



Faculty of Computer Science, Electrical Engineering and Mathematics

Decision Support Ecosystems

Assisted Low-Code Development of
Tailored Decision Support Systems

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doktor der Naturwissenschaften (Dr. rer. nat.)

submitted by

Jonas Kirchhoff

November 2023

Abstract

Decision making significantly determines the success or failure of a business. This motivates decision makers to rely on decision support systems for assistance in identifying high-quality decisions using data simulation, optimization, and visualization. However, for optimal assistance, a decision support system (DSS) must align with the decision-making process of a decision maker, which is characterized by available data, optimization goals, personal preferences, and other influences. Unfortunately, the increasing complexity and volatility of business environments prevent DSS developers to anticipate all potential decision-making processes during DSS design, and consequently, to provide decision makers with sufficient customization options. Commissioning a DSS that is tailored to a decision-making process is a cost- and time-intensive undertaking due to limited developer availability or misunderstandings between developers and decision makers. As a result, decision makers may settle for an off-the-shelf DSS that does not fully align with their decision-making process and potentially results in suboptimal decisions, thereby impairing business success.

This thesis proposes an approach that enables decision makers to create tailored decision support systems themselves, thereby avoiding the aforementioned misalignment between provided and required decision support. The approach expects DSS developers to provide partial DSS functionality in the form of reusable software services. Using a low-code approach, non-programmers can then compose these services into a holistic DSS by modeling a decision-making process with the help of an assistance system. The contribution of the thesis is fourfold: First, the thesis explains how to design a service repository to capture available decision support services for the encompassing application domain. Second, the thesis introduces the concept of a process-driven DSS as an executable composition of decision support services that is tailored to the decision-making process of an individual decision maker. Third, the thesis proposes an assistance system that validates the composed DSS with respect to functional, informational, and operational characteristics. Fourth, the thesis describes how to aggregate these concepts into a decision support ecosystem with a platform for decision makers, DSS developers, and other domain experts to promote services, requirements for decision support, and knowledge to advance the development of tailored decision support systems. The applicability and technical feasibility of the concepts are demonstrated in the domain of energy distribution network planning.

Zusammenfassung

Erfolg und Misserfolg eines Unternehmens werden maßgeblich durch getroffene Entscheidungen beeinflusst. Daher verlassen sich Entscheider oft auf Entscheidungsunterstützungssysteme, die durch Datensimulation, -optimierung und -visualisierung bei der Identifizierung von geeigneten Entscheidungen unterstützen. Für eine optimale Unterstützung muss ein Entscheidungsunterstützungssystem (EUS) jedoch auf den Entscheidungsprozess eines Entscheiders abgestimmt sein und verfügbare Daten, Optimierungsziele, persönliche Präferenzen sowie weitere Einflussfaktoren berücksichtigen. EUS-Entwickler können aufgrund der Komplexität und Volatilität von Geschäftsumgebungen allerdings nicht alle potenziellen Entscheidungsprozesse während des Entwurfs eines EUS vorhersehen, wodurch ein EUS einem Entscheider häufig nur unzureichende Anpassungsmöglichkeiten an den individuellen Entscheidungsprozess bietet. Die Einzelanfertigung eines EUS, das auf einen Entscheidungsprozess zugeschnitten ist, ist ein kosten- und zeitintensives Unterfangen aufgrund der begrenzten Verfügbarkeit von Softwareentwicklern oder Missverständnissen zwischen Entwicklern und Entscheidern während der Entwicklung. Daher geben sich Entscheider möglicherweise mit einem handelsüblichen EUS zufrieden, das nicht vollständig mit ihrem Entscheidungsprozess übereinstimmt, suboptimale Entscheidungen begünstigt und so den Unternehmenserfolg negativ beeinflusst.

In dieser Arbeit wird ein Ansatz vorgeschlagen, der es Entscheidern ermöglicht, selbst maßgeschneiderte Entscheidungsunterstützungssysteme zu entwickeln und so die Diskrepanz zwischen benötigter und tatsächlicher Entscheidungsunterstützung zu vermeiden. Dazu stellen EUS-Entwickler einen Teil der EUS-Funktionalität als wiederverwendbare Software-Dienste bereit. Nicht-Programmierer können diese Dienste dann mit einem Low-Code-Ansatz zu einem ganzheitlichen EUS kombinieren, indem sie einen Entscheidungsprozess mit Hilfe eines Assistenzsystems modellieren. Dabei ist der Beitrag dieser Arbeit viererlei: Erstens wird erklärt, wie verfügbare Dienste für die Entscheidungsunterstützung in einer Anwendungsdomäne erfasst werden können. Zweitens wird das Konzept eines prozessgesteuerten EUS als eine maßgeschneiderte, ausführbare Komposition von Entscheidungsunterstützungsdiensten vorgestellt. Drittens wird ein Assistenzsystem präsentiert, welches das komponierte EUS im Hinblick auf funktionale, informationelle und operative Eigenschaften validiert. Viertens wird in der Arbeit beschrieben, wie diese Konzepte zu einem Ökosystem für die Entscheidungsunterstützung zusammengeführt werden können, das eine Plattform für Entscheider, EUS-Entwickler und andere Domänenexperten bietet, um Dienste, Anforderungen an die Entscheidungsunterstützung und Wissen zu verbreiten und die Entwicklung maßgeschneiderter Entscheidungsunterstützungssysteme voranzutreiben. Die Anwendbarkeit und technische Machbarkeit der Konzepte wird beispielhaft für Energieverteilnetzplanung demonstriert.

Acknowledgements

I am grateful to everyone who supported me during my work on this thesis.

Special thanks go to my thesis supervisor Prof. Dr. Gregor Engels, who accepted me as his last doctoral student despite his forthcoming retirement. Gregor, thank you for your commitment and continuous encouragement, and for sharing your knowledge with me! I would also like to thank Prof. Dr. Achim Koberstein for his external review of my thesis, Dr. Christoph Weskamp for his initial feedback on my thesis topic, and Prof. Dr. Stefan Sauer and Prof. Dr. Yasemin Acar for completing my interdisciplinary examination board.

My thanks also extend to the colleagues, industry partners and students with whom I collaborated throughout the research projects *FlexiEnergy* and *Pro-LowCode*. Thank you for the successful cooperation and informative discussions! The insights I gathered throughout these projects significantly contributed to the practical grounding of my research and strengthened my interest in working on innovative solutions for real-world challenges.

Furthermore, I would like to thank my colleagues from the *Software Innovation Campus Paderborn* and the *Database and Information Systems* research group in particular for the productive and friendly working environment. Thank you for the helpful discussions and for your support in all organizational, administrative and technical matters! I enjoyed our professional collaboration during projects and networking events as much as the off-topic conversations during lunch or the occasional get-together outside of work.

Lastly, I would like to thank my family and friends for their continuous support during my work on this thesis and throughout the overall course of my studies.

Contents

List of Acronyms	xv
List of Figures	xvii
List of Tables	xxi
I Problem Identification and Solution Objectives	1
1 Introduction	3
1.1 Context: The Need for Decision Support	3
1.2 Motivation: Inappropriate Decision Support	4
1.3 Problem Statement: Tailored DSS Unavailability	7
1.4 Objective: Assisted DSS Creation by Decision Makers	11
1.5 Research Approach: Design Science	14
1.6 Research Overview: Contributions and Publications	15
1.7 Thesis Structure	20
2 Individuality of Decision Making	21
2.1 Context: Energy Distribution Network Planning	21
2.2 Decision Making Foundations	24
2.2.1 Decision Problems	24
2.2.2 Decision Quality	26
2.2.3 Decision Process	27
2.2.4 The Meta-Level of Decision Making	29
2.3 Situational Factors of Decision Making	30
2.4 VUCA and Its Effect on Situational Decision Making	34
2.5 Key Takeaways	36

3	Requirements and Related Work	37
3.1	Requirements	37
3.2	Related Trends in Software Development	42
3.2.1	Service-Oriented Computing	42
3.2.2	Low-Code Development	47
3.3	Related Work	51
3.3.1	Adaptive Decision Support Systems	52
3.3.2	DSS Generators	55
3.3.3	Service-Oriented DSS	62
3.3.4	Distantly Related Research Areas and Approaches	66
3.4	Key Takeaways	67
II	Solution Design and Development	69
4	Decision Support Ecosystems	71
4.1	Background: Digital Business Ecosystems	71
4.1.1	Definition	72
4.1.2	Characteristics	72
4.1.3	Types	74
4.1.4	Benefits	75
4.2	Towards Decision Support Ecosystems	76
4.2.1	Definition and Categorization of a DSE	76
4.2.2	Relation to the Challenges of Tailored DSS Development	77
4.2.3	Complexity of DSE Implementation	78
4.3	DSE Design Principles	79
4.4	DSE Constituents	83
4.4.1	Types of Decision Support Services	83
4.4.2	DSE Roles	84
4.4.3	DSE Lifecycle	87
4.5	Reference Architecture for a DSE Platform	89
4.6	Key Takeaways	94
5	Description of Decision Support Services	95
5.1	Context and Upfront Considerations	96
5.2	Description Requirements	97
5.3	Background and Related Work	100

5.3.1	Decision Support Ontologies	100
5.3.2	General-Purpose Service Description Languages	102
5.4	A Meta-Model for Decision Support Services	103
5.4.1	Package “Data”	104
5.4.2	Package “Computation”	106
5.4.3	Package “Optimization”	107
5.4.4	Package “Resources”	109
5.4.5	Package “Services”	110
5.4.6	Package “DSS Requirements”	113
5.5	Prototypical Implementation	115
5.6	Discussion	117
5.7	Key Takeaways	119
6	Process-Driven Decision Support Systems	121
6.1	Composition Requirements	122
6.2	Background and Related Work	125
6.3	Architectural Considerations	127
6.4	Composition of Decision Support Services with BPMN	129
6.4.1	Functional & Operational Perspective: Tasks and Subprocesses	129
6.4.2	Informational Perspective: Data and Data Associations	134
6.4.3	Behavioral Perspective: Sequence Flows, Gateways & Events	139
6.5	Prototypical Implementation	144
6.5.1	Implementation of the PD-DSS Design Application	145
6.5.2	Implementation of the PD-DSS Enactment Application	145
6.6	Discussion	147
6.7	Key Takeaways	149
7	Composition Assistance	151
7.1	Upfront Design Considerations	152
7.1.1	Approaches to Composition Assistance	152
7.1.2	Integration into the DSE Platform	156
7.1.3	Documentation of Validation Rules	157
7.2	Cross-Cutting Requirements	158
7.3	Operational Composition Assistance	159
7.3.1	Motivational Scenario	159
7.3.2	Requirements	159
7.3.3	Background and Related Work	160

7.3.4	Design and Demonstration	161
7.3.5	Discussion	163
7.4	Informational Composition Assistance	167
7.4.1	Motivational Scenario	167
7.4.2	Requirements	168
7.4.3	Background and Related Work	170
7.4.4	Design and Demonstration	173
7.4.5	Prototypical Implementation	178
7.4.6	Discussion	181
7.5	Functional and Behavioral Composition Assistance	184
7.5.1	Motivational Scenario	184
7.5.2	Requirements	186
7.5.3	Background and Related Work	188
7.5.4	Design and Demonstration	190
7.5.5	Prototypical Implementation	199
7.5.6	Discussion	201
7.6	Key Takeways	204

III Solution Discussion 205

8 Case-Study Evaluation 207

8.1	Context: Research Project “FlexiEnergy”	208
8.2	Phase 1 – Foundation	210
8.2.1	Domain Documentation: Data Types and Metadata Attributes	210
8.2.2	Domain Documentation: Computation Methods	211
8.2.3	Descriptions of Decision Support Services	214
8.2.4	Relation to Requirements for Tailored DSS Development	216
8.3	Phase 2 – Design	218
8.3.1	Cross-Sector Distribution Network Planning	218
8.3.2	Electricity Distribution Network Planning with DSM	219
8.3.3	DSM Benchmarking / Meta Decision Making	219
8.3.4	Relation to Requirements for Tailored DSS Development	221
8.4	Phase 3 – Enactment	223
8.4.1	PD-DSS Instantiation	224
8.4.2	PD-DSS Usage	224
8.4.3	Relation to Requirements for Tailored DSS Development	225

8.5	Phase 4 – Improvement	227
8.5.1	Feedback: PD-DSS and Service Reviews	227
8.5.2	Composition Knowledge: Anti-Patterns	228
8.5.3	Relation to Requirements for Tailored DSS Development	229
8.6	Threats to Validity	230
8.7	Key Takeaways	231
9	Summary and Future Work	233
9.1	Summary of Contributions	233
9.2	Future Work	237
	References	241
	Index of Definitions	263
	Appendix A Composition Assistance Supplement	267
A.1	Completeness Validation of PD-DSS Process Models	267
A.1.1	Completeness of the Operational Perspective	268
A.1.2	Completeness of the Informational Perspective	268
A.1.3	Completeness of the Functional and Behavioral Perspectives	269
A.2	Transformation Changes	271
A.2.1	Nonexistent Flows	271
A.2.2	Conditions on Nonsequential Flows	272
	Appendix B Prototypical Implementation	273

List of Acronyms

ADSS	adaptive decision support system
API	application programming interface
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CAPEX	capital expenditure
CMS	content management system
CSV	comma-separated values (file format)
DMN	Decision Model and Notation
DNO	distribution network operator
DNP	distribution network planning
DSE	decision support ecosystem
DSM	demand side management
DSS	decision support system
DSSG	decision support system generator
EUD	end-user development
EV	electric vehicle
FEEL	Friendly Enough Expression Language
GUI	graphical user interface
HTTP	Hypertext Transfer Protocol
IPO	input-processing-output
JSON	JavaScript Object Notation

LCD	low-code development
LCDP	low-code development platform
MDD	model-driven development
MMS	model management system
OPEX	operating expense
PD-DSS	process-driven decision support system
PDA	process-driven application
REST	Representational State Transfer
SO-DSS	service-oriented decision support system
SOA	service-oriented architecture
UI	user interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
VUCA	volatility, uncertainty, complexity, ambiguity
WS-*	web service technology stack
WSDL	Web Service Description Language
XML	Extensible Markup Language

List of Figures

1.1	Visualization of potential misalignments in required and provided decision support	5
1.2	Visualization of challenges for the mass availability of tailored DSSs	8
1.3	Overview of the <i>Decision Support Ecosystem</i> concept	16
1.4	Overview of the <i>Process-Driven DSS</i> concept	17
1.5	Overview of the <i>DSS Modeling Assistance</i> concept	18
1.6	Overview of contributions and associated publications	19
2.1	Schematic visualization of an energy system with a focus on electricity	22
2.2	Decision making entities visualized as a UML class diagram (default multiplicity: 1)	25
2.3	Classification of decision problems (adapted from [Sàn22])	26
2.4	Decision process structure according to [MRT76]	28
2.5	Comparison of two similar decision processes for DNP	31
3.1	SOA Publish-Find-Bind triangle (adapted from [Sch+05])	44
3.2	The two fundamental approaches to service coordination	45
3.3	Fundamental architecture of LCDPs	50
3.4	Possible approaches to implement a DSS generator	56
4.1	Overview of a digital business ecosystem	73
4.2	Types of Digital Business Ecosystems	74
4.3	Categorization of Decision Support Ecosystems	77
4.4	Types of decision support services in a DSE	84
4.5	Overview of DSE roles and their interactions	85
4.6	Overview of the continuous DSE lifecycle with involved actors and produced outputs	88
4.7	Reference Architecture for a DSE Platform	90

5.1	Focus of Chapter 5 with respect to the DSE lifecycle	95
5.2	Applications for the “Foundation” lifecycle phase as a component diagram .	96
5.3	Overview of meta-model packages	104
5.4	Overview of the meta-model package Data	105
5.5	Example instantiation of the meta-model package Data	105
5.6	Overview of the meta-model package Computation	106
5.7	Example instantiation of the meta-model package Computation	107
5.8	Overview of the meta-model package Optimization	108
5.9	Example instantiation of the meta-model package Optimization	108
5.10	Overview of the meta-model package Resources	109
5.11	Example instantiation of the meta-model package Resources	110
5.12	Overview of the meta-model package Services (Part 1)	110
5.13	Overview of the meta-model package Services (Part 2)	111
5.14	Example instantiation of the meta-model package Services	112
5.15	Overview of the meta-model package Requirements	114
5.16	Screenshot of the admin UI of the prototypical implementation	116
5.17	Overview of using the CMS <i>Payload</i> for the prototypical implementation .	116
6.1	Focus of Chapter 6 with respect to the DSE lifecycle	121
6.2	Overview of the architecture as a UML component diagram	128
6.3	Annotated excerpt of the BPMN meta-model for the functional and operational perspective of a PD-DSS (based on [Obj13])	130
6.4	Functional and operational perspective of an exemplary PD-DSS model . .	133
6.5	Annotated excerpt of the BPMN meta-model for the informational perspective of a PD-DSS process model (based on [Obj13])	135
6.6	Informational perspective of an exemplary PD-DSS process model	137
6.7	Annotated excerpt of the BPMN meta-model for the behavioral perspective of a PD-DSS process model (based on [Obj13])	140
6.8	Behavioral perspective of an exemplary PD-DSS process model	143
6.9	Mockup of the <i>PD-DSS Repository</i> application	144
6.10	Screenshot of the <i>PD-DSS Design</i> application	145
6.11	Mockup of the <i>PD-DSS Enactment</i> application for initial data selection . . .	147
6.12	Mockup of the <i>PD-DSS Enactment</i> application during service execution . .	148
7.1	Focus of Chapter 7 with respect to the DSE lifecycle	151
7.2	Integration of the composition assistance into the DSE platform	156

7.3	Example for selected violations reported by the operational composition assistance	164
7.4	Foundation of running example for the informational composition assistance	168
7.5	Demonstration of informational violations with respect to output data . . .	175
7.6	Demonstration of informational violations for input data	179
7.7	Foundation for the running example	185
7.8	Demonstration of domain-agnostic and domain-parametrized violations . .	192
7.9	Architectural overview of the functional-behavioral composition assistance	193
7.10	Demonstration of selected domain-specific anti-patterns	199
7.11	Internal architecture of the functional-behavioral composition assistance . .	200
8.1	All DSE lifecycle phases are addressed with the presented solution design .	207
8.2	Refinement of the DSE platform reference architecture based on the solution design	209
8.3	Process model excerpt for a PD-DSS supporting sector coupling	218
8.4	Excerpt of a PD-DSS supporting electricity networks with DSM	220
8.5	Process model excerpt for benchmarking DSM effects on vehicle charging .	221
8.6	Mockup of PD-DSS instantiation in the case study context	225
8.7	Mockup of interactive PD-DSS enactment in the case study context	226
9.1	Overview of thesis contributions (cf. Fig. 8.2 for data exchange)	235
A.1	Exemplary process model with <i>Missing Reversal of Network Reduction</i> anti-pattern	271

List of Tables

3.1	Summary of ADSS approaches with respect to requirements of Section 3.1	55
3.2	Evaluation of DSS generator approaches with respect to the requirements of Section 3.1	61
3.3	Evaluation of SO-DSS approaches with respect to requirements of Section 3.1	65
4.1	Mapping between design principles and requirements of Section 3.1	80
4.2	Comparison between LCDP and DSE platform components	93
7.1	(Extended) BPMN-Q elements for the graphical definition of anti-patterns .	196
8.1	Contribution of DSE lifecycle phases towards requirements of Section 3.1 .	232

PART I

Problem Identification and Solution Objectives

Introduction

Decision making is among the most important activities in a business and relies on assistance in the form of digital decision support systems (cf. Section 1.1). Although the steps performed to obtain a decision – and consequently the requirements for decision support – are very individual between different decision makers (cf. Section 1.2), the creation of a tailored decision support system that optimally addresses the individual requirements of a decision maker is still challenging (cf. Section 1.3). This chapter presents research objectives for the effective and efficient creation of tailored decision support systems (cf. Section 1.4). Furthermore, the research approach applied throughout the thesis is described (cf. Section 1.5), and an overview of the resulting contributions and associated publications is given (cf. Section 1.6). Lastly, the subsequent structure of the thesis is presented (cf. Section 1.7).

1.1 Context: The Need for Decision Support

Good business decisions help a business to thrive and to increase or maintain its competitive advantage over other businesses [BL18; MBD20]. Bad decisions result in a significant loss of economic value [SWM16]. Given these significant impacts, decision making is considered to be among the most important activities in a business [Cor80; Har96; Rol21].

The number of employees in decision-intensive occupations is currently rising [Dem21]. Hundreds to thousands of business decisions are made within a business daily [BL18]. However, most businesses do not (only) make good decisions [SWM16]. Some experts estimate that “half of the decisions made in organizations fail” [Nut02]. To some extent, this

can be attributed to the fact that decision making is becoming increasingly difficult [Pow09], in part due to the growing volatility, uncertainty, complexity and ambiguity (*VUCA*) in business environments [KQ08; BL14; Mac+16]. In short, decision makers in *VUCA* business environments must consider many unpredictable and frequently changing influencing factors with unknown cause-effect relationships when identifying an optimal decision alternative.

Due to the increasing difficulty of decision making in *VUCA* business environments, decision makers rely on decision support systems to assist them in their decision-making process [SWM16]. A *decision support system (DSS)* is an “interactive computer-based system [...] designed to help decision makers” [PI18] by “improving decision-making effectiveness and efficiency” [Pow09] using software-based simulation, optimization, and other data processing capabilities [SBM11]. In doing so, a DSS helps to address the challenges of *VUCA* business environments by providing decision makers with means for experimentation to quickly obtain decision recommendations while abstracting away a significant amount of complexity.

The previous arguments could suggest that the use of a DSS already addresses all challenges of decision making in *VUCA* business environments. Certainly, the use of a DSS is likely very beneficial – in fact, evidence accumulated from research and case studies actually shows the potential of an *appropriate and well-designed* DSS to improve decision quality [Pow09]. Yet, the explicit mention of the characteristics “appropriate” and “well-designed” in the statement suggests the existence of inappropriate systems with design flaws that have no positive impact on decision making. The next subsection explains why a DSS could be inappropriate or ill-designed, and what consequences the use of such a DSS might have on decision quality.

1.2 Motivation: Inappropriate Decision Support

For a DSS to be appropriate and well-designed, the decision support provided by the DSS must align with the requirements for decision support of a decision maker. However, different decision makers follow different decision-making processes for the same type of decision due to differences in goals, regulatory constraints, or access to resources such as data. As a result, they have varying requirements for decision support. This is demonstrated with the abstract example illustrated in Fig. 1.1, which is subsequently discussed in more detail.

Example of Misalignments in Required and Provided Decision Support

In Fig. 1.1, four decision makers of different companies want to plan the same type of infrastructure investments to meet future demands. However, they have slightly varying requirements: While the first and third decision maker require the infrastructure to be

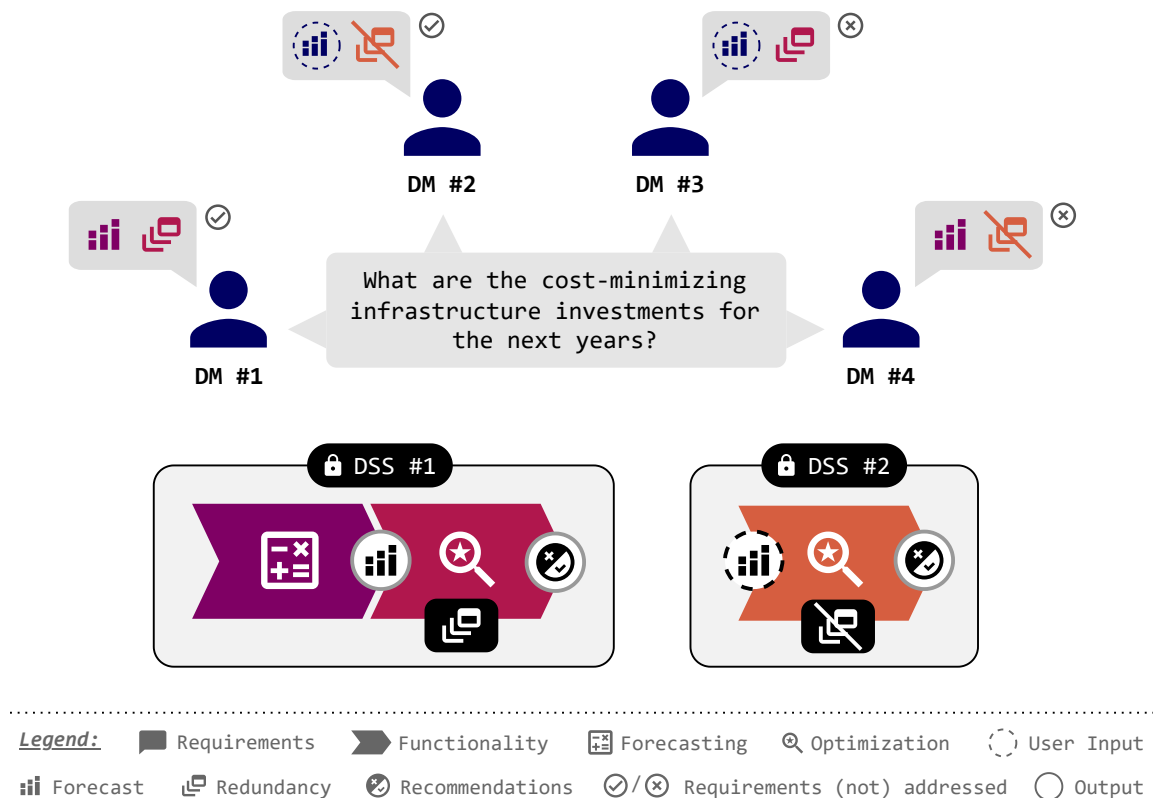


Figure 1.1: Visualization of potential misalignments in required and provided decision support

redundant to account for technical failures, the second and fourth decision maker want to avoid redundancy. Furthermore, the first and fourth decision makers also require their DSS to create a forecast for infrastructure demands, which the second and third decision maker already obtained beforehand from an external agency and want to provide as an input to their DSS. All decision makers can choose between two off-the-shelf decision support systems that are shown at the bottom of Fig. 1.1: The first DSS implements a two-stage decision-making process where first a forecast for future infrastructure demands is created, and subsequently, the cost-minimizing infrastructure expansions are computed to meet these demands *with* consideration of redundancy. The second DSS does not calculate a forecast but instead requests it as input from the decision maker before infrastructure investments are computed *without* consideration of redundancy. Consequently, both the first and second decision maker find a DSS that immediately aligns with their requirements for decision support as the first decision maker can use the first DSS, and the second decision maker can use the second DSS. Unfortunately, neither the third nor the fourth decision maker find a DSS that addresses all of their requirements for decision support. The second decision maker would prefer the optimization of the first DSS as it provides the necessary redundancy, but the forecast already

available to the decision maker cannot be used with the first DSS as it does not support any user inputs. Using the forecast would be possible with the second DSS, however, this DSS does not guarantee the required redundancy. Analogously for the fourth decision maker, the second DSS fits the non-redundancy requirement, but does not compute the missing forecast like the first DSS, which however does not optimize with consideration for redundancy. As indicated by the lock symbol next to the name of the DSS, each DSS is provided by a different developer and therefore the DSS modules are not freely interchangeable. Due to a tight coupling between DSS modules, the fourth decision maker is also unable to compute only the forecast using the first DSS to subsequently use the forecast with the second DSS.

The illustrated circumstance of potential misalignments between required and provided decision support raises the question regarding the consequences of using an inappropriate DSS during decision making, since the implemented decision-making process has a significant influence on the outcome and success of a decision (cf. [DS96; EC07; Zar13]).

Consequences of Using an Inappropriate DSS: Inefficiency and Ineffectiveness

In general, a misalignment in the desired decision-making process of a decision maker and the process supported by a DSS can result in suboptimal or delayed decisions. “Suboptimal” refers to the fact that a different *optimal decision* provides more value to the decision maker and the associated business. Often, this value corresponds to a monetary benefit [SWM16], whereby optimal solutions amount to a maximization of economic value, and suboptimal decisions to unrealized economic value. In the example previously illustrated in Fig. 1.1, the fourth decision maker might choose the first DSS because it computes the forecast for infrastructure demands that the decision maker is missing. However, the first DSS also subsequently designs and plans the infrastructure in a redundant way contrary to the requirements of the decision maker. The resulting investment recommendations are therefore likely more costly than those that do not account for redundancy, leading to a loss of economic value.

The second potential effect of misalignments between required and supported decision-making processes is a delayed decision. A delay of multiple days can be introduced by executing activities that are not needed or unnecessarily complex. A delayed decision can have two effects: First, a delayed decision can result in decreased competitiveness of the business. Especially due to the aforementioned volatility in VUCA business environments, it is crucial to make decisions fast to gain or maintain a competitive advantage over other businesses. Second, a delayed decision can turn into a suboptimal solution. This is also supported by a study of de Smet, Jost, and Weiss from McKinsey, who surveyed 1,200 managers of global companies and found “a strong correlation between quick decisions and good ones” [dSJW19]. This risk of delay affecting decision quality is especially high when the decision-making

process is altered to make up for lost time. In the example previously illustrated in Fig. 1.1, the importance of a redundant network infrastructure might force the third decision maker to use the first DSS, although it includes the computation of an unneeded demand forecast as the decision maker already has access to such a forecast. As a result, the decision maker might use a trial-and-error approach to get the forecasting module to produce the same (already available) forecast. Given a tight deadline, the delay caused by this trial-and-error execution might lose so much time that the subsequent optimization does not have enough time to identify the most optimal investment plan, thereby turning an otherwise delayed into a suboptimal decision.

In summary, an inappropriate DSS with a misalignment between desired and supported decision-making process is either *ineffective* (because suboptimal solutions are recommended) or *inefficient* (because a delay in decision making is introduced). This contradicts the purpose of a DSS, which was defined at the beginning of this chapter as “to help decision makers by improving decision-making effectiveness and efficiency” (cf. page 4). Consequently, there is a need for decision support systems whose support is tailored to the individual decision-making process of a decision maker. For example, concerning the previous example shown in Fig. 1.1, a tailored DSS for the fourth decision maker would consist of the forecasting module from the first DSS and the optimization module of the second DSS to fully address all the decision maker’s requirements for decision support. However, there is a lack of such composability.

1.3 Problem Statement: Tailored DSS Unavailability

A *tailored DSS* is a DSS that is custom-made to address the individual requirements for decision support of a specific decision maker. In other words, the decision-making process supported by a tailored DSS fully aligns with the preferred decision-making process of a decision maker. A tailored DSS thereby allows for efficient and effective decision making with timely recommendations of optimal decisions and increases the competitive advantage of a business. Unfortunately, experience from the recent research project *FlexiEnergy*¹ indicates that providing each decision maker with such a tailored DSS still presents a challenge for three reasons, which are summarized in Fig. 1.2 and subsequently explained in more detail.

Challenge 1 – Lack of Customization: The holistic approach to DSS development limits the adaption and recombination of decision support functionality

As observed throughout the *FlexiEnergy* research project for decision support in energy distribution network planning, decision makers usually turn to established DSS developers

¹ Project website: <https://flexi-energy.de>

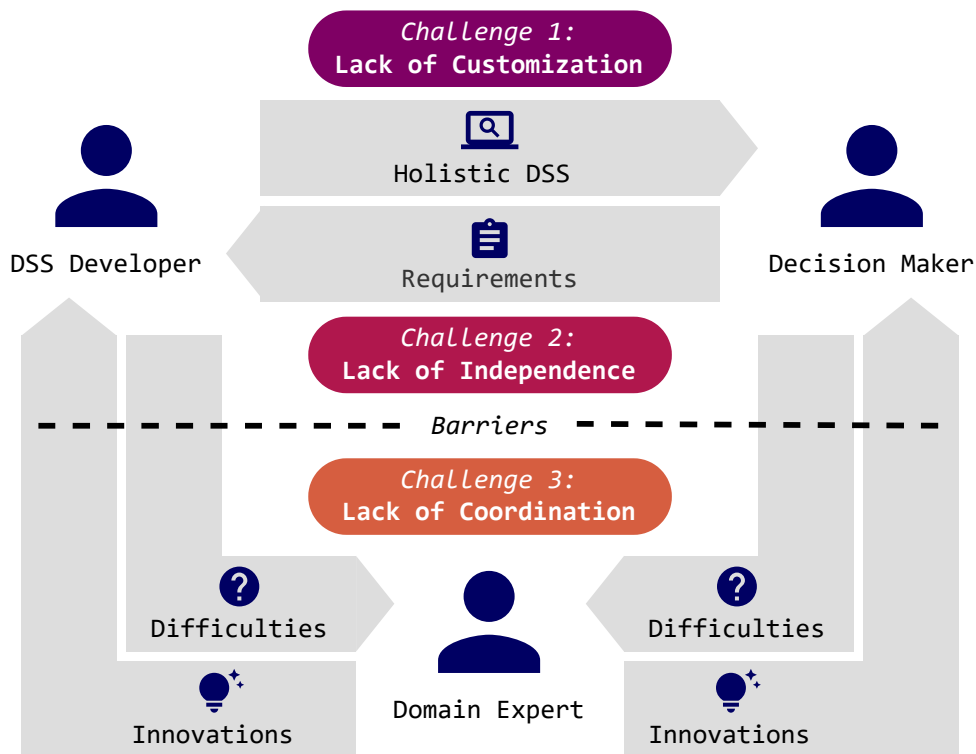


Figure 1.2: Visualization of challenges for the mass availability of tailored DSSs

to obtain an “off-the-shelf” DSS. Such an *off-the-shelf DSS* is available for purchase to all customers without any additional development effort by the DSS developer to adapt the DSS to the customer. It is *holistic* as it is a complete but closed system without the possibility for anyone besides the DSS developer to exchange parts of the DSS. This implies that all customizations required by the decision maker for an alignment between the preferred and supported decision-making process must already be provided by the DSS by some form of configuration. Consequently, any customization needs must be anticipated by the DSS developer during the design and development of the DSS, which is difficult given the previously discussed complexity in business environments and the individuality of decision making.

When a DSS lacks the required customization capabilities, some decision makers try to mimic a tailored DSS by using functionality from multiple decision support systems for assistance with different activities of their decision-making process. While this can work in principle, it comes with three disadvantages: First, the data exported from one DSS is not necessarily compatible with another DSS. The necessary data conversion is an overhead that is exceptionally inefficient and error-prone when data must be converted manually. Second, a decision maker must adapt to the characteristics of each used DSS such as keyboard shortcuts or user interface layout. This adds additional cognitive load and therefore inefficiency during

decision making. Third, decision makers must ensure the “functional compatibility” between each used DSS. For example, they must validate that the assumptions of one DSS align with the other, e.g., a second DSS continues the redundant infrastructure planning from a first DSS instead of undoing previous efforts. Otherwise, the decision support can be ineffective with the previously discussed consequences of suboptimal and/or delayed decisions.

Challenge 2 – *Lack of Independence: Relying on DSS developers is unavoidable and can result in inefficient and ineffective DSS development*

If no suitable off-the-shelf DSS is available, a decision maker can commission a DSS developer to develop a tailored DSS. This can either be approached as a development from scratch, or more time- and cost-efficient, as a modification of an existing off-the-shelf DSS that already addresses most of the decision maker’s requirements for decision support. Regardless of the chosen approach, the development of a tailored DSS has two fundamental prerequisites, namely the availability of DSS developers and successful communication between developers and decision maker. As subsequently explained, these prerequisites are not necessarily given.

The communication between a decision maker and a DSS developer is important as decision makers must first elaborate their requirements for decision support and subsequently give feedback on how the adaptations or extensions implemented by the DSS developer address these requirements. In between, the DSS developer may need to ask questions about the business environment of the decision maker for a better understanding of the requirements or for improving the design of the DSS. At any time, there is a risk of misunderstanding, e.g., due to the lack of a common understanding and consequently using the same or similar names for different domain concepts (cf. [Lie+18]). If not caught, these misunderstandings can lead to quality issues [BWR11], e.g., an undesired behavior of the DSS during usage. Furthermore, ineffective communication likely leads to “waste”, i.e., “any activity that produces no value for the customer or user” [SRP17]. Unfortunately, as Khankaew and Riddle [KR14] show for software development in small and medium enterprises, effective communication between developers and customers is often lacking. Consequently, the availability of a DSS and therefore decision making itself is delayed, resulting in an inefficiency during decision making.

Even when assuming effective communication between decision makers and DSS developers, there is still a need for trained programmers to implement the requirements of the decision maker. However, the availability of software developers is sparse [dOli+21]. This sparsity is even expected to rise in the short term due to the increased demand for digitization as a consequence of the COVID-19 pandemic [BM21]. Again, this either leads to a delay in DSS availability or a cost increase which may render the customizations of an off-the-shelf financially impossible for a decision maker. Developer availability also limits the scalability

of DSS development, i.e., providing each decision maker with a tailored DSS is unrealistic.

In summary, the dependency on DSS developers introduces the risk of miscommunication and developer unavailability. Both have been established as factors in general software development that lead to rework or even cancellation of projects if not properly managed [MGM19]. In other words with a focus on decision support, the dependency on DSS developers bears the risk of a DSS not being available in time, or not being properly aligned with the preferred decision-making process, thereby also leading to ineffective and inefficient decision making.

Challenge 3 – *Lack of Coordination*: There is no coordinated and continuous exchange between stakeholders regarding requirements, service offerings, and best practices for decision support

Although the role of a decision maker is pivotal in decision making, decision support also requires the aid of other stakeholders. In particular, the previous challenges already explained the role of a DSS developer who designs, implements, and distributes a DSS. As evident from the example for the individuality of decision-making processes in Section 1.2, there is also the role of a consultant (cf. “external agency”) who provides additional input data or other (non-software) services for decision making. Furthermore, domain experts such as researchers continuously develop new approaches to decision support, which are subsequently refined and included in a DSS by developers (in case of technical advancements) or adopted by decision makers (in case of procedural advancements). These roles must exchange information, knowledge, or artifacts among each other as summarized in Fig. 1.2.

As already elaborated for the previous challenge, the absence of a common terminology is one barrier to the efficient and effective communication and coordination between stakeholders. Another barrier is the absence of a central platform that allows stakeholders to advertise requirements for decision support, decision support systems, and technological and procedural innovations. For example, even if a DSS exists that optimally addresses the requirements for decision support of a decision maker, the absence of a central repository that documents available decision support systems makes it difficult for the decision maker to discover the DSS in the first place. This introduces the risk of settling on a suboptimal DSS recommending suboptimal decisions. Similarly, the lack of coordination between DSS developers requires each developer to implement the same fundamental decision support functionality instead of only focusing on supporting novel requirements that are not yet addressed (cf. “coopetition” instead of “competition” [Bou+15]). This results in inefficient DSS development. Furthermore, when DSS developers and domain experts only know about the requirements for decision support of a handful of selected decision makers, it is difficult for them to identify changing trends in the business environment and to develop and innovate accordingly.

1.4 Objective: Assisted DSS Creation by Decision Makers

The previous sections show that decision makers have very individual requirements for decision support that can often only be partially addressed by an off-the-shelf DSS. This observation can be attributed to a lack of DSS customization and interoperability without relying on DSS developers, and a general lack of coordination between decision makers, DSS developers, and other involved stakeholders. The result is likely inefficiency in decision making due to delayed availability of decision recommendations, ineffective decision making due to suboptimal decision recommendations, or both. Such decision making ultimately results in a competitive disadvantage for the decision maker's company and should be avoided.

This thesis aims to address the aforementioned challenges in tailored DSS development using concepts from software engineering. There are two current trends in software engineering that – at first glance – may provide a solution to the observed challenges in providing each decision maker with a tailored DSS. The first potential trend is the use of a *service-oriented architecture (SOA)*, which has received increased attention over the last years [Nik+20]. In a SOA, multiple smaller, interoperable software services can be combined into a holistic software application. Transferring this approach to the domain of decision support, a selection of multiple *decision support services* with reusable decision support functionality could be composed into a tailored DSS that optimally addresses the requirements for decision support of an individual decision maker. However, although existing approaches fundamentally show the technical feasibility of service-oriented DSS development, a SOA is nevertheless targeted at DSS developers and not at non-programmers, and therefore does not resolve the dependency on DSS developers (cf. second challenge in previous Section 1.3). Although some approaches exist to automatically select and compose services into a software application to reduce the required amount of human effort, the currently ongoing research in the Collaborative Research Centre 901 – “On-The-Fly Computing”² shows that this is a non-trivial technical challenge that also comes with other potential disadvantages such as limited trust in the automatically assembled DSS. Furthermore, a SOA per se does not facilitate multi-stakeholder coordination across multiple enterprises as described throughout the third challenge in Section 1.3.

The second potentially applicable software engineering trend is the use of a *low-code development platform* to reduce the dependency on software developers. Such a platform combines the ideas of model-driven software engineering with rapid application design and automated (cloud) deployment to enable the development of software applications by non-programmers [Sah+20]. Many platforms furthermore include a marketplace that provides reusable functionality for developing the business logic of an application by recombination. In

² Project website: <https://sfb901.uni-paderborn.de/>

doing so, low-code can potentially reduce developer dependency by enabling the customization (and perhaps even the complete development) of a tailored DSS by non-programmers and decision makers in particular, thereby addressing the first and second challenge of the previous Section 1.3. However, the application model representing a tailored DSS must be of high quality, i.e., model elements must not only address the requirements for decision support of a decision maker, but must also be interoperable and consider best practices for decision making in an application domain for effective and efficient decision support. Therefore, decision makers should be assisted during low-code development of a tailored DSS to ensure the correctness of the resulting DSS. Unfortunately, this assistance often needs to be specific to decision making in the concrete application domain of the decision maker and is therefore lacking in most development platforms, which focus on the development of general-purpose applications with limited assistance in general. The required exchange between stakeholders can also seldom be fulfilled by a marketplace alone (cf. third challenge in Section 1.3).

In summary, while both service-oriented and low-code development has the potential to address the overall challenge of providing each decision maker with a tailored DSS, both concepts have individual downsides that prevent them from completely solving this challenge. Nevertheless, combining the strengths of both concepts could potentially enable the development of a tailored DSS for (and by) each decision maker. However, the applicability and technical feasibility of aggregating the two concepts in the context of DSS development has not yet been assessed. The uncertainty regarding the applicability and technical feasibility of merging SOA and low-code for tailored DSS development in light of the previously described challenges motivates the following overall research question:

RQ₀: How can an assisted model-driven approach be applied to DSS development so that decision makers and other non-programmers can compose a tailored DSS from reusable decision support services in a multi-stakeholder context?

The answer to this research question should be an IT-based solution that makes the flexibility of a SOA accessible to non-programmers (with a focus on decision makers) by using appropriate model-based abstractions as employed by low-code, but focusing on the development of decision support systems instead of general-purpose applications. With a reference to “multi-stakeholder context”, the research question furthermore acknowledges the need for cooperation and consequently coordination between decision makers, DSS developers, and other service providers and domain experts. For additional conciseness, the overall research question RQ₀ of this thesis can be further subdivided into four smaller, enabling research questions:

RQ₁: What characteristics of provided and required decision support should be documented to enable effective service composition in an application domain?

A description of the decision support provided by a decision support service is required to document the prerequisites and effects of service invocation, which determine the compatibility between services during service composition. At the same time, decision makers need to document their requirements for decision support for assisted service selection or stakeholder coordination. The application of service orientation to DSS development requires the documentation of decision support characteristics that are not considered by traditional service description languages, e.g., optimization goals, which may be specific to a concrete application domain. In addition to identifying decision support characteristics that need documentation, the answer to this research question should also describe how to document these characteristics.

RQ₂: Which modeling approach is a suitable abstraction that enables the composition of decision support services by decision makers and other non-programmers as well as the generation of a tailored DSS?

A suitable modeling approach is necessary to abstract from the technical details of service composition in a SOA, thereby enabling DSS development by decision makers and other non-programmers during design time. At the same time, the abstraction must be chosen such that it is still possible to generate a DSS from the model that invokes the selected services during execution time when the DSS is being used by a decision maker.

RQ₃: How can decision makers and other non-programmers be assisted during the composition of decision support services during DSS development?

As elaborated before, misalignments in required and provided decision support or fundamental flaws in the service composition representing a DSS can result in delayed or suboptimal decision recommendations and consequently must be avoided. Potential mistakes in the service composition should therefore be detected at design time to avoid any runtime errors at execution time. Such assistance can furthermore ensure the implementation of best practices and other innovations contributed by domain experts.

RQ₄: How can the coordination and continuous exchange between stakeholders be supported from a technical perspective?

As previously explained, DSS development requires the involvement of multiple stakeholders, particularly decision makers, DSS developers, and domain experts. Consequently, there should be a low-barrier solution that facilitates coordination between stakeholders. This includes making stakeholders' offerings and knowledge available to decision makers, while at the same time reflecting the requirements of decision makers for future innovations. Supporting information exchange from a technical perspective is a prerequisite before discussing additional cultural and organizational approaches to encourage stakeholder coordination.

1.5 Research Approach: Design Science

The overall goal of the thesis is to identify and design IT-based solution concepts that address the previously described challenges in providing decision makers with tailored decision support systems. In alignment with this goal, the research underlying the thesis was conducted using a *design science research (DSR)* approach. DSR focuses on the creation and evaluation of IT artifacts that are intended to provide utility by supporting the design and implementation of information systems that solve organizational problems [Hev+04]. These IT artifacts can address the organizational problem either in a “unique or innovative” or a “more effective or efficient” way compared to existing approaches (if any exist) [Hev+04].

At the core of DSR is the design process as “a sequence of expert activities that produces an innovative product (i.e., the design artifact)” [Hev+04]. The design process used for this thesis aligns with the *design science research methodology* for information systems research by Peffers et al. [Pef+07], which consists of the following six phases:

- **Phase 1 – Identify Problem & Motivate:** The problem is defined and justified by describing its importance and the value of a solution.
- **Phase 2 – Define Objectives of a Solution:** Quantitative and/or qualitative objectives of a solution to the problem are inferred from the previous problem definition and knowledge of currently existing solutions (if any exist).
- **Phase 3 – Design & Development:** Design research artifacts are (conceptually) designed and subsequently created as “any designed object in which a research contribution is embedded in the design” [Pef+07].
- **Phase 4 – Demonstration:** Artifacts are used in solving one or more instances of the problem to demonstrate the fundamental applicability of the artifact.
- **Phase 5 – Evaluation:** The results of the previous demonstration are compared with the solution objectives established in the second phase to decide how well the artifact addresses the problem. Depending on the results of the evaluation, researchers may iterate back to improve the design of the artifact.
- **Phase 6 – Communication:** Lastly, the problem and the designed artifact(s) as a solution to the problem are communicated.

Of the multiple entry points into this process described by Peffers et al. [Pef+07], this thesis uses the *problem-centered initiation*, i.e., it starts with problem identification. Hevner

et al. note that the “build-and-evaluate loop [from phase 3 to 5 and back to 3] is typically iterated a number of times before the final design artifact is generated” [Hev+04]. This thesis nevertheless describes the insights gathered throughout a single iteration only as finding the most optimal solution can never be guaranteed in a given time frame assuming a sufficiently complex solution space (cf. [Hev+04]). However, a single iteration is not a limitation per se, as Hevner et al. [Hev+04] note that a “design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve”. This is fundamentally achievable within the first iteration of the design process. As Peffers et al. [Pef+07] point out, further improvements may be left to subsequent projects.

1.6 Research Overview: Contributions and Publications

Following the design science research approach, multiple intertwined concepts were derived to address the motivated challenge of providing each decision maker with a tailored DSS.

