander Ran, and Pierre America. Generalizing a model of software architecture design from five industrial approaches. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 77–88. IEEE, 2005.

[HM82] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[IBB11] Valérie Issarny, Amel Bennaceur, and Yérom-David Bromberg. Middleware-layer connector synthesis: Beyond state of the art in middleware interoperability. In *Formal Methods for Eternal Networked Software Systems*, pages 217–255. Springer, 2011.

[IEE98] IEEE. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998.

- [Jad16] Sundeep Jadav. Useful and usable API documentation: empirical evaluation for improving guidelines of API documentation, 2016.
- [Jam09] Mohammad Jamshidi. *Systems of systems engineering principles and applications*. CRC press, 2009.
- [Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.
- [K<sup>+</sup>95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.
- [KCB96] Jyrki Kontio, Gianluigi Caldiera, and Victor R Basili. Defining factors, goals and criteria for reusable component evaluation. In *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, page 21. IBM Press, 1996.
- [Kin99] Joseph R Kiniry. Leading to a kind description language: Thoughts on component specification. 1999.
- [Kit04] Barbara Kitchenham. Procedures for performing systematic reviews. Technical report, 2004.
- [KM03] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.
- [Kru92] Charles Krueger. Software reuse. 24(2):131–183, 1992.
- [Kva08] Steinar Kvale. *Doing interviews*. Sage, 2008.
- [LA99] Oliver Laitenberger and Colin Atkinson. Generalizing perspective-based inspection to handle object-oriented development artifacts. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 494–503. IEEE, 1999.
- [LASEE00] Oliver Laitenberger, Colin Atkinson, Maud Schlich, and Khaled El Emam. An experimental comparison of reading techniques for defect detection in uml design documents. *Journal of Systems and Software*, 53(2):183–204, 2000.
- [LBCC08] Rikard Land, Laurens Blankers, Michel Chaudron, and Ivica Crnkovic. Cots selection best practices in literature and in industry. In *International Conference on Software Reuse*, pages 100–111. Springer, 2008.

- [LCAC12] Claudia López, Vctor Codocedo, Hernán Astudillo, and Luiz Marcio Cysneiros. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. *Science of Computer Programming*, 77(1):66–80, 2012.
- [LD00] Oliver Laitenberger and Jean-Marc DeBaud. An encompassing life-cycle centric survey of software inspection. *Journal of Systems and Software*, 50(1):5–31, 2000.
- [LEEH01] Oliver Laitenberger, Khaled El Emam, and Thomas G Harbich. An internally replicated quasi-experimental comparison of checklist and perspective based reading of code documents. *IEEE Transactions on Software Engineering*, 27(5):387–421, 2001.
- [Lik32] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.
- [LW02] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
- [Mey92] Bertrand Meyer. Applying ‘design by contract’. *Computer*, 25(10):40–51, 1992.
- [Mik99] Anna Mikhajlova. Reasoning about object clients and distributed object interaction. In *Proceedings of the Workshop on Object-Oriented Technology*. Springer-Verlag, 1999.
- [MLM<sup>+</sup>04] Edwin Morris, Linda Levine, Craig Meyers, Pat Place, and Dan Plakosh. System of systems interoperability (sosi): final report. Technical report, DTIC Document, 2004.
- [MRPX08] Kai Mertins, Rainer Ruggaber, Keith Popplewell, and Xiaofei Xu. *Enterprise Interoperability III - New Challenges and Industrial Approaches*. Springer, 2008.
- [MS99] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.
- [MT08] Juha A Mykkänen and Mika P Tuomainen. An evaluation and selection framework for interoperability standards. *Information and Software Technology*, 50(3):176–197, 2008.
- [Mur06] Kevin P Murphy. Naive bayes classifiers. *University of British Columbia*, 2006.
- [Nai17] Thejashree Nair. Industrial evaluation for a tool extracting coins from API documentation, 2017.



- [NE02] Jeremy W Nimmer and Michael D Ernst. Automatic generation of program specifications. *ACM SIGSOFT Software Engineering Notes*, 27(4):229–239, 2002.
- [Ng04] Andrew Y Ng. Feature selection,  $l_1$  vs.  $l_2$  regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.
- [oDCIWG98] Department of Defense C4ISR Interoperability Working Group. Levels of information systems interoperability (lisi). Technical report, 1998.
- [OJ12] Michael P Oakes and Meng Ji. *Quantitative methods in corpus-based Translation Studies: A practical guide to descriptive translation research*, volume 51. John Benjamins Publishing, 2012.
- [Ove04] Sven Overhage. *UnSCom: A Standardized Framework for the Specification of Software Components*, pages 169–184. Springer Berlin Heidelberg, 2004.
- [OWR<sup>+</sup>11] Steffen Olbrich, Balthasar Weitzel, Dominik Rost, Matthias Naab, and Gilb Kutepov. Decomposing interoperability: A quality attribute in the balance of system usage, operation and development. Technical report, 2011.
- [PD91] Ruben Prieto-Diaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):88–97, 1991.
- [PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08*, pages 68–77. British Computer Society, 2008.
- [PKG99] J Payton, R Keshav, and RF Gamble. System development using the integrating component architectures process. In *Proceedings of the First Workshop on Ensuring Successful COTS Development, Los Angeles, USA*, 1999.
- [Pow08] Brenda J Powers. A multi-agent architecture for nato network enabled capabilities: enabling semantic interoperability in dynamic environments (nc3a rd-2376). In *International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 93–103. Springer, 2008.
- [Pow11] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.



- [PVG<sup>+</sup>11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [PXZ<sup>+</sup>12] Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. Inferring method specifications from natural language API descriptions. In *Proceedings of the 34th International Conference on Software Engineering*, pages 815–825. IEEE Press, 2012.
- [RB10] Ben Rubinger and Tevfik Bultan. Contracting the facebook api. *arXiv preprint arXiv:1009.3715*, 2010.
- [RH09] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.
- [Rob04] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.
- [RR07] James Robertson and Suzanne Robertson. Volere requirements specification template, 2007.
- [SA11] Mahmood Sajjad and Khan Azhar. An industrial study on the importance of software component documentation: A system integrator’s perspective. 111:583–590, 2011.
- [Sam97] Johannes Sametinger. *Software engineering with reusable components*. Springer Science & Business Media, 1997.
- [Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
- [SG03] Bridget Spitznagel and David Garlan. A compositional formalization of connector wrappers. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 374–384. IEEE, 2003.
- [Sha95] Mary Shaw. Architectural issues in software reuse: It’s not just the functionality, it’s the packaging. In *ACM SIGSOFT Software Engineering Notes*, volume 20, pages 3–6. ACM, 1995.
- [SK96] Kevin J. Sullivan and John C. Knight. Experience assessing an architectural approach to large-scale systematic reuse. pages 220–229. *Proceedings of the 18th International Conference on Software Engineering*, IEEE, 1996.

- [SMKI02] Giedre Sabaliauskaite, Fumikazu Matsukawa, Shinji Kusumoto, and Katsuro Inoue. An experimental comparison of checklist-based reading and perspective-based reading for uml design document inspection. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, pages 148–157. IEEE, 2002.
- [Spa] SparxSystems. Enterprise Architect. [Online: <http://sparxsystems.de/>; accessed 14-Feb-2017].
- [SRB00] Forrest Shull, Ioana Rus, and Victor Basili. How perspective-based reading can improve requirements inspections. *Computer*, 33(7):73–79, 2000.
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- [Tea15] LimeSurvey Project Team. Limesurvey: An open source survey tool, 2015.
- [TK01] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.
- [TN99] Sotirios Terzis and Paddy Nixon. Semantic trading: Tackling interoperability problems during system integration. In *Proceedings of the Workshop on Object-Oriented Technology*. Springer-Verlag, 1999.
- [Tru04] SA True. Planning the future of the world geodetic system 1984. In *Position Location and Navigation Symposium, 2004. PLANS 2004*, pages 639–648. IEEE, 2004.
- [Tur05] Charles Turnitsa. Extending the levels of conceptual interoperability model. In *Proceedings IEEE summer computer simulation conference, IEEE CS Press*, 2005.
- [UFBJ10] Johan Ullberg, Ulrik Franke, Markus Buschle, and Pontus Johnson. A tool for interoperability analysis of enterprise architecture models using Pi-OCL. In *Enterprise Interoperability IV*, pages 81–90. Springer, 2010.
- [UY00] Sebastian Uchitel and Daniel Yankelevich. Enhancing architectural mismatch detection with assumptions. In *Engineering of Computer Based Systems, 2000.(ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshop on the*, pages 138–146. IEEE, 2000.

- [VHT00] Antonio Vallecillo, Juan Hernández, and José M Troya. New issues in object interoperability. In *European Conference on Object-Oriented Programming*, pages 256–269. Springer, 2000.
- [VSBCR02] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.
- [VTH99] Antonio Vallecillo, José M Troya, and Juan Hernández. Object interoperability. In *European Conference on Object-Oriented Programming*, pages 1–21. Springer, 1999.
- [Woo08] RF Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, 2008.
- [WWL<sup>+</sup>13] Qian Wu, Ling Wu, Guangtai Liang, Qianxiang Wang, Tao Xie, and Hong Mei. Inferring dependency constraints on parameters for web services. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1421–1432. ACM, 2013.
- [YTB99] Daniil Yakimovich, Guilherme H Travassos, and V Basili. A classification of software components incompatibilities for cots integration. In *Proceeding of the 24th Software Engineering Workshop*, 1999.
- [Zha04] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.
- [ZZXM09] Hao Zhong, Lu Zhang, Tao Xie, and Hong Mei. Inferring resource specifications from natural language api documentation. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 307–318. IEEE Computer Society, 2009.

---

# Appendix

Appendix A: Survey on Software Interoperability Analysis	202
Appendix B: Scoping Study on Interoperability-Related Architectural Problems and Solutions	211
Appendix C: Standard Documentation Templates	213
Appendix D: COIN Extraction Templates from UML Diagrams and their Identification Algorithms	215
Appendix E: COIN Cheat Sheets	229
Appendix F: Mismatches Cheat sheet	236
Appendix G: Observation Protocol Template for the Controlled Experiment Study	237
Appendix H: Multi-Run Controlled Experiment for Evaluating the systematic Analysis Approach	238
Appendix I: Controlled Experiment for Evaluating the Guidelines for Improving API Documentation	249

# Appendix A Survey on Software Interoperability Analysis

## A.1 Questionnaire

### Survey on Software Interoperability Analysis

By filling this questionnaire and watching its short video, you would have the opportunity to learn about the faced problems in the software interoperability analysis.

Please fill this survey as accurately as possible to help us establish the state of the practice regarding "interoperability analysis" within the context of integrating externally developed software units. We will use the result for identifying and prioritizing improvements for interoperability analysis of external software units.

Answering the multiple-choice questions in this survey will take approximately 10 - 15 minutes.

Your answers will be handled anonymously.

There are 26 questions in this survey

### Part 1: Interoperability Analysis in Practice

Before filling the questionnaire, please watch this 2-minute video to introduce you to the interoperability analysis concepts and terminologies.