Contribution 1: Decision Support Ecosystems

The first and fundamental contribution of this thesis is the concept of a *decision support ecosystem (DSE)* to enforce and facilitate exchange between stakeholders, i.e., decision makers, DSS developers, domain experts and other service providers. For this purpose, a DSE provides a central platform where these stakeholders can interact to create a tailored DSS for each decision maker. A high-level overview of a DSE is shown in Fig. 1.3: Instead of developing a DSS as a tightly coupled and holistic application, a *DSS developer* makes multiple reusable and interoperable *decision support services* available via a *service repository*. Selected services from the repository are then assembled into a holistic *tailored DSS* by a *decision maker* based on their individual requirements for decision support using the *DSS composition application*. Knowledge of *domain experts* is used to provide feedback to the decision maker regarding potential improvements of the service composition. After the tailored DSS is generated from the service composition, the decision maker can interact with it like with a traditional DSS, i.e., by providing input data and receiving output data containing decision recommendations.

With respect to the overall DSE concept, the contribution of this thesis lies in the definition of the DSE concept in relation to existing ecosystem concepts, and furthermore, in the design of a reference architecture for a technical platform that enables the implementation of a DSE for a specific application domain. The DSE concept and associated platform architecture primarily address RQ₄ regarding the technical facilitation of stakeholder coordination. Additionally, the DSE concept provides the frame in which the other contributions are embedded. The

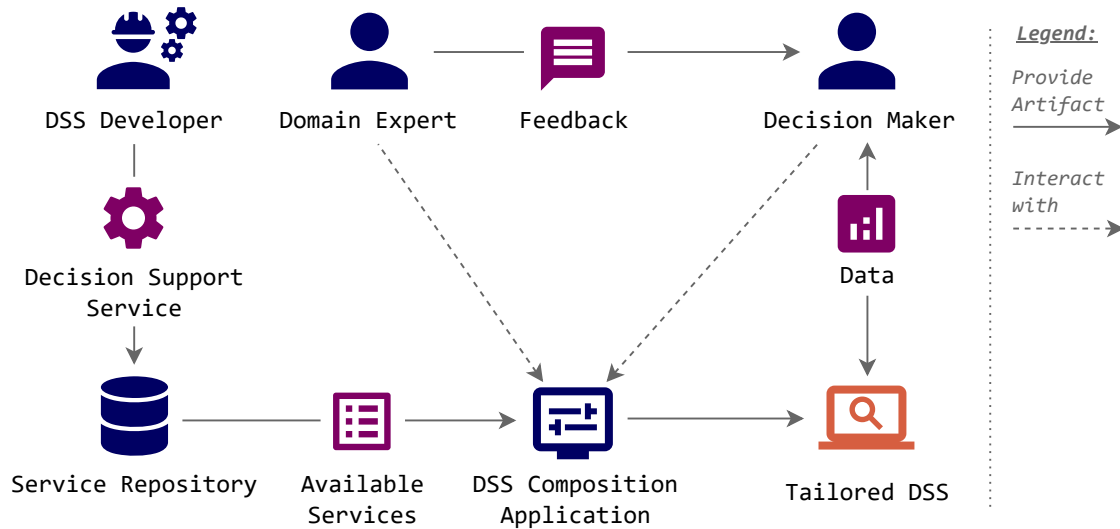


Figure 1.3: Overview of the *Decision Support Ecosystem* concept

DSE concept is primarily described in [Kir21; KWE22a]. The latter paper motivated the *EURO Working Group on Decision Support Systems (EWG-DSS)* to award the author with the “EWG-DSS 2022 Young Researcher of the Year” award.

Contribution 2: Description Language for Decision Support

The thesis contributes a description language for decision support that serves two purposes: First, the language enables domain experts to capture decision support characteristics of an application domain to ensure the portability of the overall solution design across domains. Second, the language enables service providers to document the decision support functionality provided by their decision support services (cf. “Service Repository” in Fig. 1.3) and decision makers to document their requirements for decision support. The created documentations serve as a foundation for the subsequent service composition. Besides functional characteristics, the language also supports capturing non-functional characteristics such as resource consumption or service quality. Documentations are machine-readable for automated processing across the DSE. The description language addresses RQ₁ and is primarily described in [KWE22b].

Contribution 3: Process-Driven Decision Support Systems

From a technical perspective, the success of the previously introduced DSE concept largely depends on the availability of a suitable service composition format that can be used by decision makers and other non-programmers to describe a tailored DSS. For this purposes, the thesis contributes the concept of a *process-driven decision support system (PD-DSS)*, which

uses a process model to describe a tailored DSS as a service composition (cf. Fig. 1.4).

On the one hand, a process model is suitable for this purpose as it can generally be used by non-programmers without extensive upfront training. On the other hand, a process model documents the conditional execution of software-based activities and the data exchange between them, which enables automated execution of the process model. As a result, after the *process model* was created by the decision maker via the *DSS composition application*, the execution of the model can be delegated to a *process engine* to automatically orchestrate the invocation and data exchange between the selected services in the background. This happens transparently for decision makers, i.e., they are prompted for input data and are provided with output data that passes through a generic graphical user interface that decision makers perceive as the tailored DSS. Decision makers however do not need to care about which parts of the data are processed and generated by which services in the service composition.

With the PD-DSS concept, the thesis contributes a modeling notation for the description of a tailored DSS as a composition of software services by decision makers, thereby primarily addressing RQ₂. The contribution is outlined in [KGE22; KWE22b].

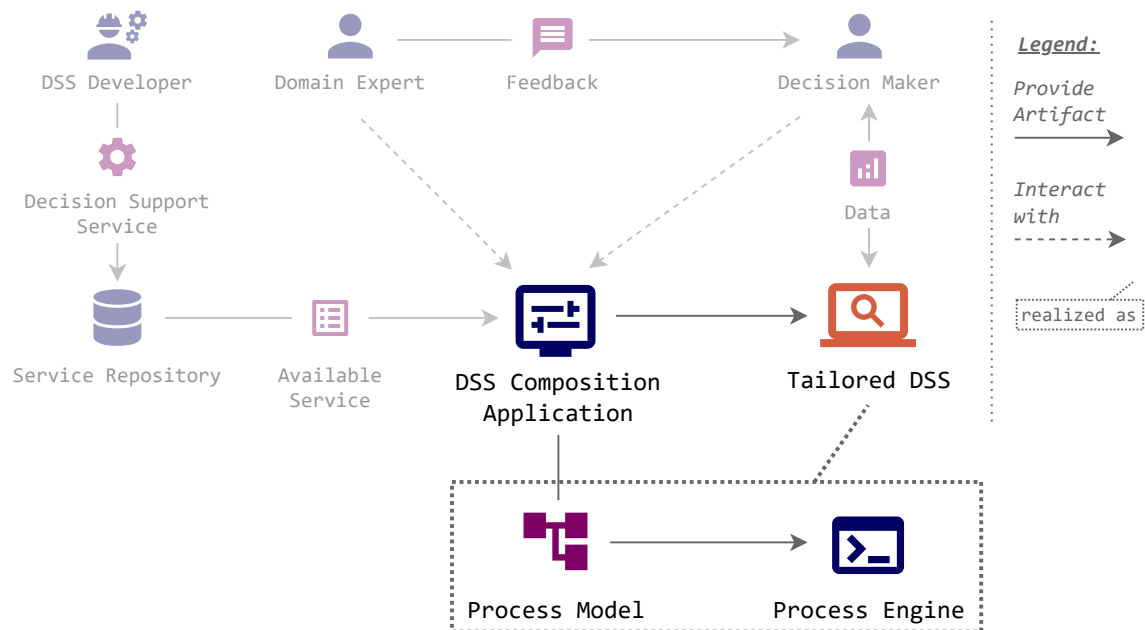


Figure 1.4: Overview of the *Process-Driven DSS* concept

Contribution 4: PD-DSS Modeling Assistance

As indicated by the explanation of the DSE concept, domain experts can guide decision makers in the creation of a tailored DSS by providing feedback regarding a service composition, either

to eliminate accidental mistakes or to include (novel) best practices that were previously not considered by the decision maker. However, manual supervision comes with the downside of creating a dependency on domain experts, thereby limiting the scalability of the ecosystem. As a countermeasure, the thesis describes an *assistance system* that automatically enforces the domain knowledge of domain experts, thereby allowing domain experts to focus on the identification of novel best practices for future innovations (cf. Fig. 1.5).

The thesis contributes multiple validation approaches for process models representing a tailored DSS to improve the quality of the associated service composition. The validation approaches check the process model for improvements regarding functionality, data exchange, or service selections. The assistance system thereby supports decision makers during the process-based creation of a tailored DSS and therefore primarily addresses RQ₃. The contribution is foremost described in [KGE22; KE22].

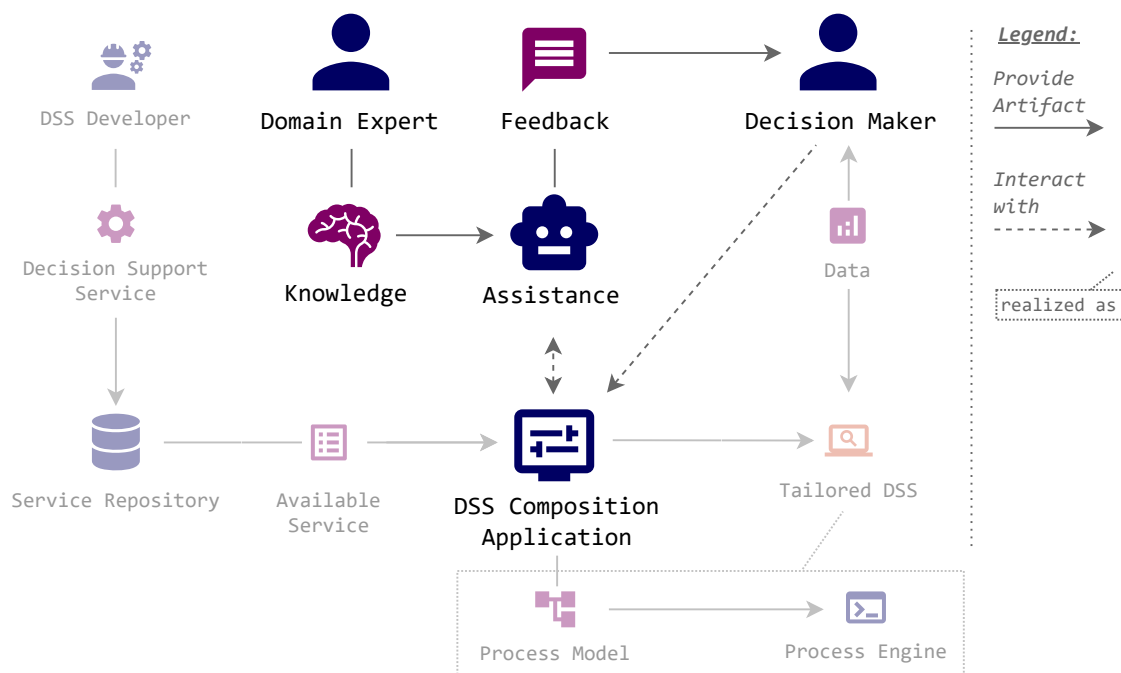


Figure 1.5: Overview of the *DSS Modeling Assistance* concept

Additional Contributions

Figure 1.6 summarizes the previously described contributions and associated publications. Additional contributions are made within the following publications:

[Kir+21] describes the fundamental functionality required for a DSS to assist decision

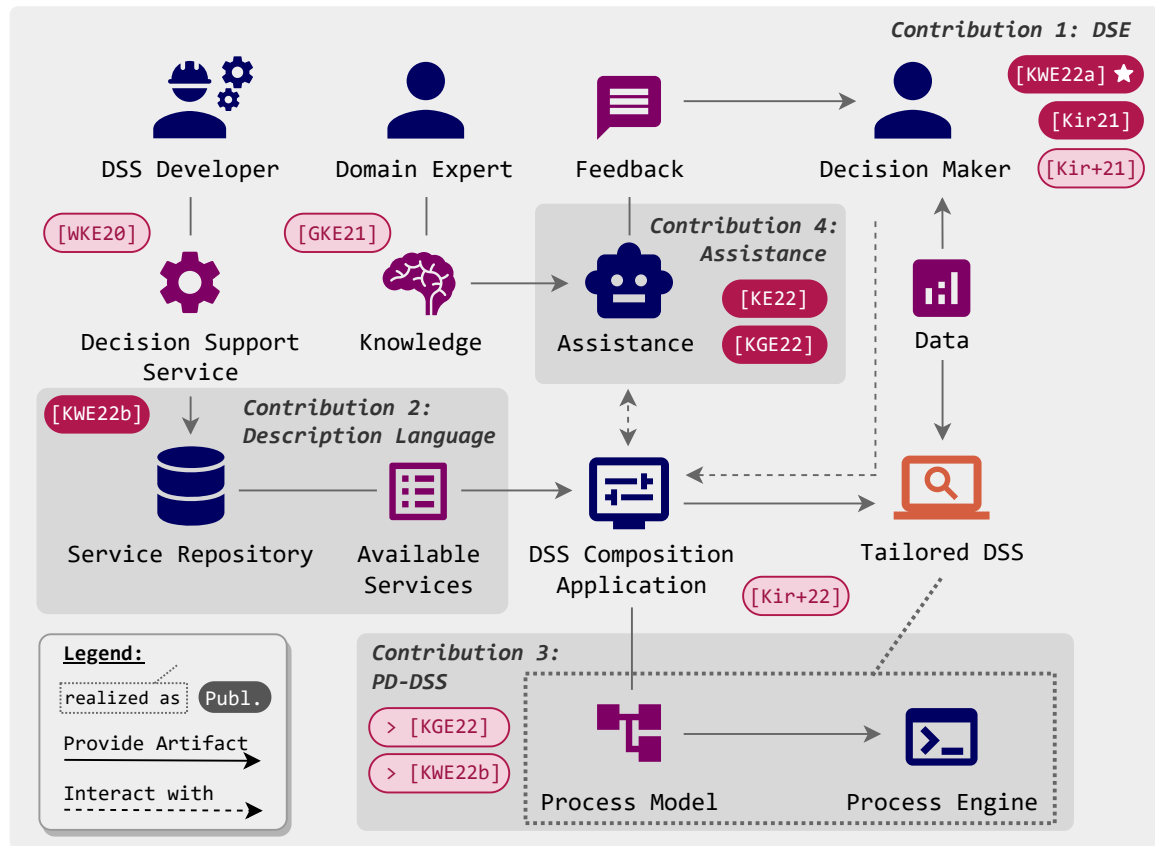


Figure 1.6: Overview of contributions and associated publications

makers in regional energy distribution network planning. As an outlook, the publication motivates the need to provide network planners with tailored decision support systems. Throughout this thesis, knowledge of the distribution network planning domain is used to derive requirements for the previously described contributions and to demonstrate the application of the contributions in a concrete domain.

[Kir+22] describes a development method for low-code applications considering the situational execution of development activities based on application requirements, features of the development platform, and skills of the development team. Throughout this thesis, the knowledge gathered as part of the publication is used to showcase the opportunities and challenges of current low-code development platforms for tailored DSS development.

[WKE20] describes a documentation-driven approach to create web-accessible services for command-line applications. The approach itself can be classified as low-code and enables the potential to quickly make existing DSS functionality available as a web service, thereby ensuring an extensive service repository and consequently a flourishing ecosystem with enough alternative services to provide each decision maker with a tailored DSS.

[GKE21] describes an approach to support business model developers with expert knowledge that was previously formalized by domain experts. This thesis generalizes some of the presented concepts for business modeling to decision support for arbitrary domains.

1.7 Thesis Structure

The thesis is structured into three parts that are closely aligned with the phases of the design science research approach previously described in Section 1.5.

Part I: Problem Identification and Solution Objectives

The first part of the thesis focuses on the identification, definition, and illustration of the research problem addressed in the thesis. The insights presented in the current Chapter 1 are extended in Chapter 2, which provides additional details and background knowledge regarding the individuality of decision making. Chapter 3 then transforms these insights into concrete requirements for a solution artifact that solves the challenges of tailored DSS development and discusses existing state-of-the-art and other related work with respect to these requirements.

Part II: Solution Design and Development

The second part of the thesis focuses on the design, development and initial demonstration of artifacts to address the previously determined solution objectives. Each previously described contribution is described in its own chapter, i.e., the decision support ecosystem concept in Chapter 4, the description language for decision support in Chapter 5, the process-driven development of decision support systems in Chapter 6, and the assistance system for service composition in Chapter 7. Each chapter can be viewed as a nested application of the design science research approach, i.e., each chapter explains the partial problem addressed as well as more fine-grained requirements before presenting the design, development, demonstration, and evaluation of the associated contribution.

Part III: Solution Discussion

The third part of the thesis describes the evaluation of the solution design in Chapter 8 based on the application of the solution design in a case study derived from practical insights gathered throughout the *FlexiEnergy* research project. Finally, Chapter 9 summarizes the thesis with respect to the initially defined research question(s) and gives an outlook on future work.

Individuality of Decision Making

This chapter discusses the individuality of decision making that creates a demand for tailored decision support systems. For this purpose, Section 2.1 first gives an overview of the domain of energy distribution network planning, which is used throughout the thesis for illustrative purposes. Next, Section 2.2 introduces fundamental concepts of decision making with a focus on decision processes. Section 2.3 presents situational factors to which decision makers are exposed, thereby influencing the composition of their decision processes and introducing variance that must be accounted for when providing decision support. Lastly, Section 2.4 describes the persistent influence of VUCA business environments on the previously described situational factors and the need for (tailored) decision support. Section 2.5 summarizes the key insights presented throughout the chapter.

2.1 Context: Energy Distribution Network Planning

The decision making that is required as part of energy distribution network planning illustrates the necessity of tailored decision support systems. This section introduces fundamental concepts of energy distribution networks, in particular their characteristics and maintenance, as a foundation for illustrative examples used throughout the remainder of the thesis. The section summarizes insights from published literature that were corroborated and extended by domain experts throughout *FlexiEnergy*, a transdisciplinary research project with partners from academia and industry for the design and development of a DSS to assist decision makers in energy distribution network planning (cf. project overview in [Kir+21]).

Role and Characteristics of Energy Distribution Networks

An *energy system* comprises “all components related to the production, conversion, delivery and use of energy” [All+14]. Energy distribution networks are essential for energy delivery, i.e., the transportation of energy from one location to another. For simplicity, the following explanations focus on electricity as an energy carrier, but analogous concepts fundamentally apply to gas and heat as well. Figure 2.1 shows a (simplified) schematic view of an energy system with a focus on electricity delivery. It is subsequently explained in more detail.

Electricity delivery is implemented using two kinds of networks. Long-range transportation of electricity is handled by *transmission networks* that operate using high-voltage. These transmission networks are used to transport the electricity generated by power plants closer to more industrial or residential areas where the electricity is consumed. *Substations* then connect the transmission network to the regional distribution network by transforming the electricity from high-voltage to medium-voltage (represented by the overlapping rings in Fig. 2.1). The *distribution network* – sometimes also referred to as *distribution grid* – is

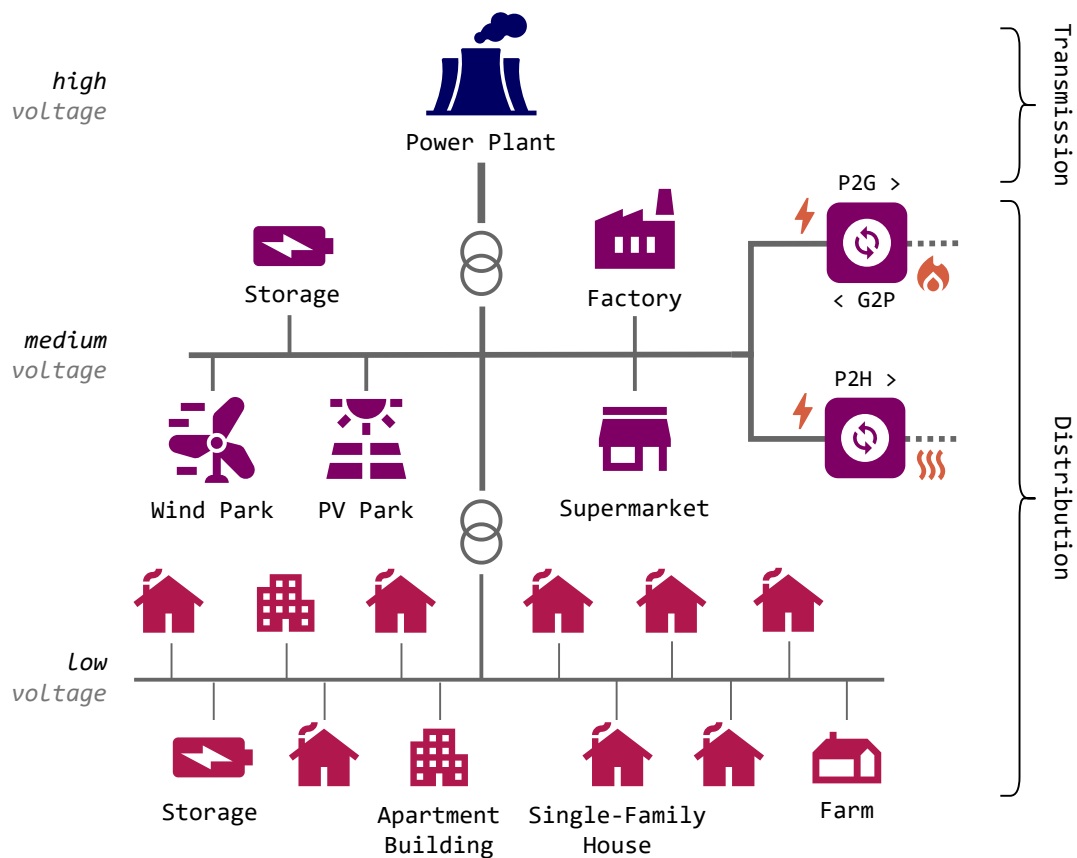


Figure 2.1: Simplified schematic visualization of an energy system with a focus on electricity

responsible for delivering the electricity to the final consumers. On the medium-voltage level are large-scale consumers such as industrial plants or large trade buildings. The final transportation of electricity to residential consumers happens after another transformation from the medium- to the low-voltage level, again using substations. [SAL18, Sect. 1.1]

The *power flow* describes how electricity travels through the interconnected networks. Traditionally, electricity delivery is designed for the unidirectional power flow described in the previous paragraph, i.e., generated electricity is fed into the network at the highest voltage level, and energy consumers are connected to the network at the lower voltage levels. However, this situation is increasingly changing as part of the energy transition towards renewable energy sources. Now, energy generation also occurs at low-voltage levels with PV systems that end consumers install on their houses. This leads to a bidirectional power flow in distribution networks, which often requires the adaptation of existing networks since these were almost exclusively designed for a unidirectional power flow. An alternative to network redesign is the use of flexibility measures to cope with the variable electricity generation from renewable energy sources. Such flexibility may be provided in the form of electricity storage for the (local) intake and release of electricity, *demand side management (DSM)* to encourage or defer the electricity demand of consumers to weaken peak demands, or *sector coupling*, i.e., the transformation from one type of energy to another using technologies such as *Power-to-Gas (P2G)* or *Power-to-Heat (P2H)*, or *Gas-To-Power (G2P)*. [ASL18, Ch. 1]

Maintenance of Distribution Networks

Regular network upgrades are necessary to ensure that a distribution network is sufficiently equipped to sustain the demand that is put on the network by energy producers and energy consumers at any given point in time. This is especially difficult for electricity distribution networks as the network itself cannot store any electricity and therefore cannot absorb any excess electricity generation or consumption. Although the aforementioned flexibility measures can be utilized for meeting consumer demands, the underlying technologies are mostly still in an early stage of development. The correct functioning of the distribution network is therefore primarily ensured by outfitting the network with properly dimensioned assets. A *network asset* is any technical component of the network such as a transformer in a substation or cables connecting electricity producers, substations, and electricity consumers. The composition of a network from network assets is referred to as the *network topology*.

A *distribution network operator (DNO)*, i.e., the company responsible for maintaining a distribution network, must decide on the network's design to meet consumer demands. Based on the aforementioned explanations, this is mostly equivalent to deciding when, where, and how to (re)place each network asset. This decision making is referred to as

the investment/reinforcement/expansion planning of distribution networks, or *distribution network planning (DNP)* for short. This long-term planning of the distribution network’s design (considering the next 20–40 years due to the lifespan of network assets [Sil16, p. 47]) is in contrast to the short-term planning of its operation, which for example considers the ad-hoc deactivation of a wind turbine if the network is temporarily overloaded. However, since the aforementioned flexibility measures are effective during network operation but need to be considered during the planning of network investments, the lines between planning the investments and operations of distribution networks are blurring (cf. [Xia+16]).

The upcoming Section 2.2 provides additional details of distribution network planning by using it as an illustrative example for fundamental definitions from the domain of decision making. The subsequent Section 2.3 provides an example that illustrates the individuality in distribution network planning processes among different distribution network operators.

2.2 Decision Making Foundations

This thesis understands *decision making* as the “process of evaluating and choosing courses of action” [BL18, p. 3]. The remainder of this subsection explains the decision making entities shown in Fig. 2.2 using examples from energy distribution network planning.

2.2.1 Decision Problems

A *decision problem* describes the need for decision making. It can be formulated as a question. For the example of energy distribution network planning, the decision problem could be characterized by the question “When, where, and how do network assets need to be (re)placed?”. The need for decision making arises from the fact that there are multiple *decision alternatives*, which are potentially a solution to the decision problem [Jes20, p. 62]. The alternatives are mutually exclusive [Sàn22]. For distribution network planning, decision alternatives correspond to different *investment plans* that describe the (re)placement of network assets (potentially in combination with the selection of flexibility measures to cope with electricity generation from renewable energy sources) [Sil16, p. 133]. A *decision* then corresponds to the decision alternative that is selected by the decision maker, e.g., the planning engineer of the distribution network operator [Sil16, p. 133].

It is possible to identify different types of decision problems. This thesis summarizes the classifications discussed by Sánchez-Marrè [Sàn22] as shown in Fig. 2.3: An *operational/programmed decision problem* is a repetitive, frequently occurring issue with limited complexity that can be supported using best practices identified from experience. Decision

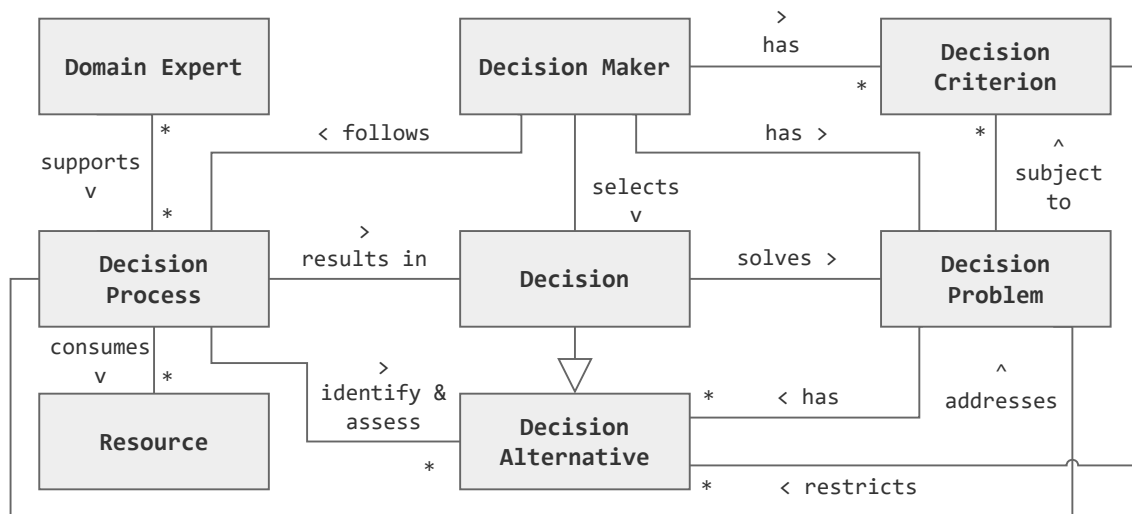


Figure 2.2: Decision making entities visualized as a UML class diagram (default multiplicity: 1)

problems of this type may occur daily. For distribution network planning, an example of such a decision problem may be how to connect a new residential consumer to the network. On the contrary, a *strategic/non-programmed decision problem* is a novel and unique issue with high complexity that only occurs very rarely in a company's lifespan. As a result, it is often not possible to identify reusable decision support. In distribution network planning, an example of such a decision problem is the decision whether to transform the distribution network into a *smart grid* for increased data collection and better demand side management to deal with peak demands. Operational/programmed and strategic/non-programmed decision problems can be understood as the respective ends of a continuous scale. A *tactical/semi-programmed decision problem* between the ends of the scale is a decision problem that this thesis assumes to happen somewhat frequently with medium complexity that it warrants the development of a reusable decision support system, yet so infrequently that it is necessary to regularly update the decision support according to interim changes in the business environment. These decision problems can be assumed to occur most frequently, as the lines between decision problem categories are blurring [Rol21], which is also corroborated by the combined investment and operational planning of distribution networks described in the previous Section 2.1. In distribution network planning, an example of a tactical/semi-programmed decision problem is the aforementioned identification of an investment plan for larger parts of the network. While this happens regularly, e.g., every year, the uncertainty due to the long foresight requires updating the planning procedure to account for interim changes in the business environment such as the previously described bidirectional power flow or the emergence of novel approaches such as flexibility measures to offset peak demands. [Sàn22]

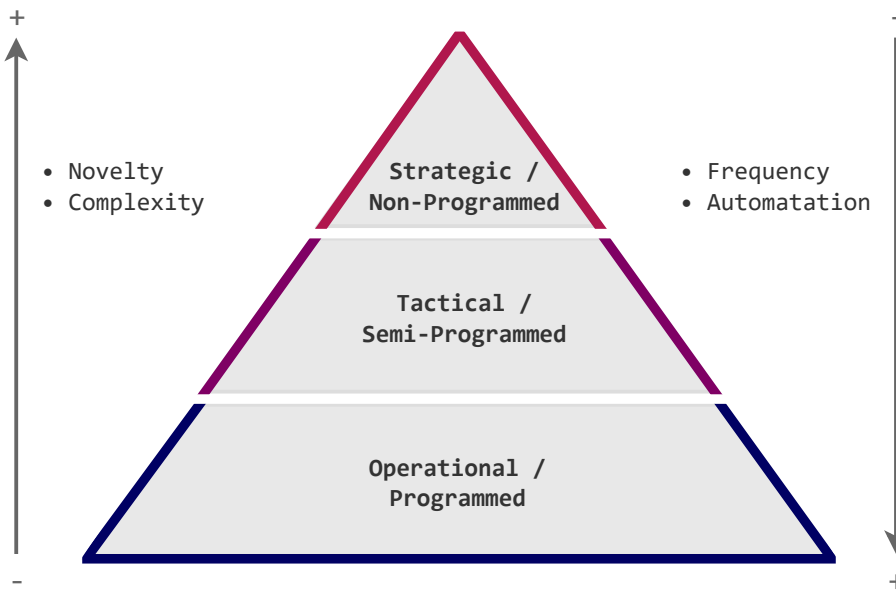


Figure 2.3: Classification of decision problems (adapted from [Sàn22])

2.2.2 Decision Quality

Each decision problem is subject to one or multiple *decision criteria* used to evaluate decision alternatives, thereby restricting the pool of decision alternatives available for selection as a decision [Jes20, p. 65]. One can differentiate between *decision objectives* that seek to minimize or maximize a certain characteristic of a decision alternative, and *decision constraints* that restrict a certain characteristic to a range of acceptable values. In distribution network planning, examples of optimization objectives are the goal to minimize the one-time investment costs (*CAPEX*) and/or recurring operational costs (*OPEX*) [Xia+16]. An example of a constraint is *(n-1)-reliability* to ensure that any given network asset of a type, e.g., a transformer, may fail without affecting the functionality of the network (cf. [Sil16, p. 69 & Fig. 5.4]).

The existence of decision criteria allows the definition of *decision validity* and *decision optimality*. A *valid decision* is a decision that meets all constraints, while an *invalid decision* fails at least one constraint. An *optimal decision* implies that no other decision alternative from the pool of available decision alternatives can be selected as the decision without negatively impacting the characteristic affected by an objective. Analogously, a *suboptimal decision* implies that deciding on another decision alternative would improve the characteristic affected by an optimization objective. Unless stated otherwise, this thesis assumes an optimal decision to also be valid, and an invalid decision to be suboptimal.

The evaluation of decision alternatives generally happens under *uncertainty*, i.e., the evaluation is based on the knowledge available at the time of decision making. However, due

to uncertain future developments, a seemingly optimal decision at the time of decision making may lead to an unwanted outcome or effect after the decision is implemented. Analogously, a seemingly bad decision might lead to the desired outcome or effect. However, it is generally assumed that “consistently making good decisions will lead to more good outcomes than otherwise” [Par+13, p. 3]. This is also referred to as *decision rationality*. [Sàn22]

2.2.3 Decision Process

The *decision process* describes the sequence of activities performed during decision making to identify, assess and select decision alternatives. It is primarily a “process of information transformation” [Zar13, p. 1] which consumes human and material resources [Zar13, p. 37] while being supported by tools [SWM16, p. 26]. The decision process followed by a decision maker significantly influences the selection of a decision alternative and consequently the quality of a decision (cf. [DS96; Zar13, p. 12; SWM16, p. 27; EC07; Jes20, p. 137]).

Roles in Decision Making

Multiple roles participate in the enactment of a decision process. Parnell et al. [Par+13] specifically describe the four roles of decision maker, stakeholder, subject matter expert, and decision analyst. The *decision maker* is the person with the “responsibility and authority to make organizational decisions” [Par+13, p. 20]. Decision makers define the decision problem and ultimately select a decision alternative at the end of the decision process. In distribution network planning, the decision maker is the leading planning engineer of the distribution network operator [Sil16, p. 133]. A *decision analyst* supports decision makers by providing them with “credible, understandable, and timely insights” (cf. [Par+13, pp. 2, 19]). For this purpose, they use social skills to extract “credible substantive knowledge” [Par+13, p. 90] related to the decision from *subject matter experts* and subsequently transform and extend the knowledge using technical skills such as data analysis. For this reason, the decision process can be viewed as a socio-technical process [Par+13, p. 3]. In distribution network planning, an example of a decision analyst is a data scientist who produces forecasts for future consumer demands. An example of a subject matter expert is a manufacturer of PV systems who may provide useful insights regarding the future market share and capabilities of these systems. A *stakeholder* in the context of decision making is “an individual or organization with a significant interest in a decision under consideration” [Par+13, p. 20].

Stakeholders are not further considered throughout the thesis as they either initiate decision making (e.g., the CEO of a distribution network operator) or are affected by the resulting decision (e.g., energy consumers), but are not directly participating in the decision process

themselves. For simplicity, the thesis furthermore assumes that decision makers are sufficiently technically adept to operate a provided DSS themselves, which implies that every decision maker is also a data analyst. Additionally, decision makers must have some subject matter knowledge in order to properly define the decision problem and the associated decision objectives. Nevertheless, additional data analysts and subject matter experts can be involved to support the decision process. The roles of data analyst and subject matter expert are subsequently aggregated into the role of a *domain expert* when it is irrelevant whether a person contributes subject matter knowledge or data analysis skills. All described roles are non-exclusive, i.e., an individual can assume multiple roles. Although the previous explanations use singular forms to refer to the roles, each role can be assumed by multiple individuals. However, for simplicity, the subsequent explanations assume the involvement of only a single decision maker in the decision process, otherwise additional consensus building between multiple decision makers is needed when deciding on a decision alternative.

Fundamental Phases of Decision Making

Various multi-phase blueprints for decision processes have been suggested in the literature with varying levels of detail (cf. [Sàn22]). Although the remainder of this thesis usually discusses decision processes on the most detailed activity level, an overview of fundamental decision process phases is given below to establish the need for certain activities described in later parts of the thesis. The decision process structure of Mintzberg, Raisinghani, and Theoret [MRT76] is used for this illustrative purpose since it represents a compromise between detail and brevity. It is visualized in Fig. 2.4 and subsequently explained. Although the shown process is sequential, it is possible to execute parts of the process iteratively, or even jump pack to a previous (sub-)phase [Sàn22].

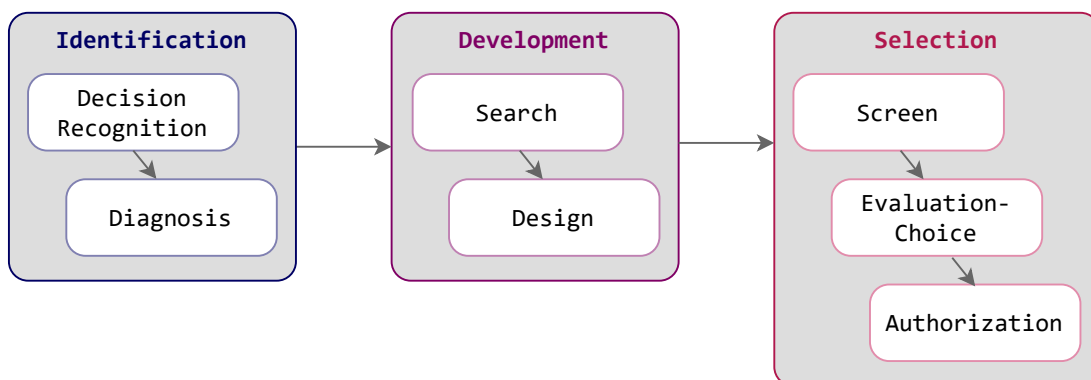


Figure 2.4: Decision process structure according to [MRT76]

Identification During the *identification* phase, the context for decision making is established. First, the need for decision making is identified throughout *decision recognition*. The need for decision making can be triggered by influences originating from the business environment, or after the completion of a time interval (e.g., in the case of yearly distribution network planning). Next, the decision problem including decision objectives is characterized throughout *diagnosis*. This includes gathering information and knowledge of the business environment that is also used throughout subsequent phases. [MRT76]

Development During the *development* phase, potential decision alternatives are collected. Already existing and potentially reusable decision alternatives are identified throughout an initial *search*. Additional decision alternatives are constructed during *design*, either from scratch or by modification of existing alternatives uncovered throughout the search. The development phase consumes the most resources throughout (strategic) decision making. [MRT76]

Selection During the *selection* phase, the final decision alternative is selected. If a large quantity of decision alternatives was identified throughout the development phase preventing a detailed evaluation of each alternative, a first superficial evaluation is conducted as part of an initial *screen*. The remaining alternatives are evaluated throughout *evaluation-choice*, usually using analytical approaches, with one alternative ultimately being selected as the decision. In case the process is mostly performed by a designated decision analyst, the agreement of the decision maker is obtained throughout *authorization*. [MRT76]

2.2.4 The Meta-Level of Decision Making

Although the previous subsection provides a fundamental structure for decision processes, each explained (sub-)phase still needs to be broken down into one or multiple concrete activities to describe the steps required to implement the phase. Each *decision activity* is a unit of work that produces an artifact supporting the identification, development, or selection phase of the decision process – either by constituting the final output of the phase or by consumption in a subsequent activity of the phase. In distribution network planning, an example of an activity may be producing a forecast for the future development of electric vehicle market shares subsequently used for forecasting consumer demands that the distribution network must sustain. Technically, the activity of “producing a forecast” can be broken down further, e.g., into collecting sources forecasting the development of market shares, extracting the data, and documenting it in a machine-readable format.

In many cases, multiple (sequences of) activities are possible to obtain a desired artifact. This can for example be attributed to the availability of multiple approaches and software tools to produce the artifact [BL18, pp. 174–175]. For example, Burmeister and Schryen [BS23]

list multiple approaches to optimize the topology of an energy distribution network. The existence of activity alternatives gives rise to the concept of “*meta decision making*” which is considered with deciding on the composition of a decision process, i.e., which activities to execute and which tools to use for assistance during activity execution. Meta decision making is often conducted experimentally by exploring and comparing multiple variations of a decision process [MKP21]. The upcoming section presents the different factors that influence decision makers to choose some activities and tools over others.

2.3 Situational Factors of Decision Making

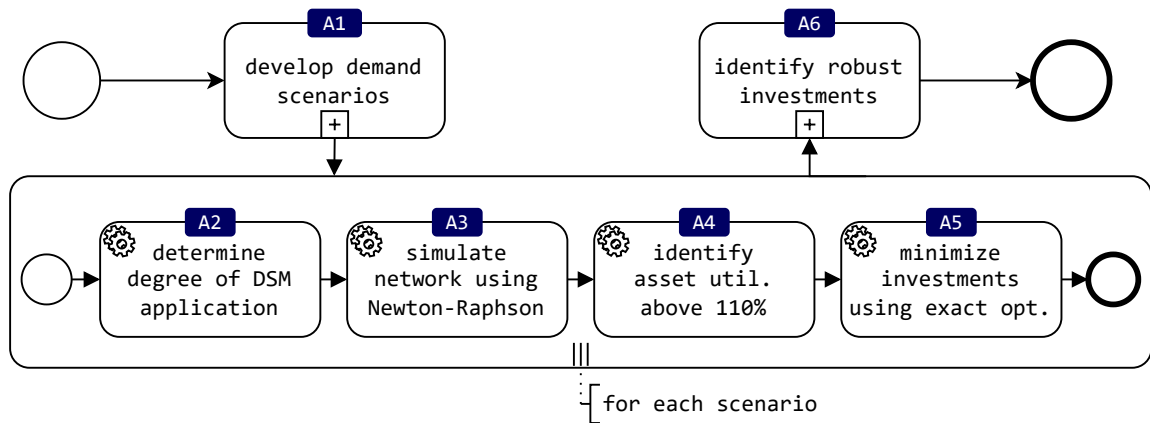
The previous section emphasizes the importance of the decision process on decision quality. In this context, the meta-level of decision making focuses on selecting or defining an appropriate decision process for the decision problem at hand. This is necessary since decision makers are subject to different *decision situations* with differences in for example decision alternatives and decision criteria [Jes20, p. 62]. These differences originate from characteristics of the company, e.g., company size, or the surrounding business environment, e.g., due to political, economic, social, technical, ecological, or legal influences (cf. [Jes20, pp. 23, 28, 155]). This section presents multiple *situational factors* that influence the constitution of a decision maker’s decision process. Since the factors are illustrated using the example of distribution network planning, the section first provides some additional details on the distribution network planning process before subsequently introducing and explaining the available factors.

Context: The Distribution Network Planning Process

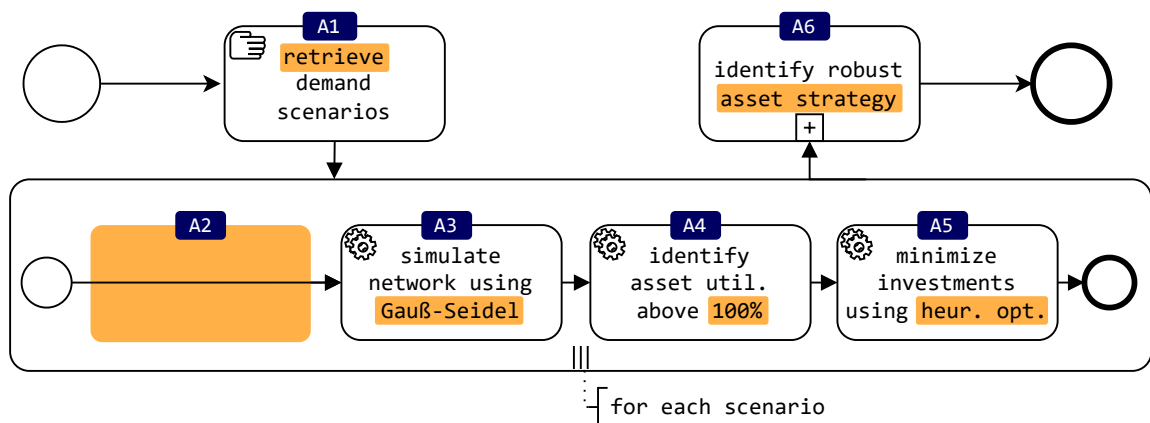
Sillaber describes an iterative planning process for distribution network planning (cf. [Sil16, Fig. 4.1]) that can be loosely mapped to the fundamental decision process phases presented in the previous section. During the *Identification* phase, the necessary information for distribution network planning is produced. This includes information about the current structure and components of the network, but also *demand forecasting*, i.e., the generation of multiple estimates for how the demands of energy producers and consumers will/could progress over the next years to account for uncertainty (cf. [Yan+18]). During the *Development* phase, different alternative investment plans are generated and subsequently evaluated during the *Selection* phase until a final decision on an investment plan is made. Due to the increasing importance of IT assistance during distribution network planning, design and choice of decision alternatives are often combined in the form of an optimization that simultaneously generates and evaluates investment plans in search of an optimal decision alternative (cf. [Sil16, p. 162; EY19]).

Six Situational Factors for Decision Process Design

Six situational factors that influence the composition of a decision process can be derived from the fundamental decision-making entities previously presented in Fig. 2.2. Factors include the availability of domain experts or other resources such as data, the availability of decision alternatives as well as methods and criteria for their evaluation, and the preferences and skills of a decision maker. Each factor is subsequently explained with respect to the two decision processes shown in Fig. 2.5. Both processes aim to identify cost-minimizing distribution network asset investments. The processes are documented using the *Business Process Model and Notation (BPMN)* [Obj13]. Although the processes in Fig. 2.5 (deliberately) look very similar at first glance, they contain significant differences as highlighted by the yellow markup in Fig. 2.5b. Each activity in the process represents the effects of one situational factor.



(a) First decision process



(b) Second decision process

Figure 2.5: Comparison of two similar decision processes for distribution network planning

- **Availability of Domain Experts – Activity A1**

The availability of domain experts to support the decision process may affect the composition of the process. For example, some activities can only be executed with the direct involvement of experts, e.g., an interview to determine the future technical developments of network assets. Other activities may require data produced by experts. An example of this factor is illustrated with activity A1 in Fig. 2.5. In both processes, the first activity is concerned with the production of multiple demand scenarios. Each scenario describes a potential future development for consumer demands. In Fig. 2.5a, these demand scenarios are developed as part of multiple activities that are aggregated into a subprocess. In Fig. 2.5b, the decision maker already has access to the demand scenarios and just needs to retrieve them, e.g., because an external agency was previously commissioned with scenario development before the planning started.

- **Availability of Decision Alternative Prerequisites – Activity A2**

Decision alternatives potentially have prerequisites before being available for selection as a decision. These prerequisites are not fulfilled for every decision maker. For the example of energy distribution network planning, demand side management (DSM) to reduce peaks in customer demands can only be used in active distribution networks where the DNO can monitor and control consumer demands in real time. Therefore, determining the degree of DSM application is only included in Fig. 2.5a, as the DNO associated with the process of Fig. 2.5b does not meet the prerequisites for DSM.

- **Methods for Evaluating Decision Alternatives – Activity A3**

The business environment may require the decision maker to use certain methods for the assessment of decision alternatives, e.g., due to regulatory constraints. Throughout distribution network planning, it is often necessary to compute the power flow throughout a network, which is also referred to as *network simulation*. This computation can be done with a variety of algorithms encapsulating different physical models or mathematical approaches. In activity A3 of Fig. 2.5a, the network simulation is performed using the Newton-Raphson algorithm. However, another DNO may be subject to different technical regulations and therefore use the Gauß-Seidel algorithm instead in Fig. 2.5b.

- **Underlying Decision Criteria – Activity A4**

Decision makers can prioritize and parametrize decision criteria differently based on the environment they (or their company) operate in. A parametrization is primarily relevant for constraints to define a threshold that a characteristic of a decision alternative cannot

exceed or fall below. In the exemplary distribution network planning process of Fig. 2.5, this affects activity A4 where network assets are identified that require a replacement. For this purpose, a capacity-to-load ratio prescribes the maximum worst-case utilization threshold for assets (cf. [Yan+18]). The threshold amounts to 110% in Fig. 2.5a and to 100% in Fig. 2.5b, e.g., due to a different risk tolerance.