**[ ] Do you (or your organization) perform interoperability analysis for integration projects? \***

Please choose **only one** of the following:

- ☐ Yes  
☐ No

**[ ] In your opinion, why isn't interoperability analysis performed in your integration projects? \***

**Only answer this question if the following conditions are met:**

Answer was 'No' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

Please choose **all** that apply:

- ☐ It is not that necessary  
☐ Unit and integration testing are performed instead  
☐ Tight schedule and limited resources  
☐ Priority is given to other tasks (e.g., implementation and testing)  
☐ It is hard to perform  
☐ There is no enough knowledge and experience about it  
☐ Other, please specify:

**[ ] When does interoperability analysis start in the integration projects of your organization? \***

**Only answer this question if the following conditions are met:**

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ At the beginning (before implementing the integration)  
☐ During implementation  
☐ After implementation  
☐ Never  
☐ I don't know  
☐ Other, please specify:

**[]Who is responsible for performing the interoperability analysis task in the integration projects of your organization? \***

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

Please choose **all** that apply:

- ☐ Project manager  
☐ Requirements engineers  
☐ Architects  
☐ Developers  
☐ Testers  
☐ System analysts  
☐ I don't know  
☐ Other, please specify:

**[]How many people does your organization assign for the interoperability analysis task? \***

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ One person  
☐ Small team (i.e., < 5 members)  
☐ Medium team (i.e., 5 - 10 members)  
☐ Large team (i.e., > 10 members)  
☐ I don't know  
☐ Other, please specify:

**[]What of the listed items do you (or your organization) use when performing the interoperability analysis task? \***

Only answer this question if the following conditions are met:

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

Please choose **all** that apply and provide a comment:

- ☐ Systematic process, please specify



- ☐ Analysis guidelines or checklist, please specify
- ☐ Analysis framework or model, please specify
- ☐ Supportive software tools, please specify
- ☐ Standard templates for documenting analysis results, please specify
- ☐ None of the above
- ☐ I don't know
- ☐ Other, please specify

**[ ] Which of the following constraints do you (or organization) aim at detecting during the interoperability analysis? \***

**Only answer this question if the following conditions are met:**

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

Please choose **all** that apply:

- ☐ Communication constraints (e.g., networking protocols, message formats, etc.)
- ☐ Syntax constraints (e.g., argument order, data types, etc.)
- ☐ Semantic constraints (e.g., terminologies, goals, rationale, etc.)
- ☐ Structure constraints (e.g., data structure, inheritance, service granularity, etc.)
- ☐ Behavior constraints (e.g., pre/post conditions, invariants, interaction protocols, control flow, etc.)
- ☐ Context constraints (e.g., stakeholders, environments, use cases, etc.)
- ☐ Quality constraints (e.g., data precision, service performance, etc.)
- ☐ None of the above
- ☐ I don't know
- ☐ Other, please specify:

**[ ] Do you document the results of the interoperability analysis (e.g., elicited constraints, detected mismatches, etc.)? \***

**Only answer this question if the following conditions are met:**

Answer was 'Yes' at question '1 [Q0]' (Do you (or your organization) perform interoperability analysis for integration projects? )

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ Yes
- ☐ No
- ☐ I don't know
- ☐ Other, please specify:

## Part 2: Problems of Interoperability Analysis in Practice

**[ ]In your experience, how much does interoperability analysis cost (out of the total cost of integration projects)? \***

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ < 10%
- ☐ 10% - 30%
- ☐ 31% - 50%
- ☐ 51% -70%
- ☐ > 70%
- ☐ I don't know
- ☐ Other, please specify:

**[ ]Have you (or your organization) faced problems in integration projects due to undetected conceptual mismatches (e.g., differences in non-technical aspects like software qualities, design decision, architectural constraints, usage context, etc.)? \***

Please choose **only one** of the following:

- ☐ 1: Never
- ☐ 2: Rarely
- ☐ 3: Sometimes
- ☐ 4: Usually
- ☐ 5: Always
- ☐ I don't know

**[ ]In your experience, how much does it cost (out of the total project cost) to resolve unexpected conceptual mismatches? \***

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ < 10%
- ☐ 10% - 30%
- ☐ 31% - 50%
- ☐ 51% -70%
- ☐ > 70%
- ☐ I don't know
- ☐ Other, please specify:

**[ ]In your opinion, what are the main difficulties in performing software interoperability analysis? \***

Please choose **all** that apply:

- ☐ Lack of focus on detecting the "conceptual" mismatches compared to the "technical" ones
- ☐ Lack of support for traceability between interoperability analysis activities and results (i.e., within a project and among projects)
- ☐ Lack of standard templates for consistent documentation of interoperability analysis results
- ☐ Lack of interoperability analysis guidelines and best practices for practitioners
- ☐ Undirected collection of information about the external software units (i.e., no plan or pre-defined data elements)
- ☐ Posterior collection of information about the external software unit (i.e., reactive collection based on rising problems along the project)
- ☐ Manual effort in analyzing description of external software units and in documenting the analysis results
- ☐ Other, please specify::

### Part 3: Input of Interoperability Analysis

**[ ]In your experience, what are the typically available sources of information about external software units that you can use to analyze their interoperability with your system? \***

Please choose **all** that apply:

- ☐ API documentation (e.g., API guidance, tutorials, blogs, wikis, etc.)
- ☐ Requirements specification documents
- ☐ High-level architecture documents
- ☐ Low-level design documents
- ☐ Source Code
- ☐ Execution log
- ☐ Test cases log
- ☐ I don't know
- ☐ Other, please specify::

**[ ]How would you describe the sufficiency of the shared information about external software systems to perform a successful interoperability analysis? \***

Please choose **only one** of the following:

- ☐ 1: Not sufficient at all
- ☐ 2: Not sufficient
- ☐ 3: Fair
- ☐ 4: Sufficient
- ☐ 5: Very sufficient
- ☐ I don't know

**[ ]What do you consider as important information that needs to be enriched in the current sources of information about external software units? \***

Please choose **all** that apply:

- ☐ Communication constraints (e.g., networking protocols, message formats, etc.)
- ☐ Syntax constraints (e.g., argument order, data types, etc.)
- ☐ Semantic constraints (e.g., glossaries, goals, rationale, etc.)
- ☐ High-level architecture view (e.g., architecture style, patterns, etc.)
- ☐ Low-level design decisions (e.g., inheritance, synchronicity, concurrency, etc.)
- ☐ Behavior constraints (e.g., pre/post conditions, interaction protocols, control flow, etc.)
- ☐ Context constraints (e.g., stakeholders, environments, use cases, etc.)
- ☐ Quality constraints (e.g., data precision, service performance, etc.)
- ☐ Other, please specify::

**[ ]What do you consider as main problems in the presentation of current sources of**

**information about external software units? \***

Please choose **all** that apply:

- ☐ Mixing conceptual and technical constraints without clear borders between them.
- ☐ Unstructured verbose of text
- ☐ Lack of easy-to-read process diagrams (e.g., flowcharts)
- ☐ Inconsistency in reporting constraints for the different data items and services
- ☐ Too low formality preventing potential automation of analysis
- ☐ Other, please specify::

**Part 4: Respondent's Information**

**[ ]What job position do you have currently? \***

Please write your answer here:

**[ ]How many years of professional experience do you have in the software integration area? \***

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ None
- ☐ < 2 years
- ☐ 2 – 5 years
- ☐ 5 – 8 years
- ☐ > 8 years

☐ Other, please specify:

**[ ]How many integration projects have you participated in? \***

**Only answer this question if the following conditions are met:**

Answer was NOT 'None' at question '18 [Q2]' (How many years of professional experience do you have in the software integration area? )

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ < 2 projects
- ☐ 2 – 5 projects
- ☐ > 5 projects

☐ Other, please specify:

**[ ]What role have you played in integration projects? \***

**Only answer this question if the following conditions are met:**

Answer was NOT 'None' at question '18 [Q2]' (How many years of professional experience do you have in the software integration area? )

Please choose **all** that apply:

- ☐ Project manager
- ☐ Requirements engineer
- ☐ System analyst
- ☐ Software architect
- ☐ Programmer
- ☐ Tester
- ☐ Technical writer

☐ Other, please specify::

**[ ]What are the typical types of software systems that your organization works on? \***

Please choose **all** that apply:

- ☐ Information systems
- ☐ Embedded systems



- ☐ Mobile systems
- ☐ Emergent systems
- ☐ Cyber-physical systems
- ☐ Ecosystems
- ☐ Other, please specify:

**[ ]What is the typical type of external software units that your organization integrates? \***

Please choose **all** that apply:

- ☐ None
- ☐ Open-source software (OSS)
- ☐ Commercial-of-the-Shelf (COTS)
- ☐ Web service APIs
- ☐ Platform APIs
- ☐ Other, please specify:

**[ ]What is the typical size of the software integration projects that your organization works on? \***

**Only answer this question if the following conditions are met:**

Answer was NOT 'None' at question '22 [Q8]' (What is the typical type of external software units that your organization integrates? )

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ Small (i.e., <3 months, <\$250K, 3-4 team members)
- ☐ Medium (i.e., 3-6 months, \$250-750K, 4-10 team members)
- ☐ Large (i.e., >6 months, > \$750K, >10 team members)
- ☐ Other, please specify:

**[ ]In which sector is your organization working? \***

Please choose **all** that apply:

- ☐ Software development
- ☐ Health
- ☐ Agriculture
- ☐ Finance
- ☐ Military
- ☐ Energy
- ☐ Automobile
- ☐ Other, please specify:

**[ ]What is the size of your organization? \***

If you choose 'Other, please specify:' please also specify your choice in the accompanying text field.

Please choose **only one** of the following:

- ☐ Small (i.e., < 50 members)
- ☐ Medium (i.e., 50-250 members)
- ☐ Large (i.e., <1000 members)
- ☐ Enterprise (i.e., >1000 members)
- ☐ Other, please specify:

**[ ]In which country are you currently working? \***

Please write your answer here:

## Appendix B Scoping Study on Interoperability-Related Architectural Problems and Solutions

### B.1 Selected Primary Studies

ID	Reference
S1	A. Brodt et al.: A mobile data management architecture for interoperability of resource and context data. In MDM (2011)
S2	A. Ojo et al.: Semantic interoperability architecture for electronic government. In dg.o (2009)
S3	A. Moulton et al.: Semantic Interoperability in the fixed income securities industry: A knowledge representation architecture for dynamic integration of web-based information. In HICSS (2003)
S4	G. Hatzisymeon et al.: An architecture for implementing application interoperation with heterogeneous systems. In DAIS (2005)
S5	L. Xianming et al.: Research on the Portlet Semantic Interoperability Architecture. In WCSE (2009)
S6	D. de Carvalho et al.: Functional and device interoperability in an architectural model of geographic information system. In SIGDOC (2007)
S7	J. Kim et al.: An enterprise architecture framework based on a common information technology domain (EAFIT) for improving interoperability among heterogeneous information systems. In SERA (2005)
S8	S. Zhu et al.: Army enterprise architecture technical reference model for system interoperability. In MILCOM (2009)
S9	F. Rabhi: Towards an open architecture for the integration and interoperability of distributed systems. In Ent-Net at SUPERCOMM (2001)
S10	B. Powers: A multi-agent architecture for NATO network enabled capabilities: enabling semantic interoperability in dynamic environments (NC3A RD-2376). In SOCASE (2008)
S11	E. Leclercq et al.: ISIS: a semantic mediation model and an agent based architecture for GIS interoperability. In IDEAS (1999)
S12	M. Paul: Enterprise geographic information system (E-GIS): A service-based architecture for geo-spatial data interoperability. In IGARSS (2006)
S13	G. Lepouras et al.: An active ontology-based blackboard architecture for web service interoperability. In ICSSSM (2005)
S14	C. Schroth et al.: UN/CEFACT Service-Oriented Architecture-Enabling Both Semantic And Application Interoperability. In KiVS (2007)
S15	P. Arapi et al.: ASIDE: An Architecture for Supporting Interoperability between Digital Libraries and ELearning Applications. In ICALT (2006)
S16	A. Bennaceur et al.: Towards an architecture for runtime interoperability. In ISoLA (2010)
S17	R. Maciel et al.: WGWSOA: A service-oriented middleware architecture to support groupware interoperability. In CSCWD (2007)
S18	Y. Demchenko et al.: Intercloud Architecture for interoperability and integration. In CloudCom (2012)
S19	D. Arize et al.: ThesIS: A semantic interoperability service for a middleware service oriented architecture. In CSCWD (2013)
S20	R. Crichton et al.: An Architecture and Reference Implementation of an Open Health Information Mediator: Enabling Interoperability in the Rwandan Health Information Exchange. In FHIES (2013)
S21	G. Komatsoulis et al.: caCORE version 3: Implementation of a model driven, service-oriented architecture for semantic interoperability. In J-BHI (2008)
S22	A. Mohtasebi et al.: Analysis of Applying Enterprise Service Bus Architecture as a Cloud Interoperability and Resource Sharing Platform. In KMO (2013)