- **Availability of Resources – Activity A5**

Decision makers have different access to resources consumed throughout the enactment of the decision process. A lack of resources may prevent decision makers to include activities in the decision process, i.e., either leaving them out completely or substituting them with alternative activities when approximate results are better than leaving out the activity entirely. An example of the latter is shown in Fig. 2.5. Cost-minimizing network asset replacements are identified throughout activity A5. For this purpose, the first decision maker uses an *exact optimization* in Fig. 2.5a which is based on integer linear programming. While this approach guarantees optimal results, it is also very complex and time-consuming (cf. [Sil16, p. 161]). Due to time constraints, the second decision maker instead uses a *heuristic optimization* in Fig. 2.5b, which is faster to execute but cannot necessarily guarantee to find an optimal solution.

- **Preferences and Skills of the Decision Maker – Activity A6**

Decision processes of different decision makers do not only vary because of influences in the business environment but also due to different preferences and skills of decision makers themselves. A lack of skills may prevent a decision maker from including a certain activity in the decision process, e.g., because the activity requires specific data analysis knowledge that the decision maker does not have. The preferences of an individual decision maker result in the decision maker either favoring or avoiding a certain activity. In the examples shown in Fig. 2.5, the trust of a decision maker in the software tools used for activity implementation influences activity A6 for identifying a robust decision among the multiple evaluated scenarios. In Fig. 2.5a, the first DNO might have sufficient trust in the quality of the previous activities and accepts a detailed investment plan that describes when, where, and how to replace individual network assets. On the contrary in Fig. 2.5b, the DNO may doubt the quality of the recommendations computed by the heuristic optimization and prefer the recommendation of a robust asset strategy that provides more generic guidelines regarding asset replacement.

The explanations of the factors “Availability of Domain Experts” and “Availability of Resources” indicate that a strict differentiation between factors is not always possible. For example, the

lack of expert availability (which requires the generation of demand scenarios by the decision maker in Fig. 2.5b) can be interpreted as a lack of resources when data is viewed as a resource consumed during process enactment. Similarly, the availability of resources can influence the selection of methods for identification and assessment of decision alternatives, as the heuristic optimization is selected as part of Fig. 2.5b due to a lack of time which would be needed for utilizing an exact optimization. Despite this potential overlap in the example, the description of each factor's origin in the previous explanations illustrates the necessity of each factor.

Combinatorial Diversity and Implications for Decision Support

The two exemplary processes shown in Fig. 2.5 showcase rather “opposite” decision processes with a difference in each activity for illustrative purposes. In practice, other decision makers would likely prefer combinations of these processes. For example, among the approximately 850 DNOs in Germany alone [Sta22], a decision maker could generally prefer the first process from activities A2 to A6, but may want to execute it with already available demand scenarios as considered by A1 of the second process. The vast possibility of combinations from the shown (and otherwise imaginable) activities illustrate the nearly impossible challenge of DSS developers in Chapter 1 to anticipate all customization options that are necessary for a decision maker to adapt the DSS to the individual decision process. Unfortunately, this is a prerequisite for creating a tailored DSS when following a holistic approach to DSS development.

Furthermore, the examples processes show that a simple parametrization of activities is not always sufficient to implement the required customization. While it is arguably possible to use the same software functionality to support activity A4 with a value of 110% and 100% respectively, this does not work for activities A1 or A6 as they would require a replacement of more complex subprocesses.

2.4 VUCA and Its Effect on Situational Decision Making

This section defines the VUCA concept introduced in Chapter 1 with respect to decision making and explains its reinforcing effect on the situational factors for decision processes presented in the previous section, thereby driving decision making individuality.

The Effects of VUCA on Decision Making Individuality

Chapter 1 already briefly introduced VUCA as an acronym for volatility, uncertainty, complexity, and ambiguity. The subsequent VUCA explanations are based on the understanding of a business environment as a collection of interdependent *influencing factors* whose future

developments determine the optimality of decision alternatives [MBD20]. For example, energy distribution network investments are highly dependent on the demands of consumers and producers, with demands depending (among others) on the market share of electric vehicles, which in turn depends on factors such as incentives for purchasing an electric vehicle.

Volatility refers to instability in the business environment due to frequent change [BL14]. This can lead to the emergence of new influencing factors or interdependencies between them, e.g., less replacement of network assets during distribution network planning due to emerging demand side management. Considering the previously described situational factors, the changes in the business environment introduced by volatility may introduce novel methods for the identification and evaluation of decision alternatives. However, since not every decision maker's business environment is necessarily impacted by the changes at the same rate, volatility increases decision making individuality.

Uncertainty is the lack of predictability for the development of future influencing factor values. This can be attributed to a lack of knowledge [BL14] or to the constant change induced by volatility that prevents the applicability of forecasting based on historical values [Law13]. Uncertainty is often addressed by considering more information throughout the decision process. This requires additional expert input or (data) resources that must be approximated in case they are not immediately available, thereby further driving decision making individuality.

Complexity refers to the vast number of influencing factors and their interdependencies that must be considered during decision making [BL14; Mac+16] (cf. the initial influencing factors example). The complexity in business environments inevitably also leads to an increased complexity of decision processes to properly portray the characteristics of the business environment. As a result, the probability to encounter the aforementioned situational factors and therefore individuality in decision processes is significantly increased.

Ambiguity implies that the cause-effect relationships between influencing factors are not fully understood [BL14]. For example, financial incentives are expected to increase the market share of electric vehicles, but it is unclear by which percentage. Analogous to the aforementioned explanations for complexity, the increasing ambiguity requires more comprehensive decision processes, which implies a higher probability to encounter individuality due to the situational factors. Furthermore, ambiguity also promotes the necessity of meta decision making, as potentially multiple decision process variants must be defined and tested to determine which variant most closely captures the characteristics of the business environment.

The Necessity of Decision Support in VUCA Environments

The increasing reliance on a DSS can be attributed to the fact that these systems are well-suited to address the characteristics of VUCA business environments. The data-driven and

simulation-based approach of a DSS creates an environment for experimentation where the implications of different algorithms and future developments can be contrasted to account for uncertainty and ambiguity [KQ08; Kow17]. Given enough computational resources and appropriate user interfaces for data input and output, the encapsulated algorithms can be arbitrarily complex without overextending the information processing capabilities of decision makers. Lastly, there is a continuous influx of new DSS research and development to ensure the availability of up-to-date decision support functionality to account for volatility in business environments – for example, the number of yearly publications on decision support systems indexed by *Web of Science* has nearly doubled in the last 10 years (922 results in 2011 vs. 1,763 results in 2021 when searching all fields for “decision support system”).

Nevertheless, these advantages of using a DSS only become accessible if the DSS is appropriate, i.e., the decision process supported by the DSS accounts for the situational factors the decision maker using the DSS is exposed to. In particular, the DSS must have inherent flexibility to provide the necessary means for experimentation, meta decision making, and for the integration of novel decision support approaches.

2.5 Key Takeaways

Decision makers follow a decision process to identify decision alternatives, evaluate them based on various decision criteria, and ultimately select an optimal decision. The composition of a decision process depends on the individual decision situation of a decision maker, which is characterized by six situational factors. In particular, decision makers are influenced by characteristics of the decision problem, available resources, and individual skills and preferences to choose certain decision activities over others when developing their decision processes. Furthermore, VUCA drives decision-making individuality across all domains and increases the demand for tailored decision support.

The quality of a decision is determined by the appropriateness of the underlying decision process. In the context of the aforementioned situational factors, anticipating an appropriate decision process for all decision makers is close to impossible for DSS developers when designing a DSS. Consequently, existing customization options of off-the-shelf decision support systems are often insufficient for effective and efficient decision making. This observation raises the question of which requirements must be fulfilled to enable the development of decision support systems that are tailored to the individual decision situations of decision makers. The next part of the thesis presents such requirements for tailored DSS development and the advantages and shortcomings of existing approaches with respect to these requirements.

Requirements and Related Work

This chapter presents requirements for an approach to provide each decision maker with a tailored decision support system (Section 3.1). The potential benefits of service-oriented computing and low-code development in addressing the requirements are discussed (Section 3.2) and related work is evaluated with respect to the established requirements (Section 3.3). The evaluation results are subsequently summarized, and a research gap is highlighted (Section 3.4), which is addressed with the solution design presented in the next part of the thesis.

3.1 Requirements

The requirements R_x for an approach to develop tailored decision support systems presented in this section are derived from the explanations provided throughout the previous Chapters 1 and 2. By extension, the requirements are derived from insights gathered throughout the research project *FlexiEnergy* (cf. [Kir+21]), supplemented with additional knowledge published in academic literature. The goal of the requirements is to ensure the effectiveness and efficiency of each tailored DSS and the associated development approach itself.

Requirement R1 – Situativity

The fundamental motivation behind the development of a tailored DSS is the alignment of the decision support functionality provided by the DSS with the requirements for decision support of an individual decision maker. In this context, six situational factors that influence the constitution of a decision process (and consequently the required decision support functionality)

are presented in Section 2.3. These factors must also be supported by an approach for tailored DSS development to ensure DSS effectiveness. The factors are subsequently summarized as:

Requirement R1.1 – Situational Decision Problems

The constitution of the decision problem influences the required decision support functionality. This includes the characterizing problem question, decision criteria, i.e., optimization objectives or constraints, and prerequisites of decision alternatives.

Requirement R1.2 – Situational Resources

The availability of resources influences the decision process and consequently the required decision support functionality. Resources include data, execution infrastructure (or funds to rent them), input from subject matter experts, and available computation time to identify a decision recommendation.

Requirement R1.3 – Situational Competences

In addition to regulatory requirements, preferences may influence the required decision support functionality. This includes preferences of a decision maker's organization, e.g., concerning service level agreements, or preferences and skills of decision makers themselves, e.g., for data visualization.

Requirement R2 – Process-Oriented

As evident from Chapter 2, decision making is a complex process consisting of multiple activities, each potentially requiring the use of a different software artifact providing the necessary decision support functionality. This process perspective of decision making should also be represented in the approach for tailored DSS development to support the mapping from decision making activities to *decision support artifacts* (i.e., software and data). This approach aligns with the mental model of decision makers and can increase development efficiency. In this context, the following sub-requirements can be derived:

Requirement R2.1 – Modularity

Decision support artifacts should be provided in a modular way such that they can be assigned to individual activities of the decision process, capturing and assisting the process as effectively as possible. This requires loose coupling and interoperability to ensure that different decision support artifacts work together.

Requirement R2.2 – Navigation

The resulting tailored DSS should support the decision maker in navigating throughout an agreed decision process, e.g., by evaluating conditions and choosing the appropriate

decision activities and associated decision support functionality. This reduces the cognitive burden of decision makers during decision making and increases decision-making efficiency. A predefined decision process aligns with (semi-)structured decision problems where a specific decision process is executed repeatedly (cf. Section 2.2).

Requirement R2.3 – Unified Execution Environment

The integration of multiple decision support artifacts in alignment with an individual decision process should be hidden from decision makers using the tailored DSS. In other words, decision makers should be under the impression that they are interacting with a traditional, holistic DSS instead of requiring them to switch between different software systems. This avoids context-switching and increases decision-making efficiency.

Requirement R3 – Variety

Creating a tailored DSS requires a sufficiently sized pool of modular decision support artifacts with software and data to choose from to effectively align the provided and required decision support. This results in the following sub-requirements:

Requirement R3.1 – Reusability

A large quantity of decision support functionality and data is already available. Encouraging the reuse of these existing artifacts for tailored DSS development can significantly increase the variety of decision support and thereby increase the chance of optimally addressing the requirements of an individual decision maker. Reusing existing decision support artifacts also improves development efficiency. Since the existing decision support functionality is likely implemented using heterogeneous software platforms, platform independence is advised to enable low-barrier reusability.

Requirement R3.2 – Extensibility

A lot of decision support functionality has already been implemented, but a lot of additional functionality will also be implemented in the next years (cf. the research trend discussed in Section 2.4). In view of the ongoing volatility in business environments, an approach for tailored DSS development must be extensible to account for advancements in decision support functionality to ensure future effectiveness.

Requirement R3.3 – Discoverability

When selecting decision support artifacts that effectively support an individual decision process, it is not only important that multiple decision support alternatives are available for selection, but the existence of alternatives must also be documented. Otherwise, a

lack of knowledge about alternatives might result in the suboptimal selection of decision support functionality, or an exhaustive discovery phase is needed that prevents the timely availability of a tailored DSS.

Requirement R4 – Suitability for Non-Programmers

As elaborated throughout Chapter 1, the dependency on DSS developers for extensive DSS customization is neither efficient nor effective due to limited developer availability and potential miscommunications. However, a lack of programming skills usually prevents decision makers from integrating available decision support artifacts on their own. To address this shortcoming, an approach for tailored DSS development should address the following sub-requirements:

Requirement R4.1 – Abstraction

Like any software application, a DSS is based on procedural code to obtain an executable software artifact. Since decision makers and other domain experts do not have the required programming skills, an abstraction is needed that maps the goal-based DSS specification usable by these non-programmers to executable application code.

Requirement R4.2 – Learnability

The approach for tailored DSS development should be easy to learn for non-programmers. Otherwise, if the approach were to require extensive upfront training, decision makers and domain experts could arguably be trained in software development instead.

Requirement R4.3 – Error Prevention

The detection of errors during DSS development is desirable regardless of whether the approach for tailored DSS development targets professional or non-programmers. However, error detection is especially critical when decision makers develop a tailored DSS themselves as the feedback loop between developers and decision makers is eliminated that would potentially have uncovered any errors before productively using the DSS. Errors can lead to ineffectiveness when the wrong decision support artifacts are selected and combined, or to inefficiency if the DSS specification is incomplete and must be fixed since the resulting DSS cannot be properly executed.

Requirement R5 – Collaboration

The increasing complexity of business environments requires cooperation between stakeholders since decision processes are often no longer constrained to decision support artifacts from a single organization. While the cooperation between directly competing companies is likely limited (although not unrealistic, see “coopetition” in Section 1.3), many companies have

suppliers or otherwise interact with other organizations and might benefit from using their decision support artifacts and experience during decision making. In this context, the following sub-requirements are relevant:

Requirement R5.1 – Common Terminology

For a successful exchange of decision support functionality, data, experiences and other insights between organizations, a common understanding of the application domain is necessary. Otherwise, misunderstandings can reduce efficiency and/or effectiveness during tailored DSS development.

Requirement R5.2 – Artifact Sharing

Organizations exchange decision support artifacts, i.e., decision support functionality or data, but potentially also other resources such as computing infrastructure. An approach for tailored DSS development should support the integration of these contributions across organizational boundaries, which likely implies a distributed environment.

Requirement R5.3 – Experience Sharing

In addition to “material” decision support artifacts that are potentially exchanged between organizations, it is also possible to share “immaterial” insights regarding the use of these artifacts such as best practices established through experience. Furthermore, experience sharing not only works from the solution perspective but also from the problem perspective as decision makers can communicate their requirements for decision support that may not yet be addressed by existing decision support artifacts.

Requirement R5.4 – Organizational Scalability

Depending on the complexity of the application domain, a sophisticated decision process may require the incorporation of decision support artifacts and experiences from a multitude of organizations. A collaborative approach to tailored DSS development should therefore consider organizational scalability such that many organizations can contribute to increasing the efficiency and effectiveness of decision making.

Requirement R6 – Compliance

Decision makers are subject to external influencing factors during decision making. This specifically includes regulatory constraints that may require decision makers to use particular decision support functionality over another. An approach for tailored DSS development should ensure that compliance with these influencing factors is given. This implies:

Requirement R6.1 – Transparency

A decision maker should be informed about the decision support functionality included in a tailored DSS. This is not only relevant for compliance, but can also increase trust in the recommendations computed by the DSS, which is desirable since decision makers often largely base their decisions on these recommendations.

Requirement R6.2 – Determinism

Given the same requirements for decision support, the development approach should result in the use of the same decision support artifacts. In addition to compliance, this is additionally motivated by the fact that a tailored DSS is specifically suited for semi-structured decision problems (cf. Section 2.2) where the effort invested to create a tailored DSS is justified by repeatedly using the tailored DSS afterwards.

Requirement R7 – Domain-Portability

Although requirements elicitation and solution design throughout this thesis primarily focus on insights from the domain of energy distribution network planning, an approach for tailored DSS development should apply to multiple application domains. Following requirement *R4 – Suitability for Non-Programmers*, configuration for supporting a new application domain should be favored over an instantiation or extension of its underlying architecture.

3.2 Related Trends in Software Development

Chapter 1 already presented service-oriented computing and low-code development as two (ongoing) trends in software development. Each trend exhibits characteristics that can potentially support tailored DSS development, but in its current state is not sufficient to address the lack of tailored decision support systems on its own. This section provides additional background information on service-oriented computing (Section 3.2.1) and low-code development (Section 3.2.1) and briefly discusses their potential benefits with respect to the requirements for tailored DSS development established in the preceding Section 3.1. The information furthermore supports the understanding of the subsequent discussion of related approaches in Section 3.3 and the explanation of the solution design in Part II.

3.2.1 Service-Oriented Computing

Papazoglou et al. define *service-oriented computing* as “the idea of assembling application components into a network of services that can be loosely coupled to create flexible,

dynamic business processes and agile applications that span organizations and computing platforms” [Pap+07]. Service-oriented computing requires a concrete service-oriented software architecture to implement the service-orientation “idea” for a concrete software application.

Service-Oriented Architectures

Historically, the output of a software development project is a *monolith* [JC19], i.e., a single executable artifact [Dra+17]. Monoliths come with disadvantages as described by Joseph and Chandrasekaran [JC19] and Dragoni et al. [Dra+17]. For a DSS, the most obstructive disadvantages include a lack of agility and evolvability preventing the adaptation of the DSS to the business environment, technology lock-in preventing the integration of the most suitable decision support functionality concerning a decision maker’s requirements for decision support, and a lack of reusability since partial functionality cannot be executed independently.

The concept of a service-oriented architecture emerged in the early to mid-nineties as a countermeasure to these and other disadvantages of monolithic applications [LL09] and is still an ongoing trend for software system modernization [Nik+20]. Laskey and Laskey define a *service-oriented architecture (SOA)* as “a paradigm for organizing and packaging units of functionality as distinct *services*, making them available across a network to be invoked via defined interfaces, and combining them into solutions to business problems” [LL09]. Additional definitions are summarized by Niknejad et al. [Nik+20]. Based on the definitions, the authors conclude that a SOA “promotes loose coupling, reusability, interoperability, agility, [and] efficiency” [Nik+20]. These benefits, in addition to the fact that network-based communication via interfaces enables services to be platform-independent [CDT17], address the challenges of monolithic (DSS) applications discussed in the previous paragraph, i.e., lack of agility, reusability, and technological openness. The benefits can be attributed to the fact that the concept of a SOA provides standardized mechanisms for the discovery and interaction between services [LL09], which are subsequently described in more detail.

Service Interfaces and Service Discovery A *service interface* documents the syntax and semantics of information exchange with a service [LL09]. This includes invocable functions with input and output formats of function parameters, constraints such as pre- or post-conditions that describe the (necessary) state of the environment before or after function invocation, and supported communication protocols (cf. [LL09]). Interfaces introduce a transparency similar to a *black box* [Nik+20], i.e., the implementation is decoupled from the interface [Dra+17] and unknown to the service consumer. This creates a loose coupling that allows service providers and service consumers to evolve independently [Val+09] and promotes *technical diversity*, i.e., allows them to use different technologies, languages, and platforms [Nik+20].

A SOA is ultimately only effective if it can match the needs of service consumers with the offerings of service providers (cf. [LL09]). *Service descriptions* consisting of service interfaces and additional non-functional metadata are therefore usually published in a *service registry* to achieve *service visibility* [LL09], i.e., to ensure that a provided service can be found by potential service consumers. This relation is summarized by the “publish-find-bind”-triangle [Sch+05] shown in Fig. 3.1: Service providers first publish a description of their offered service in the service registry. Service consumers specify their service requirements in a query to the service registry. Upon a successful match, the registry returns a suitable service description, which is then used by the service consumer to bind to the service provider and interact with its service.

Service Composition The implementation of a business process addressing a business problem usually requires the cooperation of multiple services. For example, an online shop may use three services to ship an order to a customer: a service for order management (to fetch the list of items to ship), a service for customer management (to obtain a shipping address), and a service for triggering the physical shipping. The “process of aggregating multiple services [...] to perform more complex functions” is referred to as *service composition* [She+14]. This activity can potentially be automated. The term “service composition” is also used to refer to the result of the aggregation activity, i.e., an artifact describing the cooperation of multiple services. A service composition is often described using a process model [CDT17; PL03]. The fact that a business process is implemented using multiple services is usually transparent to the stakeholders involved in the business process (cf. [LL09]).

Service Coordination Based on the previous explanations, the enactment of a business process requires the invocation of cooperating services as defined by a service composition

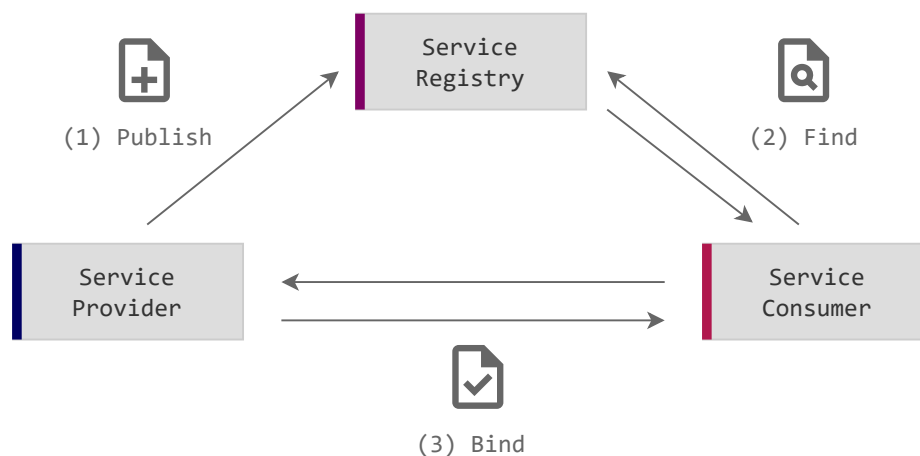


Figure 3.1: SOA Publish-Find-Bind triangle (adapted from [Sch+05])

(cf. [Dra+17]). Two fundamental approaches to service coordination have emerged, which are contrasted in Fig. 3.2. In the case of *service orchestration* shown in Fig. 3.2a, only a central Orchestrator service knows about the service composition implementing the business process (indicated in red). The Orchestrator service invokes other services and passes their response data to subsequent services as specified in the service composition. In the case of *service choreography* shown in Fig. 3.2b, there is no central entity coordinating the invocation of services. Instead, each service is responsible for acquiring the data needed for its execution, as indicated by its partial knowledge of a service composition. A service relying on and coordinating the functionality of other services is also referred to as a *higher-level service* [Dra+17], while a service without any dependencies to other services is referred to as an *atomic service* [MM18]. Message exchange between services is often based on HTTP.

In summary, the execution of a service composition is centralized for service orchestration and decentralized for service choreography [CDT17]. Service orchestration is therefore also summarized as “simple services and smart pipes” [CDT17], whereas service choreography corresponds to “smart services and simple pipes” (cf. [VKG17]). Here, “smart” and “simple” only refer to service coordination and not to the encapsulated business functionality.

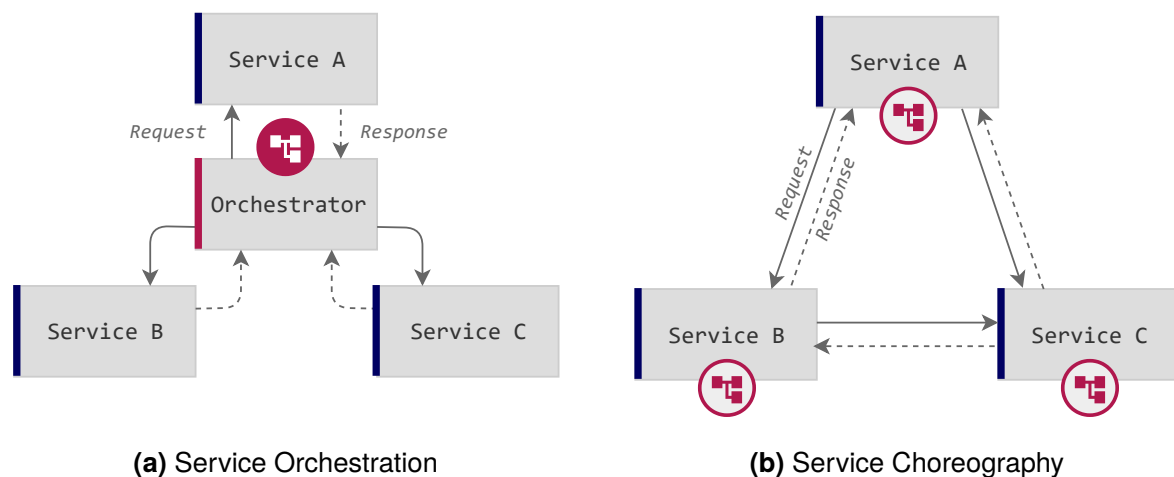


Figure 3.2: The two fundamental approaches to service coordination

SOA Implementation

In practice, web services and microservices have emerged as the two primary approaches to implementing a SOA. Both approaches are briefly introduced and compared below with a focus on characteristics that are referenced in the remainder of the thesis. A more comprehensive comparison is provided as Table 1 in the paper of Cerny, Donahoo, and Trnka [CDT17].

Origin and Adoption The technological standards underpinning web services emerged around the year 2000 to establish a global, enterprise-wide governance to improve the reuse of business functionality [CDT17]. Microservices emerged around the year 2012 throughout the industry to divide the IT landscape of a business into multiple single-purpose services with a bounded context, thereby improving the development and delivery of business functionality in cloud environments [JC19; AAE16; Dra+17]. Web services and microservices can be viewed as two subsequent iterations of SOA [Dra+17]. Nowadays, microservices are the dominant SOA implementation and experience massive adoption with many companies migrating from web services to microservices [Li+19; AAE16; CDT17].

Service Coordination Web services favor service orchestration over service choreography [CDT17]. For this purpose, services are often connected to a *enterprise service bus (ESB)*, which enables service discovery and assumes the orchestrator role shown in Fig. 3.2a, potentially converting between multiple heterogeneous communication formats [Sch+05]. Microservices favor service choreography [CDT17]. Sometimes a service mesh is used to establish communication channels between services [Li+19].

Communication Web services use a specific technology stack for SOA implementation, which is subsequently also referred to as the *WS-* technology stack*. It primarily includes the *Web Service Description Language (WSDL)* [W3C07b] for describing service functionality and *SOAP* [W3C07a] for message exchange between services. *Universal Description Discovery & Integration (UDDI)* [OAS04] can be used as a service registry, albeit a service registry is omitted by most organizations [Val+09]. Although the aforementioned technology stack provides a technically homogenous approach for service integration, it is often perceived as complex [CDT17]. Microservices use comparatively more lightweight communication mechanisms based on HTTP-APIs [JC19] such as *Representational State Transfer (REST)* [Fie00]. The use of service description languages such as *OpenAPI* [Ope21] is optional.

Independence Web services employ a “share-as-much-as-possible” concept to foster service reuse. This requires global, enterprise-wide governance. For example, web services may even share the same database, which emphasizes the need for a *canonical data model* to standardize exchanged business objects. A user interface to invoke service functionality is implemented as a holistic layer on top of the ESB. On the contrary, microservices strive for a “share-as-little-as-possible” concept (which usually leads to them being smaller [Dra+17]). Each microservice has its own data model and manages its own database. Microservices can even encapsulate their own user interface to form a *self-contained system* with complete governance over a part of the business domain. [CDT17]

3.2.2 Low-Code Development

Low-code development is another trend in software development that can potentially support tailored DSS development. This section introduces fundamental low-code terminology, compares low-code development to related approaches such as model-driven development or end-user programming, and describes the components of low-code development environments.

Low-Code Terminology

Defining low-code development is challenging since neither academia nor industry has agreed on a definition so far [Luo+21]. This can be attributed to the fact that publications on low-code only appeared fairly recently, with industry publications starting in 2014, and academic publications starting in 2018 [BF21]¹. This thesis defines low-code development based on its goals. In the following, *low-code development (LCD)* is therefore understood as an approach to enable non-programmers to develop and operate "complex software applications with little to no code" [Di +22] in order to increase productivity [BF21], i.e., to provide higher quality applications in a shorter amount of time (cf. [Mar+20]).

In the definition of low-code development, "*code*" can refer to the source code of a programming language [BF21], but also to more simple conditional expressions or algorithms [Let21]. The term *no-code development* can be used to distinguish approaches that require no code from those which require little code to create and deploy an application [Cab20]. Based on the above definition of low-code development, this thesis includes no-code development with low-code development unless explicitly stated otherwise. For uniformity, development using traditional programming languages is sometimes referred to as *high-code development*.

Low-code developers without programming background are also referred to as *citizen developers* [Sah+20]. This role is often assumed by domain experts who have sophisticated knowledge of the business issue which the developed application should address, but lack the programming skills of trained software engineers to implement such an application using traditional programming frameworks (cf. [FJF21]).

Low-Code Benefits and Related Approaches

The benefits of low-code development (i.e., providing higher-quality applications in a shorter time) come mainly from its relation to end-user development and model-driven development.

End-User Development By supporting the development of software applications by non-programmers, low-code development is similar to *end-user development (EUD)*, also referred

¹ Waszkowski [Was19] claims the low-code idea was introduced in 2011, but does not specify a source.

to as *end-user programming (EUP)* [Al +21], which provides “a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact” [Lie+06]. However, although low-code developers can be end users of the developed application, they can also be domain experts (i.e., citizen developers) without the intention to use the application themselves.

Enabling software development/adaptation by end users and domain experts comes with the advantage of “closing the gap” between stakeholders with knowledge about domain requirements and software developers who implement these requirements (cf. [Al +21]). This improves the alignment between business requirements and IT support [BF21] and results in applications with higher quality, i.e., applications that address business requirements more effectively. Additionally, the dependency on software developers is reduced, which could otherwise impose a “major obstacle” for digital transformation due to their limited availability [BF21]. By reducing the dependency on other stakeholders and communication overhead, low-code development also increases development efficiency [Kir+22].

Lieberman et al. [Lie+06] summarize three approaches with increasing complexity for EUD which can also be translated to low-code development, namely (1) *parametrization* of existing software components, (2) *composition* of multiple existing software components, and (3) *programming* of novel software components. Throughout low-code development, approaches (2) and (3) are primarily implemented using model-driven development [Sah+20].

Model-Driven Development Low-code development often utilizes abstractions and declarative programming to make software development accessible to non-programmers by applying (principles of) model-driven development [Sah+20]. *Model-driven development (MDD)* is “a software-engineering approach consisting of the application of models and model technologies to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying (making easier) and formalizing (standardizing, so that automation is possible) the various activities and tasks that comprise the software life cycle” [HT06]. In particular, *models*, i.e., “abstraction[s] over some (part of a) software product” [HT06], can be used for the specification, verification, and generation of software code [Di +22], all of which contributes to effective and efficient software development. In a low-code context, models are usually specified graphically and are used for code generation [Sah+20] and the automation of routine tasks [BF21], e.g., to provision cloud infrastructure as an execution environment for the application [Sah+20]. This allows low-code developers to focus more on the specification of the application’s business logic [Sah+20]. The automation can also increase agility throughout the development process [Al +21], which strikes an additional resemblance to *rapid application development (RAD)* [Di +22].

Novelty of Low-Code Development The previously described similarities between low-code development and existing approaches – MDD in particular – result in an (ongoing) academic debate regarding the novelty of the low-code concept. Some researchers such as Bock and Frank [BF21] and Cabot [Cab20] see limited technical contributions in the low-code concept, either going as far as denying the approach its eligibility as a scientific concept [BF21] or viewing it as a synonym for MDD [Cab20]. This critique is supported by the fact that many software tools, which nowadays market themselves as a platform for low-code development, already existed for multiple years before the low-code term was coined in 2014 [BF21].

On the contrary, researchers such as Sahay et al. [Sah+20] and Al Alamin et al. [Al +21] view MDD simply as a conceptual predecessor of low-code, or low-code as an embodiment of EUD. Di Ruscio et al. [Di +22] describe MDD and low-code development as separate concepts and highlight differences in development tools (platform- and cloud-based for LCD vs. Eclipse-based for MDD), target users (citizen developers for LCD vs. professional software developers for MDD), and target domains (business applications for LCD vs. systems engineering for MDD). Although the author of this thesis agrees that low-code development cannot simply be equated to MDD or EUD, the explanations throughout the remainder of the thesis often reference one of these paradigms or associated techniques due to their more precise definition compared to the fairly novel concept of low-code development.

Characteristics of Low-Code Development Platforms

A *low-code development platform (LCDP)* – sometimes also referred to as *low-code application platform* or simply *low-code platform* [BF21] – is an environment for the development of a *low-code application*, which is a software application developed using a low-code approach (cf. [Di +22]). LCDPs are often provided using a cloud-based *platform-as-a-service (PaaS)* approach [Sah+20]. This section describes characteristic features of LCDPs that can also be found in the solution presented throughout the remainder of the thesis. The subsequent descriptions are based on the LCDP components identified by Bock and Frank [BF21] and associated with the architectural layers described by Sahay et al. [Sah+20], both filtered and adapted based on the author's personal experience from LCDP workshops conducted throughout the *Pro-LowCode* research project (cf. [Kir+22]). An overview of LCDP architecture is given in Fig. 3.3, which is subsequently explained in more detail.

Application Layer The application layer provides all components that citizen developers directly interact with during the (graphical) specification of the software application [Sah+20].

- The *Data Modeling* component is used to define entities and their relations relevant to

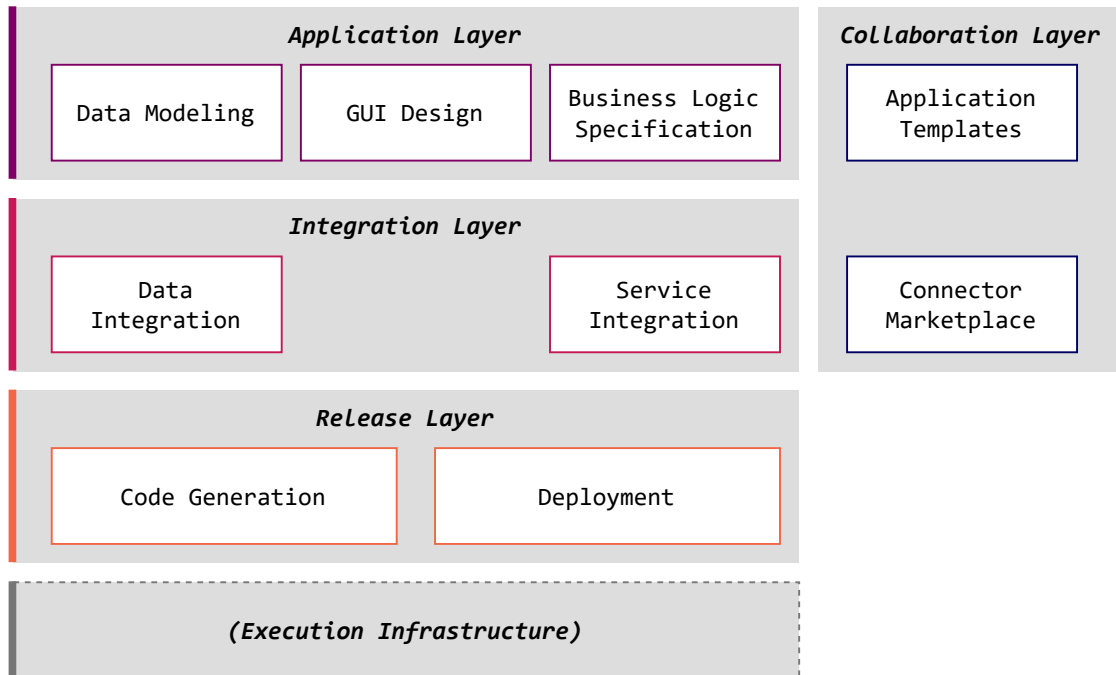


Figure 3.3: Fundamental architecture of LCDPs

the developed application as data structures. These data structures are the foundation for the other components in the layer and can support the automated generation of stubs.

- The *GUI Design* component is used to define (parts of) the application's user interface. This is usually achieved using *what-you-see-is-what-you-get (WYSIWYG)* by placing reusable UI widgets via drag&drop.
- The *Business Logic Specification* of LCDPs is used for the (often process-based) definition of the application's business logic. Business logic can be triggered by UI interactions, data manipulation events, or temporal triggers.

Integration Layer The integration layer provides connectors to establish information exchange between the low-code application and external applications similar to a SOA.

- The *Data Integration* component is used to define dependencies to external data sources such as hosted spreadsheets or a *database management system (DBMS)*. Most LCDPs reduce the need to integrate external data sources due to the existence of an automatically configured internal DBMS. The developed application can read from and/or write to connected data sources.

- The *Service Integration* component is used to define dependencies to external services that provide functionality for the specification of business logic not provided by the LCDP out-of-the-box.

Collaboration Layer The cross-cutting collaboration layer provides artifacts that support activities in the application and integration layer. These artifacts are usually made available via a central marketplace populated by the LCDP vendor as well as third-party developers.

- The *Template* component provides templates to bootstrap activities in the application layer, i.e., data modeling, GUI design, or business logic specification. It enables citizen developers to adapt (partially) complete applications to their individual needs.
- The *Connector* component provides integration adapters for data sources or services.

Deployment Layer The deployment layer is responsible for making the developed application available in the execution environment. Activities in these steps usually happen transparently to the citizen developer after triggering the deployment from the GUI of the LCDP.

- The *Code Generation* component is responsible for generating executable artifacts from the application specified in the application layer, considering the integrations specified throughout the integration layer.
- The *Deployment* component provisions the (usually cloud-based) execution environment and moves the previously generated executable artifacts into the environment. Some LCDPs support multiple execution environments, e.g., for production and testing.

3.3 Related Work

This section discusses related work and evaluates it with respect to the previously established requirements for providing decision makers with tailored decision support systems. The first three subsections discuss approaches that can be classified as an adaptive DSS (Section 3.3.1), a DSS generator (Section 3.3.2), or a service-oriented DSS (Section 3.3.3) as these concepts were previously identified to align most with the goal of this thesis. The presented approaches were identified as part of individual searches using the search engine *Web of Science* with the search string ("adaptive decision support system*" OR "adaptive DSS*") AND ("architecture" OR "design" OR "development" OR "framework") – using analogous search strings for the other concepts. The second part of the search string was added to ensure sufficient details for evaluation with respect to the previously established

requirements, to identify general-purpose approaches applicable to multiple domains, and to filter out approaches where the use of the paper's contributions as part of a DSS are only mentioned for future work. *Web of Science* was chosen over *Google Scholar* due to the possible limitation of the search on title, keywords, and abstract. The results were cross-checked with Google Scholar to account for papers not indexed by Web of Science albeit having a high amount of citations. Section 3.3.4 briefly discusses only distantly related research areas.

3.3.1 Adaptive Decision Support Systems

An *adaptive decision support system (ADSS)* is “a DSS that is able to automatically or manually modify some aspects of its structure, functionality, or interface to meet different needs in its users” [CY98]. An ADSS is characterized by some form of “activeness”, i.e., the system partially operates without user direction [Hol+08]. This also aligns with the use of “adaptive” over “adaptable” in its name, suggesting that changes are happening automatically due to internal state transitions instead of explicit user requests [Lie+06].

The primary motivation for using an ADSS is to increase decision-making effectiveness [FPV97; Hol+08] in volatile business environments [Hol+08; PS09] where stale or incomplete decision-making knowledge can result in suboptimal and consequently costly decisions [PS09]. Additionally, Holsapple et al. [Hol+08] also mention the complexity of business environments and the necessity to increase decision-making efficiency as motivation.

Detailed Evaluation of ADSS Approaches

This subsection explains the evaluation results for ADSS approaches summarized in Table 3.1 with respect to the requirements R_x presented in Section 3.1.

Holsapple et al. [Hol+08] present an ADSS concept that uses unsupervised learning to extend decision support knowledge. A suitable heuristic is initially selected based on the decision maker's objectives and the specified input data. This partially addresses *R1.1 – Situational Decision Problems* and *R1.2 – Situational Resources*, but without consideration of decision alternatives or other resources. Subsequently, the heuristic is updated or new heuristics are added to the decision support knowledge base using insights learned throughout the computation of a decision recommendation. This corresponds to an extension in the sense of *R3.2 – Extensibility*, however, manual extensibility is not explicitly mentioned. *R3.1 – Reusability* is only possible as long as the existing decision support functionality can be expressed as a heuristic that can be executed by one of the available search algorithms. The automated selection among available heuristics addresses *R3.3 – Discoverability*. The authors explicitly mention a lack of determinism and transparency introduced by utilizing learning

which fails to address *R6 – Compliance*. On the upside, this comes with the advantage of providing adaptivity without requiring additional input from the decision maker, which fulfills *R4 – Suitability for Non-Programmers*, albeit without a complete mechanism for error prevention as erroneous heuristic selections can only be identified over time via learning. The proposed ADSS concept is generic such that it can be instantiated in various application domains, however, the case study described in the paper suggests that architectural components must be specifically developed for the application domain. Consequently, *R7 – Domain-Portability* is partially fulfilled. No information is provided that suggests the fulfillment of requirement *R2 – Process Orientation* or *R5 – Collaboration*.

Chuang and Yadav [CY98] also present an ADSS using unsupervised learning to update decision support knowledge that is consumed as part of heuristic search algorithms to determine decision recommendations. For this reason, it is largely evaluated like the approach by Holsapple et al. [Hol+08] with the following differences: The approach by Chuang and Yadav [CY98] explicitly mentions the consideration of user preferences, but does not consider different decision objectives. Consequently, it fulfills *R1.3 – Situational Competences*, but not *R1.1 – Situational Decision Problems*. Furthermore, the approach can provide decision support for multiple activities, thereby addressing *R2.1 – Modularity*, but without any guidance between tasks as part of *R2.2 – Navigation*. While a documentation of the followed decision process in the sense of *R6.1 – Transparency* is not explicitly mentioned, the approach has the potential to do so due to its additional focus on adaptivity in the user interface of the ADSS. The fulfillment of *R7 – Domain-Portability* is suggested with a problem-domain component but is not demonstrated to confirm this with certainty.

Fazlollahi, Parikh, and Verma [FPV97] describe an ADSS using a rule-based approach to select models based on characteristics of decision makers (particularly considering their skills and previous use of the DSS) and characteristics of input data. The approach thereby addresses *R1.3 – Situational Competences* and *R1.2 – Situational Resources*, however, the latter only with respect to data and without considering the availability of other resources. The authors highlight the ability of the ADSS to adapt to and influence the decision process of a decision maker. However, the decision makers are responsible to navigate through the decision process themselves on a task-by-task basis, which does not fulfill requirements *R2.2 – Navigation* and *R6.2 – Determinism* (and *R6.1 – Transparency* as the followed process seems to be documented, but not presented to the user). Since the DSS selects the suitable decision support model for each task without additional input from decision makers, *R4 – Suitability for Non-Programmers* is fulfilled except for *R4.3 – Error Prevention*, which is only partially fulfilled due to a lacking feedback mechanism for correcting potential mistakes in the lookup tables. The automated selection among the available models furthermore supports *R3*

– *Variety*, although the support for *R3.1 – Reusability* and *R3.2 – Extensibility* is not explicitly mentioned. The described prototype provides limited reusability as the models are written in a proprietary rule-based environment. Collaboration as part of *R5 – Collaboration* is not mentioned. The approach supports *R7 – Domain-Portability*.

High-Level Evaluation of ADSS Approaches

Some ADSS approaches identified throughout the literature review were excluded from the detailed evaluation as they significantly differ from the goal of this thesis, or because they do not provide sufficient explanations for an adequate evaluation concerning the previously established requirements. These approaches are subsequently summarized for completeness.

The ADSS framework proposed by **Piramuthu and Shaw [PS09]** uses unsupervised learning to update the knowledge base of the DSS utilized for the identification of decision recommendations. Decision support knowledge is updated using training examples generated by a simulation component based on a sampling of an observable system or historical input data. Thus, the system is “adaptive” in the sense that the decision support knowledge is updated over time to reflect changes in the business environment. However, at a given point in time, the system does not support multiple alternatives for decision support functionality and therefore does not provide “situational” support in the sense of requirement *R1 – Situativity*. Similarly, the approach by **Mollá, Heavin, and Rabasa [MHR22]** adapts a single decision-making algorithm based on insights gathered from a continuous data stream.

Al-Qaed and Sutcliffe [AS06] describe an ADSS that utilizes different decision support tools to assist decision makers in selecting among multiple available products. However, the proposed ADSS is designed to support product selection as part of business-to-customer e-commerce scenario and therefore does not align with the semi-structured business decision problems considered throughout this thesis.

Chen, Zhou, and Hu [CZH02] describe a generator for group DSSs. Each generated DSS has an adaptive component that decomposes a decision problem into sub-problems and selects among multiple solvers to identify/assemble an optimal decision recommendation. However, the inputs and inner workings of the adaptive component are not described.

Karkanitsa [Kar20] describe an approach for decomposing a decision making problem into subproblems which are subsequently distributed as tasks among actors. However, the adaptivity of the system with respect to decision support functionality remained unclear.

Considering the distinction between “adaptable” and “adaptive” at the beginning of this section, the ADSS described by **Holm et al. [Hol+14]** is an adaptable DSS. It is therefore considered as part of the upcoming Section 3.3.2 discussing DSS generators. This also applies to the approach by **Paranagama, Burstein, and Arnott [PBA98]** and **Weller et al. [Wel+15]**.

Table 3.1: Summary of ADSS approaches with respect to the requirements of Section 3.1

Requirement	[Hol+08]	[CY98]	[FPV97]
<i>R1 – Situativity</i>			
<i>R1.1 – Situational Decision Problems</i>	◐	○	○
<i>R1.2 – Situational Resources</i>	◐	◐	◐
<i>R1.3 – Situational Competences</i>	○	●	●
<i>R2 – Process Orientation</i>			
<i>R2.1 – Modularity</i>	○	●	●
<i>R2.2 – Navigation</i>	○	○	○
<i>R2.3 – Unified Execution Environment</i>	●	●	●
<i>R3 – Variety</i>			
<i>R3.1 – Reusability</i>	◐	◐	◐
<i>R3.2 – Extensibility</i>	◐	◐	◐
<i>R3.3 – Discoverability</i>	●	●	●
<i>R4 – Suitability for Non-Programmers</i>			
<i>R4.1 – Abstraction</i>	●	●	●
<i>R4.2 – Learnability</i>	●	●	●
<i>R4.3 – Error Prevention</i>	◐	◐	◐
<i>R5 – Collaboration</i>			
<i>R5.1 – Common Terminology</i>	○	○	○
<i>R5.2 – Artifact Sharing</i>	○	○	○
<i>R5.3 – Experience Sharing</i>	○	○	○
<i>R5.4 – Organizational Scalability</i>	○	○	○
<i>R6 – Compliance</i>			
<i>R6.1 – Transparency</i>	○	◐	◐
<i>R6.2 – Determinism</i>	○	○	○
<i>R7 – Domain-Portability</i>			
	◐	◐	●

●: addressed, ◐: partially addressed, ○: not addressed

3.3.2 DSS Generators

A *decision support system generator (DSSG)* is an environment consisting of tools to develop decision support systems that are customized to a decision maker's requirements for decision

support (cf. [BSH99]). A DSSG relies on the flexible combination of reusable software modules [DN03]. As shown in Fig. 3.4, a DSS specification describing the combination of software modules is first created by a *DSS Implementor* (in alignment with the terminology used by Maturana, Ferrer, and Baraña [MFB04]), usually utilizing a (visual) editor. Next, the DSS specification is either forwarded to a code generator that generates the DSS as a standalone application (Fig. 3.4a) or – more widespread [BSH99] – the specification is interpreted as part of the DSSG environment (Fig. 3.4b). Some approaches require the DSS implementor to exhibit certain skills, while other approaches also enable decision makers to assume the role of the DSS implementor.