## B.2 Data Extraction Form

	Field	Description	RQ
F1	Title	Title of the paper	Documentation
F2	Author	Writer(s) of the paper	
F3	Year	Year of publishing the paper	
F4	Publication	Name of Journal / Proceeding	
F5	Keywords	Keywords of the paper	RQ1
F6	Objectives	Stated goals of the study by the authors- free text	RQ1
F7	IS type	Kind of IS application which the study focuses on	RQ2
F8	Interoperability problem(s)	Object of the study which the study tries to solve (i.e., problem of interest) - free text	RQ2
F9	Interoperability level	Level of LCIM that the study handles	RQ1
F10	Architectural solution(s)	Subject of the study that is proposed to solve the object (i.e., solution of problem) - free text	RQ3
F11	Solution elements	Concrete elements of the proposed subject (i.e., components of architectural solutions) - free text	RQ3
F12	Technology used	Technologies supporting implementation of proposed subjects (e.g., XML, Web Services ... etc.)	RQ3
F13	Solution evidence	Evidence provided on the quality of proposed subjects (e.g., discussion, controlled experiment, case study, etc.)	RQ4
F14	Interoperability Metric	Quantitative measures used in the study evaluation to describe the interoperability property achieved	RQ4.1
F15	Comments	Additional notes provided in the study (i.e., claimed benefits, tradeoffs, limitations, or challenges) - free text	RQ2.1

## Appendix C     Standard Documentation Templates

### C.1     COIN Portfolio Template

Interoperability Analyst/Portfolio Creator Name(s):

Creation Date:

Interoperable Software System Name:

Interoperable Software System Version:

Interoperable element (system, data, function/service/fe ature)	COIN Sheet							Comments (Concerns, information reference document or line number, etc.)
	ID (unique value)	Name (abstract title)	Category (constraint class)	Type (constraint attribute)	Value		Weight (importance rank)	Consequence (if not satisfied)
					Qualitative	quantitative		

## C.2 Mismatches List Template

Interoperability Analyst Name(s):

Analysis Date:

Integration Project Name:

Integration Project version:

First Interoperable System (S <sub>1</sub> )		Second Interoperable System (S <sub>2</sub> )	
Name:		Name:	
Version:		Version:	
COIN Portfolio Reference:		COIN Portfolio Reference:	

Interoperable element (system, data, function/service/feature)	Conceptual Mismatch				Reference COINs	
	ID (unique value)	Name (abstract title)	Type (category)	Description	COIN ID from S <sub>1</sub>	COIN ID from S <sub>2</sub>

## Appendix D COIN Extraction Templates from UML Diagrams and their Identification Algorithms

Template ID	COIN source diagram	COIN category	COIN type
t1	Component	Structure	Layering
t2	Component	Structure	Component distribution
t3	Component	Structure	DB distribution
t4	Deployment	Structure	Component distribution
t5	Deployment	Structure	DB distribution
t6	Class	Structure	Structural multiplicity
t7	Class	Structure	Inherited constraints
t8	Use case	Context	Allowed users
t9	Use case	Context	Usage multiplicity
t10	Use case	Structure	Inherited constraints
t11	Sequence	Dynamic	Interaction synchronicity
t12	All	NA	Natural language constraints

### Template (t1): Layering in component diagrams

t1 ∈ Structure COINs Category

$\forall \text{ system } (s), \exists e \in \{ \text{interoperable elements } (E) \cap$

$\text{component diagram components (COMDC)}\},$

$t1(s) = \text{True} \leftrightarrow (\text{hasBoundary}(\text{COMD}) = \text{true}) \wedge$

$((\text{length}(\text{horizontal lanes}(\text{COMD})) > 1 \mid$

$(\text{length}(\text{vertical lanes}(\text{COMD})) > 1))$



**Algorithm (t1 identification)**

**Input:** Component diagram (COMD), interoperable system (s), interoperable elements (E)

**Process:**

```

1  coinCandidates  $\leftarrow \emptyset$ 
2  coin  $\leftarrow \text{nil}$ 
3  isInteroperable  $\leftarrow \text{false}$ 
4  While (isInteroperable  $\neq \text{true}$ )
5    For each  $e \in \{\text{COMD.elements}\}$ 
6      If ( $e \in s.\text{elements} \wedge (e \in E)$ )
7        isInteroperable  $\leftarrow \text{true}$ 
8        Break For each
9      End If
10   End For each
11  End While
12  If (hasBoundary(COMD) = true)  $\wedge$ 
13    ((length(horizontal lanes(COMD)) > 1 |
14     (length(vertical lanes(COMD)) > 1))
15    coin  $\leftarrow (e, \text{"Structure"}, \text{"System Layering"}, \text{"Layered"})$ 
16    coinCandidates  $\leftarrow \{coinCandidates \cup coin\}$ 
17  End If

```

**Output:** *coinCandidates*

**Template (t2): Component distribution in component diagrams**

$t2 \in \text{Structure COINs Category}$

$\forall \text{ element } (e) \in \{ \text{interoperable elements } (E) \cap$

$\text{component diagram components (COMDC)}\},$

$t2(e) = \text{True} \leftrightarrow e.\text{stereotype} \neq \text{"database"} \wedge \text{length}(e.\text{locations}) > 1$

**Algorithm (t2 identification)**

**Input:** Component diagram (COMD), interoperable elements (E)

**Process:**

1  $\text{coinCandidates} \leftarrow \emptyset$

2  $\text{coin} \leftarrow \text{nil}$

3 For each  $e \in \{\text{COMD.elements}\}$

4   If  $(e \in E) \wedge (e.\text{stereotype} \neq \text{"database"}) \wedge (\text{length}(e.\text{locations}) > 1)$

5        $\text{coin} \leftarrow (e, \text{"Structure"}, \text{"Component distribution"}, \text{"distributed"})$

6        $\text{coinCandidates} \leftarrow \{\text{coinCandidates} \cup \text{coin}\}$

7   End If

8 End For

**Output:** coinCandidates

**Template (t3): DB distribution in component diagrams**

$t3 \in \text{Structure COINs Category}$

$\forall \text{ element } (e) \in \{ \text{interoperable elements } (E) \cap$

$\text{component diagram elements (COMDE)}\},$

$t3(e) = \text{True} \leftrightarrow e.\text{stereotype} = \text{"database"} \wedge \text{length}(e.\text{locations}) > 1$

**Algorithm (t3 identification)**

**Input:** Component diagram (COMD), interoperable elements (E)

**Process:**

1  $\text{coinCandidates} \leftarrow \emptyset$

2  $\text{coin} \leftarrow \text{nil}$

3 For each  $e \in \{\text{COMD.elements}\}$

4   If  $(e \in E) \wedge (e.\text{stereotype} = \text{"database"}) \wedge (\text{length}(e.\text{locations}) > 1)$

5      $\text{coin} \leftarrow (e, \text{"Structure"}, \text{"DB distribution"}, \text{"distributed"})$

6      $\text{coinCandidates} \leftarrow \{\text{coinCandidates} \cup \text{coin}\}$

7   End If

8 End For

**Output:** coinCandidates

**Template (t4): Component distribution in deployment diagrams**

t4 ∈ Structure COINs Category

∀ element (*e*) ∈ { interoperable elements (E) ∩

deployment diagram components (DDC)},

t4 (*e*) = True ↔ *e*.stereotype ≠ “database” ∧ length (*e*.locations) > 1

**Algorithm (t4 identification)**

**Input:** Deployment diagram (DD), interoperable elements (E)

**Process:**

1 *coinCandidates* ← ∅

2 *coin* ← nil

3 For each *e* ∈ {DD.elements}

4   If (*e* ∈ E) ∧ (*e*.stereotype ≠ “database”) ∧ (length (*e*.locations) > 1)

5       *coin* ← (*e*, "Structure", "Component distribution", “distributed”)

6       *coinCandidates* ← {*coinCandidates* ∪ *coin*}

7   End If

8 End For

**Output:** *coinCandidates*

**Template (t5): DB distribution in deployment diagrams**

t5 ∈ Structure COINs Category

∀ element (*e*) ∈ { interoperable elements (E) ∩

deployment diagram elements (DDE)},

t5 (*e*) = True ↔ *e*.stereotype = “database” ∧ length (*e*.locations) > 1

**Algorithm (t5 identification)**

**Input:** Deployment diagram (DD), interoperable elements (E)

**Process:**

1 *coinCandidates* ← ∅

2 *coin* ← nil

3 For each *e* ∈ {DD.elements}

4   If (*e* ∈ E) ∧ (*e*.stereotype = “database”) ∧ (length (*e*.locations) > 1)

5       *coin* ← (*e*, "Structure", "DB distribution", “distributed”)

6       *coinCandidates* ← {*coinCandidates* ∪ *coin*}

7   End If

8 End For

**Output:** *coinCandidates*

**Template (t6): Structural multiplicity in class diagrams**

t6 ∈ Structure COINs Category

$\forall \text{ element } (e) \in \{ \text{interoperable elements } (E) \cap$   
 $\text{class diagram elements (CDE)} \},$

t6 (e) = True  $\leftrightarrow$  e.association.multiplicity  $\neq \emptyset$

**Algorithm (t6 identification)**

**Input:** Class diagram (CD), interoperable elements (E)

**Process:**

```

1  coinCandidates ← ∅
2  coin ← nil
3  For each e ∈ {CD.elements}
4    If (e ∈ E) ∧ (e.association.multiplicity ≠ ∅)
5      coin ← (e, "Structure", "Structural multiplicity",
6          value(e.association.multiplicity))
7      coinCandidates ← {coinCandidates ∪ coin}
8    End If
9  End For

```

**Output:** coinCandidates



### Template (t7): Inherited constraints of class diagrams

t7 ∈ Structure COINs Category

$$\forall \text{ element } (e) \in \{ \text{interoperable elements (E)} \cap \text{class diagram elements (CDE)} \},$$
$$t7(e) = \text{True} \leftrightarrow e.\text{Parent} \neq \emptyset \wedge e.\text{Parent}.\text{Constraints} \neq \emptyset$$

### Algorithm (t7 identification)

**Input:** Class diagram (CD), interoperable elements (E)

### Process:

```
1 coinCandidates  $\leftarrow \emptyset$ 
```

```
2 coin ← nil
```

3 For each  $e \in \{\text{CD.elements}\}$

4 If  $(e \in E) \wedge (e.\text{Parent}, e.\text{Parent}.\text{getConstraints} \neq \emptyset)$

5 For each  $c$  in  $e.Parent.getConstraints$

```
6  coin ← (e, "Structure", "inherited constraint", value(c))
```

```
7  $coinCandidates \leftarrow \{coinCandidates \cup coin\}$ 
```

```
8      End For
```

9 End If

10 End For

**Output:** coinCandidates

**Template (t8): Allowed users in use case diagrams**

t8 ∈ Context COINs Category

∀ element ( $e$ ) ∈ { interoperable elements (E) ∩

Use case diagram elements (EDE)},

t8 ( $e$ ) = True ↔ length( $e$ .actors) > 0

**Algorithm (t8 identification)**

**Input:** Use case diagram (UD), interoperable elements (E)

**Process:**

1  $coinCandidates \leftarrow \emptyset$

2  $coin \leftarrow nil$

3 For each  $e \in \{UD.elements\}$

4   If ( $e \in E \wedge (\text{length}(e.actors) > 0)$ )

5      $coin \leftarrow (e, "Context", "Allowed users", \text{value}(e.actors))$

6      $coinCandidates \leftarrow \{coinCandidates \cup coin\}$

7   End If

8 End For

**Output:** coinCandidates

**Template (t9): Usage multiplicity in use case diagrams**

t9 ∈ Context COINs Category

∀ element ( $e$ ) ∈ { interoperable elements (E) ∩  
Use case diagram elements (UDE)},

t9 ( $e$ ) = True ↔  $e.association(actor).multiplicity \neq \emptyset$

**Algorithm (t9 identification)**

**Input:** Use case diagram (UD), interoperable elements (E)

**Process:**

```

1  coinCandidates ← ∅
2  coin ← nil
3  For each  $e \in \{UD.elements\}$ 
4    If ( $e \in E$ ) ∧ ( $e.association.multiplicity \neq \emptyset$ )
5      coin ← ( $e$ , "Context", "Usage multiplicity",
6            value( $e.association(actor).multiplicity$ ))
7      coinCandidates ← {coinCandidates ∪ coin}
8    End If
9  End For