Among the benefits of DSSG usage are the reduced effort [MFB04] and complexity [Fie99; SBM11] of procedural DSS development, which decrease DSS maintainability [Fie99] and prevent the involvement and autonomy of DSS customization by decision makers (cf. [Fie99; SBM11]). In addition, the absence of a generally accepted model [DN03] or limited DSS generalizability [Ric+97] is mentioned, e.g., due to the variety of data [Fie99] or the subjective perception of user-friendliness [SK86]. Lastly, there is also an economic benefit to implementing multiple decision support systems in the same execution environment [PBA98].

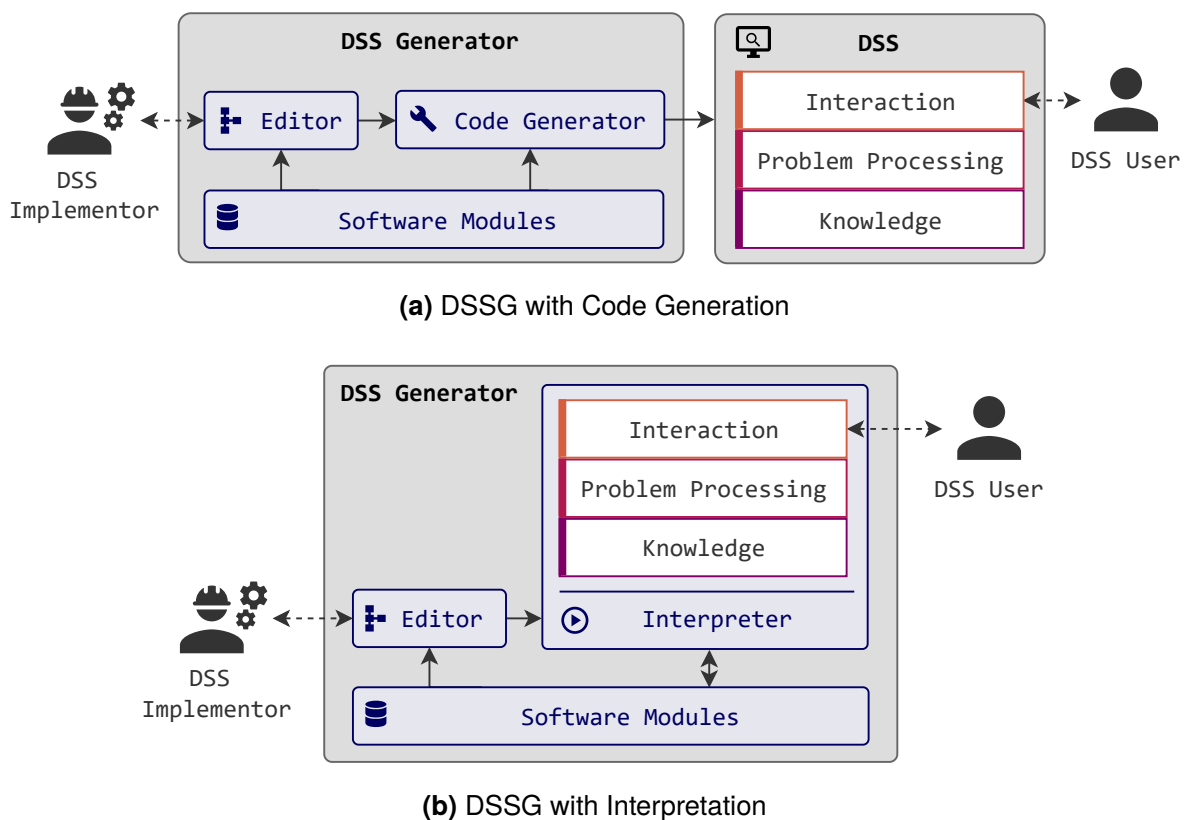


Figure 3.4: Possible approaches to implement a DSS generator

Detailed Evaluation of DSS Generator Approaches

This subsection explains the evaluation results for DSSG approaches summarized in Table 3.2 with respect to the requirements R_x of Section 3.1. In general, $R1 - Situativity$ is addressed since a DSS generator is used by a human implementor who can consider the situational factors of a decision maker. For this reason, $R1 - Situativity$ and its sub-requirements are not repeated throughout the textual explanations of evaluation results.

Ahmed, Sundaram, and Piramuthu [ASP10] describe a DSS generator with a focus on assisting in scenario-based decision making. The generator supports the component-based integration of decision support functionality from repositories of available models, solvers, and data. While $R2 - Process Orientation$ is fundamentally addressed, the activities of the decision process can only be addressed on a task-by-task basis, i.e., there is no automation as part of $R2.2 - Navigation$. $R3 - Variety$ is given, although $R3.1 - Reusability$ is limited due to the focus on a model-solver pattern. A graphical user interface with validation for component integration addresses $R4 - Suitability for Non-Programmers$. $R5 - Collaboration$ is not considered. $R6 - Compliance$ is ensured by requiring decision makers to support their decision process on a task-by-task basis. $R7 - Domain-Portability$ is addressed.

Derigs and Nickel [DN03] describe a DSS generator for financial portfolio management and optimization. The generator provides a rule base fundamentally addressing $R3 - Variety$ where decision makers can select rules and search strategies for meta-heuristics according to their decision objectives, suggesting a focus on $R1.1 - Situational Decision Problems$ and limited $R3.1 - Reusability$ and $R3.2 - Extensibility$ due to the focus on heuristic search algorithms. The generated DSS only supports a single task, i.e., selection of portfolio assets to buy/sell, consequently $R2 - Process Orientation$ is not supported, but $R6.2 - Determinism$ is. Requirement $R4 - Suitability for Non-Programmers$ is partially addressed with a SQL-based notation for rule selection via a GUI. This explicit selection of a rule also addresses $R6.1 - Transparency$. No $R5 - Collaboration$ is considered. The approach is only applicable to portfolio management and does not address $R7 - Domain-Portability$.

Dong and Loo [DL01] describe an agent-based DSS generator for the development of web-based decision support systems. A DSS is assembled by composing a pipeline of multiple decision support components consisting of data, models, and solvers. In some cases, users of the DSS generator can be supported by software agents in component selection and integration as well as composition validation. This addresses $R4 - Suitability for Non-Programmers$ only partially, since the integration of models and solvers requires programming knowledge in cases where agent support is unavailable, i.e., for tightly coupled component bindings. The involvement of agents furthermore potentially prevents $R6.2 - Determinism$. The interaction with the DSS generator happens task-/component-based, consequently, an upfront model of

the decision process cannot be used for *R2.2 – Navigation*. Only after a process has been followed it can be documented in retrospect for *R6.1 – Transparency*. The focus on the model-solver pattern limits *R3 – Variety*. Requirement *R5 – Collaboration* is not addressed. No domain restrictions are evident from the descriptions, which suggests *R7 – Domain-Portability*. However, the technical feasibility of the approach (by instantiation in a concrete application domain) is not demonstrated throughout the paper.

Fierbinteanu [Fie99] describe a visual DSS generator where users can assemble a DSS for constrained search problems from building blocks that encapsulate decision logic expressed using constrained logic programming. The explicit documentation of a building block composition using the graphical editor enables *R2 – Process Orientation*, *R6 – Compliance*, and *R3.3 – Discoverability*. Furthermore, the visual approach addresses *R4 – Suitability for Non-Programmers*, however, a proprietary visual notation is used that may decrease *R4.2 – Learnability*. With respect to *R4.3 – Error Prevention*, the approach supports validation of a building block composition, but only on a syntactical level. *R3.1 – Reusability* is not addressed since the building blocks are implemented using Prolog-like rules. There is no explicit mention of *R3.2 – Extensibility*, but it should be possible using the approach. The approach makes use of an ontology which would enable *R5.1 – Common Terminology*, however, there is otherwise no discussion of features addressing *R5 – Collaboration*. The approach can be applied in multiple domains, thereby addressing *R7 – Domain-Portability*.

Holm et al. [Hol+14] describe an approach that enables shop-floor managers to adapt the decision logic of the DSS as a (sequential) composition of “function blocks” with predetermined inputs and outputs. The explicit documentation of decision logic as a composition of function blocks addresses *R2 – Process Orientation*, *R5 – Collaboration*, and *R4.1 – Abstraction*. However, the description of the function block format by Wang [Wan08] suggests a limited *R4.2 – Learnability* by non-programmers due to the resemblance of function blocks to gates in electrical engineering and a proprietary syntax for implementing function block behavior decreasing *R3.1 – Reusability*. Also, *R4.3 – Error Prevention* is not provided. Derivation of function blocks following an existing standard fulfills *R5.1 – Common Terminology* and *R5.3 – Experience Sharing* to some extent, but only concerning domain-independent terminology and expectedly slow propagation of experience as updating a standard takes time. Cross-organizational sharing of function blocks to address *R5.2 – Artifact Sharing* is not discussed. The approach is only applicable to the domain of shop-floor management, thereby not addressing *R7 – Domain-Portability*.

Maturana, Ferrer, and Baraño [MFB04] describe a DSS generator that automatically generates an optimization-based DSS consisting of a database, solver and GUI from a model specification. As evident from the differentiation between the roles of “implementor” for the

person using the DSS generator and “user” for the person using the DSS, the DSS generator is not targeted towards decision makers but decision analysts who are familiar with SML [Geo92], a language to describe structured models in the domain of operations research used for DSS generation. Consequently, the approach only partially fulfills requirement *R4 – Suitability for Non-Programmers*. Error prevention in the form of validation is provided, but it is unclear whether the validation also considers the semantic correctness of the SML specification or only syntactical correctness. Requirement *R3 – Variety* is not fulfilled as there is no repository for models and/or solvers. Since the approach only supports a single decision-making task, *R2 – Process Orientation* and *R6 – Compliance* are not addressed. Requirement *R5 – Collaboration* is not mentioned, but *R7 – Domain-Portability* is fulfilled.

Paranagama, Burstein, and Arnott [PBA98] describe a framework to implement DSS generators for senior managers and its prototypical implementation ADAPTOR. The approach is specifically designed to support multiple users and maintains user profiles for them to supply criteria preferences for future model selections from a model base. In doing so, it is one of the few reviewed approaches that support *R5.3 – Experience Sharing*, albeit only partially as sharing novel decision support requirements is not explicitly supported. Other aspects of *R5 – Collaboration* are not considered. Since only a single model is selected and configured from the model base, *R2.2 – Navigation* and *R6 – Compliance* are not addressed. The model base only supports models for multi-criteria decision making, which limits *R3.1 – Reusability* and *R3.2 – Extensibility*. The approach fundamentally addresses *R4 – Suitability for Non-Programmers* due to model selection and configuration via a graphical user interface. The user profiles allow warnings in case of (assumed) model misconfiguration, but the wrong model selection is not considered, therefore only partially implementing *R4.3 – Error Prevention*. The approach can be applied in multiple domains, consequently *R7 – Domain-Portability* is provided.

Savić, Bicik, and Morley [SBM11] describe a DSS generator based on *Microsoft Excel* spreadsheets to encourage the participation of decision makers during (prototypical) DSS development to improve DSS adoption and performance. Since each spreadsheet only supports a single decision support task, *R2 – Process Orientation* and *R6 – Compliance* are not supported. Although multiple spreadsheets can be defined, thereby partially addressing *R2.1 – Modularity*, each spreadsheet results in a separate DSS contrary to *R2.3 – Unified Execution Environment*. The spreadsheet-based user interface of the approach fulfills *R4 – Suitability for Non-Programmers*, but *R4.3 – Error Prevention* is not described in the paper. A repository supporting *R3.3 – Discoverability* for available spreadsheets is not discussed, but (existing) decision support functionality can be integrated via *Visual Basic* scripts for *R3.1 – Reusability* and *R3.2 – Extensibility*. Requirement *R5 – Collaboration* is not addressed. The approach is advertised as “general purpose” which fulfills *R7 – Domain-Portability*.

Weller et al. [Wel+15] describe a DSS generator that composes decision support functionality (encapsulated in so-called “cognitive apps” with RESTful APIs) into a single medical decision support system. A composition of multiple cognitive apps is referred to as a “use case app” that is utilized by decision makers for corresponding the decision making tasks. The composition of cognitive apps into use case apps addresses *R2 – Process Orientation* and *R3.2 – Extensibility*. The composition of cognitive apps is based on *Data-Fu* [Sta+13], a declarative, rule-based execution language based on linked data and state transitions of resources. This addresses *R4.1 – Abstraction*, although *Data-Fu* is targeted at programmers who are able to express linked data using the *Resource Description Framework (RDF)*, therefore *R4.2 – Learnability* is not given. *R4.3 – Error Prevention* is neither mentioned for *Data-Fu* nor the overall approach. The RESTful APIs of cognitive apps support *R3.1 – Reusability*, assuming a wrapper that converts the linked data provided to the cognitive app into a format suitable for the encapsulated legacy application is possible. A repository listing all available cognitive apps is not mentioned, which hinders *R3.3 – Discoverability*. The use of linked data establishes *R5.1 – Common Terminology*, although further collaboration aspects are not mentioned. The use case apps address *R6.2 – Determinism*, however, there is no indication that the underlying composition of cognitive apps is communicated to the user to address *R6.1 – Transparency*. The approach furthermore does not fulfill *R7 – Domain-Portability* as it is targeted towards the development of medical DSS (cf. the authors’ explanation of the data tier).

High-Level Evaluation of DSS Generator Approaches

Some DSSG approaches identified throughout the literature review were excluded from the detailed evaluation as they significantly differ from the goal of this thesis, or because they do not provide sufficient explanations for an adequate evaluation concerning the previously established requirements. These approaches are subsequently summarized for completeness.

Bhargava, Sridhar, and Herrick [BSH99] evaluate eleven DSS generators and conclude that all DSS generators are meant to be used by DSS developers and not decision makers. This is in contradiction with the second challenge described throughout the problem statement in Section 1.3. Consequently, the mentioned approaches were not further considered.

Janson, Douglas Smith, and Dattero [JDD90] similarly draw a clear distinction between the roles of DSS builders and DSS users in the context of DSS generators. They present insights to improve the communication between those roles to increase the success of requirements elicitation for and implementation of a DSS.

Chen, Zhou, and Hu [CZH02] describe a DSS generator for group decision support systems that was already partially discussed in the previous section due to the adaptive component of the generated DSSs. The generator itself only seems to consider different consensus

Table 3.2: Evaluation of DSS generator approaches with respect to the requirements of Section 3.1

Requirement	[ASP10]	[DN03]	[DL01]	[Fie99]	[Hol+14]	[MFB04]	[PBA98]	[SBM11]	[Wei+15]
<i>R1 – Situativity</i>	(●: addressed by all approaches, see explanations)								
<i>R2 – Process Orientation</i>									
<i>R2.1 – Modularity</i>	●	○	●	●	●	○	◐	◐	●
<i>R2.2 – Navigation</i>	○	○	○	●	●	○	○	○	●
<i>R2.3 – Unified Exec. Env.</i>	●	●	●	●	●	●	○	○	●
<i>R3 – Variety</i>									
<i>R3.1 – Reusability</i>	◐	◐	◐	○	○	○	◐	●	●
<i>R3.2 – Extensibility</i>	●	◐	●	◐	◐	○	◐	●	●
<i>R3.3 – Discoverability</i>	●	●	●	●	◐	○	●	○	○
<i>R4 – Suitability</i>									
<i>R4.1 – Abstraction</i>	●	◐	◐	●	●	●	●	●	●
<i>R4.2 – Learnability</i>	●	◐	◐	◐	◐	◐	●	●	○
<i>R4.3 – Error Prevention</i>	●	○	●	◐	○	◐	◐	○	○
<i>R5 – Collaboration</i>									
<i>R5.1 – Common Terminology</i>	○	○	○	●	◐	○	○	○	●
<i>R5.2 – Artifact Sharing</i>	○	○	○	○	○	○	○	○	○
<i>R5.3 – Experience Sharing</i>	○	○	○	○	◐	○	◐	○	○
<i>R5.4 – Organizational Scalability</i>	○	○	○	○	○	○	○	○	○
<i>R6 – Compliance</i>									
<i>R6.1 – Transparency</i>	●	●	●	●	●	○	○	○	○
<i>R6.2 – Determinism</i>	●	○	◐	●	●	○	○	○	●
<i>R7 – Domain-Portability</i>	●	○	●	●	○	●	●	●	○

●: addressed, ◐: partially addressed, ○: not addressed

mechanisms (cf. “Group Appraisalment Method Base (GAMB)”) without consideration of selecting other decision support functionality.

Liang and Jia [LJ14] present a DSS generator framework with a focus on real-time decision making in a big data context. As their explanations are very focused on the technical implementation aspects of the approach, a detailed evaluation of the approach cannot be

performed without significant uncertainty. Nevertheless, the authors highlight the advantages of using a DSS Generator over manual DSS development throughout their demonstration.

Rico et al. [Ric+97] describe the concept for a DSS generator that provides the basis for the generator's modules described by Ramos et al. [Ram+98], Castro et al. [Cas+98], Rossi et al. [Ros+98], and Caliusco et al. [Cal+98]. However, the explanations largely focus on the domain-specific aspects of the approach for industrial companies and do not support sufficient technical depth to evaluate the approach with respect to the established requirements.

Saxena and Kaul [SK86] conceptually describe a DSS generator, albeit a short description of a prototypical implementation using the concept is also provided. However, due to the brevity of the prototype's description and the outdated technologies, an in-depth evaluation of the approach with respect to the established requirements is not possible.

Yeo and Nah [YN92] develop their DSS generator in the context of a computerized management game contrary to this thesis, which aims to improve real-world decision making.

3.3.3 Service-Oriented DSS

A *service-oriented DSS (SO-DSS)* is any DSS that utilizes a service-oriented architecture (cf. Section 3.2.1). Reusable decision support artifacts are provided in the form of loosely-coupled services that can be composed to provide decision support for a specific decision-making task.

A motivation for using a service-oriented DSS mentioned is the flexibility provided by composing services for adapting to dynamic business environments (cf. [DD13a; DS13; MKP21]). Service composition furthermore has the potential to reduce development complexity, which in the case of automated service composition can even enable decision makers to develop a DSS (cf. [MKP21]). In a distributed SOA, additional benefits are multi-enterprise integration of data and decision support functionality [DS13; YSL08], high-availability [YSL08] and efficient use of computational resources in a big data context [DD13b].

Detailed Evaluation of Service-Oriented DSSs

This subsection explains the evaluation results for SO-DSS approaches summarized in Table 3.3 with respect to the requirements R_x presented in Section 3.1.

Becker et al. [Bec+08] describe *Plato*, a service-oriented DSS for preservation planning. Based on requirements entered via a web interface, a preservation planning process is automatically assembled from loosely coupled services. The requirements capture *R1.1 – Situational Decision Problems* with respect to decision objectives, but not *R1.2 – Situational Resources* or *R1.3 – Situational Competences*. The automated composition enables *R4.1 – Abstraction* and *R4.2 – Learnability*, but *R4.3 – Error Prevention* is only partially addressed

since the automated composition will likely prevent errors, but potential errors cannot be corrected manually. The composition is executed with a BPEL-based workflow engine in alignment with *R2 – Process Orientation*, albeit some external tools are executed outside the DSS, thus only partially addressing *R2.3 – Unified Execution Environment*. Service-Oriented enables *R3 – Variety*. Recurring planning situations are supported with reusable patterns and templates stored in a knowledge base to support *R5.3 – Experience Sharing*. The approach partially addresses *R6 – Compliance* since compositions are documented for *R6.1 – Transparency* and can be saved for future use, which supports *R6.2 – Determinism*, but only at the end of the process. *R5 – Collaboration* across organizational boundaries is not mentioned. The approach is only applicable for preservation planning, consequently *R7 – Domain-Portability* is not provided.

Dong and Srinivasan [DS13] provide an adaptation of a previous approach (Dong and Loo [DL01] discussed as part of Section 3.3.2) with an additional focus on composing distributed decision components to address decision makers' individual requirements for decision support. The approach describes a service-oriented environment where distributed decision components (models, solvers, visualizations, and data) are coordinated using software agents. The authors improve on the previous version of their approach in the following ways: *R4 – Suitability for Non-Programmers* is improved by providing a graphical user interface for service composition for *R4.1 – Abstraction* and *R4.2 – Learnability*. However, the explicit mention of validation responsible for *R4.3 – Error Prevention* in the previous approach is missing. Manual composition enables *R6 – Compliance*. *R5 – Collaboration* is improved by providing an ontology to enable *R5.1 – Common Terminology* and a distributed, agent-based computation model enables *R5.4 – Organizational Scalability*.

Shafiei, Sundaram, and Piramuthu [SSP12] describe a multi-enterprise collaborative DSS that corresponds to a service-oriented DSS since DSS functionality is composed of multiple web services across organizational boundaries. The approach uses BPEL [OAS07] for invoking the different services, thereby addressing *R2 – Process Orientation* and *R6 – Compliance*. A BPEL process is specified via a GUI that addresses *R4.1 – Abstraction* and *R4.2 – Learnability*, but not *R4.3 – Error Prevention* as no validation is mentioned. The manual composition of services addresses *R1 – Situativity*. *R3 – Variety* is addressed and platform independence for services is specifically mentioned for *R3.1 – Reusability*, but services still must fit the model-solver pattern. Albeit utilizing decision support functionality from multiple organizations, *R5.1 – Common Terminology* is not considered. Also *R5.3 – Experience Sharing* is not supported for sharing decision support requirements and best practices. The approach addresses *R7 – Domain-Portability*.

Axelsson et al. [Axe+17] describe a software ecosystem that provides decision support for

system component selection. Using an ecosystem specifically focuses on *R3 – Variety* and *R5 – Collaboration*, fully addressing all of its sub-requirements except *R5.3 – Experience Sharing* since previous decision cases can be shared among participants, but novel decision support requirements cannot be shared. *R4.3 – Error Prevention* is also only performed for checking the interoperability of newly added services with existing services, but not as part of service composition. The approach supports the composition of multiple ecosystem services into decision processes, thereby addressing *R2 – Process Orientation*, but there is no indication that service composition can be done by decision makers or other domain experts to address *R4 – Suitability for Non-Programmers*. There seems to be some recommendation for service selection, but it only considers *R1.1 – Situational Decision Problems*. Decision processes provide *R6.2 – Determinism*, but a sufficient explanation of decision processes to decision makers for *R6.1 – Transparency* is not mentioned. *R7 – Domain-Portability* is not addressed since the approach exclusively focuses on decision support for system component selection.

High-Level Evaluation of Service-Oriented DSSs

Some SO-DSS approaches identified throughout the literature review were excluded from the detailed evaluation as they significantly differ from the goal of this thesis, or because they do not provide sufficient explanations for an adequate evaluation with respect to the previously established requirements. These approaches are subsequently summarized for completeness.

Demirkan and Delen [DD13b] motivate the need and requirements for SO-DSS, describe the conceptual architecture of a SO-DSS, and define the concepts of data-as-a-service, information-as-a-service, and analytics-as-a-service in combination with future research directions. Since the authors focus on conceptual explanations, the paper does not provide enough technical detail for an evaluation with respect to the requirements established in Section 3.1. The authors provide additional (conceptual) background information in [DD13a].

Yang and Calmet [YC06] describe a service-oriented DSS that is built upon an ontology-driven uncertainty model. While the explanation of the approach has certain characteristics that align with the requirements established for this thesis, e.g., platform independence for services providing decision support functionality, the provided information is ultimately too superficial for a detailed evaluation with respect to the solution requirements, in particular regarding *R4 – Suitability for Non-Programmers* and *R6 – Compliance*.

Ye, Song, and Li [YSL08] describe the service-oriented architecture of a DSS for crisis management and its implementation using the WS-* technologies discussed in Section 3.2.1. However, since the authors' motivation for choosing a SOA is primarily influenced by high availability of decision support functionality during a crisis, the adaptivity of the system is limited for non-developers (or at least not sufficiently discussed throughout the publication).

Table 3.3: Evaluation of SO-DSS approaches with respect to the requirements of Section 3.1

Requirement	[Bec+08]	[DS13]	[MKP21]	[SSP12]	[Axe+17]
<i>R1 – Situativity</i>					
<i>R1.1 – Situational Decision Problems</i>	◐	●	○	●	●
<i>R1.2 – Situational Resources</i>	○	●	●	●	○
<i>R1.3 – Situational Competences</i>	○	●	○	●	○
<i>R2 – Process Orientation</i>					
<i>R2.1 – Modularity</i>	●	●	●	●	●
<i>R2.2 – Navigation</i>	●	●	●	●	●
<i>R2.3 – Unified Execution Environment</i>	◐	●	●	●	●
<i>R3 – Variety</i>					
<i>R3.1 – Reusability</i>	●	◐	◐	●	●
<i>R3.2 – Extensibility</i>	●	◐	●	◐	●
<i>R3.3 – Discoverability</i>	●	●	●	●	●
<i>R4 – Suitability for Non-Programmers</i>					
<i>R4.1 – Abstraction</i>	●	●	●	●	○
<i>R4.2 – Learnability</i>	●	●	●	●	○
<i>R4.3 – Error Prevention</i>	◐	○	◐	○	◐
<i>R5 – Collaboration</i>					
<i>R5.1 – Common Terminology</i>	○	●	●	○	●
<i>R5.2 – Artifact Sharing</i>	○	●	●	●	●
<i>R5.3 – Experience Sharing</i>	●	○	○	○	◐
<i>R5.4 – Organizational Scalability</i>	○	●	●	●	●
<i>R6 – Compliance</i>					
<i>R6.1 – Transparency</i>	○	●	○	●	●
<i>R6.2 – Determinism</i>	◐	●	◐	●	○
<i>R7 – Domain-Portability</i>					
	●	●	●	●	○

● : addressed, ◐ : partially addressed, ○ : not addressed

Stănescu, Ștefan, and Filip [SSF15] describe a DSS for renewable energy providers where decision makers can integrate additional data sources not considered by developers at

design time. However, the paper does not indicate if and how decision support functionality can be customized by end users, therefore *R1 – Situativity* is assumed to be unfulfilled.

3.3.4 Distantly Related Research Areas and Approaches

This section explains additional research areas that initially seem related to the goal of providing decision makers with tailored decision support systems. However, upon closer examination, these research areas do not sufficiently address the challenges of Section 1.3.

Collaborative and Group Decision Support Systems A *collaborative DSS* assists with collaborative decision making where multiple participants must agree on a decision despite having potentially conflicting objectives (cf. [Zar13]). The focus of a collaborative DSS is therefore on establishing a consensus between participants of the decision process and not a recombination of distributed decision support functionality or an advancement thereof. This fundamentally also holds for a *group DSS*, despite the focus potentially being more on cooperation compared to competition in the context of a collaborative DSS [Gra87; Zar13].

Intelligent and Web-Based Decision Support Systems An *intelligent DSS (IDSS)* is a DSS that utilizes some form of artificial intelligence technique during the computation of a decision recommendation (cf. [Phi13]). Holsapple et al. [Hol+08] point out that not every intelligent DSS is able to adapt to the requirements for decision support of an individual decision maker. Since the research of IDSS approaches is rather broad and the discussion of adaptive decision support systems in Section 3.3.1 already includes multiple approaches that use AI to address certain situational factors in decision making, no further IDSS approaches are evaluated.

Similarly, web-based decision support systems are not further evaluated. A *web-based DSS* is accessible to decision makers via a web browser (cf. [DL01]). However, similar to the use of AI in an intelligent DSS, using is a technical choice to implement (part of) a DSS and does not guarantee adaptivity to account for different decision support requirements of decision makers. Instead, a web-based DSS is primarily used to reach a wider audience [ZSJ08].

Model Management Systems A *model management system (MMS)* is part of a (model-driven) DSS to store and manipulate decision-making models. These models are subsequently instantiated with data for a concrete decision-making task and provided to a solver to obtain a decision recommendation (cf. [ED13]). Some approaches such as those proposed by El-Gayar and Deokar [ED13] and Stapel [Sta16] consider ontology-based concepts to enable distributed model management across organizational boundaries or model recombination. However, since an MMS is only part of a DSS and requires additional components for data management, solvers, and visualization that may also be subject to the situational factors of decision making, standalone MMS approaches are not evaluated.

Business Intelligence and Analytics *Business intelligence* and/or *business analytics* (referred to as *BIA* in combination) refer to methods and their software implementations that enable the collection, integration, transformation, and analysis of data [LCC13]. Delen and Demirkan [DD13a] present a taxonomy of business analytics that categorizes BIA approaches from descriptive (“What happened?”) over predictive (“What will happen?”) to prescriptive (“What should be done?”). Since decision support ultimately only aligns with the last level of prescriptive analytics and the explanations of Lim, Chen, and Chen [LCC13] suggest a focus of BIA on analyzing existing data (as opposed to the generation of new data that may also be part of a DSS, e.g., as part of a simulation for energy distribution network planning), BIA approaches are not extensively evaluated.

Decision Model and Notation (DMN) The *Decision Model and Notation (DMN)* aims to “provide the constructs that are needed to model decisions” [Obj21]. DMN is often used in conjunction with the *Business Process Model and Notation (BPMN)* [Obj13] to describe how decision tasks within a business process are implemented. Decision logic is specified such that it can be automatically executed, either based on a structured description of business rules or using decision services that are equivalent to the execution of a function with decision logic as provided by a web service. However, as previously described, decision support is not only characterized by decision logic but also by data generation in the form of simulation or visualization-based data analysis. Although this could in theory be implemented outside DMN with other business process tasks, the limitations of general-purpose process modeling for the development of tailored decision support systems have already been discussed in Section 1.4 in the context of low-code development.

Situational Method Engineering Similar to decision processes, the effectiveness and efficiency of software development processes are influenced by situational factors originating from the context of a specific development project or the encompassing organization. The goal of *situational method engineering (SME)* is the creation of situation-specific software development processes, potentially by assembling existing fragments with accompanying tools support from a repository [FE16]. However, SME focuses on the design of software development processes and not decision (support) processes. Only recently, Gottschalk [Got22] applied SME concepts to decision support for business model development.

3.4 Key Takeaways

The requirements presented in Section 3.1 call for an approach to tailored DSS development that considers the situational factors of decision making using a modular, process-oriented

approach suitable for non-programmers. A prerequisite for a tailored DSS is a sufficient variety of reusable decision support functionality that is potentially obtained and extended across multiple organizations in a collaborative fashion utilizing shared experience. Although assistance during the development of a tailored DSS is desirable for error prevention, the development approach must still be transparent and deterministic for compliance and should work across multiple application domains. The subsequent evaluation of approaches presented in Section 3.3 with a focus on adaptive and service-oriented decision support systems as well as DSS generators shows that existing work only partially addresses these requirements.

As evident from Table 3.1, all ADSS approaches only consider a subset of situational factors as part of *R1 – Situativity*. Their automated adaptivity is often an advantage for unstructured decision problems (cf. Section 2.2) where the decision process is not completely known upfront and is discovered with the help of the ADSS. However, this dynamism fails to address *R2.2 – Navigation* and *R6 – Compliance* in particular. Furthermore, none of the reviewed ADSS approaches consider *R5 – Collaboration*, which may in part be attributed to the comparatively early publication date of the papers.

The evaluation of DSS generators summarized in Table 3.2 significantly deviates from the ADSS approaches since *R1 – Situativity* is addressed for every approach. This is assumed since there is a human implementor using the DSSG who should be able to interpret the situational factors of the decision maker the DSS is generated for. However, not every approach considers the implementor's role to be assumed by decision makers and instead requires a specific data analysis or technical skillset, thus *R4 – Suitability for Non-Programmers* is not always addressed. Furthermore, *R5 – Collaboration* is only very sparsely addressed.

As indicated by Table 3.3, SO-DSS approaches usually address most of the requirements of Section 3.1. Compared to the previous two research areas, they can significantly improve with respect to *R5 – Collaboration* and *R3 – Variety*. Nevertheless, the approaches either target developers and do not address *R4 – Suitability for Non-Programmers*, or they use automated service composition with shortcomings regarding *R6 – Compliance*. Furthermore, they (partially) fail to address *R4.3 – Error Prevention* and sub-requirements of *R1 – Situativity*.

In conclusion, there is a research gap for an approach that addresses all established requirements for tailored DSS development. The previous summary suggests the use of a service-oriented DSS generator to combine the respective benefits regarding *R1 – Situativity*, *R3 – Variety*, and *R5 – Collaboration* with the potential to address *R4 – Suitability for Non-Programmers*. In addition to addressing all other requirements, a special focus should be given to *R4.3 – Error Prevention*, *R5.3 – Experience Sharing*, and *R6.1 – Transparency*, which is often not or only partially addressed.

PART II

Solution Design and Development

Decision Support Ecosystems

This chapter introduces the concept of a decision support ecosystem to address the previously presented requirements and research gap for tailored DSS development. The chapter discusses the benefits of the concept for providing each decision maker with a tailored DSS and focuses on initial design recommendations to guide the implementation of a decision support ecosystem. The presented insights extend the paper “Decision Support Ecosystems: Definition and Platform Architecture” by Kirchhoff, Weskamp, and Engels [KWE22a].

The chapter first introduces the related concept of a digital business ecosystem (Section 4.1). It serves as a foundation for the subsequent definition of a decision support ecosystem and the discussion of its expected benefits in addressing the challenges of tailored DSS development (Section 4.2). Next, five design principles are presented to guide implementors during the realization of a decision support ecosystem for a specific application domain (Section 4.3). The design principles are refined into characteristics of a decision support ecosystem such as actor roles and lifecycle activities (Section 4.4), which are ultimately aggregated into a reference architecture for the shared platform of a decision support ecosystem (Section 4.5). The insights presented in the chapter are correlated to the upcoming chapters explaining one approach for the technical implementation of the reference architecture (Section 4.6).

4.1 Background: Digital Business Ecosystems

This section describes the concept of a digital business ecosystem as a foundation for the introduction of the decision support ecosystem concept in the upcoming section. The (digital

business) ecosystem concept is defined (Section 4.1.1) and its primary characteristics are discussed (Section 4.1.2). Two sub-categories of digital business ecosystems are presented (Section 4.1.3) and the benefits of digital business ecosystem are summarized (Section 4.1.4).

4.1.1 Definition of Digital Business Ecosystems

Inspired by biological ecosystems, the ecosystem metaphor refers to “multiple and varying interrelationships between many actors and infrastructure that contribute to [the creation of] a resource (e.g., business, service or software)” [OL18]. The ecosystem metaphor has been used in different disciplines, for example, to describe business and digital ecosystems [SLE19], or human, social, and commercial ecosystems [Bos09]. The remainder of this section focuses on the concept of a digital business ecosystem since it is the most common ecosystem variant [LBB12], has been applied in many disciplines on practical as well as academic level [SLE19], and aligns with the overall focus of the thesis. Based on a literature review, Senyo, Liu, and Effah define a *digital business ecosystem (DBE)* as “a socio-technical environment of individuals, organisations and digital technologies with collaborative and competitive relationships to co-create value through shared digital platforms” [SLE19]. In the definition, *value* refers to any “financial or non-financial benefit” [SLE19], e.g., a service fee or need satisfaction respectively [Man16].

4.1.2 Characteristics of Digital Business Ecosystems

Oliveira and Lóscio [OL18] and Senyo, Liu, and Effah [SLE19] describe similar characteristics for digital business ecosystems that expand on the concept’s definition in the previous Section 4.1.1. The characteristics are visually summarized in Fig. 4.1 and subsequently explained in more detail.

Symbiosis of Networked Actors

A digital business ecosystem contains multiple actors, i.e., individuals and organizations, with cooperative and competing relationships across organizational boundaries [SLE19]. As indicated by Fig. 4.1, these actors can be grouped into roles where each actor can potentially assume multiple roles [Man16]. The roles chosen in the figure are based on the descriptions of Karl et al. [Kar+20] and Schwichtenberg and Engels [SE20]. The *platform provider* contributes and maintains the shared digital platform where a *service provider* publishes available ecosystem services. These *ecosystem services* can range from reusable software components [Bos09] to activities that are not supported by software [OL18], e.g, consulting

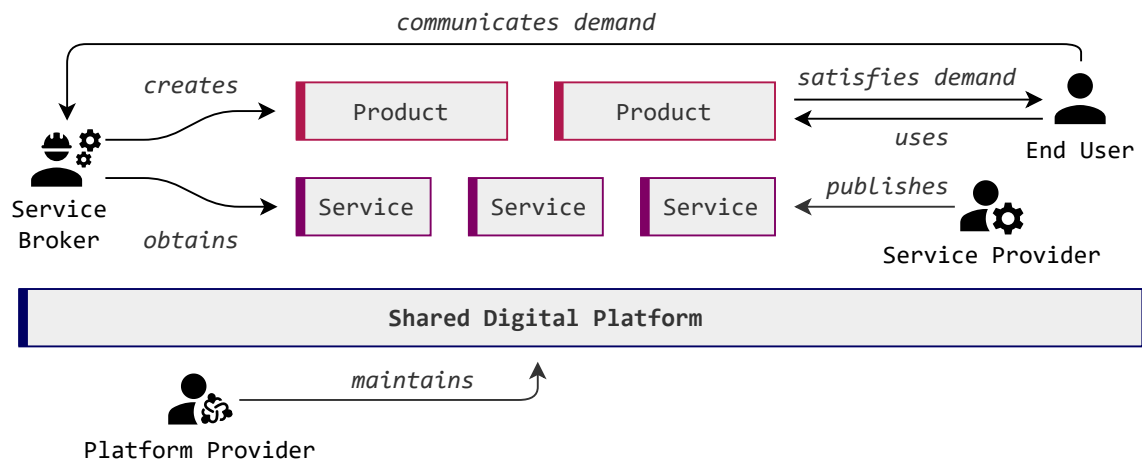


Figure 4.1: Overview of a digital business ecosystem

services¹. A subset of available services is then recombined by a *service broker* into a *product*, i.e., a material good or an immaterial (software-unrelated) service offering, to meet the demands of an *end user*. This separation of concerns between roles creates a network of (implicit) dependencies between actors [SLE19], e.g., the end user depends on the service broker for demand satisfaction, and the service broker depends on the services contributed by service providers for product development. Service providers and service brokers in particular depend on the shared digital platform of the platform provider. All actors participating in an ecosystem can provide technical and non-technical contributions [Man16], e.g., the platform provider can contribute infrastructure for the shared digital platform as well as human resources for platform governance. It is assumed that the sum of actors' contributions creates greater value than what would be achievable for a single organization on its own [SLE19].

Shared Digital Platform

The shared digital platform of a digital business ecosystem is the technical infrastructure to find and connect ecosystem services [LBB12] via supported information and communication technologies [SLE19]. While the shared digital platform exhibits parallels to the concept of a service-oriented architecture discussed in Section 3.2.1, the platform supersedes a SOA by supporting software-unrelated service offerings and by putting a special emphasis on the cross-organizational relationships between ecosystem participants [LBB12]. In other words,

¹ The meaning of “service” in the context of a digital business ecosystem that also includes software-unrelated service offerings can vary from the meaning of “service” in a service-oriented architecture (cf. Section 3.2.1), which always refers to a software artifact. For this reason, the term “ecosystem service” will be used to include all types of service offerings in contrast to a “software service” of a SOA.

the shared digital platform consists of tools and service offerings that actors participating in a digital business ecosystem can use to collaboratively create value (cf. [SLE19]).

Self-Organized (Co-)Evolution

Changes within the surrounding business environment as well as changes within the digital business ecosystem, e.g., changing relationships between ecosystem participants, can lead to new business opportunities or business threats [SLE19; OL18]. These changes require ecosystem participants to adapt and evolve. This evolution occurs mostly self-organized as the aforementioned symbiosis between ecosystem participants and the associated dependency network between actors result in the adaptation of one actor triggering the adaptation of dependent actors [SLE19]. For example, when a service provider changes an ecosystem service, all end users who use a product that integrates this service are affected by this change. This results in co-evolution, i.e., the collective and virtually simultaneous evolution of all ecosystem participants [SLE19]. Self-organization is further promoted by enabling interaction or feedback between ecosystem participants through the shared digital platform [OL18].

4.1.3 Types of Digital Business Ecosystems

Oliveira and Lóscio [OL18] summarize different types of digital business ecosystems that can be distinguished based on the types of ecosystem services exchanged via the shared digital platform and the types of products that are developed from these ecosystem services and provided to end users. Figure 4.2 shows popular variants of digital business ecosystems that are subsequently explained in more detail.

Software Ecosystem In a *software ecosystem*, the main contributions of ecosystem participants are software components or software services to support business activities. Bosch [Bos09] describes a software ecosystem as an advancement of a software product line that spans across organizational boundaries to further address the mass-customization need of

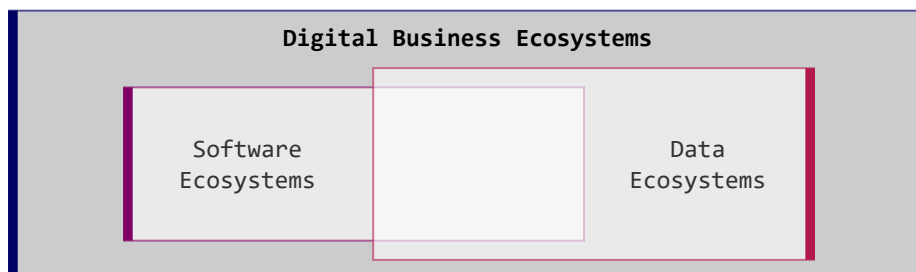


Figure 4.2: Types of Digital Business Ecosystems

end users, potentially using end-user development. The *Salesforce* platform² is an example of a cloud-based software ecosystem for customer relationship management. [Bos09]

Data Ecosystem In a *data ecosystem*, the main contribution of ecosystem participants is data, but ecosystem services can furthermore encompass related technologies such as software services or infrastructure. The goal of a data ecosystem is the delivery of intelligence to industry, academia, and governments. An example of a data ecosystem is *GovData*³, the open data portal of Germany. [OL18]

Hybrid Ecosystems As explained by Oliveira and Lóscio [OL18] and indicated in Fig. 4.2, a clear distinction between ecosystem variants is not always possible. As evident from the previous introduction of data ecosystems, a data ecosystem can also include the exchange of data-related software, thereby potentially encompassing a software ecosystem. Analogously, a software ecosystem does not necessarily exclude the exchange of related data to be consumed with exchanged software components and services. As a result, hybrid ecosystems that cover multiple variants of digital business ecosystems are possible. However, due to the heterogeneity of potential contributions by ecosystem participants, the complexity in the design and development of such hybrid ecosystems increases.

4.1.4 Benefits of Digital Business Ecosystems

The previously presented ecosystem characteristics support the explanation of three benefits associated with ecosystem usage that are published in academic literature.

Mass Customization [Bos09] A digital business ecosystem can enable mass customization. Ecosystem services provided by additional actors outside of an organization provide service brokers with more alternatives to address the product requirements of end users. The promotion and discovery of alternatives are furthermore facilitated via the shared digital platform of the ecosystem. Thus, digital business ecosystems support the development of effective applications. Depending on the design of the ecosystem, end users can potentially develop customized products themselves using an end-user development approach (cf. Section 3.2.2).

Efficient Development [SLE19] The utilization and recombination of existing ecosystem services contributed by service providers via the shared digital platform enable service brokers to achieve lower development costs and faster time-to-market (cf. [MP14]). This increases development efficiency considering development time and spending. Efficient development is also an enabler of the aforementioned mass customization. Analogous to previous explanations,

² Website: <https://www.salesforce.com/eu/>

³ Website: <https://www.govdata.de/>

efficiency is maximized when the ecosystem (or more specifically, the underlying shared digital platform) supports end-user development.

Accelerated Innovation [OL18; Bos09] An ecosystem achieves accelerated innovation in multiple ways: First, innovators among actors participating in a network contribute novel ecosystem services or ideas for service integration. Due to the symbiosis and network character of ecosystems enabled by the shared digital platform, these innovations spread across ecosystem participants. The accompanying co-evolution furthermore ensures that all ecosystem participants benefit from innovations. The collaboration between ecosystem participants also enables a shared cost of innovation, which in turn achieves the retention of funds that can be spent on research and development of further innovations.

4.2 Towards Decision Support Ecosystems

This section translates the characteristics and benefits of digital business ecosystems presented in the previous section to the domain of tailored DSS development. For this purpose, the section provides an initial definition for a decision support ecosystem (Section 4.2.1), explains how a decision support ecosystem can potentially address the challenges for tailored DSS development motivated at the beginning of the thesis (Section 4.2.2), and presents a challenge that prevents the immediate implementation of a decision support ecosystem (Section 4.2.3).

4.2.1 Definition and Categorization of a DSE

The definition of a digital business ecosystem in Section 4.1.1 referred to the co-creation of value as the central goal of a digital business ecosystem. As motivated in Chapter 1, this thesis focuses primarily on the value of efficiently developing a tailored DSS to effectively address the individual requirements for decision support of a decision maker. Thus, in consideration of the previous definition of digital business ecosystems and the discussion of their characteristics and benefits, an initial definition for the decision support ecosystem is formulated as follows:

Decision Support Ecosystem – Conceptual Definition

A decision support ecosystem (DSE) is an evolving socio-technical network of individuals and organizations with collaborative and competitive relationships who provide technical and non-technical contributions for the efficient and effective co-development of tailored decision support systems via a shared digital platform.

In alignment with previous ecosystem definitions and characteristics, the above definition of a decision support ecosystem provides a conceptual view of the DSE concept, hence it is

labeled as a “conceptual definition”. A refined definition based on subsequently explained design choices for the implementation of a DSE will be discussed at the end of the chapter.

The technical contributions of DSE participants can already be summarized as software for data analysis (i.e., for data transformation, simulation, optimization, and visualization), data to run analyses with, and infrastructure to run analyses on. Considering the variants of digital business ecosystems discussed in Section 4.1.3, i.e., software ecosystems and data ecosystems, a decision support ecosystem can therefore be classified as a hybrid ecosystem in the overlapping space between software and data ecosystems (cf. Fig. 4.3).

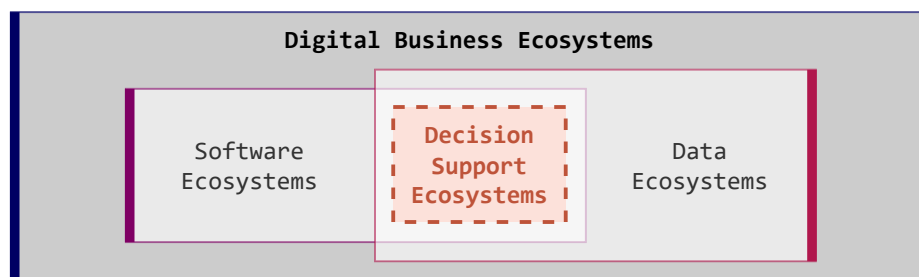


Figure 4.3: Categorization of Decision Support Ecosystems

4.2.2 Relation to the Challenges of Tailored DSS Development

The motivation behind the use of an ecosystem approach for tailored DSS development lies in the alignment of previously presented ecosystem characteristics and benefits with the three challenges for tailored DSS development presented in Section 1.3. By closely aligning the definition of a decision support ecosystem with the definition of a digital business ecosystem in the previous subsection, a DSE can be expected to address the challenges of tailored DSS development with similar characteristics and benefits:

Challenge 1 – Lack of Customization is immediately addressed with the *mass customization* benefit of (digital business) ecosystems. By establishing a cross-organizational *network of actors*, the availability of decision analysis functionality and other artifacts available for integration into a tailored DSS is increased. The *shared digital platform* furthermore facilitates the discovery of decision support functionality and artifacts. As a result, the number of decision processes and associated individual requirements for decision support that can be supported with a tailored DSS is significantly increased, and consequently, the efficiency and effectiveness of decision making is increased as well.