```

**Output:** *coinCandidates*

**Template (t10): Inherited constraints in use case diagrams**

t10 ∈ Structure COINs Category

$\forall \text{ element } (e) \in \{ \text{interoperable elements } (E) \cap$

Use case diagram elements (UDE)),

$t10(e) = \text{True} \leftrightarrow e.\text{Parent} \neq \emptyset \wedge e.\text{Parent}.\text{Constraints} \neq \emptyset$

**Algorithm (t10 identification)**

**Input:** Use case diagram (UD), interoperable elements (E)

**Process:**

1  $\text{coinCandidates} \leftarrow \emptyset$

2  $\text{coin} \leftarrow \text{nil}$

3 For each  $e \in \{\text{UD.elements}\}$

4   If  $(e \in E) \wedge (e.\text{Parent}, e.\text{Parent}.\text{getConstraints} \neq \emptyset)$

5       For each  $c$  in  $e.\text{Parent}.\text{getConstraints}$

6            $\text{coin} \leftarrow (e, \text{"Structure"}, \text{"inherited constraint"}, \text{value}(c))$

7            $\text{coinCandidates} \leftarrow \{\text{coinCandidates} \cup \text{coin}\}$

8       End For

9   End If

10 End For

**Output:** coinCandidates

**Template (t11): Interaction synchronicity in sequence diagrams**

t11 ∈ Dynamic COINs Category

$\forall$  system ( $s$ ),  $\exists e \in \{ \text{interoperable elements (E)} \cap$

Sequence diagram elements (SDE)},

t11 ( $s$ ) = True  $\leftrightarrow \exists$  SDE.message.type  $\neq \emptyset$

**Algorithm (t11 identification)**

**Input:** Sequence diagram (SD), interoperable elements (E)

**Process:**

1 *coinCandidates*  $\leftarrow \emptyset$

2 *coin*  $\leftarrow \text{nil}$

3 *messagesSynchronicity*  $\leftarrow \emptyset$

4 For each *message*  $\in \{\text{SD.elements}\}$

5   If (*m.type*  $\neq \emptyset$ )

6     *messagesSynchronicity*  $\leftarrow \{\text{messagesSynchronicity} \cup m.\text{type}\}$

7   End If

8   If (*messagesSynchronicity*  $\neq \emptyset$ )

9     *coin*  $\leftarrow (e, \text{"Dynamic"}, \text{"Interaction synchronicity"},$

10         *messagesSynchronicity*)

11     *coinCandidates*  $\leftarrow \{\text{coinCandidates} \cup \text{coin}\}$

12   End If

**Output:** *coinCandidates*

**Template (t12): Natural language constraint in all diagrams**

t12  $\notin$  any COINs Category

$\forall e \in \{ \text{interoperable elements (E)} \cap \{$

    Component diagram elements (COMDE)  $\cup$

    Deployment diagram elements (DDE)  $\cup$

    Class diagram elements (CDE)  $\cup$

    Use case diagram elements (UDE)  $\cup$

    Sequence diagram elements (SDE)  $\} \}$ ,

t12 ( $e$ ) = True  $\leftrightarrow$  ( $e.\text{Note} \neq \emptyset$ )  $\mid$  ( $e.\text{Constraint} \neq \emptyset$ )  $\mid$  ( $e.\text{Comment} \neq \emptyset$ )

**Algorithm (t12 identification)**

**Input:** Component diagram (COMD), Deployment diagram (DD),  
 Class diagram (CD), Use case diagram (UD), Sequence  
 diagram (SD), interoperable elements (E)

**Process:**

1 *coinCandidates*  $\leftarrow \emptyset$

2 *coin*  $\leftarrow \text{nil}$

3 For each  $e \in \{\text{COMD.elements} \cup \text{DDE.elements} \cup \text{CDE.elements} \cup$   
 4                      UDE.elements  $\cup$  SDE.elements}

5   If ( $e \in E$ )  $\wedge$  ( $e.\text{Note} \neq \emptyset$ )

6     *coin*  $\leftarrow (e, \text{NA}, \text{"Natural language constraint"}, \text{value}(e.\text{Note}))$

7     *coinCandidates*  $\leftarrow \{\text{coinCandidates} \cup \text{coin}\}$

8   End If

9   If ( $e \in E$ )  $\wedge$  ( $e.\text{Constraint} \neq \emptyset$ )





```
10  coin  $\leftarrow$  (e, NA, "Natural language constraint", value(e.Constraint))
11  coinCandidates  $\leftarrow$  {coinCandidates  $\cup$  coin}
12  End If
13  If (e  $\in$  E)  $\wedge$  (e.Comment  $\neq \emptyset$ )
14    coin  $\leftarrow$  (e, NA, "Natural language constraint", value(e.Comment))
15    coinCandidates  $\leftarrow$  {coinCandidates  $\cup$  coin}
16  End If
17 End For
```

**Output:** *coinCandidates*

## Appendix E COIN Cheat Sheets

### E.1 COIN Cheat Sheet for UML Diagrams

Perspective	COIN Category	COIN type	Description	Where to find (probably)?
<b>System</b>	<b>Structure</b>	Layering	Layered, not layered	- In component and deployment diagrams when components appear organized in horizontal or vertical lanes
		Distribution	Distributed, centralized	- In component and deployment diagrams when components appear on multiple nodes
<b>Data/ service</b>	<b>Semantic</b>	Semantic constraints of data	Unit of measurement, scale of measurement, sorting style (e.g., ascending, descending, categorized, etc.), etc.	- In NL “note” or “constraint” elements in any diagram
		Input and output of service	Non-technical description of expected input and output (no format, no data type)	- In input and output elements of activity diagram, sequence diagram, or interaction overview diagram
	<b>Structure</b>	Data structural constraints	Data invariants, inherited constraints, capacity limits/ranges, or multiplicity constraints	- In a relation’s multiplicity information in class diagrams, object diagrams. Or in NL “note” or “constraint” elements in any diagram
		Service distribution and data distribution	Distributed, centralized	- In component and deployment diagrams when a data or service element appears in multiple sites
		Dependency	Underlying dependencies (i.e., additional required functions or data)	- In a relation’s type information (aggregation, composition, etc.) in class diagrams, object diagrams, or in NL “note” or “constraint” elements in any diagram
		Redundancy	Single or multiple units or interfaces dedicated to a specific element	- In component and deployment diagrams when a data or service element appears duplicated
		Service and data encapsulation	Encapsulated with interfaces or not encapsulated with direct access	- Interfaces in component diagrams, deployment diagrams, or class diagrams
		Data concurrency	Single-user or shared access	- In state machine diagrams and activity diagrams
	<b>Dynamic</b>	Service contracts	Invariants, pre-, or post-conditions	- In NL “note” or “constraint” elements in any behavioral diagram (e.g., activity diagram, sequence diagram, communication diagram, etc.)

		Interaction time constraints	Session timeline, acknowledgment timeline, response timeline, etc.	- In timing diagrams, NL “note”, or “constraint” elements in any behavioral diagram (e.g., activity diagram, sequence diagram, communication diagram, etc.)
		Interaction property	State(ful/less), (a)synchronous, etc.	- In the type of messages in sequence diagrams (i.e., a <i>synchronous</i> message is denoted by a <i>solid</i> arrowhead  ; an <i>asynchronous</i> message by a <i>line</i> arrowhead  )
		Interaction protocol	Description of process flow/activities and usage steps	- In activity diagrams, sequence diagrams, or interaction overview diagrams
		Communication style	Messaging, procedure call, blackboard, streaming, etc.	- In message types and properties in activity diagrams, sequence diagrams, or interaction overview diagrams
	Context	Usage multiplicity	Access rate or session multiplicity	- In a relation’s type information (aggregation, composition, etc.) in class diagrams, object diagrams, or in NL “note” or “constraint” elements in any diagram
		Usage environment	Device specification, wired/wireless, etc.	- In component and deployment diagrams when a data or service element appears duplicated
		Intended end user	Human/machine, gender, location, age, etc.	- Interfaces in component diagrams, deployment diagrams, or class diagrams
	Quality	Service quality	Availability, response time, throughput, #parallel connections, etc.	- In state machine diagrams and activity diagrams
		Data quality	Security, trust, accuracy, etc.	- In NL “note” or “constraint” elements in any behavioral diagram (e.g., activity diagram, sequence diagram, communication diagram, etc.)

## E.2 COIN Cheat Sheet for SRS in IEEE Template 830

Perspective	COIN Category	COIN type	Description	Where to find (probably)?
<b>System</b>	<b>Syntax</b>	Terminology definition	Glossary items or inline definition for special terms or abbreviations	- In "Definitions or Glossary" subsection under "Introduction" section
	<b>Semantic</b>	Goal of system	Why to interoperate with this system	- For the system goal, look in "Purpose" subsection under "Introduction" section
	<b>Structure</b>	Layering	Layered, not layered	- In "Design and Implementation Constraints" subsection under "Introduction" section
		Distribution	Distributed, centralized	- In "Design and Implementation Constraints" subsection under "Introduction" section
	<b>Dynamic</b>	Dynamicity	Static, periodic change, irregular change, continuous change, etc.	- In "Design and Implementation Constraints" subsection under "Introduction" section
		Communication style	Messaging, procedure call, blackboard, streaming, etc.	- In "Design and Implementation Constraints" subsection under "Introduction" section
	<b>Context</b>	Application domain	The domain area where the system will be used (e.g., health, banking, education, etc.)	- In "Project Scope" subsection under "Introduction" section
		Usage quota	Access rate or quota (e.g., limited, unlimited, etc.)	- In "Design and Implementation Constraints" subsection under "Introduction" section
		Usage mode	Online/offline, wired/wireless, dynamic/static, etc.	- In "Operating Environment" subsection under "Overall Description" section
	<b>Quality</b>	Quality attributes	Reliability, response time, ease of use, security, etc.	- In "Non-Functional Requirements" subsection under "Specific Requirements" section
<b>Data/service</b>	<b>Semantic</b>	Semantic constraints of data	Unit of measurement, scale of measurement, sorting style (e.g., ascending, descending, categorized, etc.), etc.	- In "Constraints, assumptions and dependencies" subsection under "Overall/General description" section. Or, within "input/output" description of use cases in the "Specific Requirement" section
		Goal of service	Why to interoperate with this service	- For a function goal, look in "Goal" description of its use cases in the "Specific Requirement" section
		Input and output of service	Non-technical description of expected input and output (no format no data type)	- In "input/output" description of a function use case in the "Specific Requirement" section
	<b>Structure</b>	Data structural constraints	Data invariants, inherited constraints, capacity limits/ranges, or multiplicity constraints	- In "Constraints, assumptions and dependencies" subsection under "Overall/General description" section

		Service distribution and data distribution	Distributed, centralized	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section especially under design constraints
		Dependency	Underlying dependencies (i.e., additional required functions or data)	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section especially under design constraints
		Redundancy	Single or multiple units or interfaces dedicated to a specific element	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section especially under design constraints
		Service and data encapsulation	Encapsulated with interfaces or not encapsulated with direct access	- In “Software Interfaces” under “External Interface Requirements” subsection
		Data concurrency	Single-user or shared access	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section, especially under design constraints
	<b>Dynamic</b>	Service contracts	Invariants, pre-, or post-conditions	- In “pre-condition and post-condition” description of a function use case in the “Specific Requirement” section
		Interaction time constraints	Session timeline, acknowledgment timeline, response timeline, etc.	- In “pre-condition and post-condition” description of a function use case in the “Specific Requirement” section
		Interaction property	State(ful/less), (a)synchronous, etc.	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section - In “Communication Interfaces” under “External Interface Requirements” subsection
		Interaction protocol	Description of process flow/activities and usage steps	- In the “Specific Requirement” section look for a function’s “Flow of events” in its use case or for process details in the “Functional requirements”
		Communication style	Messaging, procedure call, blackboard, streaming, etc.	- In “Design and Implementation Constraints” subsection under “Introduction” section
	<b>Context</b>	Usage multiplicity	Access rate or session multiplicity	- In “Constraints, assumptions and dependencies” subsection under “Overall/General description” section
		Usage environment	Device specification, wired/wireless, etc.	- In “Project Scope” subsection under “Introduction” section. Or, in “Operating Environment” subsection under “Overall Description” section
		Intended end user	Human/machine, gender, location, age, etc.	- In “User Classes and Characteristics” subsection under “Overall Description” section

	<b>Quality</b>	Service quality	Availability, response time, throughput, #parallel connections, etc.	- In “Non-functional Requirements” subsection under “Specific Requirements” section
		Data quality	Security, trust, accuracy, etc.	- In “Non-functional Requirements” subsection under “Specific Requirements” section



## E.3 COIN Cheat Sheet for API Documents

Perspective	COIN Category	COIN type	Description	Where to find (probably)?
<b>System</b>	<b>Syntax</b>	Terminology definition	Glossary items or inline definition for special terms or abbreviations	In “Overview or Introduction” section of the system, or within the introduction of services
	<b>Semantic</b>	Goal of system	Why to interoperate with this system	
	<b>Structure</b>	Layering	Layered, not layered	
		Distribution	Distributed, centralized	
	<b>Dynamic</b>	Dynamicity	Static, periodic change, irregular change, continuous change, etc.	
		Communication Style	Messaging, procedure call, blackboard, streaming, etc.	
	<b>Context</b>	Application domain	The domain area where the system will be used (e.g., health, banking, education, etc.)	
		Usage quota	Access rate or quota (e.g., limited, unlimited, etc.)	
		Usage mode	Online/offline, wired/wireless, dynamic/static, etc.	
	<b>Quality</b>	Quality attributes	Reliability, response time, ease-to-use, security, etc.	
<b>Data/service</b>	<b>Semantic</b>	Semantic constraints of data	Unit of measurement, scale of measurement, sorting style (e.g., ascending, descending, categorized, etc.), etc.	- In the description of data input or returned output of a service
		Goal of service	Why to interoperate with this service	- In the service-dedicated description
		Input and output of service	Non-technical description of expected input and output (no format, no data type)	- In the service-dedicated description and process models associated with it
	<b>Structure</b>	Data structural constraints	Data invariants, inherited constraints, capacity limits/ranges, or multiplicity constraints	- For input/output constraints, look in the service-dedicated description
		Service distribution and data distribution	Distributed, centralized	- In the service-dedicated description and any models associated with it
		Dependency	Underlying dependencies (i.e., additional required functions or data)	- In the service-dedicated description and any models associated with it

		Redundancy	Single or multiple units or interfaces dedicated to a specific element	- In the service-dedicated description and any models associated with it
		Service and data encapsulation	Encapsulated with interfaces or not encapsulated with direct access	- In the service-dedicated description and any models associated with it
		Data concurrency	Single-user or shared access	- For data of a specific service, look in its dedicated description
	<b>Dynamic</b>	Service contracts	Invariants, pre-, or post-conditions	- For a service contract, look in its dedicated description and conditional elements on process models associated with it
		Interaction time constraints	Session timeline, acknowledgment timeline, response timeline, etc.	- In the service-dedicated description
		Interaction property	State(ful/less), (a)synchronous, etc.	- In the service-dedicated description
		Interaction protocol	Description of process flow/activities and usage steps	- In the service-dedicated description and process models associated with it
		Communication style	Messaging, procedure call, blackboard, streaming, etc.	- In the service-dedicated description and process models associated with it
		Usage multiplicity	Access rate or session multiplicity	- In the service-dedicated description
	<b>Context</b>	Usage environment	Device specification, wired/wireless, etc.	- In the service-dedicated description
		Intended end user	Human/machine, gender, location, age, etc.	- In the service-dedicated description
		Service quality	Availability, response time, throughput, #parallel connections, etc.	- In the service-dedicated description
	<b>Quality</b>	Data quality	Security, trust, accuracy, etc.	- In the service-dedicated description

## Appendix F Mismatches Cheat Sheet

Mismatch Type		How to find?	Causing COINs	Examples
<b>Direct</b>		COINs of <i>similar category and type</i> with explicit <i>contradicting values</i> for corresponding interoperable element.	<i>All</i> types of COINs can be the cause of direct mismatches.	<ul style="list-style-type: none"> <li>S<sub>1</sub> has a “size of lists” constraint that the returned object has a maximum capacity of 100 items.</li> <li>S<sub>2</sub> has a “size of lists” constraint that the maximum size of the lists used in the system is 50 items.</li> <li>This leads to a “direct mismatch” on the structure level.</li> </ul>
<b>Indirect</b>		COINs with values that may <i>influence</i> the requirements of <i>other</i> COINs in the corresponding interoperable element.	Mostly, <i>Structure</i> COINs and <i>Dynamic</i> COINs are the cause of indirect mismatches and they mainly affect <i>Quality</i> COINs of the other software unit.	<ul style="list-style-type: none"> <li>S<sub>1</sub> has a “synchronicity” constraint that the independent tasks are processed synchronously.</li> <li>S<sub>2</sub> has a “quality” constraint that the system requires high response time.</li> <li>This leads to an “indirect mismatch” on the quality level.</li> </ul>
<b>Potential</b>	<b>Adherence</b>	A COIN that has <i>no corresponding or conflicting</i> COINs in the other system/service, but it <i>d demands being satisfied</i> .	<i>All</i> types of COINs can be behind the adherence mismatches.	<ul style="list-style-type: none"> <li>S<sub>1</sub> has a “redundancy” constraint that there should be a backup interface to ensure availability of service.</li> <li>S<sub>2</sub> has no constraints regarding interface redundancy.</li> <li>This may lead to “potential adherence mismatches” if developers of S<sub>2</sub> do not build a backup interface. Hence, it has to be reported to ensure they will satisfy this constraint when reusing S<sub>1</sub>.</li> </ul>
	<b>Consensus</b>	A COIN that has <i>no corresponding or conflicting</i> COINs in the other system, but <i>d demands a common understanding or agreement</i> .	<i>Semantic</i> and <i>Context</i> COINs are the main causes of the consensus mismatches.	<ul style="list-style-type: none"> <li>S<sub>1</sub> has “terminology” constraints that it uses domain-specific terms (e.g., for a farm application, the “Growing season” is the period of time from April until October or November for European farms).</li> <li>S<sub>2</sub> has no corresponding constraint to define the aforementioned term.</li> <li>This may lead to a “potential consensus mismatch” if users of S<sub>2</sub> misunderstand the definition (e.g., farmers in Finland will understand the growing season as the period of time from June to September only).</li> </ul>

## Appendix G      Observation Protocol Template for the Controlled Experiment Study

Possible disturbing factors' codes	
<b>Emotions</b> E1. Un-concentrated / Unfocused E2. Unconfident/doubtful E3. Bothered/ frustrated E4. Tired	<b>Events</b> E5. Participant(s) is (are) inactive E6. Participant arrives late E7. Participant leave early E8. Cell-phone call E9. Noise E10. Interruption

Controlled experiment study information				
Study date				
Study time				
Study location				
Group (A,B)				
Controlled experiment observations				
Participant ID	Time	Emotion code	Event code	Comments

## Appendix H Multi-Run Controlled Experiment for Evaluating the Systematic Analysis Approach

### H.1 Experimental Material

For all example inputs (e1, e2, e3) from all sessions and for the introduction tutorial for each group, please go to this webpage: <http://abukwaik.com/site/multi-run-experiment16>

(use password: abukwaik\_experiment).

#### H.1.1 Non-Disclosure Agreement and Informed Consent



Session: 1  
Group: A  
Case-ID: «CaseID»

**Non-Disclosure Agreement & Informed Consent**

Last name (Student): \_\_\_\_\_  
First name: \_\_\_\_\_

The student is taking part in the GSE course during the Winter Semester 2015/2016.

To ensure the success of the case study the student will keep confidential all information gained during the study and shall disclose any information about given materials especially to other students joining the course.

The student also accepts that the case study is recorded for the purpose of anonymous analysis. All gathered data will be kept in confidence. The performance or the results of the case study will not influence the grading of the student in the GSE course.

Kaiserslautern, \_\_\_\_\_ (Date)  
\_\_\_\_\_ (Student signature)

1

## H.1.2 Experimental Procedure Description



Session: 1  
Group: A  
Case-ID: «CaseID»

### Reuse Analysis Case Study – Procedures

First of all, **thank you** for participating in the study!

Please read this document carefully and contact your experimenter in case you have fundamental problems in understanding the given information or tasks.

Please remember, don't talk about the system details to other participants of the course before all sessions of the study are conducted!

Please try to perform the tasks *as fast as possible*, but aim at achieving results of *high quality*. The maximum allocated time for executing the case study is 90 minutes.

#### Preparation

- Fill the briefing questionnaire (page 3 and 4)
  - Typically, this step takes about 5 minutes
- Read the Introduction (page 5)
  - Typically, this step takes about 10 minutes

#### Execution

- Read the task description and conduct it (page 6)
  - Typically, the execution takes about 55 minutes (activity1: 20 min, activity2: 20 min, and activity3: 15 min)

#### Finalization

- Fill in the Debriefing Questionnaire (page 35 - 36)
  - Typically, the finalization takes about 10 minutes



## H.1.3 Briefing Questionnaire



Session: 1  
Group: A  
Case-ID: «CaseID»

## Briefing Questionnaire

## Remark

The following questions help us in analyzing the results in a more detailed manner. Your answers will be treated anonymously. The ID you provide will not be used to connect your questionnaire to your results.

Please answer the questions as complete and honest as possible. Thank you for your collaboration!

## Background information

Age _____ Gender <input type="radio"/> Male <input type="radio"/> Female What is your subject of study (e.g., computer science)? _____ What are you currently studying? <input type="radio"/> bachelor <input type="radio"/> master <input type="radio"/> diploma <input type="radio"/> PhD In which semester (Fachsemester) of your studies are you at the moment? _____
Have you participated in software development projects before? <input type="radio"/> Yes <input type="radio"/> No If yes, for how many <i>years</i> have you been working in the software industry (also as a "Hiwi")? <input type="radio"/> 1-2 <input type="radio"/> 3-4 <input type="radio"/> 4-6 <input type="radio"/> 6-8 <input type="radio"/> more than 8 If yes, how many industrial software <i>projects</i> have you participated in? <input type="radio"/> 1-2 <input type="radio"/> 3-4 <input type="radio"/> 4-6 <input type="radio"/> 6-8 <input type="radio"/> more than 8 If yes, what roles have you had in the project(s) (e.g., architect, programmer, tester, etc.)? _____ _____ Do you have experience in software reuse projects? <input type="radio"/> Yes <input type="radio"/> No If yes, for how many <i>years</i> have you participated in software reuse related tasks? <input type="radio"/> 1-2 <input type="radio"/> 3-4 <input type="radio"/> 4-6 <input type="radio"/> 6-8 <input type="radio"/> more than 8 If yes, how many <i>projects</i> have you participated in their software reuse related tasks? <input type="radio"/> 1-2 <input type="radio"/> 3-4 <input type="radio"/> 4-6 <input type="radio"/> 6-8 <input type="radio"/> more than 8

How do you rate your experience about reusing:

	Built (coded)	Analyzed (reviewed)	Learned (at school)	Read (individually)	I don't know it
Commercial-of-the-Shelf (COTS)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Web service APIs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Platforms APIs (SDKs)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Open-source software (OSS)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Is the GSE course the only source of your software reuse knowledge?  
☐ Yes    ☐ No

If no, where did you earn your further knowledge on software reuse? (e.g., projects, books, bachelor thesis work, master thesis work, etc.)