Challenge 2 – Lack of Independency is (partially) addressed by the introduction of the *service broker* as an intermediate role between service providers (e.g., traditional DSS

developers who develop decision analysis functionality), and end users (e.g., decision makers who utilize the decision support functionality). This introduces a separation of concerns that allows developers to focus on the development of decision support functionality while the dedicated service brokers can focus on the interaction with decision makers and composition of ecosystem services, thereby potentially improving the communication between stakeholders and consequently *efficiency* and effectiveness of tailored DSS development. However, to fully address *Challenge 2 – Lack of Independency* and the associated shortage of trained programmers, the composition of services should be possible without programming skills, potentially even by decision makers themselves in the sense of end-user development. The support for such a development by non-programmers largely depends on the technical implementation of the *shared digital platform*.

Challenge 3 – Lack of Coordination is addressed by the *socio-technical network* that is established between ecosystem participants based on the *shared digital platform*. The platform allows ecosystem participants to interact and exchange feedback, e.g., regarding provided decision support functionality or data, which results in improvements of ecosystem services and consequently *accelerated innovation*. As a result, decision makers profit from the integration of proven as well as innovative ecosystem services in their tailored DSS. The socio-technical network between ecosystem participants and associated symbiosis also enables a *self-organizing co-evolution* such that ecosystem participants can profit from regular updates to their tailored DSS in case of available innovations to guarantee decision-making effectiveness and efficiency over time.

4.2.3 Complexity of DSE Implementation

The previous subsection explains the potential of an ecosystem approach to address the challenges of tailored DSS development presented in Chapter 1. Throughout the discussion of (expected) ecosystem benefits, the shared digital platform in particular has emerged as a fundamental necessity to address all three challenges of tailored DSS development as it enables the integration of ecosystem services and facilitates the symbiosis of ecosystem participants. However, the conceptual explanations so far provide only limited guidance for implementors to establish a decision support ecosystem by implementing its shared digital platform. In addition to an absence of technical implementation details, the DSE definition leaves antecedent questions unanswered that influence platform design, e.g., *Which stakeholder roles participate in a decision support ecosystem, and what are (potential) relationships between roles?*, *Which activities are performed by stakeholder roles?*, and *What kind of services and other contributions are brought into the ecosystem by stakeholders?*

Because the shared digital platform is essential to the success of any ecosystem, there is a danger of being unable to achieve the previously discussed benefits for tailored DSS development with a decision support ecosystem due to poor design decisions when developing the shared DSE platform. Implementors working on the realization of a DSE should therefore be provided with guidelines that assist them in the implementation of a DSE platform. Unfortunately, such validated guidelines, e.g., in the form of a reference architecture, are even largely unavailable for the older and more established concept of digital business ecosystems (cf. [SLE19; LBB12; OBF19]). Therefore, the remainder of this chapter and subsequent chapters provide conceptual and practical insights on the implementation of the shared digital platform of a decision support ecosystem.

4.3 DSE Design Principles

Design principles describe “fundamental propositions that aid designers in achieving a successful transfer of requirements to design” [MGO20]. Consequently, DSE design principles provide implementors with guidelines to implement the shared digital platform of a decision support ecosystem and subsequently achieve the benefits of the DSE concept for tailored DSS development discussed in the previous section. Since design principles are fundamental design propositions, the five design principles presented in this section can be viewed as a conceptual framework for more detailed (architectural and technical) explanations throughout the upcoming sections and chapters. Considering the approaches for design principle derivation summarized by Möller, Guggenberger, and Otto [MGO20], the subsequently presented design principles are derived as a response to the solution requirements for tailored DSS development motivated in Section 3.1 while considering the advantages and disadvantages of related approaches discussed in Section 3.3.

Table 4.1 documents the relationship between the subsequently explained design principles and the requirements for tailored DSS development of Section 3.1. In the table, a filled circle represents the expected fulfillment of a requirement by a design principle, while a partially filled circle represents that a design principle is expected to support the fulfillment of a requirement but cannot fully address it on its own. An empty circle represents that the influence of a design principle is dependent on a specific technical implementation and cannot be determined on a conceptual level. The last column indicates that the aggregation of all design principles into the holistic DSE concept is expected to address all solution requirements. Nevertheless, the subsequent explanation of relations between design principles and solution requirements is only argumentative at this point and details as well as the feasibility of the design principles still need to be developed and validated throughout the thesis.

Requirement	DP1 – Generation	DP2 – Svc.-Orient.	DP3 – MDD	DP4 – Assistance	DP5 – Knowl. Base	Coverage
<i>R1 – Situativity</i>						
<i>R1.1 – Situational Decision Problems</i>	●	○	○	○	○	✓
<i>R1.2 – Situational Resources</i>	●	○	○	○	○	✓
<i>R1.3 – Situational Competences</i>	●	○	○	○	○	✓
<i>R2 – Process Orientation</i>						
<i>R2.1 – Modularity</i>	◐	●	○	○	○	✓
<i>R2.2 – Navigation</i>	●	◐	○	○	○	✓
<i>R2.3 – Unified Execution Environment</i>	●	○	○	○	○	✓
<i>R3 – Variety</i>						
<i>R3.1 – Reusability</i>	○	●	○	○	○	✓
<i>R3.2 – Extensibility</i>	○	●	○	○	○	✓
<i>R3.3 – Discoverability</i>	○	●	○	◐	○	✓
<i>R4 – Suitability for Non-Programmers</i>						
<i>R4.1 – Abstraction</i>	○	○	●	○	○	✓
<i>R4.2 – Learnability</i>	○	○	●	◐	○	✓
<i>R4.3 – Error Prevention</i>	○	○	◐	●	◐	✓
<i>R5 – Collaboration</i>						
<i>R5.1 – Common Terminology</i>	○	○	◐	○	●	✓
<i>R5.2 – Artifact Sharing</i>	◐	●	○	○	○	✓
<i>R5.3 – Experience Sharing</i>	○	○	○	◐	●	✓
<i>R5.4 – Organizational Scalability</i>	○	●	○	○	◐	✓
<i>R6 – Compliance</i>						
<i>R6.1 – Transparency</i>	◐	○	●	○	○	✓
<i>R6.2 – Determinism</i>	●	○	○	○	○	✓
<i>R7 – Domain-Portability</i>						
	○	○	○	○	●	✓

●: addresses requirement, ◐: supports addressing of requirement, ○: no contribution

Table 4.1: Mapping between design principles and requirements of Section 3.1

Design Principle DP1 – Generation

The evaluation of existing approaches in Section 3.3 shows that DSS generators can consider all situational factors summarized in *R1 – Situativity*. This is due to the fact that the selection and composition of decision support artifacts are done by a human implementor who can consider the holistic decision situation of a decision maker. On the contrary, automated adaptation or composition of decision support artifacts is often limited to a subset of situational factors. A DSS generator also addresses *R2 – Process Orientation* since it combines multiple decision support artifacts (*R2.1 – Modularity*) into a holistic DSS (*R2.3 – Unified Execution Environment*) that invokes and utilizes the artifacts in alignment with the decision process of a decision maker (*R2.2 – Navigation*). The combination of decision support artifacts also supports *R5.2 – Artifact Sharing*, but not necessarily across organizational boundaries. Aside from innovations, the implementor using the DSS generator can also be expected to select the same decision support artifacts for the same decision support requirements, thereby addressing *R6.2 – Determinism*. The selections and reasons for selection can also be documented for *R6.1 – Transparency*, however, potentially separate from the DSS itself.

Design Principle DP2 – Service-Orientation

The introduction of service-oriented architectures in Section 3.2.1 and evaluation of service-oriented DSS approaches in Section 3.3 highlight the strength of these approaches to address *R3 – Variety*. Potentially platform-dependent implementation details are hidden behind public interfaces and accessed via platform-independent communication protocols based on HTTP, thereby supporting the integration of already existing decision support artifacts (*R3.1 – Reusability*). Service registries collecting descriptions of all available services support the discovery of existing services (*R3.3 – Discoverability*) and the addition of new services (*R3.2 – Extensibility*). Since multiple interoperable services are usually combined into an executable composition to achieve the desired functionality for a business process, service orientation provides the foundation for *R2.1 – Modularity* and *R2.2 – Navigation*. With the goal to promote reusability, service orientation (and specifically service registries) furthermore can address *R5.2 – Artifact Sharing* and *R5.4 – Organizational Scalability*, although – as discussed during Section 4.1.2 – the latter requires additional requirements such as *R5.1 – Common Terminology* to be addressed for cross-organizational collaboration.

Design Principle DP3 – Model-Driven Development

Although not every model-driven approach is able to address *R4 – Suitability for Non-Programmers*, the discussion of low-code development in Section 3.2.2 shows that it is fundamentally possible to design model-driven approaches suitable for software development of tailored applications by non-programmers. In particular, a suitable modeling approach is

both human-readable for specifying DSS functionality (*R4.2 – Learnability*) and machine-readable for generating an executable DSS software artifact (*R4.1 – Abstraction*). Learnability is further increased by using a (familiar) graphical modeling notation. *R4.3 – Error Prevention* can be partially addressed by validating models against the associated meta-model and by using validated model-to-code transformations to guarantee the syntactical correctness of the generated program code, but additional model verification on a semantic level usually requires the separate implementation of model checking. Models can also serve as documentation to address the shortcoming of *DP1 – Generation* for *R6.1 – Transparency*. This advantage is even increased when model-driven development is combined with end-user development since decision makers know the models they created and why. The underlying meta-model for a specified model can provide a foundation for standardized information exchange (*R5.1 – Common Terminology*), but not necessarily for domain-specific entities.

Design Principle DP4 – Assistance System

In the context of a DSS generator where multiple decision support artifacts are composed together, a human implementor may accidentally introduce errors such as selecting artifacts that are incompatible with each other or with the requirements for decision support of a decision maker. An assistance system can support a human implementor in identifying and avoiding such errors during DSS design, thereby improving effectiveness and efficiency during DSS development (*R4.3 – Error Prevention*). By identifying and potentially even providing corrections for syntactic and semantic errors, the assistance system may teach decision makers how to avoid certain errors in the future, thereby improving *R4.2 – Learnability*. Since *DP2 – Service-Orientation* suggests the existence of a service registry that documents all available decision support artifacts, the assistance system could even simplify *R3.3 – Discoverability* by highlighting those artifacts that best align with the requirements for decision support of a decision maker. Lastly, the assistance system can also facilitate *R5.3 – Experience Sharing* if the feedback provided by decision makers and other stakeholders can be incorporated into the computation of suggestions by the assistance.

Design Principle DP5 – Knowledge Base

By documenting fundamental application domain knowledge in a knowledge base accessible to all ecosystem participants, a shared understanding of the application domain entities, their relations, and associated decision-making tasks is established (*R5.1 – Common Terminology*, *R5.4 – Organizational Scalability*). In addition to this fundamental application domain knowledge, participants can also contribute experience from previous tailored DSS development projects to the knowledge base (*R5.3 – Experience Sharing*). This knowledge can be manually accessed by DSS implementors, or as already discussed throughout *DP4 – Assistance System*,

potentially be enforced automatically by the assistance system to avoid errors in the DSS design (R4.3 – *Error Prevention*). By exchanging the knowledge base for other application domains, it is possible to apply the DSE concept in multiple domains (R7 – *Domain-Portability*).

4.4 DSE Constituents

The remainder of this chapter focuses on the refinement of the previously presented design principles into a reference architecture for the shared DSE platform to further assist stakeholders in the implementation of a DSE. Before the reference architecture is presented in the next section, this section provides background information to support the upcoming explanations of the reference architecture. In particular, this section describes potential ecosystem services provided within a DSE (Section 4.4.1), roles and their interactions within a DSE (Section 4.4.2), and a lifecycle of activities throughout a DSE (Section 4.4.3).

4.4.1 Types of Decision Support Services

Based on the DSE definition in Section 4.2.1, participants of a DSE can provide any technical and non-technical contribution that supports tailored DSS development. Consequently, an ecosystem service in the context of a DSE is fundamentally any artifact or human activity that supports the decision process of a decision maker. Certain types of ecosystem services can be identified as visualized in Fig. 4.4. Fundamentally, a *decision support service* can be a *functional service* providing any decision support functionality for the implementation of an activity in the decision process using an *input-processing-output (IPO)* pattern, or a *data service* providing data that is consumed by a functional service for the implementation of a decision support activity. Functional services can further be refined into *automated services* that require no further input from the decision maker to provide decision support, and *interactive services* that require active involvement of the decision maker.

Examples of automated services for energy distribution network planning include a power flow analysis algorithm or an optimization model for network topologies. These are examples of a *software-based service*, which combines a software artifact and the associated execution infrastructure. The combination is done in alignment with the reusability concept of service-oriented architectures (cf. Section 3.2.1 and *DP2 – Service-Orientation*) as it makes the combined service platform-independent and improves integration with other services. Another example of an automated service is the offering of an agency to create multiple scenarios forecasting the energy demands of consumers as an example for a *delegated service*. Although the agency may internally require human effort to realize the offering, the service is

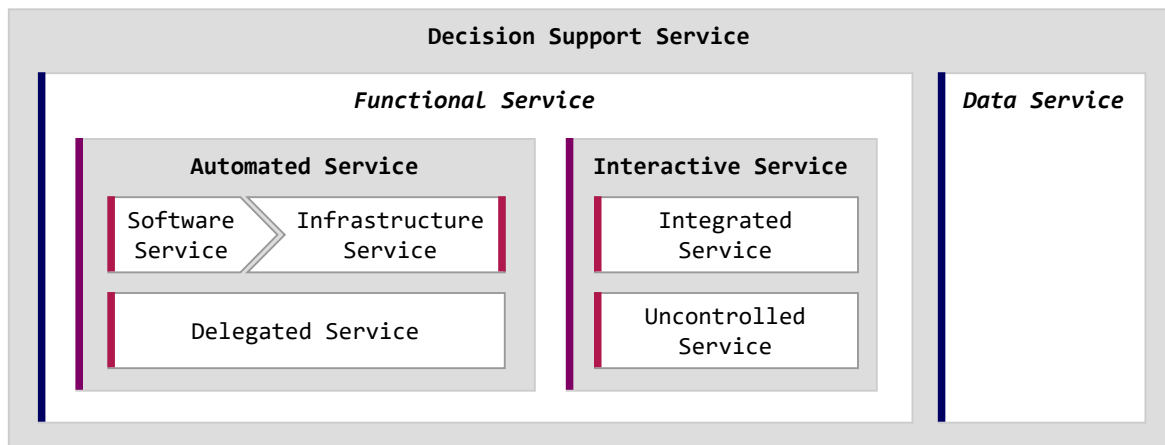


Figure 4.4: Types of decision support services in a DSE

nevertheless automated from a decision maker’s point of view if no further input from the decision maker is required during the execution of the activity.

An example of an interactive service is a service associated with a user interface where decision makers can input their own assumptions about the development of customers’ energy demands. This is an example for an *integrated service*, as the user interface should be included in the tailored DSS. To account for decision support functionality provided by software applications that are not yet available as an integrated service, the *uncontrolled service* is provided as a fallback that describes how decision makers can use external tools for supporting an activity in the decision process.

4.4.2 DSE Roles

This section describes six roles that participate in a DSE and are assumed by human actors. The automated assistance system providing recommendations for the composition of decision support services can also be viewed as an interactive actor. However, it does not contribute novel artifacts or knowledge into the DSE but instead improves the quality of a tailored DSS by enforcing existing knowledge. It is therefore not considered as an additional role. The contributions of the human-actor roles in the context of a DSE as well as the relationships between the roles are summarized in Fig. 4.5 and subsequently explained in detail.

Six DSE Roles

Since the responsibility of a **platform provider**, i.e., maintaining the ecosystem’s shared digital platform, is virtually identical between a decision support ecosystem and a digital business ecosystem (cf. Section 4.1), the role of the platform provider is not explained again.

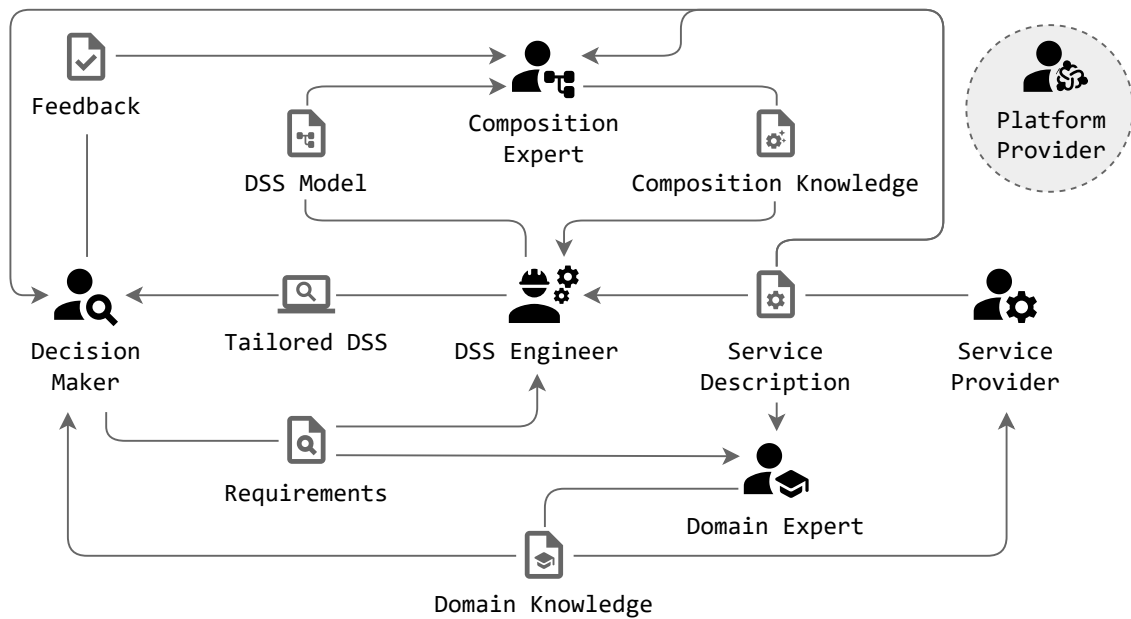


Figure 4.5: Overview of DSE roles and their interactions

For simplification, it is separated from the other roles in Fig. 4.5 to indicate that the role must exchange artifacts, knowledge, and feedback with all other roles.

Decision Maker A decision maker is a person for whom a tailored DSS is developed. Each decision maker has (partially) individual requirements for decision support that are influenced by the decision situation of the decision maker and introduce the need for the development of a tailored DSS. After using a specific tailored DSS, decision makers can evaluate the suitability of the DSS with respect to their situational requirements for decision support and provide corresponding feedback to other DSE participants.

Domain Expert A *domain expert* is a person with substantial knowledge of decision making in a concrete application domain where a DSE is employed. This knowledge includes entities of the application domain with their attributes and relationships as well as knowledge about decision-making tasks including prerequisites of decision alternatives and decision criteria with their susceptibility to influences of situational factors (cf. Section 2.2). The domain knowledge of domain experts is fundamentally important for all other roles, however, Fig. 4.5 focuses on the most important interactions of domain experts with decision makers and service providers for better clarity. The domain knowledge of domain experts helps decision makers to identify and specify their requirements for decision support. Domain experts also need to be informed about novel requirements of decision makers to account for changes in the business environment. Analogously, the domain knowledge of domain experts helps service

providers to specify the contributions of their existing decision support services and to identify the demand for novel decision support services. Simultaneously, they need to inform domain experts about novel services due to changes in the business environment.

Service Provider A *decision support service provider* (subsequently referred to as *service provider* for brevity) contributes one or multiple decision support services encapsulating any artifact or activity to support part of a (tailored) decision process. Considering the types of decision support services discussed in the previous Section 4.4.1, the role of the service provider could be further refined into *data provider*, *software provider*, *infrastructure provider* etc., however, this distinction is not used throughout this thesis. While most services will be used by the subsequently explained DSS engineer, service providers may also advertise data services to decision makers. A service provider either identifies services based on existing decision support capabilities or designs and develops services from scratch. The latter is important to address novel requirements for decision support of decision makers due to changes in the business environment. Identification of such changes requires a service provider to monitor the application domain captured in the form of domain knowledge by domain experts. The domain knowledge also facilitates the documentation of service characteristics in the form of a service description to make the service accessible for composition into a tailored DSS.

DSS Engineer A *DSS engineer* models the composition of decision support services contributed by service providers into a tailored DSS that is subsequently used by a decision maker. In the case of end-user development, the role of the DSS engineer is assumed by the decision maker and would not warrant a separate role. However, the role of the DSS engineer could also be assumed by a data analyst or subject matter expert (cf. Section 2.2) without programming skills but with an interest to provide a decision maker with a tailored DSS. For the development of a tailored DSS, the DSS engineer needs to know about the requirements for decision support of a decision maker, the available decision support services contributed by service providers (with a focus on functional services), and ideally also recommendations on how to select and compose these services for different decision situations.

Composition Expert A *composition expert* aggregates experiences with tailored DSS development and usage into composition knowledge that is used by the DSS engineer to improve future compositions of decision support services. The composition expert receives feedback from decision makers regarding their experiences with a tailored DSS. This feedback is specified for (and therefore includes) the decision maker's requirements for decision support that motivated the need for tailored DSS development. Additionally, composition experts need the associated *DSS model* created by the DSS engineer describing the composition of decision support services used for the generation of the tailored DSS, and the descriptions of

decision support services themselves. The resulting composition knowledge extracted by the composition expert is then provided to the DSS engineer, most likely via the assistance system supporting the DSS engineer during service composition.

Remarks on DSE Roles

The previous explanations often use the singular form when referring to a role. Nevertheless, a role can (and should) be assumed by multiple actors to benefit from heterogeneous requirements and offerings for decision support contributed by decision makers and service providers as well as improved knowledge gathering by multiple experts.

In addition to this one-to-many relationship between a role and actors assuming the role, there is also a converse many-to-one relationship, i.e., the roles are non-exclusive and one actor can assume multiple roles. As already suggested during *DP3 – Model-Driven Development* and repeated during the explanation of the DSS engineer, it may even be desirable that the responsibilities of the DSS engineer are assumed by the decision maker. Decision makers and service providers are likely also good domain experts as their requirements and offerings for decision support shape the application domain with respect to decision making. Nevertheless, a strict separation between roles may be desirable to ensure scalability as the ecosystem grows with respect to the participating organizations and individuals.

The roles of domain expert and composition expert fundamentally serve a similar purpose, i.e., to improve the quality of ecosystem activities and to enable ecosystem evolution. However, the roles have different perspectives: While the domain expert focuses on evolution triggered by changes in the surrounding business environment, the composition expert focuses on evolution triggered by interactions within the ecosystem itself.

4.4.3 DSE Lifecycle

This subsection expands on the role descriptions of DSE participants by describing a lifecycle that establishes an order on participants' activities to fulfill the aforementioned responsibilities. The lifecycle is summarized in Fig. 4.6 and subsequently explained in more detail.

Foundation The goal of the foundation phase is to provide the basis for the subsequent activities of the lifecycle. Throughout *domain documentation*, domain experts collect application domain knowledge, either based on their own perspective of the application domain or based on interactions with other DSE participants. The documented domain knowledge is then available for service providers to describe their decision support services with respect to characteristics of the application domain during *service registration*, thereby making the services available to other DSE participants.

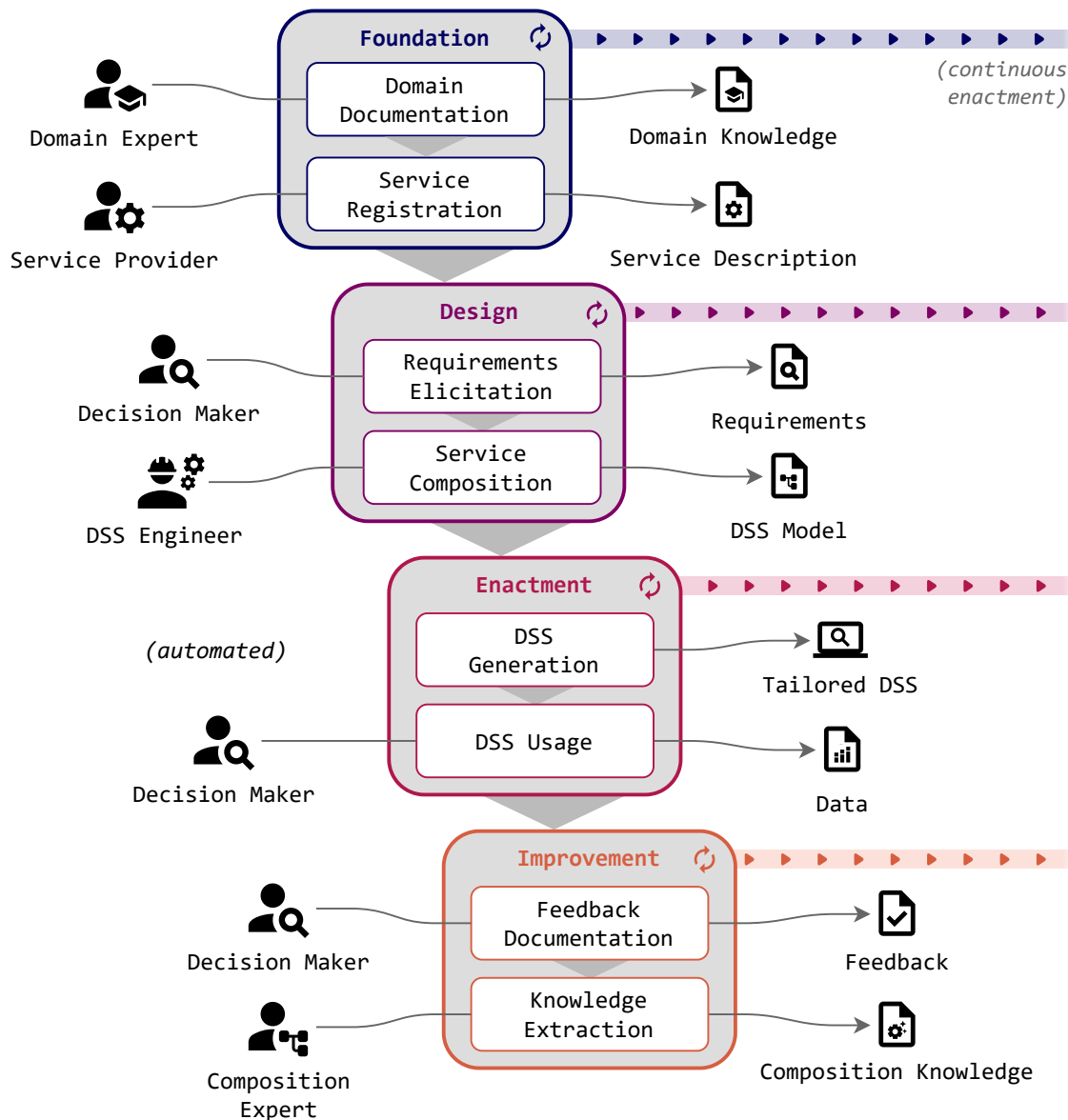


Figure 4.6: Overview of the continuous DSE lifecycle with involved actors and produced outputs

Design The goal of the design phase is to model a tailored DSS that addresses the individual decision support requirements of a decision maker. For this purpose, decision makers first document their requirements for decision support with respect to the previously formalized characteristics of the application domain throughout *requirements elicitation* to establish a common understanding between with DSS engineers. Next, during *service composition*, a DSS engineer selects a subset of available decision support services and models their integration into a holistic tailored DSS. During composition, the DSS engineer can utilize best practices documented in the form of composition knowledge.

Enactment The goal of the enactment phase is to provide decision makers with effective decision recommendations in alignment with their individual decision processes. The first activity of the phase is *DSS generation* where the tailored DSS is automatically generated from decision support services as prescribed by the previously created DSS model. Generation is the prerequisite for the subsequent *DSS usage* where the decision maker interacts with the generated DSS and provides as well as receives data.

Improvement The goal of the improvement phase is to gather experience that can improve efficiency and effectiveness during future tailored DSS development. For this purpose, decision makers first capture feedback regarding the tailored DSS during *feedback documentation*. This feedback, in addition to the associated DSS model created by a DSS engineer, is then analyzed by the composition expert throughout *knowledge extraction* and documented as composition knowledge to be considered during the design of future service compositions.

Although the previous explanations suggest a waterfall-like enactment of the lifecycle, this at most applies immediately after the initiation of the DSE when the problem and solution space have not yet been described and ecosystem participants have not yet gathered experience from which composition knowledge can be extracted. Afterwards, the enactment of lifecycle phases and activities likely happens continuously and in parallel as indicated in Fig. 4.6. For example, the experiences of decision makers may trigger both the extraction of new composition knowledge and an update of the domain documentation if changes in the business environment cannot be mapped to existing DSE capabilities.

4.5 Reference Architecture for a DSE Platform

This section aggregates the explanations of previous sections into a reference architecture for a shared DSE platform to further assist the implementation of a DSE. The reference architecture is depicted in Fig. 4.7 and closely aligned with the lifecycle explained in Section 4.4.3, i.e., every lifecycle phase is an architectural layer and every activity is often directly translated into an interactive software component within the architecture. Given these commonalities and to avoid redundancy, the explanations in this section primarily focus on the exchange of artifacts throughout DSE components. The explanations furthermore map the architectural components to the design principles introduced in Section 4.3.

Foundation Layer

The entry point into the architecture is the *Domain Documentation* component where domain experts characterize decision making in the application domain. The structured domain

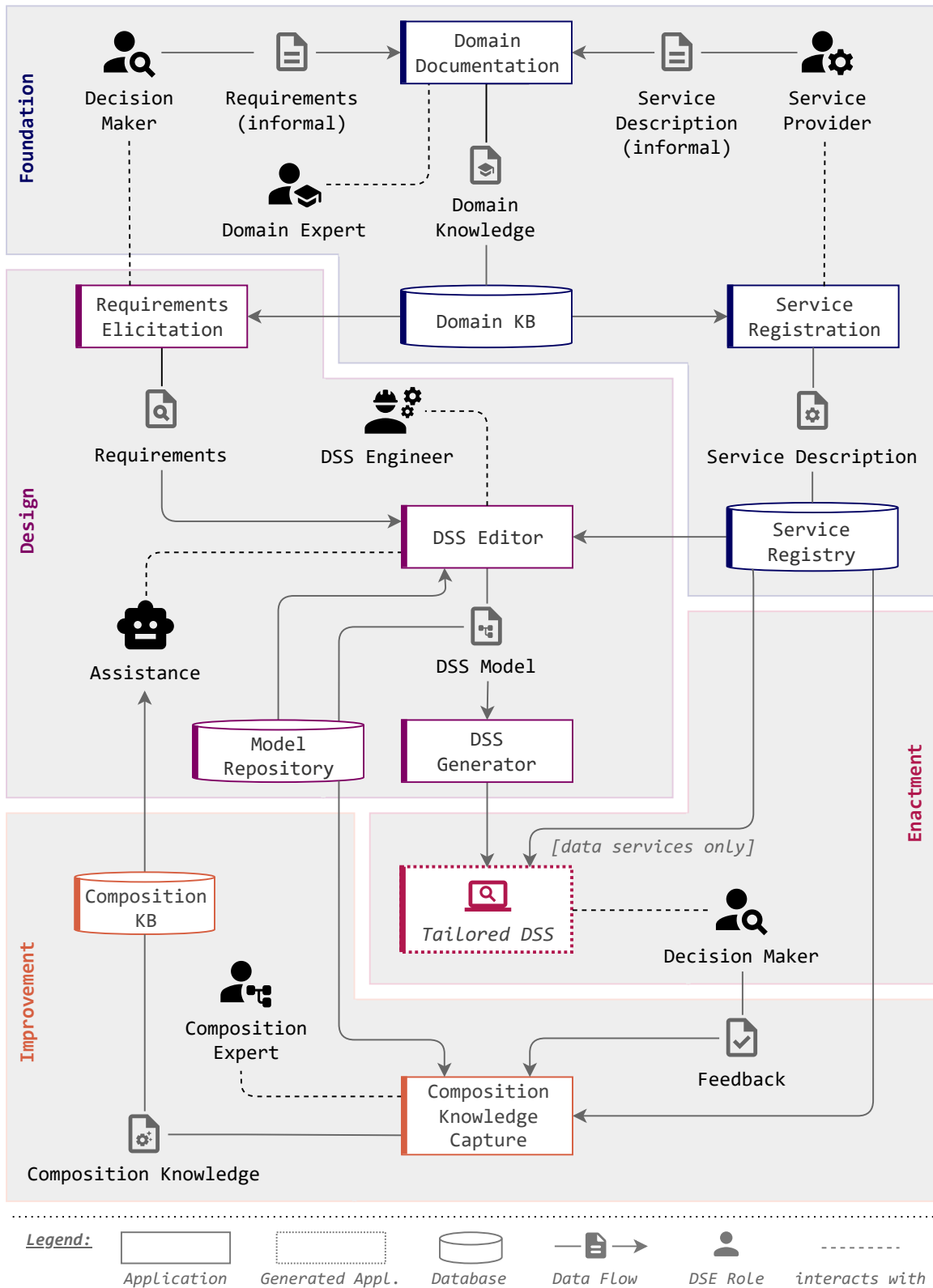


Figure 4.7: Reference Architecture for a DSE Platform

knowledge is stored in the *Domain Knowledge Base* where it is subsequently used for the structured documentation of decision makers' requirements for decision support and service characteristics. Since the structured documentation of requirements and services depends on the structured domain knowledge, domain experts must initially rely on informal documentation of requirements and service characteristics during the documentation of the application domain. The domain knowledge base partially addresses *DP5 – Knowledge Base* with a focus on requirements *R5.1 – Common Terminology* and *R7 – Domain-Portability*.

Service providers make their decision support services available within the DSE using the *Service Registration* component. During registration, they describe the characteristics of their services based on the structured application domain knowledge. This results in a structured service description that is published in the *Service Registry* to facilitate service discovery. The service registry primarily addresses *DP2 – Service-Oriented* with a focus on requirements *R2.1 – Modularity*, *R3.1 – Reusability*, *R3.2 – Extensibility*, *R3.3 – Discoverability*, *R5.2 – Artifact Sharing*, and *R5.4 – Organizational Scalability*.

Design Layer

Decision makers use the *Requirements Elicitation* component for structured documentation of their situational requirements for decision support based on the structured domain knowledge documentation. A structured documentation of requirements is a prerequisite for future processing of the requirements, both to enable a common understanding with other ecosystem participants and also for automated processing throughout later components.

Such automated processing happens as part of the *DSS Editor* component where the DSS engineer composes decision support services with the support of an *assistance* to address the situational requirements for decision support of a decision maker. The output of the *DSS Editor* is the *DSS Model* describing the tailored DSS for the given decision support requirements. The DSS model can be stored in the *Model Repository* for future reuse. The DSS editor in combination with the assistance primarily addresses *DP3 – Model-Driven Development* and *DP4 – Assistance System* with a focus on requirement *R4 – Suitability for Non-Programmers*, and also *R6.1 – Transparency* when the role of the DSS engineer is assumed by the decision maker in the context of end-user development. The DSS editor furthermore partially addresses *DPI – Generation* with a focus on requirements *R1 – Situativity* and *R2.1 – Modularity*.

Enactment Layer

The *DSS Generator* receives the DSS model previously designed by the DSS engineer and generates the tailored DSS from the described service composition. Although the tailored DSS

is shown as a separate application in Fig. 4.7, the DSS specification could be processed either by a code generator or an interpreter similar to the approaches discussed in Section 3.3.2. The tailored DSS is subsequently available to the decision maker who can interact with the DSS and select additional data services from the service registry. The *DSS Generator* partially addresses *DPI – Generation* with a focus on requirement *R2 – Process Orientation*.

Improvement Layer

The *Composition Knowledge Extraction* component is used by the composition expert to extract and document composition knowledge. The component relies on previously designed DSS models stored in the *Model Repository* and the descriptions of utilized services. In addition to the identification of frequently occurring patterns within models throughout the repository, the composition expert can also consider feedback for specific compositions provided by decision makers. The structured composition knowledge is stored in the *Composition Knowledge Base* where it is accessed by the automated *Assistance* to support the DSS engineer during DSS design with the DSS editor. The knowledge base partially addresses *DP5 – Knowledge Base* with a focus on requirements *R5.3 – Experience Sharing* and *R4.3 – Error Prevention*.

Relation to Low-Code Development Platforms

The previous description of the DSE platform reference architecture shows similarities to the fundamental architecture of low-code development platforms (LCDPs) previously presented in Section 3.2.2. The relation between components is summarized in Table 4.2.

The *Domain Documentation* and *Domain Knowledge Base* components of a DSE platform correspond to the *Domain Modeling* component of LCDPs, with the difference that domain experts define reusable data structures for the whole DSE and not just for a single application in the case of LCDPs. While the DSE platform has no component for the assembly of graphical user interfaces from primitives such as buttons or text input fields similar to the *GUI Design* component of LCDPs, the user interface of a DSS is nevertheless influenced by selecting among (alternative) interactive decision support services when specifying the DSS using the *DSS Editor* component. The selection and composition of decision support services using the editor also corresponds to the *Business Logic Specification* component of LCDPs.

The *Data Integration* and *Service Integration* components of LCDPs are implemented in cooperation between a DSE platform's *Service Registry* and *DSS Editor* components, which support the integration of functional and data decision support services.

The *Model Repository* of the DSE platform providing reusable DSS models can be compared to the *Application Templates* component of LCDPs. The *Service Registry* of a DSE

platform corresponds to the *Connector Marketplace* of an LCDP as it enables and promotes the reuse of (external) services.

The *DSS Generator* of a DSE platform corresponds to the *Code Generation* component of an LCDP. In the DSE platform, the functionality of the LCDP *Deployment* component is assumed to be contained in the *DSS Generator* component. This is due to the DSE concept not explicitly requiring the execution of the DSS in a cloud environment – the *DSS Generator* could generate and deploy web applications or native desktop applications (although a web-based deployment is potentially more beneficial for low-barrier DSE adoption).

The reference architecture for a DSE platform furthermore defines components without a counterpart in the fundamental LCDP architecture. These components include the *Requirements Documentation* component and experience exchange using the *Composition Knowledge Capture* and *Composition Knowledge Base* components. Furthermore, although the *Assistance* is not an application that DSE participants directly interact with, it is nevertheless an automated application triggered by other components (similar to code generation).

The comparison demonstrates that a DSE platform can be viewed as a low-code development platform with some extensions to the fundamental LCDP architecture as well as a dedicated focus on the development of tailored decision support systems.

Table 4.2: Comparison between LCDP and DSE platform components

LCDP Component	DSE Platform Component
Data Modeling	Domain Documentation and Domain Knowledge Base
GUI Design	DSS Editor *
Business Logic Specification	DSS Editor
Data Integration	Service Registry and DSS Editor
Service Integration	Service Registry and DSS Editor
Application Templates	Model Repository
Connector Marketplace	Service Registry
Code Generation	DSS Generator
Deployment	DSS Generator *
-	Requirements Elicitation
-	Composition Knowledge Capture and Knowledge Base
-	Assistance

* : closest mapping, see textual explanations for details

4.6 Key Takeaways

A decision support ecosystem is a variant of a digital business ecosystem in which ecosystem participants collaborate to provide each decision maker with a tailored DSS for effective and efficient decision making. A key enabler of a DSE is its shared digital platform, which is used by ecosystem participants to publish and integrate various decision support services into a tailored DSS. Although the shared digital platform is essential to address the challenges of tailored DSS development presented in Chapter 1, there is a lack of design knowledge that guides implementors during the realization of a DSE and its platform. For this reason, the chapter presented five fundamental design principles to define a conceptual framework for the development of a DSE platform. Furthermore, the chapter presented the types of decision support services contributed to a DSE, the six roles of actors participating in a DSE, and a lifecycle that describes the activities of ecosystem participants. In summary, domain experts and service providers characterize the problem and solution space for decision support in an application domain, DSS engineers model a tailored DSS as a service composition with support from an assistance system, and decision makers use the generated DSS and subsequently provide feedback to composition experts who extract best practices for future DSS development. These presented insights were subsequently aggregated into a reference architecture to further guide the implementation of a DSE platform. Based on the design recommendations, the conceptual definition of a DSE provided in Section 4.2.1 can be refined into the following design definition:

Decision Support Ecosystem – Design Definition

A decision support ecosystem (DSE) is an evolving socio-technical network with collaboration between decision makers, service providers, DSS engineers, and experts who contribute requirements, decision support services (including software, data, computing infrastructure, and other offerings), and best practices for decision support via a shared digital platform that enables the assisted low-code development of tailored decision support systems.

Despite the reference architecture, there is still a lack of conceptual and technical details regarding the implementation of architecture components or the format of artifacts exchanged between DSE participants. The subsequent chapters address this knowledge gap by explaining how to capture required and provided decision support for a concrete application domain (Chapter 5), how to model and generate a tailored DSS (Chapter 6), and how to describe and enforce composition knowledge during DSS design with an assistance system (Chapter 7).

Description of Decision Support Services

This chapter presents an approach for the structured description of decision support artifacts (i.e., data and functionality for data transformation, simulation, optimization, and visualization) to document both required and provided decision support services for automated processing in subsequent DSE lifecycle activities. The presented insights extend the paper “Requirements-Based Composition of Tailored Decision Support Systems” by Kirchhoff, Weskamp, and Engels [KWE22b] using modeling concepts introduced in the paper “Detecting Data Incompatibilities in Process-Driven Decision Support Systems” by Kirchhoff, Gottschalk, and Engels [KGE22].

As shown in Fig. 5.1, the insights presented in this chapter support the first three activities of the DSE lifecycle, i.e., domain documentation, service registration, and requirements elicitation. For this purpose, the chapter contributes: (1) a meta-model that describes the necessary concepts for the documentation of required and provided decision support in an application domain, and (2) recommendations for the development of a software application that supports the form-based instantiation of meta-model elements by DSE participants.

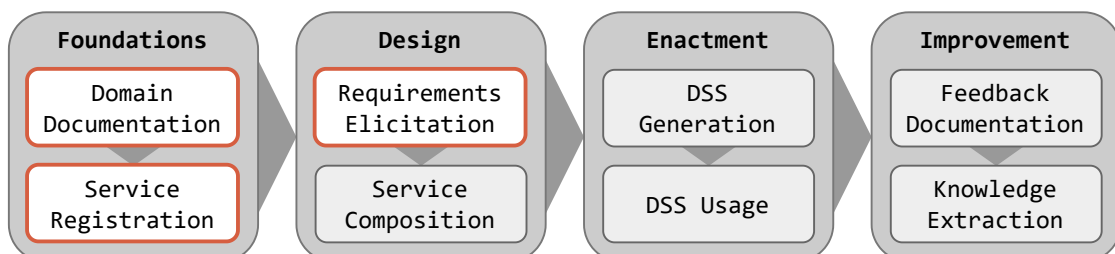


Figure 5.1: Focus of Chapter 5 with respect to the DSE lifecycle

After a presentation of upfront design considerations (Section 5.1), the chapter introduces requirements for a structured documentation of required and provided decision support (Section 5.2) and discusses existing decision support ontologies and service description languages with respect to these requirements (Section 5.3). Next, a meta-model for describing the characteristics of the application domain and associated decision support is introduced and illustrated with examples (Section 5.4). The chapter also presents a concrete syntax and prototypical implementation for the representation and form-based manipulation of meta-model instantiations (Section 5.5). The insights are discussed with respect to the initially presented requirements for the description format (Section 5.6) and summarized (Section 5.7).

5.1 Context and Upfront Considerations

Section 4.4.1 associates a functional decision support service with the *input-processing-output (IPO) principle*, i.e., the service receives input data from the decision maker or a previously executed service, processes it, and returns output data (unless the service solely focuses on interactive data visualization). The same IPO principle can also be observed for the overall decision support system, albeit it only operates on data provided by the decision maker. With respect to the black-box view assumed by the IPO principle, a (tailored) DSS can therefore be viewed as a single fictional/required decision support service. Consequently, a decision maker should be able to document required decision support functionality very similar to service providers documenting provided decision support functionality. This is reflected in the upcoming explanations as well as the architectural overview in Fig. 5.2, where cross-cutting concepts are defined in the *Domain Registry* and reused in the *Service Registry* and *Requirements Elicitation* applications to address the additional needs of service providers and decision makers for the documentation of provided and required decision support.

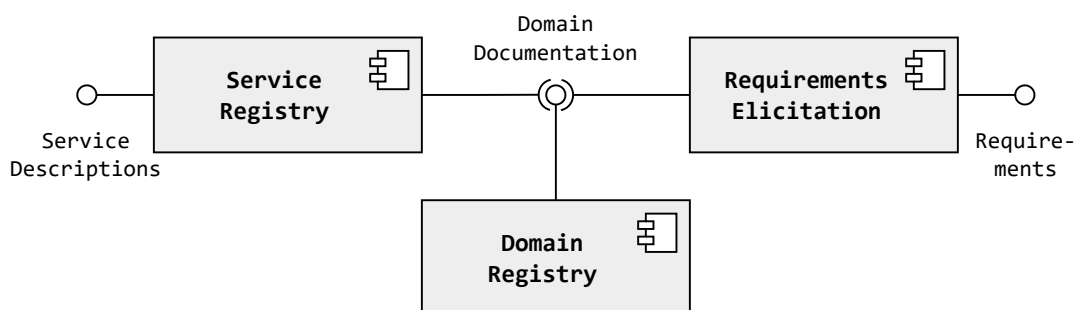


Figure 5.2: Applications for the “Foundation” lifecycle phase as a UML component diagram

5.2 Description Requirements

This section introduces requirements *DRx* for the structured description of required and provided decision support functionality and data. The requirements are primarily derived from the characteristics of (situational) decision making discussed in Chapter 2 and illustrated using examples from the domain of energy distribution network planning.

Domain Perspective

The domain perspective aggregates the requirements of domain experts to provide fundamental information about the application domain as a basis for further descriptions.

Description Requirement DR1 – Types of Domain Data

The description approach must support the documentation of fundamental types of domain data including available serialization formats to describe what kind of data can be processed throughout a decision process, i.e., provided to or by decision support services. Examples in the domain of energy distribution network planning are “network topology” or “consumer demands” with associated (generic) data formats such as XML, JSON, or CSV.

Description Requirement DR2 – Characteristics of Domain Data

The description approach must support the documentation of characteristic attributes for data types in the application domain to enable the specification of additional information regarding decision making for these entities. For example, without knowing that a network topology is associated with operational costs and can be designed in an (n-1)-reliable way, it is not possible to capture that decision makers may want to minimize operational costs without sacrificing a certain level of reliability. Such characteristics can also be useful to describe data constraints of services, e.g., to document that a service only supports network topologies up to a certain size. As suggested by the example, it is sufficient to capture characteristic *metadata* to reduce the effort and complexity of documentation, i.e., the size of the network topology as an integer instead of each network asset included in a topology.

Functional Perspective

The functional perspective aggregates the requirements of decision makers and service providers to describe their required/provided decision support functionality.

Description Requirement DR3 – Computation Approach

The description approach must support the documentation of the computation approach that is used to implement decision support functionality. A computation approach can be

characterized by its overall goal, e.g., power flow simulation for a network topology, which can be achieved using multiple algorithms, e.g., Newton-Raphson or Gauss-Seidel, each potentially requiring different input data.