\_\_\_\_\_

\_\_\_\_\_

How do you rate you English proficiency in understanding text and speaking?

☐ 1 (Beginner: can understand basic words and expressions)

☐ 2 (Elementary: can understand frequently used expression and phrases)

☐ 3 (Intermediate: can understand the main idea of complex text )

☐ 4 (Professional: can understand with ease everything heard or read)

☐ 5 (Mother language)

## H.1.4 Task Description

## For Ad-hoc Approach



Session: 1  
Group: A  
Case-ID: «CaseID»

**Task Description****Task summary**

Your task is to play the role of a reuse analyst and to *first* find the “Conceptual Reuse Constraints” of a software system (which needs reuse) and a software unit (which is a reuse candidate) by analyzing their given documentation. *Next*, you will find the “Conceptual Reuse Mismatches” between them based on comparing their constraints that you have found.

**Input**

- The *Software Requirements Specification (SRS)* of the *Notification App* (Material 1)
- *Reuse Constraints Template for the Notification App* (Material 2): You will use this template to document the conceptual reuse constraints you find in the Notification App. The template is supported with an illustrating example.
- The *API documentation of AppleWatch* (Material 3)
- *Reuse Constraints Template for the AppleWatch API documentation* (Material 4): You will use this template to document the conceptual reuse constraints you find in the AppleWatch API documentation. The template is supported with an illustrating example.
- *Reuse Mismatches Template* (Material 5): You will use this template to document the conceptual reuse mismatches you find between the Notification App and AppleWatch. The template is supported with an illustrating example.

**Activities**

Please conduct the following activities:

- Activity 1: Finding the “Conceptual Reuse Constraints” in the SRS of the Notification App
  - Details provided on page 7
- Activity 2: Finding the “Conceptual Reuse Constraints” in the API documentation of the AppleWatch
  - Details provided on page 21
- Activity 3: Finding the “Conceptual Reuse Mismatches” between the reuse constraints of the Notification App and the AppleWatch
  - Details provided on page 31

## For Half COINA Approach



Session: 2  
Group: C  
Case-ID: C1

### Task Description

#### Task summary

Your task is to play the role of a reuse analyst and to *first* find the “Conceptual Reuse Constraints” of a software system (which needs reuse) and a software unit (which is a reuse candidate) by analyzing their given documentation. *Next*, you will find the “Conceptual Reuse Mismatches” between them based on comparing their constraints that you have found.

#### Input

You have the following *input* for your work:

- *Reuse Constraints Cheat Sheet for SRS* (Material 1): You will use this sheet to ensure that you have looked for all possible types of conceptual reuse constraints in the given software requirements specification (SRS) document. It is possible to find multiple instances of the same constraint type in the document.
- *The Software Requirements Specification (SRS) of the Notification App* (Material 2).
- *Reuse Constraints Template for the Notification App* (Material 3): You will use this template to document the reuse constraints you find in the Notification App. The template is supported with an illustrating example.
- *Reuse Constraints Cheat Sheet for API Documentation* (Material 4): You will use this sheet to ensure that you have looked for all possible types of conceptual reuse constraints in the given API documentation. It is possible to find multiple instances of the same constraint type in the document.
- *The API documentation of AppleWatch* (Material 5).
- *COINs List Template for the AppleWatch API documentation* (Material 6): You will use this template to document the COINs you find in the AppleWatch API documentation. The template is supported with an illustrating example.
- *Reuse Mismatches Template* (Material 7): You will use this template to document the reuse mismatches you find between the Notification App and AppleWatch. The template is supported with an illustrating example.

#### Activities

Please conduct the following activities:

- Activity 1: Finding the “Conceptual Reuse Constraints” in the SRS of the Notification App
  - Details provided on page 5
- Activity 2: Finding the “Conceptual Reuse Constraints” in the API documentation of the AppleWatch
  - Details provided on page 21
- Activity 3: Finding the “Conceptual Reuse Mismatches” between the reuse constraints of the Notification App and the AppleWatch
  - Details provided on page 33

## For Full COINA Approach



Session: 3  
Group: B  
Case-ID: B1

### Task Description

#### Task summary

Your task is to play the role of a reuse analyst and to *first* find the “Conceptual Reuse Constraints” of a software system (which needs reuse) and a software unit (which is a reuse candidate) by analyzing their given documentation. *Next*, you will find the “Conceptual Reuse Mismatches” between them based on comparing their constraints that you have found.

#### Input

You have the following *input* for your work:

- *Reuse Constraints Cheat Sheet for SRS* (Material 1): You will use this sheet to ensure that you have looked for all possible types of conceptual reuse constraints in the given software requirements specification (SRS) document. It is possible to find multiple instances of the same constraint type in the document.
- *The Software Requirements Specification (SRS) of the Notification App* (Material 2).
- *Reuse Constraints Template for the Notification App* (Material 3): You will use this template to document the reuse constraints you find in the Notification App. The template is supported with an illustrating example.
- *Reuse Constraints Cheat Sheet for API Documentation* (Material 4): You will use this sheet to ensure that you have looked for all possible types of conceptual reuse constraints in the given API documentation. It is possible to find multiple instances of the same constraint type in the document.
- *The API documentation of AppleWatch* (Material 5).
- *COINs List Template for the AppleWatch API documentation* (Material 6): You will use this template to document the COINs you find in the AppleWatch API documentation. The template is supported with an illustrating example.
- *Mismatches Cheat Sheet* (Material 7): You will use this sheet to ensure that you have looked for all possible types of reuse mismatches between the two systems. It is possible to find multiple instances of the same Mismatch type between the systems.
- *Reuse Mismatches Template* (Material 8): You will use this template to document the reuse mismatches you find between the Notification App and AppleWatch. The template is supported with an illustrating example.

#### Activities

Please conduct the following activities:

- Activity 1: Finding the COINs in the SRS of the Notification App
  - Details provided on page 5
- Activity 2: Finding the COINs in the API documentation of the AppleWatch
  - Details provided on page 21
- Activity 3: Finding the Mismatches between the COINs of the Notification App and the AppleWatch
  - Details provided on page 33



## H.1.5 Debriefing Questionnaire for Full COINA Approach



Session: 3  
Group: B  
Case-ID: B1

## Debriefing Questionnaire

**Remark**

The information you provide is important for evaluating the applicability and usefulness of the conceptual reuse analysis approach. Your answers will be treated anonymously. The ID you provide will not be used to connect your questionnaire to your results.

Please answer the questions as complete and honest as possible. Thank you for your collaboration!

1. How <i>well</i> did you <i>understand</i> the task?					
<input type="radio"/> 1 (Very bad) <input type="radio"/> 2 (Bad) <input type="radio"/> 3 (Moderate) <input type="radio"/> 4 (Good) <input type="radio"/> 5 (Very good)					
2. How <i>difficult</i> was it to <i>perceive</i> the task?					
<input type="radio"/> 1 (Very easy) <input type="radio"/> 2 (Easy) <input type="radio"/> 3 (Moderate) <input type="radio"/> 4 (Difficult) <input type="radio"/> 5 (Very difficult)					
3. How <i>difficult</i> was it to <i>perform</i> the task?					
<input type="radio"/> 1 (Very easy) <input type="radio"/> 2 (Easy) <input type="radio"/> 3 (Moderate) <input type="radio"/> 4 (Difficult) <input type="radio"/> 5 (Very difficult)					
4. How <i>interesting</i> was it to <i>perform</i> the task?					
<input type="radio"/> 1 (Very boring) <input type="radio"/> 2 (Boring) <input type="radio"/> 3 (Neutral) <input type="radio"/> 4 (Interesting) <input type="radio"/> 5 (Very interesting)					
5. How do you estimate the <i>quality</i> of your <i>results</i> (found reuse constraints and mismatches)?					
<input type="radio"/> 1 (Very low) <input type="radio"/> 2 (Low) <input type="radio"/> 3 (Moderate) <input type="radio"/> 4 (High) <input type="radio"/> 5 (Very high)					
6. During the reuse analysis task, I found the <i>Reuse Constraints Cheat Sheets</i> :					
	1 (Strongly disagree)	2 (Disagree)	3 (Agree)	4 (Strongly agree)	I don't know
<i>clear and understandable</i> how to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>easy to learn</i> how to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>flexible</i> to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>easy</i> for me to become <i>skillful</i> at using it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Using the *Reuse Constraints Cheat Sheets* during the reuse analysis:

	1 (Strongly disagree)	2 (Disagree)	3 (Agree)	4 (Strongly agree)	I don't know
made the task of finding the conceptual reuse constraints <i>easier</i>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me produce a <i>more complete</i> list of conceptual reuse constraints	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me produce a <i>more correct</i> list of <i>relevant</i> conceptual reuse constraints	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me produce <i>more consistent/ comparable</i> lists for the two systems	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me <i>save time</i> in producing the lists of conceptual reuse constraints	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. During the reuse analysis task, I found the *Mismatches Cheat Sheet*:

	1 (Strongly disagree)	2 (Disagree)	3 (Agree)	4 (Strongly agree)	I don't know
<i>clear</i> and <i>understandable</i> how to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>easy to learn</i> how to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>flexible</i> to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<i>easy</i> for me to become <i>skillful</i> at using it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. Using the *Mismatches Cheat Sheet* during the reuse analysis:

	1 (Strongly disagree)	2 (Disagree)	3 (Agree)	4 (Strongly agree)	I don't know
made the task of comparing the lists of reuse constraints <i>easier</i>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me produce a <i>more complete</i> list of Mismatches	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me produce a <i>more correct</i> list of <i>relevant</i> Mismatches	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me <i>save time</i> in producing the Mismatches lists	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

10. Do you have further comments about the case study or the used approach?

---



---



---



---



---



## H.2 Raw Data

		precision task1 (%)	recall task1 (%)	precision task2 (%)	recall task2 (%)	precision task3 (%)	recall task3 (%)	time (min)
Run II e1	Ad hoc	87.50	20.00	76.92	28.57	66.67	7.69	60
		87.50	20.00	100.00	17.14	100.00	15.38	60
		87.50	20.00	100.00	14.29	0.00	0.00	61
		60.00	17.14	100.00	25.71	100.00	11.54	67
		100.00	14.29	100.00	11.43	100.00	7.69	63
		69.23	25.71	83.33	28.57	60.00	11.54	68
		25.00	2.86	33.33	2.86	20.00	3.85	60
		75.00	8.57	25.00	2.86	60.00	11.54	66
		85.71	34.29	81.82	25.71	60.00	11.54	69
	Half COINA	92.86	37.14	66.67	5.71	0.00	0.00	45
		100.00	11.43	50.00	2.86	0.00	0.00	38
		90.00	25.71	100.00	11.43	100.00	7.69	50
		100.00	28.57	100.00	20.00	40.00	7.69	50
		75.00	34.29	85.00	48.57	0.00	0.00	44
		100.00	31.43	66.67	5.71	50.00	3.85	55
		90.91	28.57	100.00	8.57	100.00	7.69	55
		88.89	22.86	71.43	14.29	100.00	11.54	55
		90.91	28.57	75.00	25.71	50.00	3.85	46
	Full COINA	92.31	34.29	87.50	20.00	66.67	7.69	54
		91.67	31.43	90.00	25.71	100.00	3.85	64
		92.86	37.14	70.00	20.00	85.71	23.08	50
		50.00	2.86	100.00	11.43	100.00	11.54	51
		100.00	25.71	100.00	20.00	75.00	11.54	43
		100.00	11.43	87.50	20.00	100.00	11.54	48
		87.50	40.00	100.00	25.71	62.50	19.23	44
		88.24	42.86	100.00	25.71	83.33	19.23	41
		87.50	20.00	85.71	17.14	50.00	3.85	47
Run II e1	Ad hoc	83.33	38.46	80.00	25.00	50.00	5.26	63
		70.00	53.85	100.00	15.63	100.00	15.79	60
		90.91	38.46	80.00	12.50	50.00	5.26	60
		80.00	61.54	100.00	18.75	100.00	5.26	63
		80.00	15.38	100.00	28.13	50.00	5.26	55
		75.00	23.08	100.00	15.63	0.00	0.00	69
		57.14	15.38	100.00	21.88	0.00	0.00	65
		91.67	42.31	75.00	9.38	100.00	10.53	57
		80.00	46.15	87.50	21.88	75.00	15.79	64
	Half COINA	90.91	38.46	100.00	15.63	50.00	5.26	65
		100.00	26.92	100.00	25.00	100.00	10.53	60
		85.71	46.15	78.57	34.38	60.00	15.79	63
		90.91	38.46	85.71	18.75	75.00	15.79	50
		93.75	57.69	100.00	31.25	100.00	10.53	55
		100.00	46.15	100.00	25.00	100.00	21.05	59

			76.19	61.54	100.00	31.25	100.00	21.05	55
			71.43	38.46	100.00	34.38	100.00	15.79	56
			90.48	73.08	100.00	34.38	100.00	26.32	45
			92.31	92.31	100.00	40.63	100.00	63.16	60
			92.31	46.15	100.00	25.00	100.00	10.53	57
			91.67	42.31	100.00	12.50	100.00	21.05	60
			92.86	50.00	100.00	37.50	100.00	26.32	55
			100.00	34.62	100.00	18.75	100.00	26.32	55
			90.91	38.46	100.00	15.63	100.00	21.05	60
		<b>Full COINA</b>	93.33	53.85	100.00	18.75	100.00	21.05	51
			86.96	76.92	100.00	31.25	85.71	31.58	50
			100.00	46.15	100.00	21.88	100.00	26.32	47
			100.00	46.15	90.00	28.13	66.67	10.53	63
			100.00	42.31	100.00	15.63	100.00	21.05	50
			91.30	80.77	100.00	34.38	100.00	36.84	49
			92.00	88.46	100.00	28.13	100.00	52.63	49
			90.91	38.46	90.00	28.13	100.00	15.79	50
			100.00	38.46	100.00	12.50	100.00	21.05	51
			86.67	50.00	100.00	34.38	88.89	42.11	48
			100.00	42.31	100.00	15.63	100.00	47.37	50
			86.67	50.00	100.00	40.63	100.00	36.84	51
	<b>Run II e2</b>	<b>Ad hoc</b>	100.00	14.81	100.00	26.92	50.00	8.00	58
			90.91	37.04	70.00	26.92	83.33	20.00	59
			76.92	37.04	81.82	34.62	60.00	12.00	58
			70.59	44.44	63.64	26.92	80.00	16.00	58
		<b>Half COINA</b>	95.00	70.37	88.89	30.77	92.31	48.00	55
			94.12	59.26	85.71	46.15	87.50	28.00	57
			92.86	48.15	92.86	50.00	80.00	16.00	55
			88.24	55.56	77.78	26.92	75.00	12.00	55
		<b>Full COINA</b>	94.12	59.26	83.33	38.46	92.86	52.00	49
			94.44	62.96	91.67	42.31	83.33	20.00	50
			92.86	48.15	88.89	30.77	66.67	16.00	48
			94.44	62.96	94.74	69.23	90.00	36.00	50

# Appendix I Controlled Experiment for Evaluating the Guidelines for Improving API Documentation

## I.1 Experimental Material

For the experiment input for both the control and the treatment group (i.e., the original SoundCloud API document and the modified version), please go to this webpage: <http://abukwaik.com/site/api-guidelines16>

(use password: abukwaik\_experiment).




**Non-Disclosure Agreement & Informed Consent**

Case-ID: «CaseID»

Last name (Student): \_\_\_\_\_  
 First name: \_\_\_\_\_

To ensure the success of the study the student will keep confidential all information gained during the study and shall disclose any information about given materials especially to other students joining the course.

The student also accepts that the study is recorded for the purpose of anonymous analysis. All gathered data will be kept in confidence.

Kaiserslautern, \_\_\_\_\_ (Date)  
 \_\_\_\_\_ (Student signature)

Copy Rights to AGSE – Hadil Abukwaik

Page 1



Case-ID: «CaseID»

### Reuse Analysis Study – Procedures

First of all, **thank you** for participating in the study!

Please read this document carefully and contact your experimenter in case you have fundamental problems in understanding the given information or tasks.

Please remember, don't talk about the system details to other participants of the course before all sessions of the study are conducted!

Please try to perform the tasks *as fast as possible*, but aim at achieving results of *high quality*. The maximum allocated time for executing the case study is 60 minutes.

#### Preparation

- Fill the briefing questionnaire (page 3 and 4)
  - Typically, this step takes about 10 minutes

#### Execution

- Read the task description and conduct it (page 5)
  - Typically, the execution takes about 30 minutes

#### Finalization

- Fill in the Debriefing Questionnaire (page 8 - 9)
  - Typically, the finalization takes about 10 minutes

Case-ID: «CaseID»

**Briefing Questionnaire****Remark**

The following questions help us in analyzing the results in a more detailed manner. Your answers will be treated anonymously.

Please answer the questions as complete and honest as possible. Thank you for your collaboration!

**Background information**

Age \_\_\_\_\_

Gender ☐ Male ☐ Female

What is your subject of study (e.g., computer science)? \_\_\_\_\_

What are you currently studying?

☐ bachelor ☐ master ☐ diploma ☐ PhD

In which semester (Fachsemester) of your studies are you at the moment? \_\_\_\_\_

Have you participated in software development projects before?

☐ Yes ☐ NoIf yes, for how many *years* have you been working in the software industry (also as a "Hiwi")?☐ 1-2 ☐ 3-4 ☐ 4-6 ☐ 6-8 ☐ more than 8If yes, how many industrial software *projects* have you participated in?☐ 1-2 ☐ 3-4 ☐ 4-6 ☐ 6-8 ☐ more than 8

If yes, what roles have you had in the project(s) (e.g., architect, programmer, tester, etc.)?

---



---

Do you have experience in software reuse projects?

☐ Yes ☐ NoIf yes, for how many *years* have you participated in software reuse related tasks?☐ 1-2 ☐ 3-4 ☐ 4-6 ☐ 6-8 ☐ more than 8If yes, how many *projects* have you participated in their software reuse related tasks?☐ 1-2 ☐ 3-4 ☐ 4-6 ☐ 6-8 ☐ more than 8

How do you rate your experience about reusing:

	Built (coded)	Analyzed (reviewed)	Learned (at school)	Read (individually)	I don't know it
Commercial-of-the-Shelf (COTS)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Web service APIs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Platforms APIs (SDKs)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Open-source software (OSS)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

If you have any further knowledge of software reuse, where did you earn it from? (e.g., projects, books, bachelor thesis work, master thesis work, etc.)?

\_\_\_\_\_

\_\_\_\_\_

Do you have experience in reading UML diagrams?

☐ Yes ☐ No

How do you rate your experience about UML:

	Built (coded)	Analyzed (reviewed)	Learned (at school)	Read (individually)	I don't know it
Flow Chart Diagram	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Class Diagram	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Use Case Diagram	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

How do you rate your English proficiency in understanding text and speaking?

☐ 1 (Beginner: can understand basic words and expressions)

☐ 2 (Elementary: can understand frequently used expression and phrases)

☐ 3 (Intermediate: can understand the main idea of complex text )

☐ 4 (Professional: can understand with ease everything heard or read)

☐ 5 (Mother language)





Case-ID: «CaseID»

### Task Description

#### Task summary

Your task is to play the role of a developer or reuse analyst and to answer questions about the API services by analyzing their given API documentation.

#### Input

- *The API documentation of SoundCloud:* This document offers information about the offered API services.
- *Task sheet:* This sheet has questions related to the reuse constraints of the API services.

#### Activities

Please conduct the following activities:

- Activity 1: Read the questions in the task sheet carefully
- Activity 2: Read the API documentation of SoundCloud to find the answers for the questions in the task sheet
- Activity 3: Write the answers for the questions in the given area in the table.



Case-ID: «CaseID»

Write start time (e.g., 10:10 am): \_\_\_\_\_

## Task sheet

[illegible]

What are the <i>conditions</i> that should be satisfied to <i>Upload</i> an audio?		
What are the main <i>steps</i> towards <i>Uploading</i> an audio?		
What happens after successfully processing the request for <i>uploading</i> an audio file?	<input type="checkbox"/> Nothing <input type="checkbox"/> Confirmation message <input type="checkbox"/> Specific action; please specify:	
What is the returned value of most SoundCloud APIs?	Name:	
	Max no. of items:	
	Max value:	

Write end time (e.g., 10:40 am): \_\_\_\_\_

**Debriefing Questionnaire****Remark**

The information you provide is important for evaluating the usefulness and ease-of-use of the API documents that you read. Your answers will be treated anonymously.

Please answer the questions as complete and honest as possible. Thank you for your collaboration!

1. How *well* did you *understand* the task?

☐ 1 (Very bad)   ☐ 2 (Bad)   ☐ 3 (Moderate)   ☐ 4 (Good)   ☐ 5 (Very good)

2. How *difficult* was it to *perceive* the task?

☐ 1 (Very easy)   ☐ 2 (Easy)   ☐ 3 (Moderate)   ☐ 4 (Difficult)   ☐ 5 (Very difficult)

3. How *difficult* was it to *perform* the task?

☐ 1 (Very easy)   ☐ 2 (Easy)   ☐ 3 (Moderate)   ☐ 4 (Difficult)   ☐ 5 (Very difficult)

4. How *interesting* was it to *perform* the task?

☐ 1 (Very boring)   ☐ 2 (Boring)   ☐ 3 (Neutral)   ☐ 4 (Interesting)   ☐ 5 (Very interesting)

5. How do you estimate the *quality* of your *results* (your task answers)?

☐ 1 (Very low)   ☐ 2 (Low)   ☐ 3 (Moderate)   ☐ 4 (High)   ☐ 5 (Very high)

6. The API document that I read for answering the task questions:

	1 (Strongly disagree)	2 (Disagree)	3 Neutral	4 (Agree)	5 (Strongly agree)
was <i>easy</i> to <i>understand</i>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
was <i>easy</i> to <i>learn</i>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
didn't take <i>effort</i> to find the answers in it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
didn't take long <i>time</i> to find the answers in it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me to produce <i>complete</i> answers to the questions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
helped me to produce <i>correct</i> answers to the questions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. Do you have further comments about the study or the API documents that you read?