Description Requirement DR4 – Optimization Goals

For provided/required optimization functionality, the description approach must support the documentation of optimization goals defining the validity and optimality of a decision recommendation (cf. decision criteria in Section 2.2). Otherwise, there is a risk of recommending and implementing suboptimal decisions. Optimization goals should be documented with respect to the previously defined data characteristics, e.g., minimizing investment costs of a network topology while ensuring that it is (n-1)-reliable.

Description Requirement DR5 – Decision Alternative Generation

Since decision alternatives are determined throughout the decision process, usually as part of an optimization model, the description approach must support the documentation of the supported/utilized types of decision alternatives. Otherwise, there is a risk of recommending decisions that a decision maker cannot implement, e.g., placing battery storage systems although a DNO is prohibited from operating them.

Non-Functional Perspective

The non-functional perspective focuses on characteristics other than required/provided functionality that can influence whether a decision support service can be utilized or not.

Description Requirement DR6 – Resources

The description approach must document the resources consumed by decision support services as resources are only available to decision makers to a limited extent, and an overextension of available resources must be avoided. For functional services, resource consumption at least includes the execution time of a service. Additionally, and also relevant for data services, a service fee may be charged or a license may be required. Resource consumption may depend on the characteristic of an input provided to the service, e.g., the runtime of an energy network optimization service may be exponential in the size of the network topology.

Description Requirement DR7 – Data Constraints

The description approach must support the documentation of additional constraints on inputs and/or outputs of a functional decision support service to ensure the compatibility of a service with the data provided by the decision maker when using the resulting tailored DSS. These constraints can be interpreted as data requirements for inputs and data guarantees on outputs.

For example, a network optimization service may only support topologies up to a certain amount of assets, or a network reduction service may guarantee a certain aggregation of assets as a percentage of the provided input topology's size.

Description Requirement DR8 – Service Level Objectives

The description approach must support the documentation of service quality in the form of *service level objectives* that describe permissible value ranges on quantitative service characteristics, i.e., *service level indicators*, such as availability or response time [Bey+16, Ch. 4]. In the context of decision support where potentially sensitive data is processed, these characteristics can also include qualitative aspects of the execution environment. For example, it may be desirable to prioritize an in-house service over other services.

Description Requirement DR9 – Usage Information

The description approach must support documentation of how to invoke a decision support service. Additionally, it should be possible to document *exceptions* that can occur during service execution and may prevent the service from producing the promised output. For example, a power flow simulation service may find that a network topology is invalid due to unconnected assets. Documentation of possible exceptions enables the definition of reactions to these exceptions during service composition such as automated fixes or restarts.

Description Requirement DR10 – Access Control

Although collaboration between ecosystem participants is a key benefit of any digital business ecosystem (cf. Section 4.1), some service providers may refrain from making their services available to all ecosystem participants. This may even be required by law for data services, for example, historical energy demands of consumers are subject to data protection regulations. Consequently, it must be possible to restrict service discovery and access.

Integration Perspective

While the previous requirements focus on the expressiveness of the description approach, additional requirements can be derived from the need to integrate the approach into the holistic concept of a decision support ecosystem introduced in the previous Chapter 4.

Description Requirement DR11 – Machine-Readability

The described required/provided decision support services must be documented in a machine-readable way such that the descriptions can be further processed throughout later activities of the DSE lifecycle, e.g., by the composition assistance to validate a service composition.

Description Requirement DR12 – Tool Support

The documentation of required/provided decision support services should be possible using software tools with graphical user interfaces. Compared to exclusively textual service descriptions, a GUI can provide an abstraction that enables more domain experts, service providers, and decision makers to participate in the ecosystem, thereby facilitating its adoption. Tool support can also ensure data integrity by restricting user input to valid values.

Description Requirement DR13 – Human-Readability

Although the previous requirement for tool support suggests that service descriptions do not need to be modeled by hand, a description should still be human-readable to support the development of software tools for subsequent DSE lifecycle activities (e.g., the assistance system for service composition), for example, to assist in debugging.

5.3 Background and Related Work

This section discusses existing approaches for the description of reusable (decision support) functionality to assess their suitability with respect to the previously presented description requirements. Discussed approaches are classified as decision support ontologies (Section 5.3.1) and general-purpose service description languages (Section 5.3.2)

5.3.1 Decision Support Ontologies

An *ontology* documents the vocabulary of an application domain by defining basic concepts and their relationships among each other as well as axioms constraining their interpretation and enabling the derivation of new knowledge [KV11]. Concepts and relationships can be defined on a type-level (*T-Box*, e.g., a person can be married to another person) and instance-level (*A-Box*, e.g., person A is married to person B) [RS13]. Thus, an ontology could potentially include both the concepts that are needed to describe required and provided decision support services as well as descriptions of services and requirements themselves. Considering the description requirements established in the previous section, the subsequent discussion of decision support ontologies focuses on approaches that use an ontology to describe decision support functionality and associated data and does not include approaches that use an ontology to encode data and decision support functionality itself (e.g., [MGK07]).

Rockwell et al. [Roc+09] describe a decision support ontology (DSO) for collaborative product design. The DSO concepts largely align with the decision-making entities introduced in Section 2.2. However, the DSO is primarily designed to capture past product design decisions with respect to associated product requirements, design alternatives, and their

evaluation results. It does not capture (reusable) decision process activities describing how the evaluation results are derived and therefore does not address *DR3 – Computation Approach* in particular as well as any requirement from the “non-functional perspective”.

Rospocher and Serafini [RS13] describe an ontology to capture the decision support request of a user and the data processed and produced by a DSS. The ontology consists of a *problem component* describing all aspects of decision support problems that users can submit to the DSS, a *data component* describing the data consumed by the DSS, and a *conclusions component* describing the data produced by the DSS. However, these components only influence the construction of a decision support ontology for a specific application domain as demonstrated in the paper. This does not align with the reusability of the description approach across domains as suggested by *R7 – Domain-Portability*.

Chai and Liu [CL10] describe an ontology-based approach to decompose a decision problem into tasks that are executed by human actors during group decision making. The vaguely described approach lacks consideration for the software-based execution of tasks, which is specifically considered by the “non-functional perspective” of the previously introduced description requirements. Similarly, Bennani et al. [Ben+18] describe a (high-level) meta-model (comparable to an ontology without knowledge derivation) for describing collaborative group decision making without software support.

Bhrammanee and Wuwongse [BW08] describe an ontology to document optimization models for the model base of a DSS. While this ontology addresses most of the requirements documented under the “functional perspective” in the previous section, it only addresses one requirement from the “non-functional perspective” by enabling the documentation of consumed resources. Similar restrictions can be observed for semantic model management systems discussed in Section 3.3.4 or the approach by Deokar and El-Gayar [DE13].

Zagorulko and Zagorulko [ZZ10] present an approach to provide decision support for reducing power consumption of oil-and-gas production enterprises. Their approach uses a *subject domain ontology* to capture entities of the application domain, and a *task ontology* to describe the functionality of decision support modules. In addition to only addressing requirements from the “functional perspective”, the ontology is domain-specific and uses the model-solver pattern already criticized during the discussion of related work in Section 3.3.

Blomqvist [Blo14] describe the results of a literature survey and expert interviews to identify the potential benefits of using (ontology-based) semantic web technologies for DSS development. However, the summary of expert interviews only mentions the usage of ontologies to capture reusable decision support artifacts as a future outlook.

In summary, none of the discussed decision support ontologies is comprehensive enough to address the description requirements presented in the previous section.

5.3.2 General-Purpose Service Description Languages

While some of the previously described decision support ontologies (partially) address requirements from the “functional perspective” of the description requirements, they especially lack the expressiveness to also address requirements from the “non-functional perspective”. Given this observation, the subsequent discussion focuses on the suitability of existing service description languages to address the description requirements, which traditionally support both perspectives, albeit without a focus on decision support.

A *service description language* supports the documentation of a software service’s characteristics to enable its use in the context of a service-oriented architecture (cf. Section 3.2.1). Nawaz, Mohsin, and Janjua [NMJ19] conducted a systematic review of service description languages and classified them into languages for documenting service deployment and provisioning, modeling and composition, discovery and selection, and service level agreements. However, all listed service description languages are general-purpose languages that address most of the introduced description requirements from the non-functional perspective, but not from the functional perspective with a focus on decision support. The missing support for describing decision support goals also applies to the *OpenAPI Specification (OAS)* [Ope21], which defines a programming-language agnostic standard for describing HTTP APIs. Although not listed in the review by Nawaz, Mohsin, and Janjua [NMJ19] – perhaps due to the poor representation of the OAS in academic publications – it is the service description language with the most practical adoption based on the author’s personal experience. However, reusing the OAS for the description of decision support services would require the semantic redefinition of language elements as well as comprehensive language extensions for the documentation of decision support functionality. For example, the specification supports the description of data types with their attributes, which aligns with *DR2 – Characteristics of Domain Data* at first glance but would result in a semantic misalignment since the OAS expects service providers to describe the actual structure of data. On the contrary, *DR2 – Characteristics of Domain Data* only focuses on the documentation of metadata characteristics (which may not even be explicitly contained within the actual data). Because of these potential misalignments and/or comprehensive extension efforts, the OAS is not further considered.

Of the service-oriented DSS approaches reviewed in Section 3.3.3, Mustafin, Kopylov, and Ponomarev [MKP21] use *OWL-S* [W3C04], an application of the *OWL Web Ontology Language* [W3C12] for the semantic description of web services. Their approach relies on two other ontologies: an application domain ontology to define domain concepts and properties, and a DSS functional blocks ontology to categorize and describe characteristics of services for visualization, data management, model management, and solvers. Unfortunately, the

DSS functional blocks perspective, which corresponds to the functional perspective of the description requirements, is not described in detail. Furthermore, the model-solver approach was already criticized during the evaluation of the overall approach in Section 3.3.3 as it does not consider simulation, statistical computations, or utility functionality such as data conversion. This also holds for the approach by Shafiei, Sundaram, and Piramuthu [SSP12] using the “outdated” *WSDL* (cf. Section 3.2.1) and Dong and Srinivasan [DS13] without details regarding service description. Explanations of the service description format are also missing from the approaches described by Becker et al. [Bec+08] (which is based on *Java Beans*), Stănescu, Ștefan, and Filip [SȘF15] (which uses SOAP and REST for communication, but does not specify a standard for service description) and Axelsson et al. [Axe+17] (whose prototypical implementation suggests that only a service category, name, and URL are stored in the discovery service). Initial experiments within this thesis to utilize the extensibility of OWL-S to address the description requirements showed limitations for automated processing and particularly *DR13 – Human-Readability* due to the verbosity of the underlying *Resource Description Framework (RDF)* [W3C14], which uses a subject-predicate-object structure to describe data and requires specialized databases (“*triple stores*”) for data storage and specialized query languages for data retrieval and manipulation. These disadvantages are often accepted due to the associated advantage of knowledge inference from RDF data using a reasoner. However, there is no indication that such knowledge inference capabilities would benefit the implementation of later DSE components.

In summary, none of the reviewed service description languages are suitable to address the description requirements introduced in the previous section.

5.4 A Meta-Model for Decision Support Services

The previous section shows that existing decision support ontologies and (software) service description languages are only partially suitable to describe required and provided decision support. This section introduces a meta-model as an abstract syntax of a decision support description language to address the description requirements of Section 5.2. The concepts defined by the meta-model aim to strike a balance between a goal-oriented description of decision support suitable for decision makers and DSS engineers, and a solution-oriented description suitable for service providers to highlight unique features of their services.

The meta-model is visualized as a UML class diagram [OMG17] and each class is textually described. Classes are aggregated into the packages shown in Fig. 5.3. The class diagram for each package is accompanied by a UML object diagram to illustrate how these classes would be instantiated in the context of energy distribution network planning.

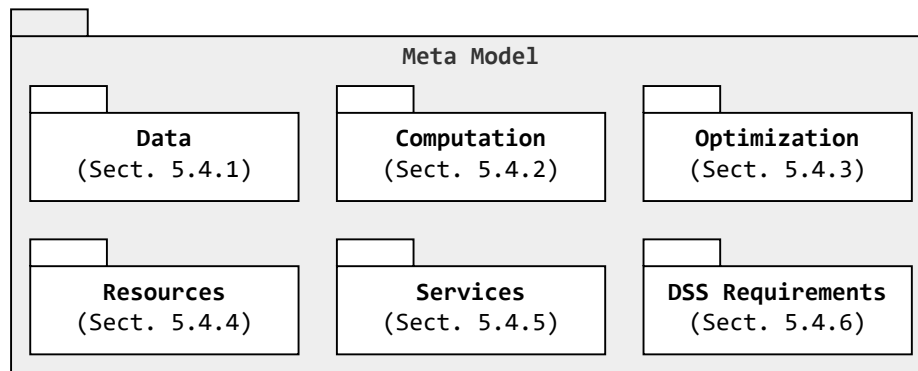


Figure 5.3: Overview of meta-model packages

The class diagrams are subject to the following conventions: Each class has attributes for a human-readable name and description, which are omitted in the diagram for comprehensibility. The default cardinality for the start of an association is $0..*$, for an aggregation $0..1$, and for a composition 1. A composition indicates that instances of the associated class cannot exist without the composing instance. An aggregation indicates that instances of the associated class can temporarily exist without an aggregating instance. This differentiation is primarily done for the subsequent explanations of the prototypical implementation. Previously discussed packages, classes, and objects are shown transparently to distinguish them from newly introduced concepts. For compactness, class names in object diagrams are sometimes abbreviated and not all associated classes are instantiated in the illustrating object diagrams.

5.4.1 Package “Data”

The class diagram for the package Data is shown in Fig. 5.4 with the associated object diagram showing an example instantiation in Fig. 5.5. The goal of the package is to capture the characteristics of data that is utilized throughout a decision process. For this purpose, data is associated with a *DataType* such as *network topology*. A data type can be stored using one or multiple available *DataFormats*, which may refer to a specific version. For example, a network topology can be expressed in the JSON-based *PNET* format developed throughout the *FlexiEnergy* project, or the *Pickle*-format defined by the *PandaPower* application¹. Characteristics of a data type are captured by its *MetadataAttributes*. As suggested by the name, the attributes of a data type do not fully describe the data, but only selected characteristics that are relevant for the formalization of decision support functionality. For example, while a network topology describes multiple network assets, the *size* of the

¹ Tool website: <https://www.pandapower.org/>

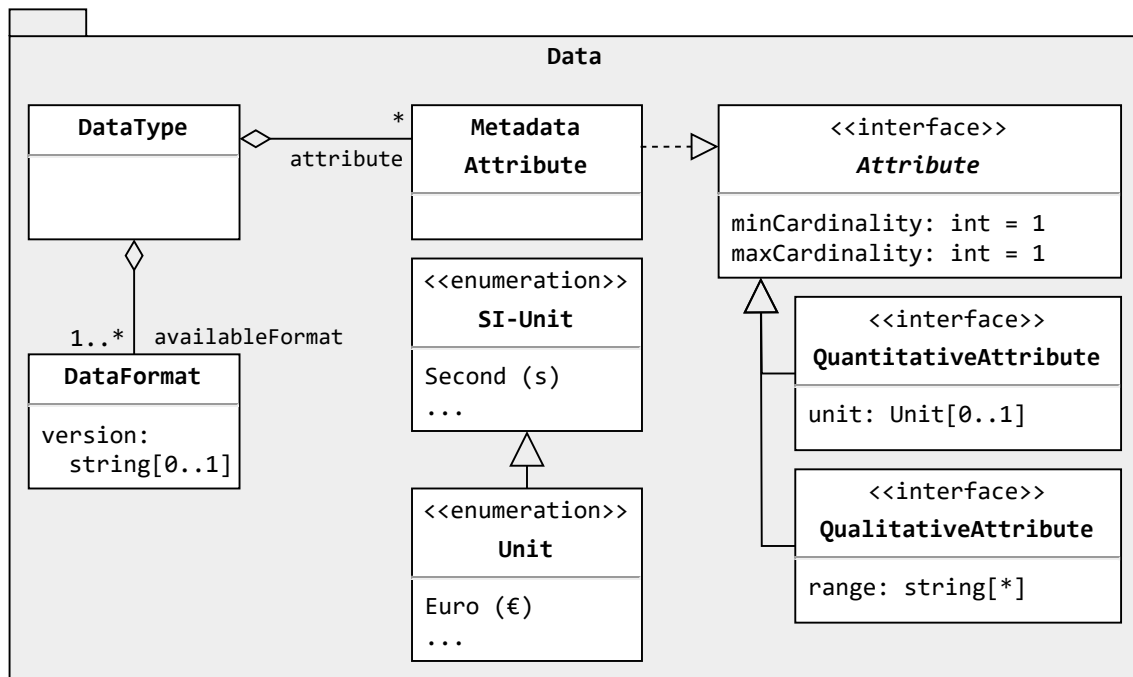


Figure 5.4: Overview of the meta-model package Data

topology with respect to the number of assets is sufficient to filter services that are unable to process the topology due to its size. Or the fact that a network can be $(n-1)$ -reliable with respect to different asset types such transformers, cables, etc., can be used to specify that a certain level of reliability should be achieved when optimizing network investments. These examples for attributes showcase the difference between a quantitative and qualitative attribute. An Attribute can have a minimum and maximum cardinality to represent lists. A QuantitativeAttribute can specify a measurement unit from the list of available (SI-)Units, and a QualitativeAttribute can specify a range of admissible values.

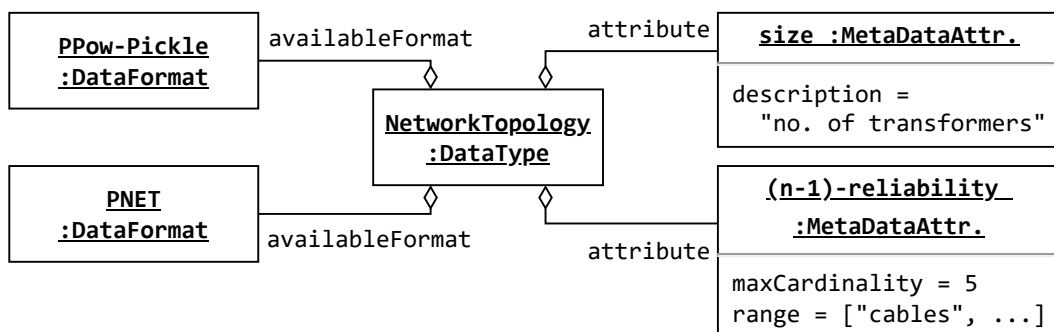


Figure 5.5: Example instantiation of the meta-model package Data

5.4.2 Package “Computation”

The class diagram for the package `Computation` is shown in Fig. 5.6 with the associated object diagram showing an example instantiation in Fig. 5.7. The goal of the `Computation` package is to describe different approaches to process the previously documented data types. The `ComputationMethod` class captures such a computation approach and the `MethodData` it consumes and produces (as indicated by the `MethodDataKind` enumeration). An example of a method is a *mathematical exact optimization of network assets*, which receives a *network topology*, *consumer demands* and *asset types* available for replacement as input, and computes a list of *investments* that describes when, where and how to replace assets. Methods can also have multiple outputs, e.g., a *network reduction* to reduce the size of a network topology produces the reduced topology as well as information on how to undo the reduction. While output method data describes intended outputs, a `ComputationException` defines a potential error that may occur during the execution of the computation and is optionally accompanied by a payload with additional error information, e.g., because a topology is *incomplete*.

Computation methods with different inputs but a similar goal can be grouped into a `ComputationGoal`, e.g., mathematical exact and heuristic asset optimization can be associated with the *asset optimization* goal. Alternatively, information about (technical) details

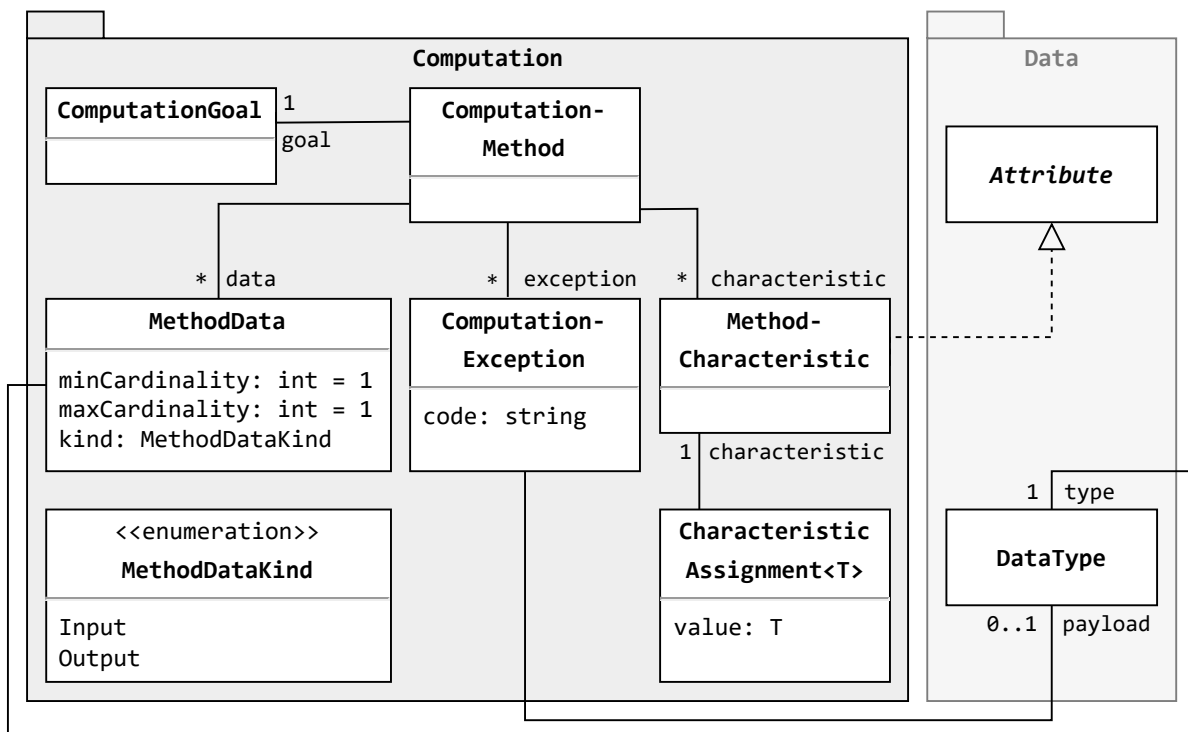


Figure 5.6: Overview of the meta-model package `Computation`

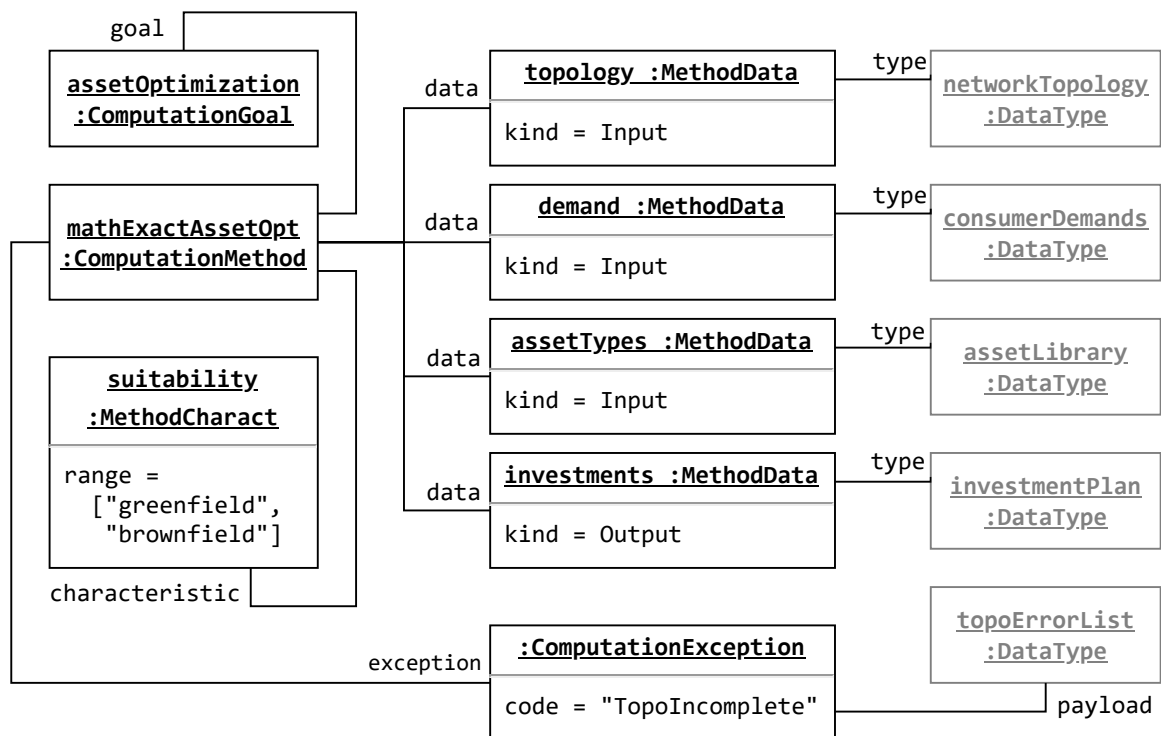


Figure 5.7: Example instantiation of the meta-model package Computation

of the computation method can be defined as a **MethodCharacteristic**. For the example of network asset optimization, some methods may only be *suitable* for designing networks from scratch (“greenfield”), while others also support the re-design of existing networks (“brownfield”). The **CharacteristicAssignment** class will later be used to assign values to method characteristics based on the type of the **Attribute** (i.e., quantitative or qualitative). Definitions for method data, characteristics, and exceptions may be reused across methods.

5.4.3 Package “Optimization”

The class diagram for the package **Optimization** is depicted in Fig. 5.8 with the associated object diagram showing an example instantiation in Fig. 5.9. The goal of the optimization package is to enable the description of optimization characteristics for required and provided optimization-based decision support services. These characteristics are later described at service-level and not at method-level since a service implementing a method often only supports a subset of theoretically possible optimization characteristics.

An **OptimizationProblem** is characterized by one or multiple **Objectives** to minimize or maximize quantitative data characteristics, and an arbitrary amount of **Constraints** on data characteristics to restrict the set of valid optimization solutions. In the example instantiation, the

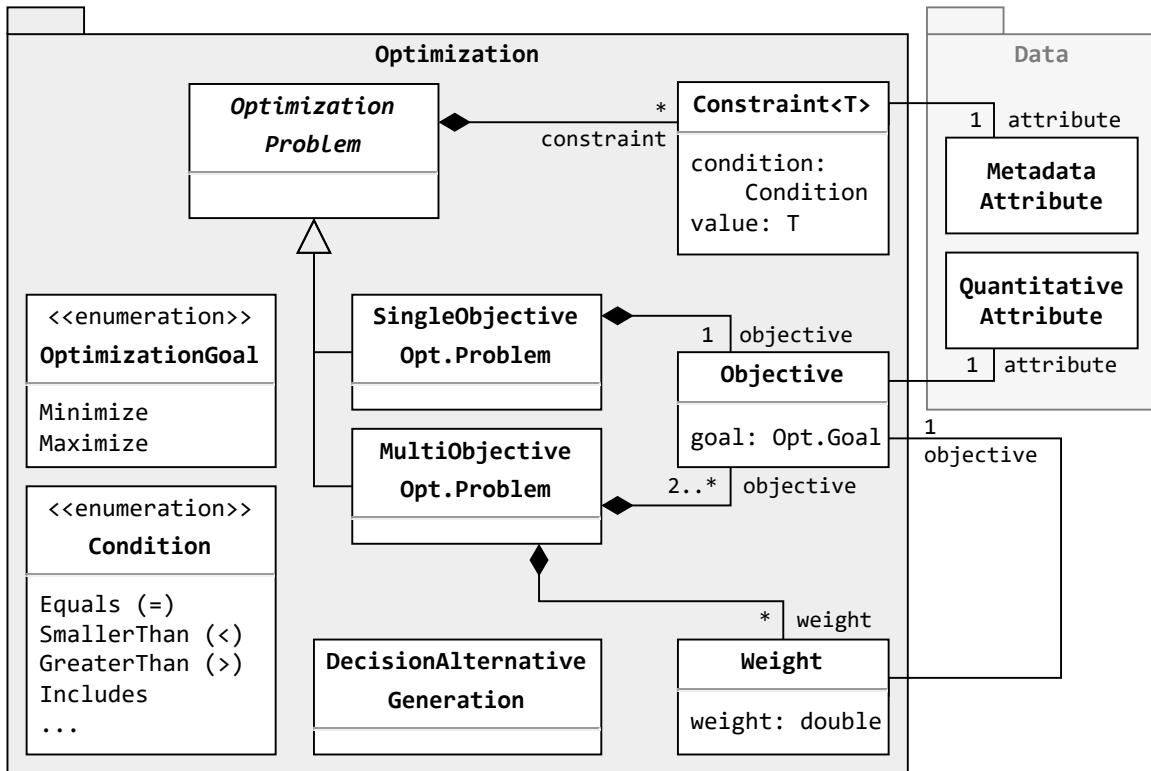


Figure 5.8: Overview of the meta-model package Optimization

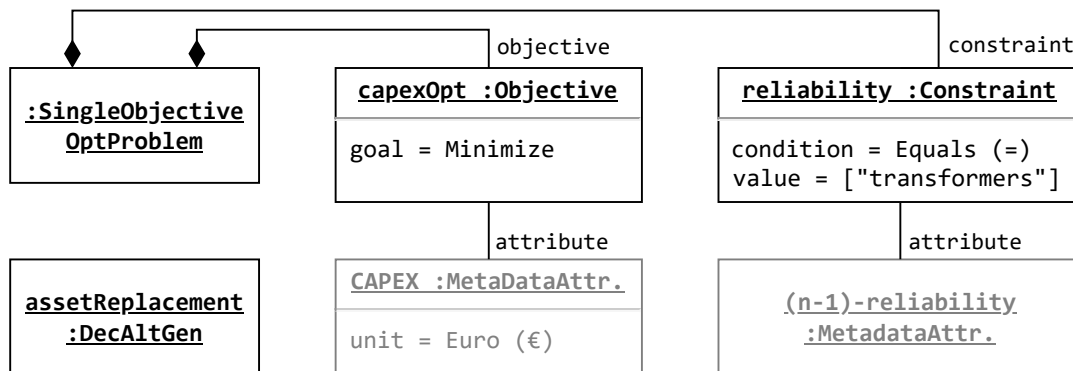


Figure 5.9: Example instantiation of the meta-model package Optimization

investment costs (CAPEX) of a network topology are minimized while ensuring that the network is (n-1)-reliable in its transformers. If multiple opposite objectives are optimized, they can be prioritized by a weight, which can for example be implemented by services using scalarization or weighted goal programming [CD20]. The DecisionAlternativeGeneration class specifies approaches for the generation of decision alternatives during optimization, e.g., traditional *asset replacement* or placement of battery storage systems.

5.4.4 Package “Resources”

The class diagram for the package Resources is depicted in Fig. 5.10 with the associated object diagram showing an example instantiation in Fig. 5.11. The goal of the package is to enable the documentation of resources available/required for consumption during the execution of decision support services. The Resource class is used to describe any consumable resource such as *time* or *money* with an associated unit. In the simplest case, an *AbsoluteResourceQuantity* specifies an absolute value for the available or consumed resource quantity. For example, a service invocation may always incur a *monetary fee* of 10 Euros. A *ReferenceResourceQuantity* can only be used for functional decision support services implementing a computation method. In this case, the resource consumption can be specified with respect to an input prescribed by the method. The relation between the quantitative input data attribute and the resource quantity is described by a *GrowthFunction*. For example, the *execution time* of a network optimization provided by a service may be *exponential* in the *size* of the provided network *topology*. Other considered growth functions are *Linear* and *Polynomial*, which can be parametrized accordingly.

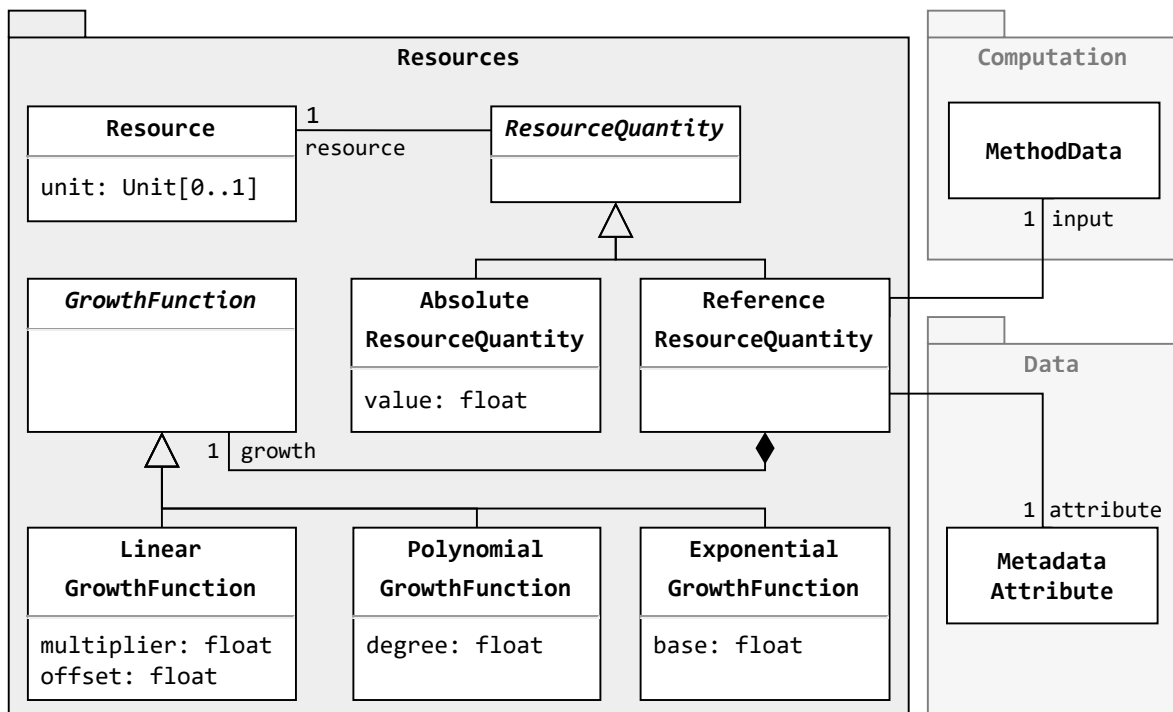


Figure 5.10: Overview of the meta-model package Resources

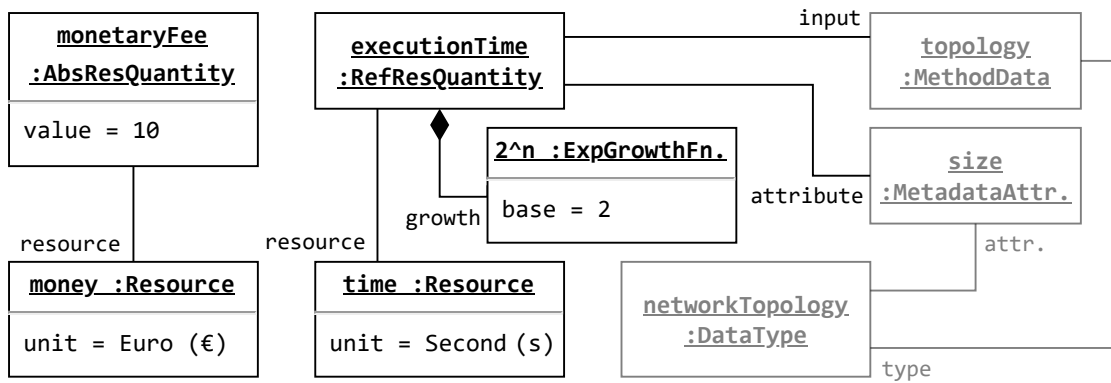


Figure 5.11: Example instantiation of the meta-model package Resources

5.4.5 Package “Services”

The class diagram for the package Services is split into Figures 5.12 and 5.13 with the associated object diagram showing a (partial) example instantiation in Fig. 5.14. Unlike the previous packages, whose concepts support both the description of provided as well as required decision support functionality, the concepts of the Services package are intended for the description of existing decision support services only.

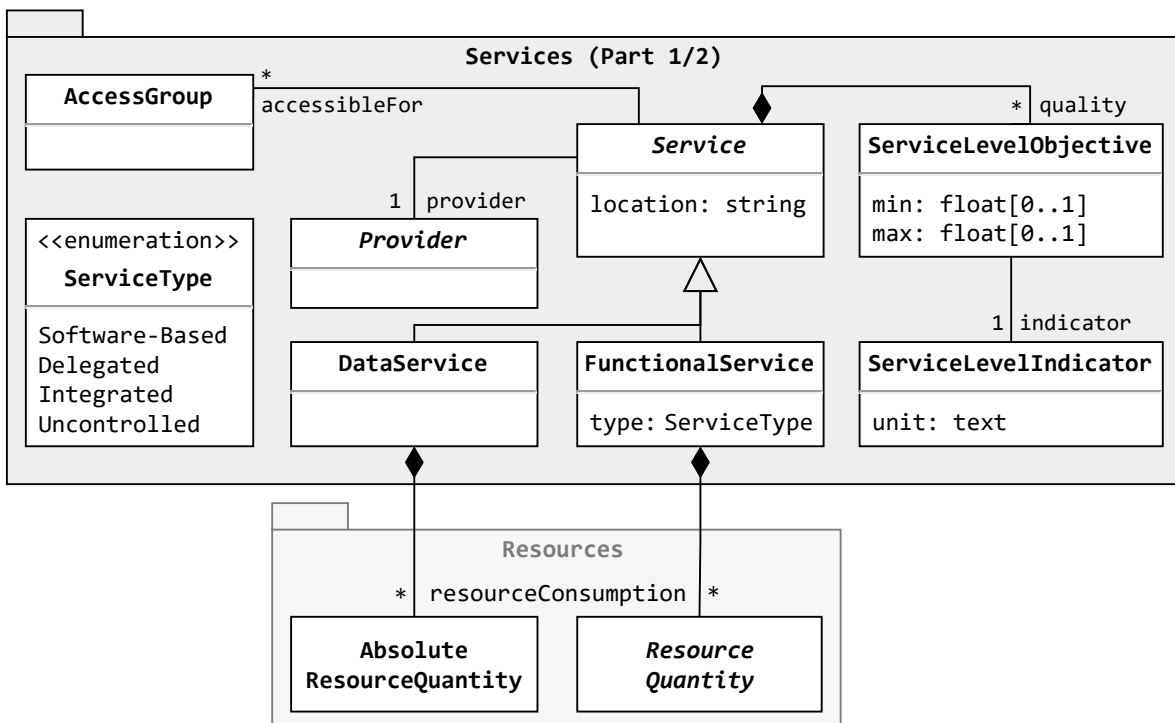


Figure 5.12: Overview of the meta-model package Services (Part 1)

A service can either provide a dataset (DataService) or implement a computation method (FunctionalService). Each service is assumed to be available as a microservice at a URL documented as `location`, documents the service Provider, and is associated with multiple AccessGroups permitted to use the service. The description of service providers and the association of ecosystem participants to access groups is not part of the meta-model as it covered with existing description languages such as the *OpenAPI Specification* [Ope21]. Services are associated with ServiceLevelObjectives that specify an upper or a lower bound on a ServiceLevelIndicator, e.g., to guarantee that a service has an *availability of 99%* or higher. For a data service, resource consumption can only be specified absolutely, while a functional service can also specify resource consumption with respect to input parameters of the implemented computation method as illustrated in the previous Section 5.4.4.

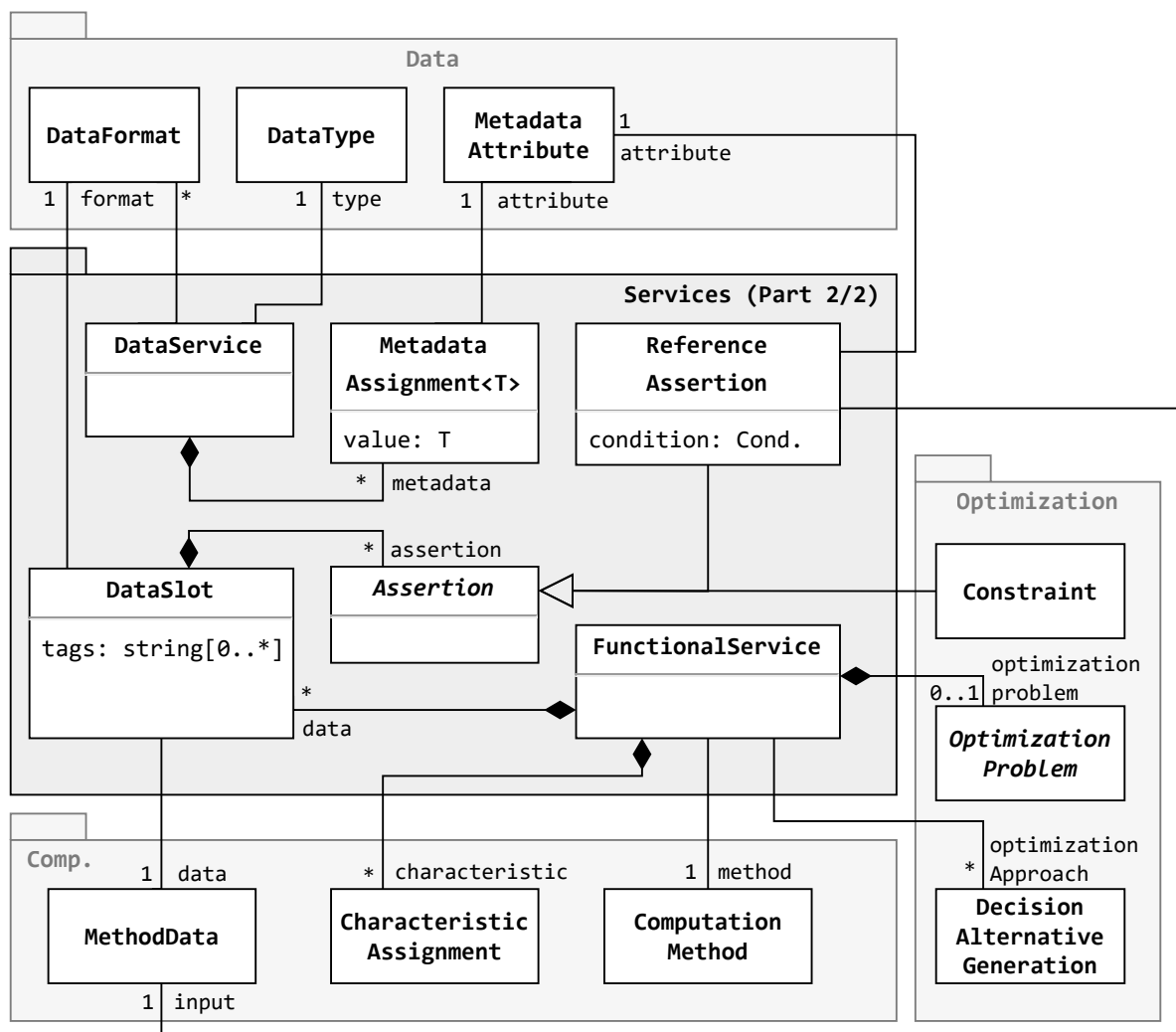


Figure 5.13: Overview of the meta-model package Services (Part 2)

A `DataService` only specifies the data type and the data format of provided data via associations, and its metadata via the `MetadataAssignment` class. The type of a metadata value depends on whether the metadata attribute is of quantitative or qualitative nature. For example, a network topology for a *fictitious city* with a *size* of 394 network assets described using the *PNET* data format may be published *globally* within the DSE for testing purposes.

A `FunctionalService` is characterized by the computation method it implements. The example used for illustration purposes is the *PowOpt* algorithm developed throughout the *FlexiEnergy* project, which implements a mathematically exact optimization of network assets given future consumer demands (cf. Section 5.4.2). Each input or output data of the method results in a `DataSlot` for the service, although only the one for the input *topology* is shown in Fig. 5.14 for clarity. In addition to the method data a data slot represents, a data slot furthermore documents supported data formats and potential `Assertions` on the data, e.g., to ensure that the provided network topology has a *size below 1000 nodes*. For such absolute assertions, the `Constraint` class from the `Optimization` package is reused. Although not demonstrated in the object diagram, the value of an assertion can also be specified as a `ReferenceAssertion` to an input method data, e.g., to express for a consumer demand simulation service that both load profiles and market shares of consumer

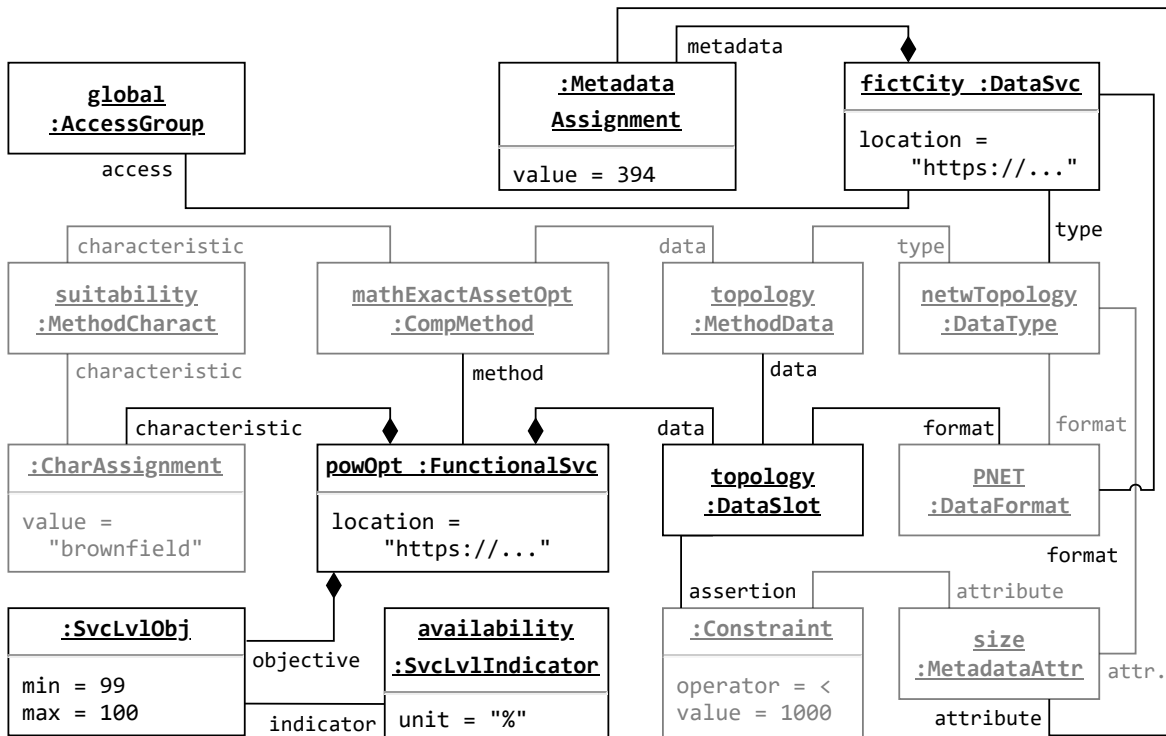


Figure 5.14: Example instantiation of the meta-model package Services

technologies must have the same temporal resolution. A `DataSlot` can furthermore specify tags that must be present on provided data or will be added to the produced data as metadata to document that data has been processed by a specific service. A functional service providing optimization functionality can document the addressed optimization problem as described in Section 5.4.3. A `CharacteristicAssignment` assigns a value to a characteristic of the computation method, e.g., to express that the *PowOpt* service works for *brownfield* networks.