---

---

---

---

## I.2 Reference Solution for Experiment Task



Case-ID: «CaseID»

Write start time (e.g., 10:10 am): \_\_\_\_\_

### Task sheet

Question	Answer	
What <i>types of Apps</i> can use the SoundCloud APIs?	Smart Phone Apps (iOS and Android) Web Apps Desktop Apps	
Can the Apps access Soundcloud <i>core services</i> (e.g., uploading sounds) <i>directly</i> ?	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No, it is encapsulated	
Can the Apps access Soundcloud <i>data</i> (e.g., track) <i>directly</i> ?	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No, it is encapsulated	
Is SoundCloud a <i>layered</i> system?	<input checked="" type="checkbox"/> Yes; please specify the layers: <ol style="list-style-type: none"> <li>1. API (interface)</li> <li>2. Service</li> <li>3. Data (compound and basic)</li> </ol> <input type="checkbox"/> No	
What are the two <i>data categories</i> of Soundcloud?	1. Compound (whatever of the same meaning)	Example: user profile
	2. Basic (whatever of the same meaning)	Example: sound audio
What are the two main scenarios for the <i>Authentication</i> service?	1. Sign in with soundcloud	



	2. Sign in without soundcloud
What are the two scenarios that need to <i>Refresh Tokens</i> ?	1. Client-side Javascript Applications  2. Server-side Web Application
What are the <i>conditions</i> that should be satisfied to <i>Upload</i> an audio?	1. Registered and authenticated 2. Request body <500 MB 3. Request processed successfully
What are the main <i>steps</i> towards <i>Uploading</i> an audio?	1. Send a Post request to /tracks endpoint 2. Queue track for encoding 3. Check progress state of track 4. Stream/embed track
What happens after successfully processing the request for <i>uploading</i> an audio file?	<input type="checkbox"/> Nothing <input type="checkbox"/> Confirmation message <input checked="" type="checkbox"/> Specific action; please specify: Track queued for encoding
What is the <i>returned value</i> of most SoundCloud APIs?	Name: <b>Collection</b>
	Max no. of items: <b>50</b>
	Max value: <b>200</b>

Write end time (e.g., 10:40 am): \_\_\_\_\_



---

## Lebenslauf (CV)

<b>Name</b>	Hadil Abukwaik	
<b>Address</b>	Am Harzhuebel 126 Kaiserslautern, Germany	
<b>Birth date</b>	16.08.1984	
<b>Birth place</b>	Gaza, Palestine	
<b>Marital status</b>	Married, 2 children	
<b>Nationality</b>	Palestinian	
<b>Education</b>	1990-1996	Primary School
	1996 -1999	Preparatory School
	1999 - 2002	High School
	2002 - 2007	Bachelor of Computer Systems Engineering, Al-Azhar University, Gaza, Palestine
	2008 - 2010	Master of Computer Science, California State University, Sacramento, CA, USA
	2012 - 2017	PhD candidate of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany
	2007 - 2008	Lecturer at Al-Azhar University, Gaza, Palestine
	2008 - 2009	Software Developer at iScan Services Center, Sacramento, CA, USA
	2011 - 2016	Research assistant (part-time) at University of Kaiserslautern, Kaiserslautern, Germany
<b>Professional experience</b>	2017 - 2017	Wissenschaftlicher Mitarbeiter at University of Kaiserslautern, Kaiserslautern, Germany
	2018	Scientist in "Software Systems and Architectures" at ABB Corporate Research

Kaiserslautern, 25. February. 2018



# PhD Theses in Experimental Software Engineering

- Volume 1**      **Oliver Laitenberger** (2000), *Cost-Effective Detection of Software Defects Through Perspective-based Inspections*
- Volume 2**      **Christian Bunse** (2000), *Pattern-Based Refinement and Translation of Object-Oriented Models to Code*
- Volume 3**      **Andreas Birk** (2000), *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*
- Volume 4**      **Carsten Tautz** (2000), *Customizing Software Engineering Experience Management Systems to Organizational Needs*
- Volume 5**      **Erik Kamsties** (2001), *Surfacing Ambiguity in Natural Language Requirements*
- Volume 6**      **Christiane Differding** (2001), *Adaptive Measurement Plans for Software Development*
- Volume 7**      **Isabella Wieczorek** (2001), *Improved Software Cost Estimation A Robust and Interpretable Modeling Method and a Comprehensive Empirical Investigation*
- Volume 8**      **Dietmar Pfahl** (2001), *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*
- Volume 9**      **Antje von Knethen** (2001), *Change-Oriented Requirements Traceability Support for Evolution of Embedded Systems*
- Volume 10**    **Jürgen Münch** (2001), *Muster-basierte Erstellung von Software-Projektplänen*
- Volume 11**    **Dirk Muthig** (2002), *A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*
- Volume 12**    **Klaus Schmid** (2003), *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*
- Volume 13**    **Jörg Zettel** (2003), *Anpassbare Methodenassistenz in CASE-Werkzeugen*
- Volume 14**    **Ulrike Becker-Kornstaedt** (2004), *Prospect: a Method for Systematic Elicitation of Software Processes*
- Volume 15**    **Joachim Bayer** (2004), *View-Based Software Documentation*
- Volume 16**    **Markus Nick** (2005), *Experience Maintenance through Closed-Loop Feedback*

- Volume 17**     **Jean-François Girard** (2005), *ADORE-AR: Software Architecture Reconstruction with Partitioning and Clustering*
- Volume 18**     **Ramin Tavakoli Kolagari** (2006), *Requirements Engineering für Software-Produktlinien eingebetteter, technischer Systeme*
- Volume 19**     **Dirk Hamann** (2006), *Towards an Integrated Approach for Software Process Improvement: Combining Software Process Assessment and Software Process Modeling*
- Volume 20**     **Bernd Freimut** (2006), *MAGIC: A Hybrid Modeling Approach for Optimizing Inspection Cost-Effectiveness*
- Volume 21**     **Mark Müller** (2006), *Analyzing Software Quality Assurance Strategies through Simulation. Development and Empirical Validation of a Simulation Model in an Industrial Software Product Line Organization*
- Volume 22**     **Holger Diekmann** (2008), *Software Resource Consumption Engineering for Mass Produced Embedded System Families*
- Volume 23**     **Adam Trendowicz** (2008), *Software Effort Estimation with Well-Founded Causal Models*
- Volume 24**     **Jens Heidrich** (2008), *Goal-oriented Quantitative Software Project Control*
- Volume 25**     **Alexis Ocampo** (2008), *The REMIS Approach to Rationale-based Support for Process Model Evolution*
- Volume 26**     **Marcus Trapp** (2008), *Generating User Interfaces for Ambient Intelligence Systems; Introducing Client Types as Adaptation Factor*
- Volume 27**     **Christian Denger** (2009), *SafeSpection – A Framework for Systematization and Customization of Software Hazard Identification by Applying Inspection Concepts*
- Volume 28**     **Andreas Jedlitschka** (2009), *An Empirical Model of Software Managers' Information Needs for Software Engineering Technology Selection  
A Framework to Support Experimentally-based Software Engineering Technology Selection*
- Volume 29**     **Eric Ras** (2009), *Learning Spaces: Automatic Context-Aware Enrichment of Software Engineering Experience*
- Volume 30**     **Isabel John** (2009), *Pattern-based Documentation Analysis for Software Product Lines*
- Volume 31**     **Martín Soto** (2009), *The DeltaProcess Approach to Systematic Software Process Change Management*
- Volume 32**     **Ove Armbrust** (2010), *The SCOPE Approach for Scoping Software Processes*



- Volume 33**     **Thorsten Keuler** (2010), *An Aspect-Oriented Approach for Improving Architecture Design Efficiency*
- Volume 34**     **Jörg Dörr** (2010), *Elicitation of a Complete Set of Non-Functional Requirements*
- Volume 35**     **Jens Knodel** (2010), *Sustainable Structures in Software Implementations by Live Compliance Checking*
- Volume 36**     **Thomas Patzke** (2011), *Sustainable Evolution of Product Line Infrastructure Code*
- Volume 37**     **Ansgar Lamersdorf** (2011), *Model-based Decision Support of Task Allocation in Global Software Development*
- Volume 38**     **Ralf Carbon** (2011), *Architecture-Centric Software Producibility Analysis*
- Volume 39**     **Florian Schmidt** (2012), *Funktionale Absicherung kamerabasierter Aktiver Fahrerassistenzsysteme durch Hardware-in the-Loop-Tests*
- Volume 40**     **Frank Elberzhager** (2012), *A Systematic Integration of Inspection and Testing Processes for Focusing Testing Activities*
- Volume 41**     **Matthias Naab** (2012), *Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems*
- Volume 42**     **Marcus Ciolkowski** (2012), *An Approach for Quantitative Aggregation of Evidence from Controlled Experiments in Software Engineering*
- Volume 43**     **Igor Menzel** (2012), *Optimizing the Completeness of Textual Requirements Documents in Practice*
- Volume 44**     **Sebastian Adam** (2012), *Incorporating Software Product Line Knowledge into Requirements Processes*
- Volume 45**     **Kai Höfig** (2012), *Failure-Dependent Timing Analysis – A New Methodology for Probabilistic Worst-Case Execution Time Analysis*
- Volume 46**     **Kai Breiner** (2013), *AssistU – A framework for user interaction forensics*
- Volume 47**     **Rasmus Adler** (2013), *A model-based approach for exploring the space of adaptation behaviors of safety-related embedded systems*
- Volume 48**     **Daniel Schneider** (2014), *Conditional Safety Certification for Open Adaptive Systems*
- Volume 49**     **Michail Anastasopoulos** (2013), *Evolution Control for Software Product Lines: An Automation Layer over Configuration Management*
- Volume 50**     **Bastian Zimmer** (2014), *Efficiently Deploying Safety-Critical Applications onto Open Integrated Architectures*

- Volume 51**      **Slawomir Duszynski** (2015), *Analyzing Similarity of Cloned Software Variants using Hierarchical Set Models*
- Volume 52**      **Zhensheng Guo** (2015), *Safe Requirements Engineering: A Scenario-based Approach for Identifying Complete Safety-oriented Requirements*
- Volume 53**      **Bo Zhang** (2015), *VITAL – Reengineering Variability Specifications and Realizations in Software Product Lines*
- Volume 54**      **Norman Riegel** (2016), *Prioritization in Incremental Requirements Engineering*
- Volume 55**      **Pablo Oliveira Antonino de Assis** (2016), *Improving the Consistency and Completeness of Safety Requirements Specifications*
- Volume 56**      **Thomas Bauer** (2016), *Enabling Functional Integration Testing by Using Heterogeneous Models*
- Volume 57**      **Michael Kläss** (2016), *HyDEEP: Transparent Combination of Measurement and Expert Data for Defect Prediction*
- Volume 58**      **Liliana Katherine Guzmán Rehbein** (2017), *Empirically-based Method for Performing Qualitative Synthesis in Software Engineering*
- Volume 59**      **Michael Roth** (2017), *Qualitative Reliability Analysis of Software-Controlled Systems using State/Event Fault Trees*
- Volume 60**      **Hadil Abukwaik** (2017), *Proactive Support for Conceptual Interoperability Analysis of Software Units*

Software Engineering has become one of the major foci of Computer Science research in Kaiserslautern, Germany. Both the University of Kaiserslautern's Computer Science Department and the Fraunhofer Institute for Experimental Software Engineering (IESE) conduct research that subscribes to the development of complex software applications based on engineering principles. This requires system and process models for managing complexity, methods and techniques for ensuring product and process quality, and scalable formal methods for modeling and simulating system behavior. To understand the potential and limitations of these technologies, experiments need to be conducted for quantitative and qualitative evaluation and improvement. This line of software engineering research, which is based on the experimental scientific paradigm, is referred to as 'Experimental Software Engineering'.

In this series, we publish PhD theses from the Fraunhofer Institute for Experimental Software Engineering (IESE) and from the Software Engineering Research Groups of the Computer Science Department at the University of Kaiserslautern. PhD theses that originate elsewhere can be included, if accepted by the Editorial Board.

Editor-in-Chief: Prof. Dr. Dieter Rombach

Director Business Development of Fraunhofer IESE and Head of the AGSE Group of the Computer Science Department, University of Kaiserslautern

Editorial Board Member: Prof. Dr. Peter Liggesmeyer

Executive Director of Fraunhofer IESE and Head of the SEDA Group of the Computer Science Department, University of Kaiserslautern

Editorial Board Member: Prof. Dr. Frank Bomarius

Deputy Director of Fraunhofer IESE and Professor for Computer Science at the Department of Engineering, University of Applied Sciences, Kaiserslautern