5.4.6 Package “DSS Requirements”

The class diagram for the package `DSS Requirements` is shown in Fig. 5.15. The goal of the package is to enable the structured documentation of a decision maker’s requirements for decision support for subsequent processing by DSS engineers or the composition assistance system. An object diagram with example instantiations is not provided to avoid redundancy as the package primarily describes relationships to classes introduced in previous packages.

Following the upfront design considerations of Section 5.1 discussing the similarities between describing provided and required decision support functionality, the central `DssRequirements` class inherits from the `FunctionalService` class described in Section 5.4.5 to reuse the existing concepts for describing available resources for consumption, available access groups, and desired service level objectives. Additional requirements can be specified with respect to concepts from previously described packages (via `DataRequirement`, `OptimizationRequirement`, `MethodRequirement` and `GoalRequirement`). The motivation for using these “intermediate classes” instead of completely defining DSS requirements as a `FunctionalService` is the support of `RequirementLevels` to control the (mandatory) inclusion/exclusion of functionality using the vocabulary defined in RFC 2119 [IET97]. Requirement levels allow contradictions, e.g., a decision maker could declare a computation method as `MUST` be included in one method requirement and as `MUST NOT` be included in another. Naturally, the meta-model also does not guarantee that a DSS addressing the documented requirements can be composed from the available decision support services.

A `DataRequirement` extends a `DataSlot` to document that a certain input or output should (not) be provided by or to the decision maker in a specific data format with certain metadata. The reference of method data is optional as it does provide additional semantic meaning regarding the provided/computed data, but also comes with the risk that the data requirement cannot be exactly fulfilled due to some pre-/post-processing. For example, a decision maker may reference the optimized network topology computed by a computation method for asset optimization, but this method may be succeeded by a reversal of a previously executed network reduction that actually provides the final optimized network topology.

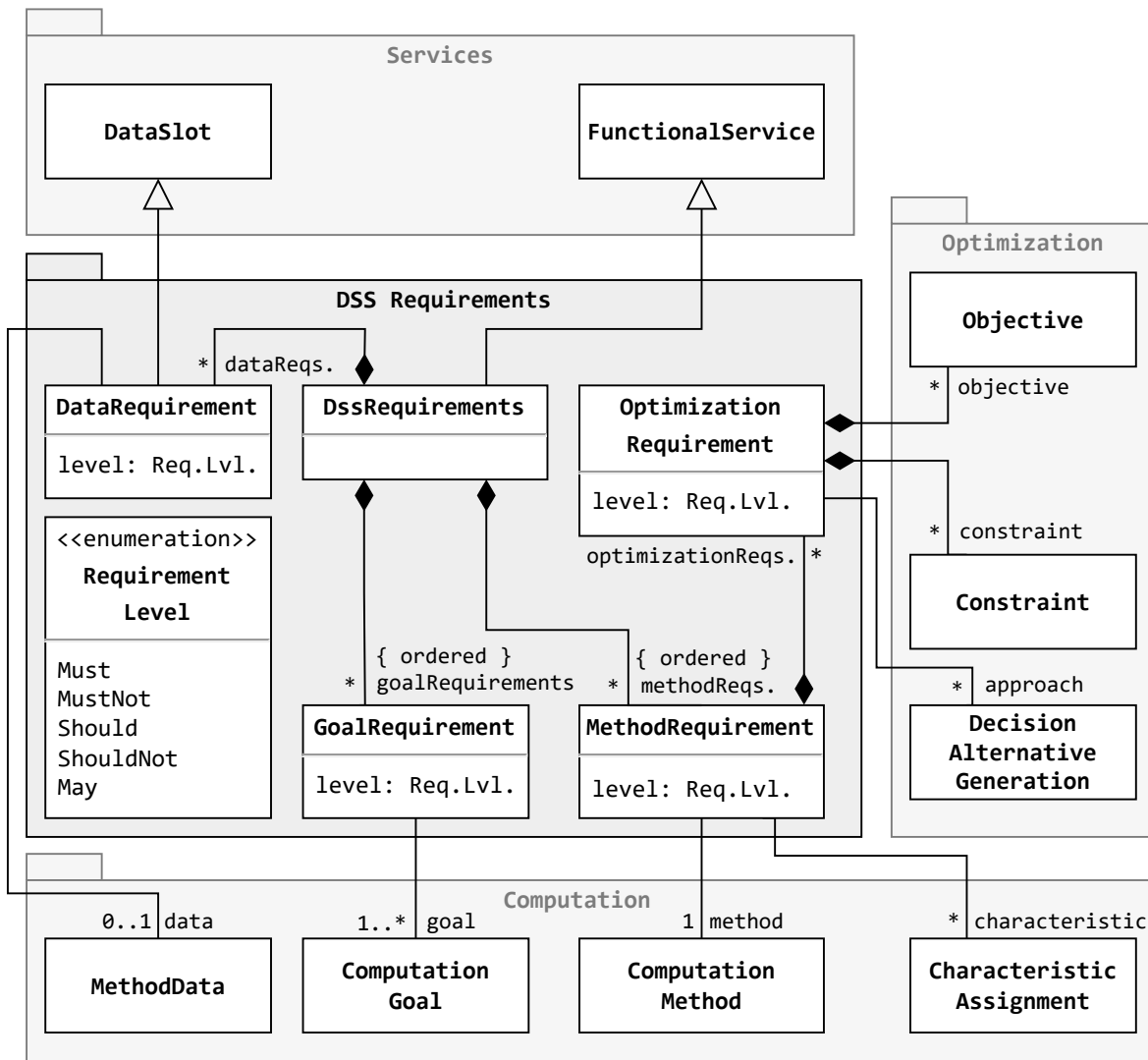


Figure 5.15: Overview of the meta-model package Requirements

A *GoalRequirement* documents the use of a specific computation approach throughout the supported decision process. It can be further refined using a *MethodRequirement* to restrict functionality to computation methods with certain characteristics. The order of goal and method requirements can be used to determine their desired execution sequence. For optimization functionality, the optimization characteristics can be further constrained using an *OptimizationRequirement*. Instead of reusing the *OptimizationProblem* class, the requirement uses the associated classes to support requirements levels for individual parts, e.g., to express that (n-1)-reliability should be achieved but is not mandatory.

A subset of requirement classes may be used based on the knowledge level of a decision maker. For example, a decision maker who already has a (planned) decision process may be

able to specify a sequence of computation methods to execute with restrictions on their defined characteristics, while another decision maker may only specify the (desired) decision process on a high-level using the `GoalRequirement` class. Even more high-level, a decision maker could simply specify the input and output that should be produced via `DataRequirements`, essentially describing a fictitious computation method with additional information on the available data formats. Although these distinctive approaches to requirement documentation are imaginable, they can be arbitrarily combined to support the documentation of requirements as precisely as possible. Requirements for the usage of specific services are not supported due to the assumption that a decision maker with such technical knowledge can immediately assume the role of the DSS engineer.

5.5 Prototypical Implementation

This section demonstrates how the previously described meta-model supports the implementation of the *Domain Registry*, *Service Registry* and *Requirements Elicitation* applications introduced in Section 5.1. The subsequently presented prototypical implementation uses a headless *content management system (CMS)*. A headless CMS enables content creators to create entities using forms. Unlike a traditional CMS, a headless CMS does not transform the entered data into static webpages to be viewed in a web browser. Instead, the created entities are made available via an API for further processing by other applications. This combines the advantages of using forms to enable ecosystem participants to define instances of meta-model elements with the flexibility of arbitrarily accessing and processing these elements in subsequent DSE components. Additionally, using an existing headless CMS framework reduces implementation effort during (prototypical) implementation.

The headless CMS framework *Payload*² was chosen for the prototypical implementation since the previously described meta-model can be translated into a *Payload configuration*. This configuration is then used by the framework to provision the form-based *admin user interface* accessible via a web browser for data manipulation (cf. Fig. 5.16), the *API server* for data access via REST [Fie00] or GraphQL³, and the database for data storage (cf. Fig. 5.17). The configuration is provided as supplementary material with the thesis (cf. Appendix B). The data format of API requests and responses also provides a concrete syntax for the representation of meta-model instances based on the *JavaScript Object Notation (JSON)* [Ecm17].

The *Payload* configuration is derived from the previously described meta-model as follows: Each meta-model class that is not tied to another class by composition is mapped to a *Payload*

² *Payload* website: <https://payloadcms.com/>

³ *GraphQL* website: <https://graphql.org/>

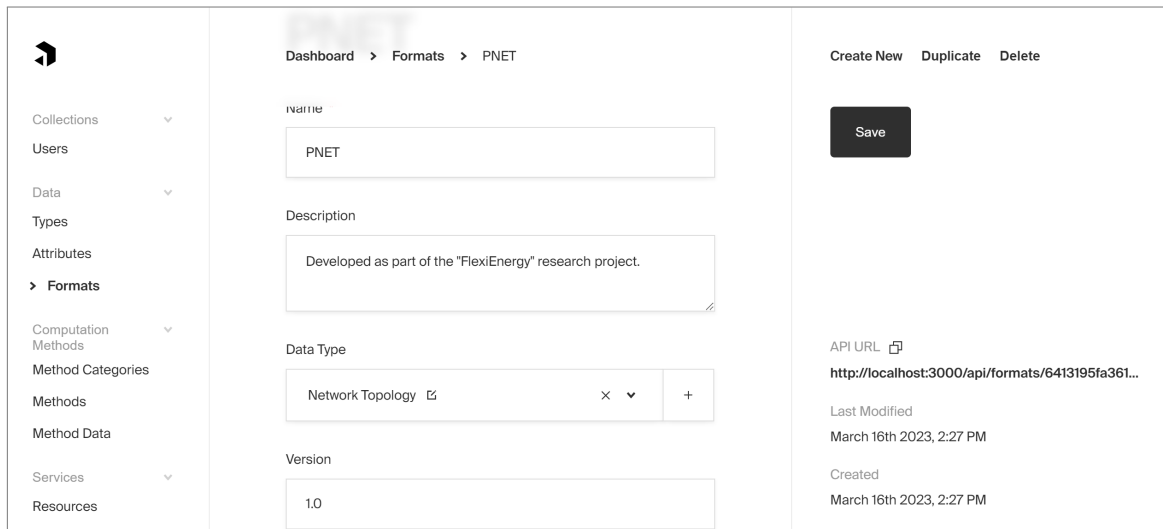


Figure 5.16: Screenshot of the admin UI of the prototypical implementation

collection that can store multiple instances of the meta-model class. Each instance is stored as a JSON object (referred to as a *document*) with a (nested) key-value structure as prescribed by the *field definitions* of the encompassing collection. Based on the field definition, the value for the field can either be specified as an absolute value (which is used to document attribute values), an ID-based reference to a document in another collection (which is used for associations and aggregations starting from the class), and nested fields (which correspond

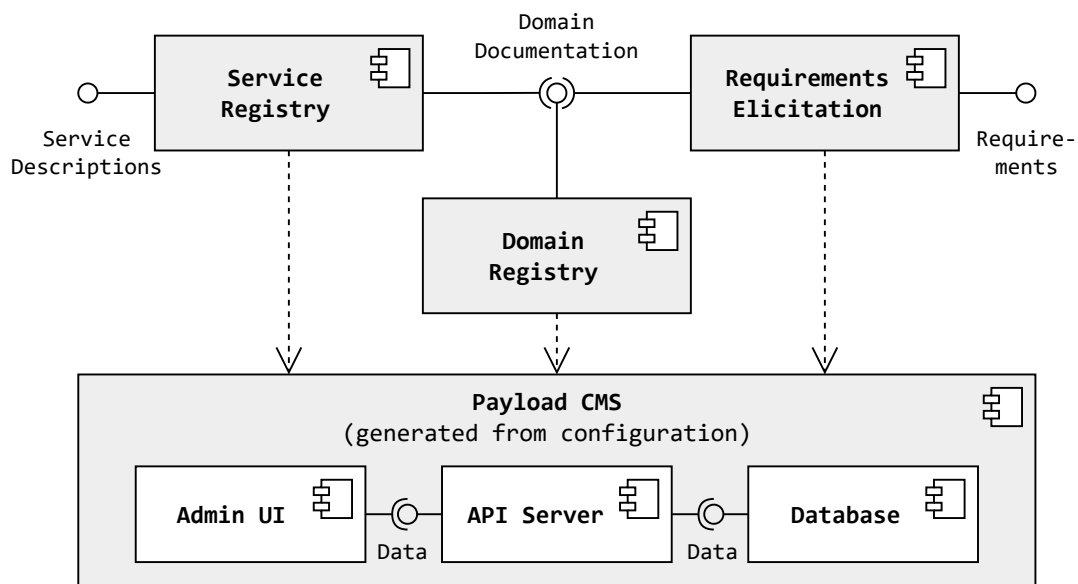


Figure 5.17: Overview of using the CMS *Payload* for the prototypical implementation

to a class that is related to the represented class by composition). Example excerpts for the *Payload* configuration and meta-model instances are shown in Listings 5.1 and 5.2 to document functional services with their service level objectives (cf. Fig. 5.12).

Listing 5.1: Partial *Payload* config (JavaScript)

```
// "functional-service" collection:
{ fields: [
  // ...
  { name: "quality",
    type: "array",
    fields: [
      { name: "indicator",
        type: "relationship",
        relationTo: "indicators" },
      { name: "min", type: "number" },
      { name: "max", type: "number" },
    ],
  },
] }
```

Listing 5.2: Partial instances (JSON)

```
// in "f.-s." collection:
{ "id": "some-service",
  // ...
  "quality": [ {
    "indicator":
      "sli-availability",
    "min": 0.95,
    "max": 1
  } ]
}
// in "indicators" collection:
{ "id": "sli-availability",
  "name": "Service Availability" }
```

5.6 Discussion

The object diagrams throughout Section 5.4 demonstrate the sufficient expressiveness of the meta-model to capture (required and provided) decision support services and characteristics of the application domain for energy distribution network planning. Furthermore, the explanations of Section 5.5 demonstrate the technical feasibility of transforming the meta-model into software for the form-based documentation of domain and service characteristics. Cross-cutting threats that also potentially affect the validity of these demonstrations are discussed in Chapter 8. The remainder of this section summarizes the relationship between the presented insights and the requirements for describing provided and required decision support functionality and data established in Section 5.2.

Domain Perspective The domain perspective consists of requirements *DR1 – Types of Domain Data* and *DR2 – Characteristics of Domain Data*. These requirements map directly to the Data package, in particular the `DataType` and `MetadataAttribute` classes respectively.

Functional Perspective Requirement *DR3 – Computation Approach* primarily maps to the Computation package, which allows the definition of `ComputationGoals` and associated

`ComputationMethods` with their inputs, outputs, and additional characteristics. The requirement also affects the `Service` package by associating each `FunctionalService` with the computation method it implements and service-specific characteristics. The meta-model explicitly does not support the documentation of services that implement multiple methods as this would run counter to the DSE goal of fostering reusability.

Requirements *DR4 – Optimization Goals* and *DR5 – Decision Alternative Generation* directly map to the `Optimization` package with classes `Objective` and `Constraint` addressing *DR4 – Optimization Goals*, and `DecisionAlternativeGeneration` addressing *DR5 – Decision Alternative Generation*. Again, these classes are referenced by a `FunctionalService` to specify the optimization capabilities of the service (if relevant). A previously published version of the meta-model in [KWE22b] also considered the specification of a percentage of the theoretical optimum that can be achieved with an optimization method. This was extended to the more comprehensive `MethodCharacteristics`.

Non-Functional Perspective Requirement *DR6 – Resources* primarily maps to the `Resources` package for the specification of `Resources` and `ResourceQuantities`. These classes are then referenced during the definition of a `Service` to document resource consumption for data- and functionality-based decision support services or available resources for `DssRequirements`.

Requirements *DR7 – Data Constraints*, *DR8 – Service Level Objectives*, *DR9 – Usage Information*, and *DR10 – Access Control* map to the `Services` package. Data constraints are defined as `Assertions` for a `FunctionalService`, but the specification of `MetadataAssignment` of a `DataService` also supports the later evaluation of whether a data service meets the constraints of functional service. Similar to *DR6 – Resources*, the specification of `ServiceLevelObjectives` and `AccessGroups` can be interpreted as “required” for a functional service, and as “provided” for a required DSS. Requirement *DR8 – Service Level Objectives* could be extended to cover complete service level agreements and also include the specification of penalties if SLOs are not met. However, since this is not specific to the domain of decision support, existing approaches may be reused (cf. [NMJ19]).

DR10 – Access Control must be enforced during service invocation, but could also be already considered during service discovery. For example, some service providers may prefer their services not to be advertised to users with missing access rights, while other service providers may prefer an advertisement to motivate potential consumers to obtain access rights if this is associated with the payment of a fee. This behavior could be toggled with an additional `Service` attribute. Furthermore, a hierarchical relation between access groups may be desirable in practice to reduce the effort of multi-group assignments.

The parameterization of functional services is not explicitly considered in the meta-model (except for potentially supporting configurable weights for optimization targets). For

example, a service's considered level of (n-1)-reliability may be configured as part of a service composition. However, this is not a shortcoming of the meta-model as different configurations of a service would simply be represented as different functional services. Nevertheless, this lack of service configuration might lead to more complex data management in practice to account for potential updates in all service descriptions.

Integration Perspective Requirements from the integration perspective are addressed by the design choices described in Section 5.5. Using a JSON-based format to document models instantiated from the meta-model addresses both requirements *DR11 – Machine-Readability* and *DR13 – Human-Readability*, although the latter should ideally be confirmed by a user/developer study. Requirement *DR12 – Tool Support* is addressed using the form-based instantiation of meta-model elements based on a headless CMS.

5.7 Key Takeaways

Existing service description languages and decision support ontologies only partially address the thirteen established requirements for documenting provided and required decision support in the context of a decision support ecosystem. The meta-model proposed throughout this chapter addresses this shortcoming by defining the concepts required for modeling provided and required decision support services. The meta-model consists of six packages for the description of (1) data provided to and by services, (2) decision support functionality in the form of computation methods, (3) optimization characteristics, (4) resources to be consumed by services, (5) the data- and functionality-based decision support services themselves, and (6) decision support requirements. Illustrative examples provided during explanations of the meta-model demonstrate its sufficient expressiveness for the example application domain of energy distribution network planning. A prototypical demonstration shows how the instantiation and distribution of meta-model elements can be realized with form-based data manipulation using a headless content management system to support the DSE lifecycle activities *domain documentation*, *service registration*, and *requirements elicitation*.

The presented description approach only supports the discovery of provided decision support services and requirements elicitation for desired decision support. However, as established in Chapter 4, a tailored DSS requires the integration of multiple decision support services to address a decision process from start to finish. Since the proposed meta-model does not support the description of such service integrations, the next chapter presents an approach to describe a tailored DSS as a composition of decision support services.

Process-Driven Decision Support Systems

The previous chapter introduces a description format for decision support services to document software functionality and data that is available to support individual activities of a decision process. Holistic support of a decision process from start to finish by a tailored DSS still requires the composition of multiple decision support services, i.e., a description of the (conditional) execution sequence of functional services and the data exchange between them. As indicated in Fig. 6.1 for the overall DSE lifecycle, this chapter fills this gap by contributing a process-driven approach for the composition of decision support services by DSS engineers, including conceptual and technical explanations of how a tailored DSS can be generated from the service composition and how decision makers interact with the generated DSS. The insights presented in this chapter are coarsely outlined in the paper “Detecting Data Incompatibilities in Process-Driven Decision Support Systems” by Kirchhoff, Gottschalk, and Engels [KGE22] and “Requirements-Based Composition of Tailored Decision Support Systems” by Kirchhoff, Weskamp, and Engels [KWE22b].

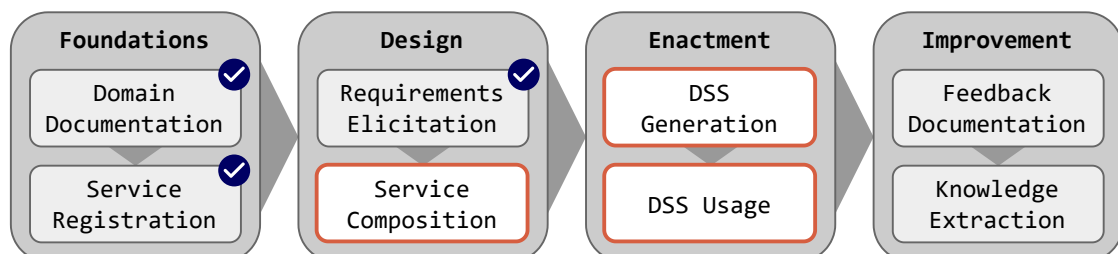


Figure 6.1: Focus of Chapter 6 with respect to the DSE lifecycle

Section 6.1 introduces fundamental requirements for the composition of decision support services into a tailored DSS. Next, Section 6.2 discusses existing approaches to service composition with respect to these requirements to assess their reusability in the DSE context. After an architectural overview of the composition approach in Section 6.3, the mapping from DSE concepts onto the BPMN standard is explained in Section 6.4 and demonstrated using examples from energy distribution network planning. Section 6.5 elaborates the technical feasibility of the composition approach. The presented insights are discussed in Section 6.6 with respect to the composition requirements, and the chapter is summarized in Section 6.7.

6.1 Composition Requirements

This section presents composition requirements CR_x for an approach to compose decision support services into a tailored DSS supporting the holistic decision process of a decision maker. The requirements are based on the insights presented in the previous chapters, in particular, the fundamental characteristics of decision-making processes presented in Section 2.2, the overall requirements for tailored DSS development and associated DSE design principles discussed in Sections 3.1 and 4.3 respectively, and the description of provided and required decision support services introduced in Chapter 5. The requirements are categorized and illustrated based on realistic examples from the domain of energy distribution network planning.

Process Category

The process category aggregates requirements that are derived from the fact that decision making is a process with each process activity being supported by a decision support service.

Composition Requirement CR1 – Functional Perspective

The composition approach must capture the *functional perspective* of a decision process documenting the activities performed throughout the process (cf. [Sun+06]). Consequently, the functional perspective of a decision process documents the computation goals and computation methods such as “(Newton-Raphson) power flow simulation” (cf. Chapter 5).

Composition Requirement CR2 – Operational Perspective

The composition approach must capture the *operational perspective* of a decision process describing the software application used for the enactment of an activity (cf. [Sun+06]). In the DSE context, the operational perspective documents the functional decision support service that is used to implement a computation method associated with an activity in the decision process. For example, the “Newton-Raphson power flow simulation” method is implemented by the *PandaPower* application that can be made available as a software-based service.

Composition Requirement CR3 – Behavioral Perspective

The composition approach must capture the *behavioral perspective* of a decision process describing the execution sequence of activities, optionally with conditions when to execute or skip certain activities (cf. [Sun+06]). An example of such a conditional execution is the use of a network reduction technique to lower the size of a network topology if it were otherwise too large for processing by a network optimization approach. In addition to such a proactive conditional execution, the composition approach should also consider runtime exceptions that may occur during the execution of a service, e.g., to handle an erroneous network topology.

Composition Requirement CR4 – Informational Perspective

The composition approach must capture the *informational perspective* of a decision process describing the data consumed and produced by each activity in the process (cf. [Sun+06]). Since the description of a decision support service already includes the definition of input and output method data via the associated computation method, it is primarily important to capture the *data flow*, i.e., how output data from one service is used as input data of a subsequent service (e.g., the results of power flow simulation are used to identify the necessary network investments during a subsequent asset optimization). Additionally, the composition approach must document which data is provided by/to decision makers as input/output of the decision process (e.g., the network topology to analyze/optimize).

Enactment Category

The enactment category aggregates requirements regarding the application of the decision (support) process for a concrete decision problem. This requires transforming a composition of decision support services into an executable DSS.

Composition Requirement CR5 – Instantiation

As previously discussed, a tailored DSS primarily benefits semi-structured decision problems where the same decision process is applied to multiple instances of a decision problem (cf. Section 2.2). For example, the same decision process for identifying cost-optimal network investments can be applied to different regional distribution networks. In this example, the network topology must initially be provided as input to the decision process by the decision maker. The composition approach should consequently differ between the reusable design of a decision process and the enactment of a process instance with concrete data.

Composition Requirement CR6 – Orchestration

The enactment of a decision support service composition should include the automated (conditional) invocation of the selected decision support services with the necessary data. By

eliminating the need for decision makers to navigate through the decision process themselves, the cognitive load of decision makers can be reduced and efficiency can be increased.

Composition Requirement CR7 – Service Types

The enactment of a decision support service composition must consider the different types of functional decision support services (cf. Section 4.4.1). For example, while an automated decision support service such as a power flow simulation can be invoked via an HTTP request, an interactive decision support service (e.g., for manually fixing errors in the network topology) requires the active participation of the decision maker and therefore must be integrated into the user interface of the tailored DSS.

Design Category

The design category aggregates requirements that support the needs of a DSS engineer who designs a tailored DSS as a service composition.

Composition Requirement CR8 – Visual Programming

The composition approach should be accessible for DSS engineers without potentially any programming education to facilitate participation in a decision support ecosystem. In modern low-code development platforms, visual (model-driven) development has been proven successful for this purpose (cf. Section 3.2.2). This requires the transformation of a visual composition of decision support services into an executable DSS.

Composition Requirement CR9 – Learnability

The composition approach should require limited upfront training to reduce learning effort for DSS engineers and thereby increase the adoption of the DSE approach. The need for upfront training is significantly reduced by using a widespread notation for the documentation of a service composition that is likely already familiar to DSS engineers.

Integration Category

The integration category aggregates requirements for the integration of the composition approach into the overall DSE concept introduced in Chapter 4.

Composition Requirement CR10 – Service Discovery

The service composition approach must interact with the DSE service registry introduced in Chapter 5 to inform a DSS engineer about available decision support services for an application domain. A decision maker must also know about available data services in the service registry during the instantiation of the decision process when using the tailored DSS.

Composition Requirement CR11 – Model Repository

The service composition approach should be integrated into a model repository that documents the relationship between the requirements for decision support previously specified by a decision maker and the associated service composition created by a DSS engineer. Such a model repository provides a foundation for future DSE lifecycle activities, i.e., to identify best practices in addressing decision makers' requirements for decision support.

Composition Requirement CR12 – Machine Readability

A produced service composition must be documented in a machine-readable format for the orchestration of decision support services (cf. *CR6 – Orchestration*) and any (subsequent) automated processing, e.g., by the composition assistance envisioned by the DSE concept.

6.2 Background and Related Work

This section discusses existing service composition and orchestration approaches to assess their reusability for the development of tailored decision support systems considering the previously presented composition requirements.

Approaches Using OWL-S, BPEL, or Custom-Made Notations

A variety of approaches for service composition already exists, as summarized by the taxonomy of Lemos, Daniel, and Benatallah [LDB16], the literature review on RESTful service composition by Garriga et al. [Gar+16], or the discussion of semantic web service composition by Alwasouf and Kumar [AK19]. The classification of approaches within these papers suggest that most service composition approaches use some (business) process-based or workflow-based description of a service composition. This observation also aligns with the service-oriented DSS approaches previously reviewed in Chapter 3: Becker et al. [Bec+08] and Shafiei, Sundaram, and Piramuthu [SSP12] use the *Business Process Execution Language (BPEL)* [OAS07] for describing an executable service composition. Mustafin, Kopylov, and Ponomarev [MKP21] use OWL-S, which was previously introduced in Section 5.3. Dong and Srinivasan [DS13] and Axelsson et al. [Axe+17] use a custom-made format to describe a service composition without providing further details on the description format itself.

All mentioned approaches have limitations in terms of the composition requirements introduced in the previous section. The custom-made formats do not benefit from the documentation and examples available for existing notations, which limits *CR9 – Learnability*. OWL-S, in addition to the disadvantages already discussed in Chapter 5, does not have any tool support for the execution of a service composition other than the proof-of-concept work

by Paolucci et al. [Pao+03] (who refer to OWL-S by its previous name DAML-S [W3C04]). Consequently, OWL-S cannot address *CR6 – Orchestration* without extensive implementation effort. BPEL does not natively support human activities [Len+18] or a visual notation [Ley10] and therefore does not address *CR7 – Service Types* and *CR8 – Visual Programming*. It is also tightly coupled to the web-service implementation of SOA and is thus decreasing in importance due to the recent focus on microservices (cf. Section 3.2 and [Len16]). Although there are some approaches to address these disadvantages of BPEL such as *WS-HumanTask* [OAS10] or *BPEL4REST* [Pau08], a more comprehensive alternative to BPEL with a visual notation is the *Business Process Model and Notation (BPMN)* [Obj13], which is discussed next.

Approaches using BPMN

BPMN is a superset of BPEL that also supports human tasks and comes with a graphical notation [Ley10]. It is popular among domain experts due to its comprehensibility [LN12], which also makes it easy to learn without extensive upfront training for both business and IT professionals [Rec08]. It furthermore supports the business process lifecycle from process modeling to process execution via a process engine [Sch+21a]. These advantages make BPMN the “de-facto standard” of business process modeling [ADL10; BL17].

The fundamental applicability of BPMN for service composition in the context of a service-oriented architecture is already shown in existing work. For example, Stiehl [Sti14] describes how BPMN can be used to define a *process-driven application (PDA)* where a process model describes an executable sequence of services to invoke. However, the approach is not immediately applicable to the DSE context as it heavily relies on the (outdated) *WS-** technologies, which do not align with the more lightweight communication approach of decision support services (cf. *CR7 – Service Types* and Chapter 5). The description of the prototypical implementation suggests that the technical information required for service invocation must be provided by a programmer and cannot be done by a domain expert without programming skills, consequently not addressing *CR8 – Visual Programming*. This disadvantage also holds for the extension of the PDA concept to support RESTful web services described by Schäffer et al. [Sch+21a]. The approach by Valderas, Torres, and Pelechano [VTP20] divides a BPMN model into fragments consisting of multiple activities, with each fragment being executed by a single microservice. This mapping from multiple activities to a single microservice does not align with the previous definition of a decision support service as an aid for a single activity in the decision process (cf. *CR2 – Operational Perspective*). The approach by Gutiérrez-Fernández, Resinas, and Ruiz-Cortés [GRR17] primarily focuses on using a BPMN process engine for the development of individual microservices, but not for the orchestration of services. Their approach furthermore requires the manual development of a

graphical user interface for a microservice, which is either not in alignment with *CR8 – Visual Programming*, or prevents the support of interactive decision support services as required by *CR7 – Service Types*. Larrinaga et al. [Lar+22] use BPMN for the composition of services provided by hardware components. This does not align with software- and human-based decision support services summarized in *CR7 – Service Types*. Amiri [Ami18], Blal et al. [Bla+18], and Zafar et al. [Zaf+19] describe approaches that support the design of new microservices based on a BPMN process model. This is the opposite of the goal pursued in this chapter, i.e., to use a process model to compose existing decision support services.

Summary of Existing Service Composition Approaches

The initially provided description of BPMN lists multiple characteristics that show its potential to address many of the previously established composition requirements. These characteristics particularly include its widespread visual notation to describe process models (potentially addressing the requirements from the “Process” category as well as *CR8 – Visual Programming* and *CR9 – Learnability*), its (repeated) executability with a process engine based on an XML representation (potentially addressing *CR5 – Instantiation*, *CR6 – Orchestration* and *CR12 – Machine Readability*), and its potential support for different kinds of decision support services (*CR7 – Service Types*). Thus, based on the previous discussion of existing service composition approaches, BPMN is inherently more suitable for describing a composition of decision support services than alternative notations such as OWL-S, BPEL, or custom-made notations. Nevertheless, all previously discussed approaches based on BPMN do not address all composition requirements established in the previous section. However, their limitations seem to be originating from how BPMN is utilized within the approach and not from the fundamental characteristics of BPMN itself. Therefore, the remainder of this chapter focuses on conceptual and technical descriptions of how BPMN can be applied in the DSE context to describe a tailored DSS as a composition of decision support services.

6.3 Architectural Considerations

The discussion of related work throughout the preceding Section 6.2 suggests the use of a BPMN process model to describe a composition of decision support services representing a tailored DSS that is subsequently enacted by a BPMN process engine. Such a (tailored) DSS is referred to as a *process-driven DSS (PD-DSS)* due to its relation to the more generic concept of a process-driven application. The realization of the PD-DSS concept has two prerequisites: First, a refinement of the DSE reference architecture introduced in Section 4.5 is needed to

determine if and how a BPMN-based composition approach can be integrated into the DSE platform. Second, given a suitable architectural framework, the expressiveness of BPMN must be evaluated to determine if and how DSE concepts can be represented using BPMN while addressing the previously established composition requirements. This section focuses on the architectural integration of a BPMN-based composition approach into the DSE platform by (partially) refining the reference architecture for the DSE platform as shown in Fig. 6.2.

The *PD-DSS Repository* application corresponds to the *Model Repository* of the DSE platform reference architecture. It acts as a central hub where DSS engineers receive the requirements for decision support previously documented by a decision maker using the *Requirements Elicitation* application. For a selected requirements documentation, a DSS engineer uses the *PD-DSS Design* application to create a BPMN process model that describes a composition of decision support service addressing the requirements. Thus, the *PD-DSS Design* application corresponds to the *DSS Editor* of the DSE platform reference architecture, and the *PD-DSS Process Model* corresponds to the *DSS Model*. Information about available decision support services is obtained from the *Service Registry*.

The designed process model is then stored in the *PD-DSS Repository* application until a decision maker wants to use the resulting PD-DSS. In that case, a decision maker uses the *PD-DSS Enactment* application, particularly its *Interaction* component, to instantiate the process model with data that should be used throughout the decision process. This parametrization with data enables the reuse of the PD-DSS for similar decision problems. The

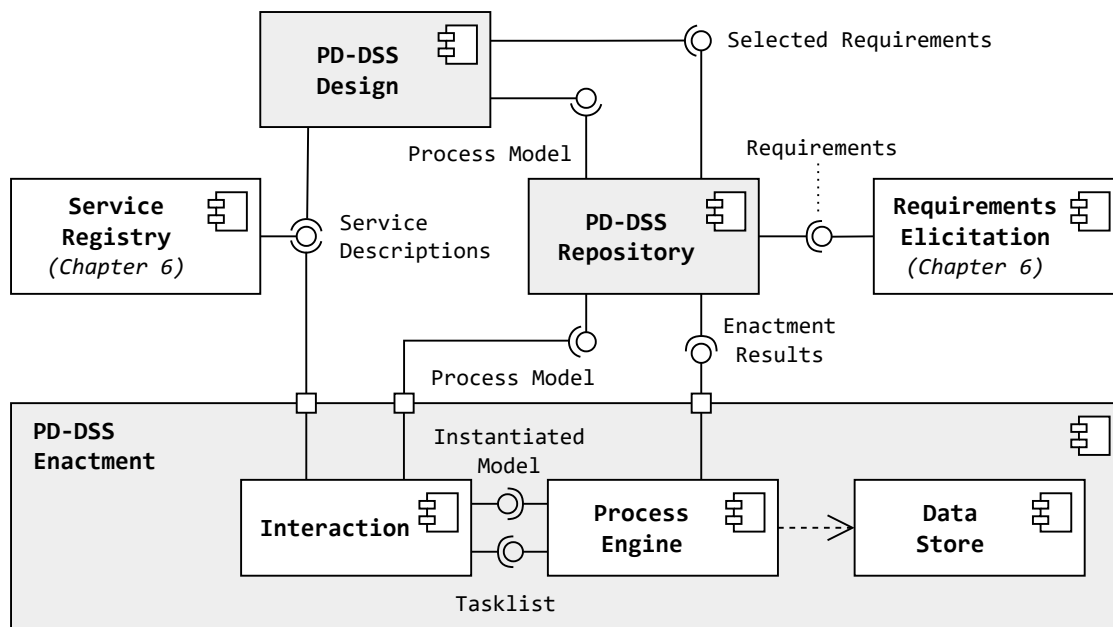


Figure 6.2: Overview of the architecture as a UML component diagram

invocation of decision support services is then coordinated using an (off-the-shelf) BPMN *Process Engine*, which either directly invokes automated decision support services as described in the instantiated process model, or provides a list of tasks that require manual input from the decision maker via the *Interaction* module for interactive decision support services. Thus, the *PD-DSS Enactment* application corresponds to both the *DSS Generator* and *Tailored DSS* of the DSE platform reference architecture, albeit the DSS is realized by interpreting a process model instead of a generated standalone application. Data exchanged between services is temporarily stored in the *Data Store*, while the computed decision recommendations and other output data are saved in the *PD-DSS Repository* for future reference.

6.4 Composition of Decision Support Services with BPMN

Although the preceding Section 6.3 shows how a BPMN-based composition approach can be integrated into a DSE platform, it is still necessary to determine if and how the concepts defined by the BPMN specification can be utilized to describe a tailored PD-DSS as a composition of decision support services to address the composition requirements previously established in Section 6.1. Based on the BPMN meta-model and its associated semantics described in the BPMN specification ([Obj13]), this section presents a mapping from DSE concepts to BPMN concepts to enable the reuse of existing BPMN editors and process engines in an environment for the design and enactment of process-driven decision support systems. Furthermore, the section presents the responsibilities of DSS engineers, decision makers, and the process engine during the design and enactment of a BPMN-based PD-DSS. The subsequent explanations are structured according to the process modeling perspectives introduced in Section 6.1, i.e., the functional and operational perspective (Section 6.4.1), the informational perspective (Section 6.4.2), and the behavioral perspective (Section 6.4.3) of a PD-DSS.

6.4.1 Functional & Operational Perspective: Tasks and Subprocesses

A PD-DSS process model must contain information about the work the PD-DSS should perform to support the decision process of a decision maker. Consequently, a PD-DSS process model must document the activities contained in the targeted decision process (= functional perspective) and how each activity is supported by a functional decision support service (= operational perspective). In BPMN, the functional and operational perspective are captured using concepts associated with an **Activity**, which represents any “work that is performed within a business process” [Obj13]. A **Task** is a special activity describing work which “cannot be broken down to a finer level of detail” [Obj13] – contrary to a **SubProcess**, an

activity which itself consists of multiple other activities. The excerpt of the BPMN meta-model for activities depicted in Fig. 6.3 shows entities that are relevant for the discussion on how to represent the functional and operational perspective of a PD-DSS in BPMN to enable the reuse of existing BPMN process engines for PD-DSS enactment. Transparent entities are relevant to the discussion, but not specified by DSS engineers during the design of a PD-DSS. Refined cardinalities are highlighted. The excerpt is subsequently discussed in more detail.

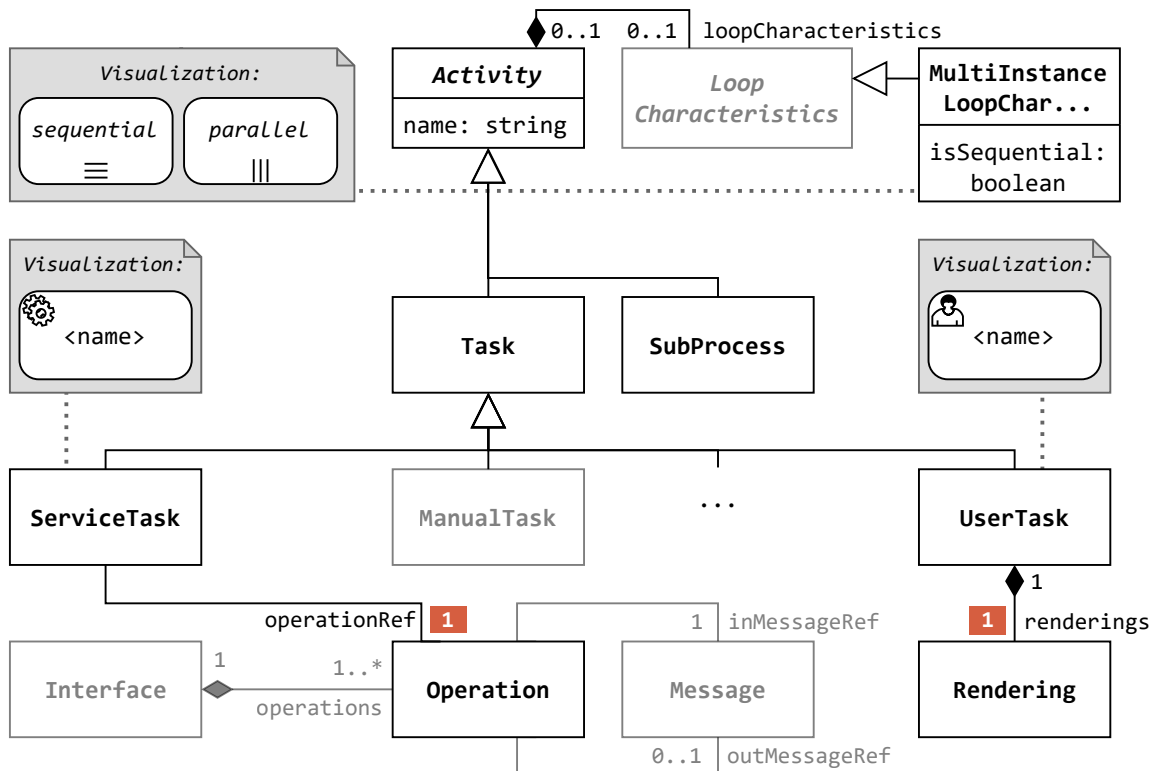


Figure 6.3: Annotated excerpt of the BPMN meta-model for the functional and operational perspective of a PD-DSS (based on [Obj13])

Mapping Decision Support Services to BPMN Tasks

In the DSE context, the invocation of a single functional decision support service corresponds to the atomic work represented by a BPMN task. Section 4.4.1 describes different types of functional decision support services, i.e., *automated services* that do not require any participation from the decision maker and can be provided by a software application deployed on associated infrastructure (*software-based service*) or by a third-party agency (*delegated service*), and *interactive services* that require the active participation of the decision maker (or an appointed employee), usually via a GUI, and should be integrated into the generated

PD-DSS (*integrated service*) or must be executed separately (*uncontrolled service*). As indicated by the excerpt of the BPMN meta-model shown in Fig. 6.3, the BPMN specification defines different types of tasks with different characteristics, which are differently well suited to represent certain types of decision support services as subsequently discussed.

Service Tasks for Automated Decision Support Services A `ServiceTask` uses a web service or an automated application to implement the work represented by the task. A service task references the implementing service `Operation` documented in a service `Interface`. The operation defines the `Message`-based communication with the service. In the DSE context, each decision support service only provides a single operation characterized by the associated computation method. Consequently, each decision support service can be modeled as an `Operation` of a single `Interface` representing the DSE service registry. The input message of each operation includes (references to) the data required by the decision support service, and the output message includes (references to) the data produced by the decision support service. The structure of these messages can be automatically derived from the descriptions of decision support services stored in the DSE service registry, particularly from the associated computation method (cf. Chapter 5). As a result, a DSS engineer only needs to select the operation representing the desired decision support service for a service task during PD-DSS design. Since the selection of a decision support service is mandatory to support an activity in the decision process, the cardinality of `operationRef` is updated from `0..1` to `1`.

The automated request-response interaction of service tasks naturally aligns with a software-based decision support service provided as a software application on the associated infrastructure. However, a service task can also represent a delegated decision support service since the black-box characteristic of services makes it indistinguishable whether the assignee implements the service using (unpublished) software, manual work, or a combination of both. Thus, from a decision maker's point of view, a delegated decision support service also seems "automated" in the sense of the definition provided for a BPMN service task: A request (message) with input data is provided to the service implementor, and a response (message) with output data is returned to the decision maker.

User Tasks for Interactive Decision Support Services Contrary to an automated `ServiceTask`, a `UserTask` or `ManualTask` represents work that requires active human involvement. However, the start and completion of a manual task are not tracked by the business process engine during process enactment. Consequently, a manual task is not suitable for the composition of decision support services as subsequent services would be executed by the process engine without the results computed during the manual task.

A `UserTask` is more suitable to represent an interactive decision support service since not only the start and completion of a user task are tracked by a process engine, but the

BPMN specification furthermore envisions the process engine to assist the human performer during the enactment of a user task. This assistance includes the display of a graphical user interface during task enactment as specified in a *Rendering* associated with the user task. A rendering can be automatically generated for each interactive decision support service considering the technical capabilities of the selected business process engine. Consequently, the DSS engineer can document the implementing interactive decision support service for a user task by selecting its corresponding rendering during the design of the PD-DSS. Since the selection of a decision support service is mandatory to support an activity in the decision process, the cardinality of renderings is updated from * to 1.

The technical capabilities of a process engine selected for the implementation of the *PD-DSS Enactment* application are likely limited to specific types of renderings such as forms and may be insufficient to integrate for example interactive decision support services provided as web applications. In this scenario, the task manager of the process engine, which continuously publishes outstanding user tasks, can be used to display interactive decision support services outside the process engine. From an architectural point of view, this is already considered in the component diagram of Fig. 6.2 where the *Interaction* component prompts the decision maker for input based on the tasklist received from the process engine.

Repeated Service Invocation

When processing a data collection of multiple data instances, e.g., multiple demand scenarios for energy distribution network planning, it may be necessary to repeatedly invoke a decision support service (or a sequence thereof) for each data instance in the collection. For this purpose, an *Activity* can document *MultiInstanceLoopCharacteristics* for sequential or parallel execution as visualized by three lines at the bottom of the activity. In case a sequence of decision support services is executed in each loop, the corresponding tasks should be aggregated into a *SubProcess* that is associated with the loop characteristics.

Implications for the Design of a PD-DSS

In a BPMN-based *PD-DSS Design* application, the previously described considerations require a DSS engineer and the application to perform the following actions:

1. The DSS engineer places a “generic” task on the modeling canvas and specifies a human-readable name (and optional description) for the task.
2. The DSS engineer selects the implementing decision support service for the task. Based on the description of the selected service in the registry, the *PD-DSS Design* application

transforms the task into a service task with the selected operation for automated services, or into a user task with the selected rendering for interactive services.

3. The DSS engineer identifies tasks that are executed multiple times for different data instances of a data collection. For a single affected task, the DSS engineer specifies whether task instances are executed in parallel or sequentially. For multiple consecutive tasks, the DSS engineer wraps them in a subprocess with the desired loop characteristics.

Figure 6.4 shows the functional and operational perspective of an exemplary PD-DSS model. The gray circles indicate examples of the previously described actions. The IDs of services selected by a DSS engineer are visualized as BPMN comments. First, the creation of multiple potential scenarios describing the future energy demands of consumers is implemented using an (integrated) interactive decision support service. Each scenario is subsequently evaluated in parallel without decision maker intervention by computing the effects on asset loads and determining optimal network investments with respective (software-based) automated decision support services. Lastly, robust investments are identified by the decision maker as part of an (uncontrolled) interactive user task.

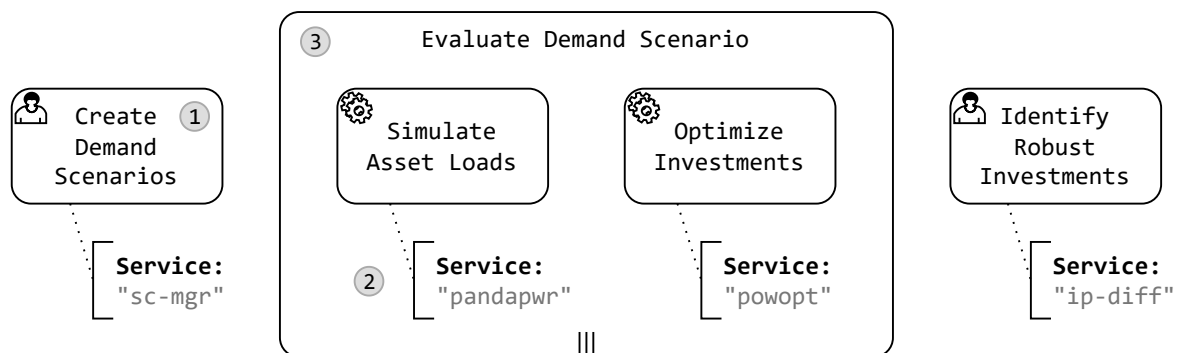


Figure 6.4: Functional and operational perspective of an exemplary PD-DSS process model

Implications for the Enactment of a PD-DSS

Since the different types of decision support services can be mapped to BPMN tasks, the BPMN engine of the *PD-DSS Enactment* application can simply execute the tasks as prescribed by the BPMN specification. Thus, an automated decision support service represented as a service task can be invoked using an HTTP request. For an interactive decision support service represented as a user task, the process engine halts execution and notifies the decision maker about required input such that the decision maker is not required to actively wait

in front of the PD-DSS during the enactment of non-interactive automated services. If a user task is implemented by an integrated decision support service, the UI of the service is either directly shown by the process engine via the associated rendering, or it is shown in cooperation with the *Interaction* component of the *PD-DSS Enactment* application. This requires each integrated decision support service to provide a button in its user interface that a decision maker can use to signal the completion of the task so that the engine can continue the enactment of other tasks. If a user task is implemented by an uncontrolled decision support service that is implemented outside of the PD-DSS, the engine simply shows the decision maker the task description and a UI to indicate the completeness of the task, including a form to submit the data that was generated throughout the task.

6.4.2 Informational Perspective: Data and Data Associations

The decision support services associated with BPMN tasks follow the IPO principle, i.e., they require input data and produce output data. Input data for a service can either be provided by the decision maker as input to the overall decision process, or by a previously executed service. This informational process perspective – also referred to as *data flow* – can be captured with the excerpt of the BPMN meta-model shown in Fig. 6.5. As before, entities that can be automatically generated are shown transparent, and updated cardinalities are highlighted.

Mapping of Computation Methods to Input/Output Specifications

Modeling the data exchange in BPMN requires each Activity (cf. Fig. 6.3) to have an `InputOutputSpecification`, which documents the data required for the execution of the activity as `DataInputs`, and the data produced during the execution of the activity as `DataOutputs`. These data inputs and outputs align with the input and output method data of the computation method implemented by a decision support service. Consequently, after a decision support service has been selected by the DSS engineer for the implementation of a task, the associated I/O specification can be generated from the service description stored in the service registry (cf. Chapter 5). Multiple `InputSets` can be defined to specify that only a subset of `DataInputs` may be required for the execution of the activity, each potentially producing only a subset of `DataOutputs` as specified by an `OutputSet`. This can be used to model the optional input and output method data of computation methods.

Data Exchange

Once the data inputs and outputs of an activity have been defined, data exchange can be modeled by specifying the origin of a data input and the target of a data output using

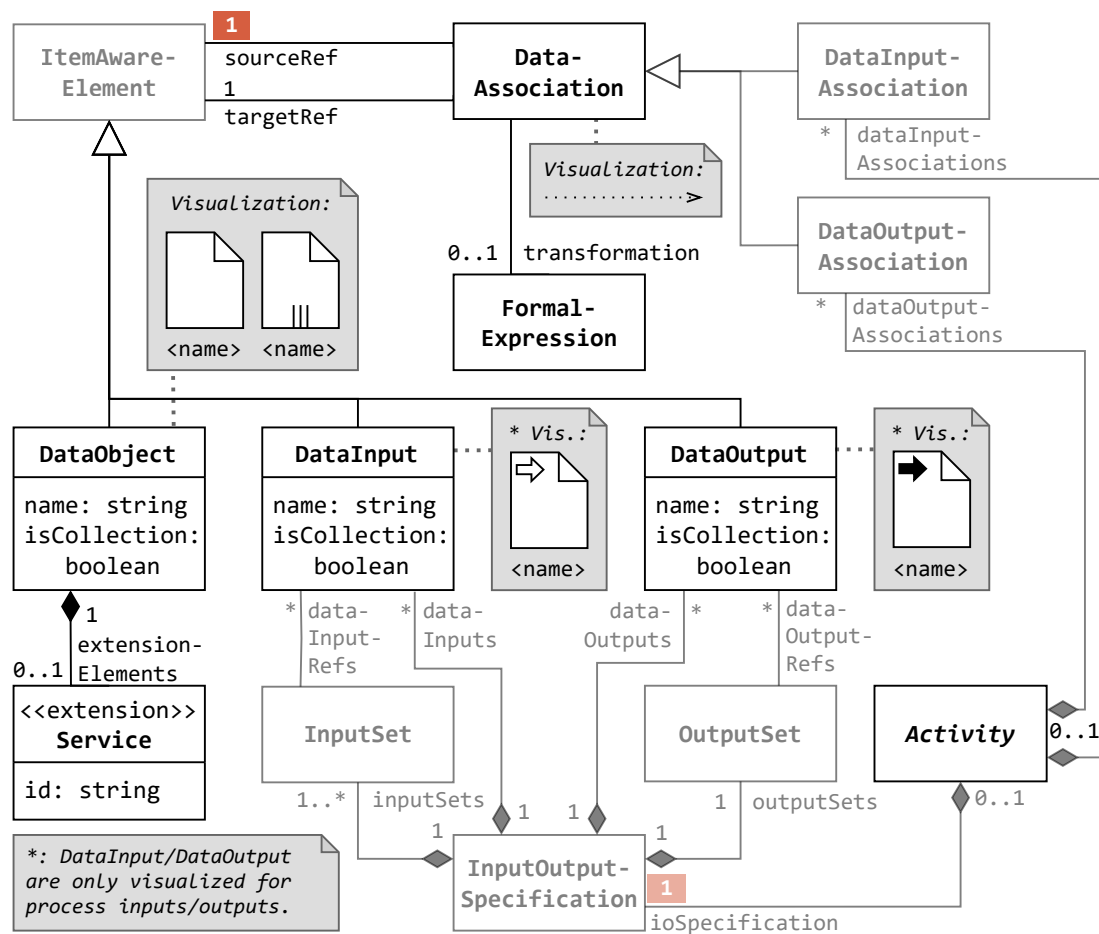


Figure 6.5: Annotated excerpt of the BPMN meta-model for the informational perspective of a PD-DSS process model (based on [Obj13])

a **DataAssociation**. In particular, a **DataInputAssociation** describes how data is provided to a data input of an activity, and a **DataOutputAssociation** describes how a data output is utilized. Data is copied from **sourceRef** to **targetRef** as documented by each **DataAssociation**. For simplicity, data can only be obtained from one source.

Data associations can directly specify the data input of an activity as the target and the data output of an activity as the source. However, since data inputs and outputs are only visualized for the overall process and not for individual activities, output data is instead copied into an intermediate **DataObject** that is shown in a process model as a visual representation of a process variable. Afterwards, the data can be copied from the data object into the designated data input of an activity with a second data association. As a result, the source of a data input association can be a data object or a process data input, and the target of a data output association can be a data object or a process data output. During the enactment of the PD-DSS,

process data inputs must be provided by the decision maker and process data outputs are provided to the decision maker. For configuration data of a service, which is neither provided by the decision maker nor generated by another decision support service, a DSS engineer can also specify the ID of a data service that should be copied into a data object during PD-DSS enactment. The corresponding `Service`-class uses the extension mechanism of BPMN.

A data object/input/output can be a collection of multiple data instances as indicated by the boolean `isCollection` flag, which results in the addition of three vertical lines at the bottom of the document representing the data. If the data-receiving activity documents multi-instance loop characteristics, the activity is executed once for each data instance in the collection. The output of repeated subprocess execution is also aggregated into a data collection.

Data Format Conversion

Data provided by a decision maker or generated by a decision support service may use a data format that is incompatible with the supported data formats of a subsequent service. While it is possible to simply add tasks for data format conversion to the process model, repeated use of conversion tasks potentially adds visual clutter to the graphical representation process model. Therefore, the `transformation` attribute of a data association can be used to specify the ID of a functional service that provides the necessary data conversion.

Implications for the Design of a PD-DSS

In a BPMN-based *PD-DSS Design* application, the previously described considerations require a DSS engineer and the application to perform the following actions:

1. When a DSS engineer selects a decision support service for the implementation of a task, the application automatically creates the I/O specification for the task based on the description of the selected decision support service in the DSE service registry.
2. The DSS engineer adds process data inputs and outputs for data that is provided by or to the decision maker before and at the end of PD-DSS enactment.
3. The DSS engineer adds data objects for data that is exchanged between activities or for static data that is provided by a fixed data service selected during design time.
4. The DSS engineer declares data objects/inputs/outputs as data collections where applicable. This may be done automatically by the *PD-DSS Design* application based on the description of the selected decision support service.

- The DSS engineer connects data objects/inputs/outputs and activities with data associations. For a data (output) association from an activity to a data object or process data output, the DSS engineer specifies the name of the output method data to copy into the target data object/output as the source. For a data (input) association from a data object or process data input to an activity, the DSS engineer specifies the name of the input method data to copy the data into as the target. The DSS engineer selects a service for data format transformation if necessary.

Figure 6.6 shows the informational perspective of an exemplary PD-DSS process model excerpt for energy distribution network planning. The tasks themselves are shown transparently as they originate from the previously discussed functional perspective. The gray circles indicate examples of the previously described actions. A demand scenario provided by the decision maker is used as the demands input of the service simulating asset loads. The asset loads produced by the service under the output loads are used for the subsequent optimization service as input with (coincidentally) the same name. The available asset types are selected by the DSS engineer during the design of the PD-DSS via the Service property. The investment plan produced by the optimization service as output `invPlan` is returned to the decision maker as it constitutes the output of the process. It is previously transformed from CSV to PDF format using the specified decision support service.

Implications for the Enactment of a PD-DSS

Before a composition of decision support services can be enacted, the *PD-DSS Enactment* application must first prompt the decision maker for process data inputs via its *Interaction*

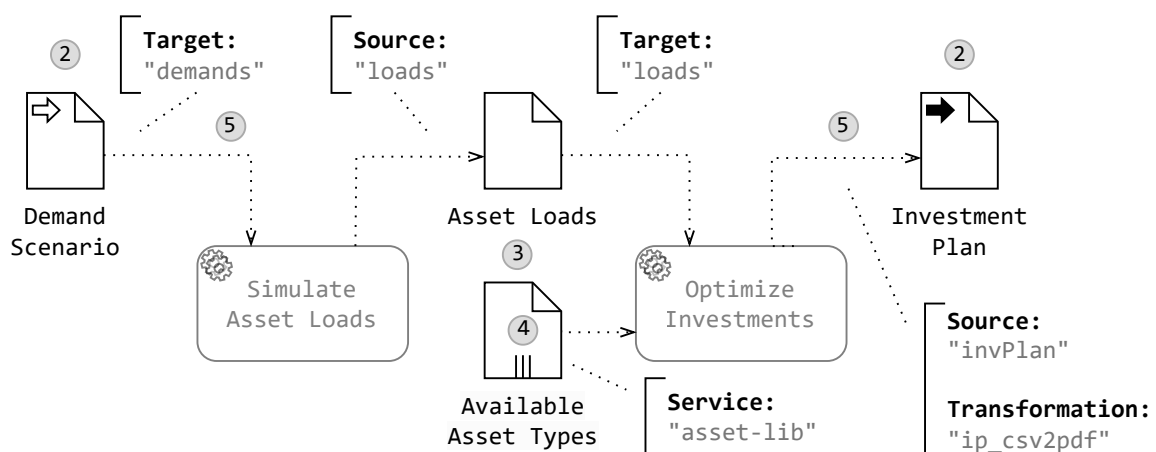


Figure 6.6: Informational perspective of an exemplary PD-DSS process model

component. For this purpose, the decision maker selects data services from the DSE service registry. Temporary data services can be created ad-hoc from uploads if the data will not be reused in the future. The selected data is then made available to the BPMN process engine for the enactment of the PD-DSS process model, which coordinates the data exchange between activities. The results of the enactment, i.e., process data outputs, are made available to the decision maker as a download in the model repository (cf. Fig. 6.2).

Data Exchange per Service Type For service tasks implemented by an automated decision support service, the required data is included in the HTTP request to invoke the service. For user tasks implemented by an integrated decision support service, the input data must be provided to the integrated software module, and the module must forward the data to the engine once the decision maker indicates the completion of the task. For user tasks implemented by an uncontrolled decision support service, the data must be provided to the decision maker for manual processing, e.g., for import into an external software application. The decision maker must furthermore be able to provide the engine with the data produced during task execution.

Reference-Based Data Exchange The explanations of the informational perspective provided so far suggest that data objects (including process data inputs and outputs) hold the complete data that is provided to or produced by a decision support service implementing a task. However, the BPMN process engine would potentially receive large payloads in response to a service invocation, just to subsequently include the payload in a request to (one or multiple) other services. Also, for data services that specify a URL to the data set as part of their service description, the engine would need to initially download the data before immediately forwarding it to the relevant services. This data passing with the engine as a “middleman” is an unnecessary back and forth that can be avoided by providing a central data store where services can download the data required for their invocation and upload the data produced during their execution. As a result, invocation of services corresponds to a single message with key-value data, where each key represents an input method data of the computation method implemented by the decision support service, and the associated value is a URI [IETF05] pointing to the data. The response of a service has the same key-value structure, except that the keys correspond to output method data. Although this reference-based data exchange introduces a minor overhead for service providers as they have to implement initial data fetching for each of their services (which could potentially be simplified using generic adapters), it comes with two significant advantages: First, assuming proper versioning, services can cache data instances of data services and do not need to download them again. Second, if decision support services and the data store for (temporary) data generated during service execution are placed in the same data center, data exchange is significantly faster compared to the initially described “middleman engine”. However, this advantage can result in a cloud vendor lock-in.

Consideration of Metadata The previously discussed reference-based data exchange also comes with a disadvantage: When the BPMN process engine no longer receives any provided or produced data, it cannot evaluate the data to decide on the conditional invocation of decision support services as documented in the upcoming behavioral perspective. However, as indicated during the explanations of the language for describing decision support services in the previous Chapter 5, access to the complete data is not necessary to specify constraints of decision support services that may influence their conditional execution. Instead, metadata such as the size of a network topology is sufficient. Thus, data objects (and, consequently, the information passed to a decision support service during invocation) should not only store references to the required data but also the associated metadata. Similarly, the metadata for each method data should be returned in the response of a decision support service.

6.4.3 Behavioral Perspective: Sequence Flows, Gateways & Events

The functionality of a PD-DSS is not only characterized by the activities of the supported decision process and their implementation with decision support services, but also whether these activities are enacted in sequence or in parallel, or only under certain conditions. So far, the order of activity enactment has only been documented implicitly, e.g., with the left-to-right placement of activities in Fig. 6.4 or the data dependencies in Fig. 6.6. The behavioral perspective of a PD-DSS provides explicit documentation for the order of activity and service enactment. The relevant excerpt of the BPMN meta-model is shown in Fig. 6.7. The sequential execution of model elements is made explicit by connecting them with a `SequenceFlow`.

Gateways

A BPMN Gateway is used to control “how Sequence Flows interact as they converge and diverge within a Process” [Obj13]. Thus, a gateway can be used to describe the nonsequential invocation of decision support services. For this purpose, three types of gateways are relevant:

Parallel Execution of Decision Support Services A `ParallelGateway` initially splits an incoming sequence flow into multiple sequence flows that are subsequently executed in parallel. This can be used for the parallel execution of decision support services without data dependencies between each other for increased efficiency. The parallel sequence flows are later joined with another parallel gateway to ensure all parallel flows are completed before subsequent activities are executed.

Conditional Execution of Decision Support Services An `ExclusiveGateway` also has multiple outgoing sequence flows, but each flow is associated with a `FormalExpression` that

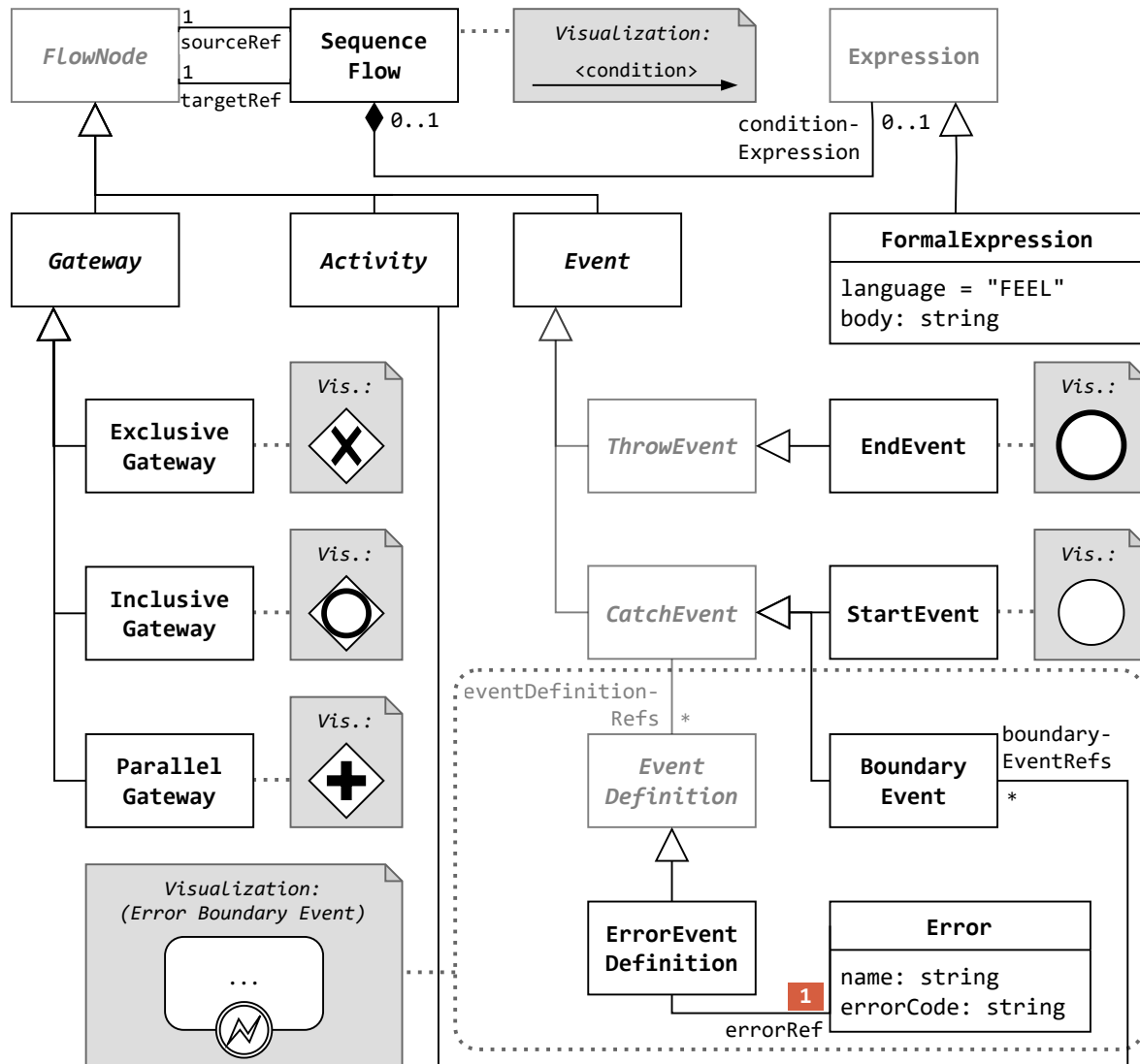


Figure 6.7: Annotated excerpt of the BPMN meta-model for the behavioral perspective of a PD-DSS process model (based on [Obj13])

uses a predefined expression language to formally document a condition that must evaluate to true in order to proceed with the sequence flow. During process execution, only the first sequence flow whose condition evaluates to true is selected. A human-readable form of the condition is written over the arrow representing the sequence flow. In the DSE context, an expression can be defined with respect to the metadata of a provided/produced data object. For example, the “size” attribute of a network topology can be evaluated to decide whether the network can still be optimized using a mathematical exact optimization approach, or whether a heuristic approach should be used to reduce execution time. For the documentation

of a condition as a formal expression, the *Friendly Enough Expression Language (FEEL)* of the BPMN-related standard *Decision Model and Notation (DMN)* is used [Obj21]. The relevant excerpt of the language grammar in extended Backus-Naur form (EBNF) is shown in Listing 6.1. The excerpt additionally introduces the clauses `object` holding all valid names of data objects of the BPMN process model, and `attribute` to reference a metadata attribute defined by the data type associated with the data stored in the data object. For example, the choice of using a mathematical exact optimization approach can be expressed as `NetworkTopology.Size < 1000`. In addition to data objects, a qualifier can also refer to a resource to support the execution of different activities based on resources available to the decision maker, e.g., to execute a heuristic optimization instead of a mathematical exact optimization if `Resources.Time < '1h'`.

Combined Conditional and Parallel Execution of Services An `InclusiveGateway` combines an exclusive and parallel gateway: All conditions of outgoing sequence flows are evaluated and potentially executed in parallel if multiple conditions evaluate to “true”.

Events

A BPMN Event describes something that can happen during process execution and requires or allows for a reaction, thus impacting the execution of tasks (cf. [Obj13]). Although BPMN defines multiple types of events (based on `EventDefinition`), the most relevant event for a composition of decision support services is an error event. An `ErrorEvent` as a `BoundaryEvent` is associated with an `Activity` and has an outgoing sequence flow

Listing 6.1: EBNF for expressing conditions based on FEEL [Obj21]

```

qualifier      = ( object, ".", attribute ) | resource ;
operator       = "=" | "!=" | "<" | "<=" | ">" | ">=" ;
literal        = string | integer | boolean | datetime ;
unarycomparison = qualifier, operator, (literal | qualifier) ;
list           = "[", ( (string, {",", string}) |
                        (integer, {",", integer}) ), "]" ;
listcomparison = (qualifier | list), " in ", (qualifier | list) ;
comparison     = unarycomparison | listcomparison ;
conjunction    = "(", condition, " and ", condition, ")" ;
disjunction    = "(", condition, " or ", condition, ")" ;
condition      = comparison | conjunction | disjunction ;

```

that is executed instead of the regular sequence flow attached to the activity if an error is caught. The error event only catches instances of the specified `Error` if referenced, otherwise, the alternative sequence flow is executed for all errors. This aligns with the exceptions that can potentially occur during the execution of a decision support service implementing a computation method (cf. Chapter 5). For example, a service task “mathematical exact optimization” for a network topology could throw an exception that the network topology is incomplete, which would trigger a user task to fix the topology and later restart the optimization. A boundary error event can potentially throw data that can be stored in a data object using a data association. Other relevant events include the `StartEvent` and `EndEvent`, which document the start and end of a process respectively.

Implications for the Design of a PD-DSS

In a BPMN-based *PD-DSS Design* application, the previously described considerations require a DSS engineer and the application to perform the following actions:

1. When creating a new process model, the *PD-DSS Design* application automatically adds a start event and end event for documenting the start and end of the process.
2. The DSS engineer places gateways before and after (sequences of) tasks that are conditionally executed or executed in parallel.
3. The DSS engineer adds boundary error events to activities that can throw an error and optionally specifies an error code unless all potentially thrown errors should be caught and handled in the same way. If data is associated with the error that is subsequently processed, the DSS engineer adds a data object representing error data and connects it to a subsequent activity with a data association.
4. The DSS engineer connects tasks, subprocesses, gateways, and events with sequence flows to document the order of their execution.
5. The DSS engineer adds conditions to sequence flows from gateways to activities by defining a FEEL-expression and (optionally) a human-readable label.

Figure 6.8 shows the behavioral perspective of an exemplary PD-DSS process model for energy distribution network planning. Again, process model elements from previously discussed perspectives are shown transparently. Based on the size of the network topology provided by the decision maker, either a mathematically exact or a heuristic optimization is performed. In case of an error related to the topology, the topology is fixed manually by the decision maker

based on the information provided with the error. If fixing the topology fails for any reason, the enactment of the process/composition is stopped.

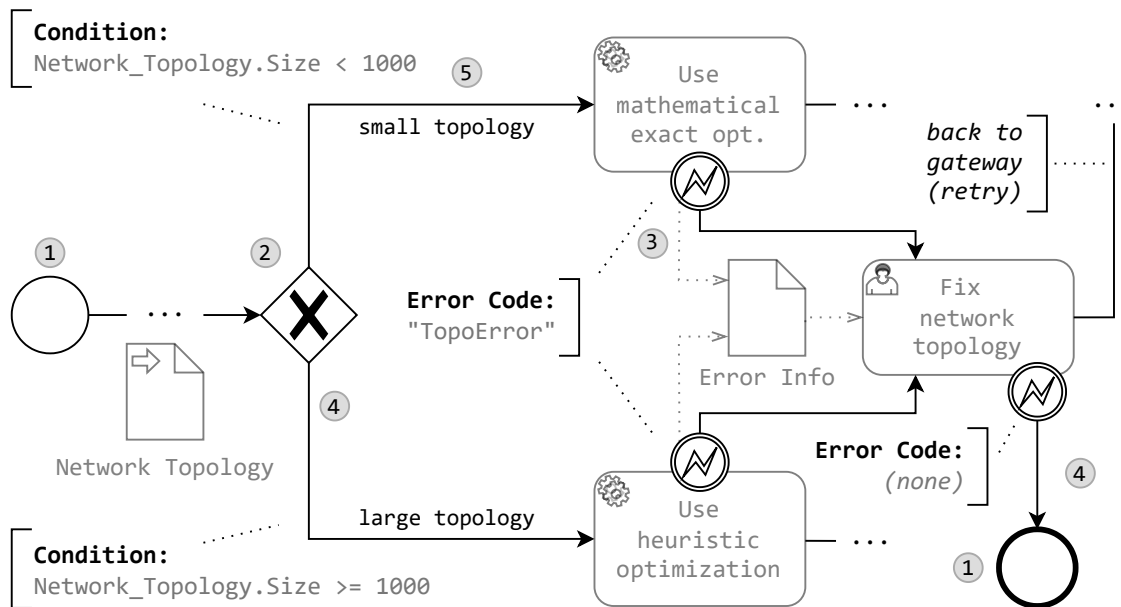


Figure 6.8: Behavioral perspective of an exemplary PD-DSS process model

Implications for the Enactment of a PD-DSS

The previously described concepts have no implications on the *PD-DSS Enactment* application other than the underlying BPMN process engine being required to support the FEEL expression language for the interpretation of conditions assigned to sequence flows. If the utilized BPMN process engine only supports a different expression language with similar expressiveness, FEEL-expressions can be converted to the supported expression language or the expression language can be used directly in the PD-DSS process model.

Since the BPMN process engine encapsulated in the *PD-DSS Enactment* application coordinates the invocation of decision support services, the perspective does not require any additional effort from decision makers during the enactment of the PD-DSS process model.

Decision support services should explicitly indicate in their response whether their execution was successful or not, as this implies whether the response should be interpreted as the regular output of the associated computation method or an error payload. For this purpose, the existence of the error code in the response is sufficient to indicate that an error occurred. Similar to a regular response, the error payload is returned as a reference to an item in the data store, and only associated metadata is included in the response.

6.5 Prototypical Implementation

This section provides technical insights to discuss the potential reusability of existing BPMN editors and process engines for the implementation of the previously described concept of using BPMN to model a tailored PD-DSS as a composition of decision support services. Due to this emphasis on reusability, the remainder of this section focuses on the implementation of the *PD-DSS Design* application (Section 6.5.1) and the *PD-DSS Enactment* application (Section 6.5.2). The *PD-DSS Management* application works fundamentally similar to a content management system and could therefore be built upon the headless CMS *Payload* that was already used for the prototypical implementation in Chapter 5. Nevertheless, a mockup showing a skeleton for the potential user interface of the *PD-DSS Management* application is shown in Fig. 6.9. The figure depicts the view for editing a single PD-DSS with references to the requirements for decision support that motivate the development of the PD-DSS, a preview of the PD-DSS process model with a button to edit the model using the *PD-DSS Design* application, and the results of previous PD-DSS enactments with a button to start a new enactment using the *PD-DSS Enactment* application.

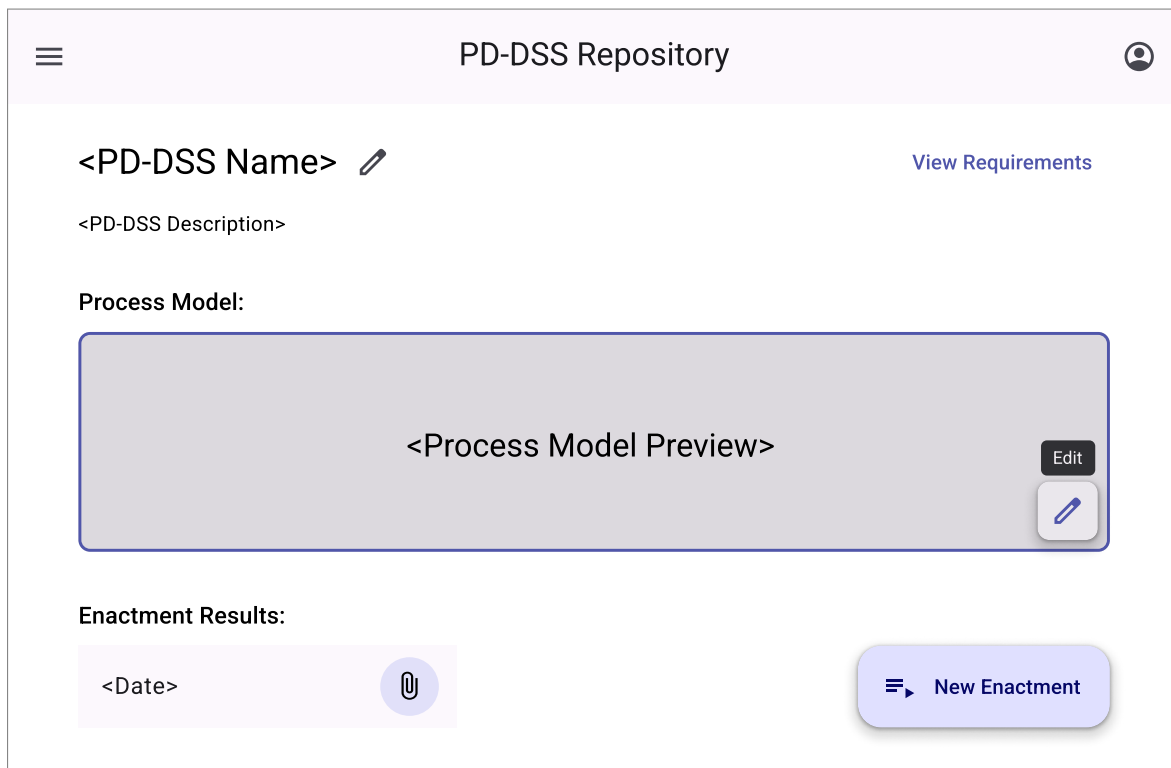


Figure 6.9: Mockup of the *PD-DSS Repository* application

6.5.1 Implementation of the PD-DSS Design Application

The *bpmn-js*¹ library was evaluated for the visual editing of a BPMN process model within the *PD-DSS Design* application. As shown in the screenshot depicted in Fig. 6.10, the application consists of the modeling canvas where the DSS engineer assembles the PD-DSS process model from the elements shown in the left toolbox, and a property editor on the right where the currently selected element is configured (shown as comments in Figs. 6.4, 6.6 and 6.8).

The prototypical implementation demonstrates the potential reusability of the *bpmn-js* library for the implementation of the *PD-DSS Design* application. However, for simplicity, the property editor saves properties as BPMN extension elements instead of manipulating the native BPMN entities. For example, the editor saves the ID of a selected decision support service using a service extension instead of creating/linking to an instance of *Operation* or *Rendering* respectively. This is because a conversion of the process model is necessary anyway due to limitations of existing BPMN process engines as explained during the upcoming discussion of the *PD-DSS Enactment* application, and implementing the property editor based on extension elements increases code reusability and thereby reduces implementation effort.

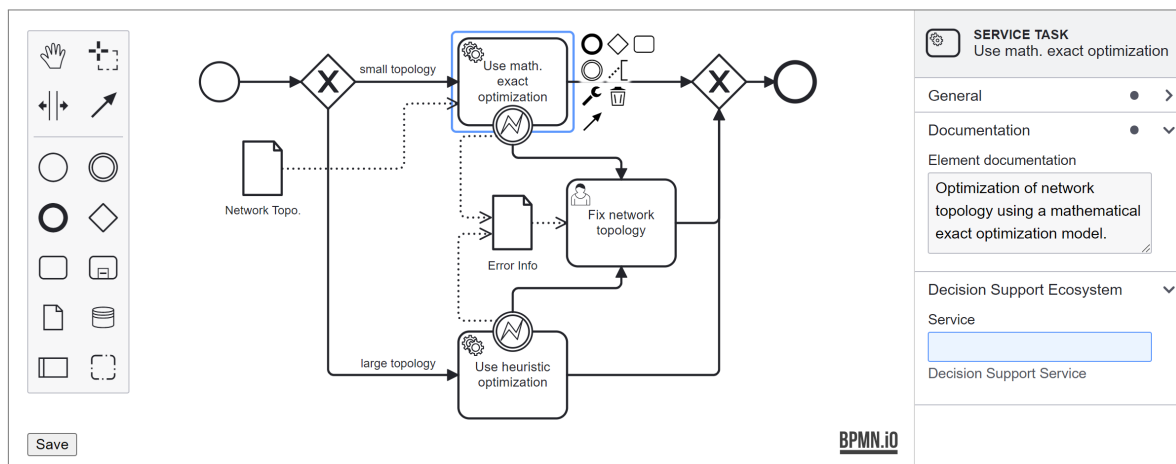


Figure 6.10: Screenshot of the *PD-DSS Design* application

6.5.2 Implementation of the PD-DSS Enactment Application

The core of the *PD-DSS Enactment* application is the BPMN process engine. It invokes decision support services as specified in the PD-DSS process model. The most practically

¹ Library website: <https://bpmn.io/toolkit/bpmn-js/>

relevant BPMN process engines include *Activiti*², *Camunda*³, and *jBPM*⁴ [Len16; Len+18]. However, all engines are subject to two challenges that require a workaround to enable their reuse for the implementation of the *PD-DSS Enactment* application:

Challenge 1: Different Implementations of the BPMN Specification

Although subject to the same BPMN specification, BPMN process engines have different capabilities. According to Lenhard et al. [Len+18], this can be attributed to inconsistencies and ambiguities in the BPMN specification, process engines deviating from the specification or only implementing parts of it, and the overall absence of a certification process that guarantees the compatibility of a process engine with the BPMN specification. For example, all described engines use an engine-specific custom extension for the invocation of external services via HTTP instead of relying on the message-based communication approach included in the BPMN specification. The *Camunda* engine even uses an engine-specific approach to document the data exchange between activities [Cam22]. Attempts to define a compatibility layer across engines in the form of *BPMN-I* [Sil11, Ch. 18] have not been successful so far.

The previously described observation implies that a PD-DSS process model created with the *PD-DSS Design* application must first be converted into an engine-specific process model before it can be enacted. The *PD-DSS Enactment* application can transparently perform this conversion after instantiation, i.e., after the decision maker has selected the input data for enactment (cf. Fig. 6.11). The prototypical evaluation demonstrates the fundamental feasibility of such a conversion for the *Camunda* engine, which was selected due to its accessible documentation and support for FEEL-expressions.

Challenge 2: Interactive User Tasks

A second challenge that affects all three previously listed BPMN process engines is the representation of interactive decision support services as user tasks. Both *Camunda* and *Activiti* only support the display of forms associated with a user task, which does not support the integration of arbitrary (web-based) user interfaces as required for an integrated interactive decision support service. The *jBPM* engine supports the *WS-HumanTask* [OAS10] specification, which – similar to the BPMN specification – does not include concrete details on how to define user interface renderings for user tasks.

As a workaround, the selected *Camunda* process engine includes the *Tasklist* application⁵,

² Website: <https://www.activiti.org/>

³ Website: <https://camunda.com/>

⁴ Website: <https://www.jbpm.org/>

⁵ *TaskList* documentation: <https://docs.camunda.io/docs/components/tasklist/introduction-to-tasklist/>

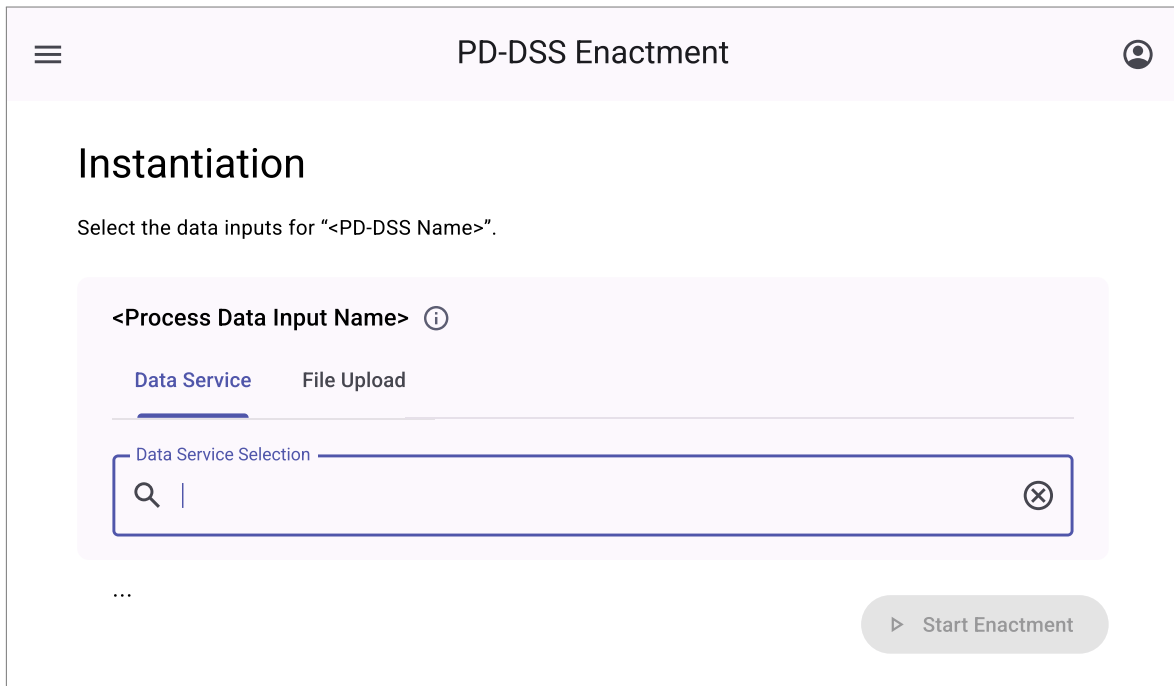


Figure 6.11: Mockup of the *PD-DSS Enactment* application for initial data selection

which provides an API to list and complete user tasks. This API can be used by the *Interaction* component of the *PD-DSS Enactment* application to query open user tasks, display the associated interactive decision support services to the decision maker, and return the results of the interaction to the process engine to continue enactment of the service composition (cf. *UI Container* in Fig. 6.12, which gets replaced by the user interface of the currently executed integrated decision support service, a generic user interface for the download and upload of data required for or produced by an uncontrolled decision support service, or a placeholder indicating the execution of an automated decision support service). Since the implementation of alternative web-based frontends for the *Tasklist* application have already been discussed in the *Camunda* blog [Mue18], they were not considered throughout the prototypical implementation.

6.6 Discussion

This section discusses the presented conceptual and technical insights with respect to the composition requirements introduced in Section 6.1. Since the requirements of the “Process” category directly map to the subsections of Section 6.4 (i.e., *CR1 – Functional Perspective* and *CR2 – Operational Perspective* to Section 6.4.1, *CR4 – Informational Perspective* to

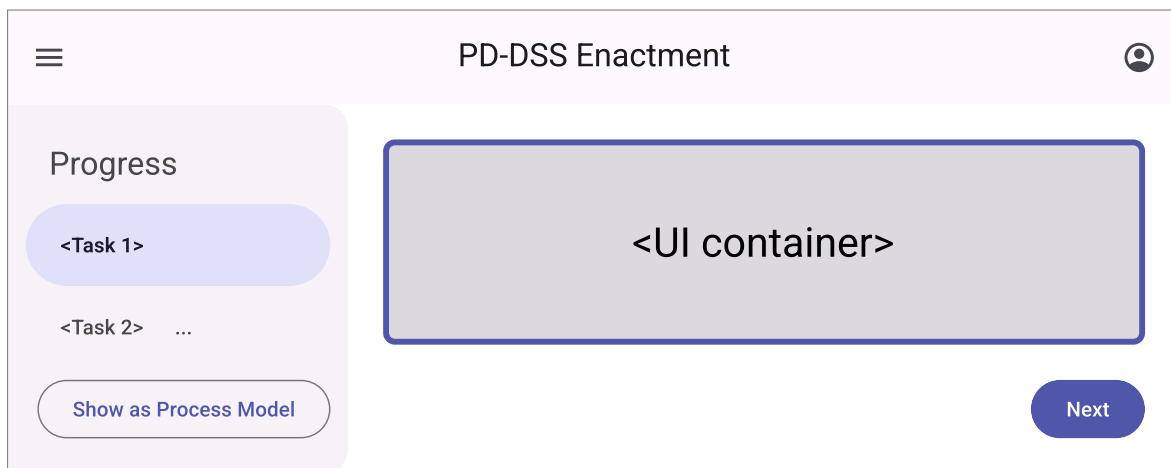


Figure 6.12: Mockup of the *PD-DSS Enactment* application during process model execution

Section 6.4.2, and *CR3 – Behavioral Perspective* to Section 6.4.3), the associated insights are not repeated here to avoid redundancy. The fundamental applicability of the described concepts was demonstrated with the excerpts of exemplary PD-DSS process models in the domain of energy distribution network planning. In this context, it is important to note that the strict separation of perspectives and design actions among sections was chosen for presentation purposes. In a real-world situation, a DSS engineer will likely iteratively refine a process model for all perspectives, e.g., add two tasks, connect them with a sequence flow and document their data exchange before adding additional activities.

The reusability of a designed PD-DSS, which is required as part of *CR5 – Instantiation*, is achieved with process-level data inputs during the design of the PD-DSS for which the decision maker must select (temporary) data services when instantiating the PD-DSS process model before its enactment. The reuse of an existing BPMN process engine for the enactment of the process model furthermore addresses *CR6 – Orchestration*. Both the design and enactment of a PD-DSS process model support (web-based) automated and interactive decision support services, which addresses *CR7 – Service Types*. In particular, the option to integrate custom user interface elements into the PD-DSS differentiates the approach from other tools for the definition of data engineering pipelines such as *Apache Airflow*⁶. A feature that is prominent in BPMN but not considered so far is the assignment of different user tasks to different workers.

The visual notation provided by BPMN, in addition to the fact that BPMN is widespread and applicable for domain experts without extensive upfront training, addresses both *CR8 – Visual Programming* and *CR9 – Learnability*. Learnability is further increased by transforming a PD-DSS process-model created using the *PD-DSS Design* application into a representation

⁶ Website: <https://airflow.apache.org/>

that is specific to the utilized BPMN process engine, allowing DSS engineers to focus on the visual specification of the PD-DSS process model without considering its technical implementation. The scalability of the notation, i.e., its suitability to represent extensive process models, should be further evaluated, especially for the informational perspective as many data objects and associations may become too “visually verbose”. However, potential scalability issues can be addressed by using subprocesses as those can be collapsed within a single process model, `LinkEvents` to add additional layout flexibility, or a `CallActivity` to invoke a (sub)process specified in a different process model. The latter can also foster the reusability of partial PD-DSS process models.

As evident from the architectural overview given in Section 6.3, the composition approach integrates with other DSE applications. In particular, the integration includes interfacing with the DSE service registry to obtain information about available decision support services (*CR10 – Service Discovery*) and the *PD-DSS Management* application integrates with the *Requirements Elicitation* application to document the requirements for decision support that motivated the development of a tailored PD-DSS (*CR11 – Model Repository*). However, the traceability between requirements and design currently is only provided textually with the description of the PD-DSS and could be transformed into a structured representation for further (automated) processing. Lastly, the XML representation of a BPMN process model addresses *CR12 – Machine Readability*.

6.7 Key Takeaways

A tailored DSS can be represented as a BPMN process model that describes a composition of decision support services. Such a DSS is referred to as a process-driven decision support system (PD-DSS). A BPMN process model is especially suitable for composing decision support services due to its ability to capture the functional, operational, informational, and behavioral aspects of a DSS. Although a subset of the BPMN specification is sufficient to model a tailored PD-DSS, the prototypical implementation of the PD-DSS concept indicates that the reuse of an existing BPMN process engine to support the enactment of a PD-DSS requires model transformation to address the limitations of available engines. The final architecture of the implementation consists of three applications for the design, enactment, and management of PD-DSS process models.

The upcoming chapter describes a composition assistance that integrates with the *PD-DSS Design* application to support DSS engineers in designing the functional, operational, informational, and behavioral characteristics of a PD-DSS.

Composition Assistance

The process-driven approach to DSS development presented in Chapter 6 already enables DSS engineers to provide each decision maker with a tailored (PD-)DSS as a composition of decision support services. Nevertheless, the overall DSE design proposed in Chapter 4 recommends an additional assistance system that supports DSS engineers during service composition by automatically suggesting improvements to the efficiency and effectiveness of a PD-DSS service composition. The composition knowledge propagated by the composition assistance is gathered throughout the last DSE lifecycle phase, for example, from feedback provided by decision makers after using a PD-DSS (cf. Fig. 7.1).

This chapter contributes a design for such a supporting composition assistance based on the papers “Detecting Data Incompatibilities in Process-Driven Decision Support Systems” by Kirchhoff, Gottschalk, and Engels [KGE22] and “Anti-pattern Detection in Process-Driven Decision Support Systems” by Kirchhoff and Engels [KE22]. In addition to composition knowledge manually contributed by composition experts, the assistance can also automatically

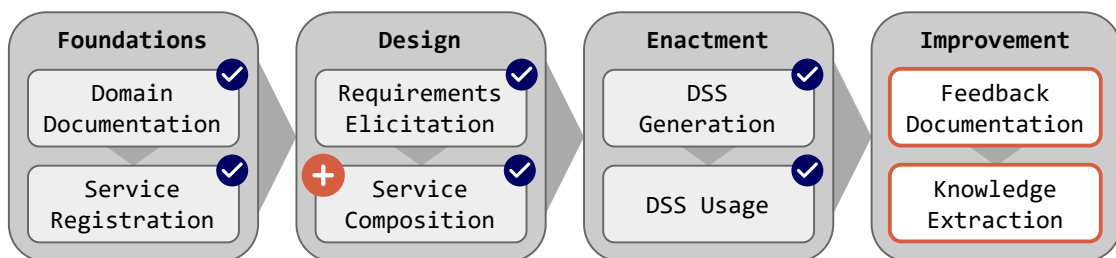


Figure 7.1: Focus of Chapter 7 with respect to the DSE lifecycle

derive composition knowledge from the documentation of the application domain and its decision support services. The proposed assistance design integrates with the *PD-DSS Design* application used by DSS engineers and continuously provides feedback to improve functional, behavioral, informational, and operational characteristics of a PD-DSS process model.

The chapter first discusses some fundamental design considerations for the composition assistance, including its architectural integration into the PD-DSS development environment described in Chapter 6 and a documentation format that is used throughout subsequent sections (Section 7.1). The insights of the discussion are then transformed into fundamental requirements for the composition assistance (Section 7.2). The design of the composition assistance consists of three individual assistance implementations, which focus on the validation of different perspectives in a PD-DSS process model documenting the composition of decision support services, i.e., the operational (Section 7.3), informational (Section 7.4) and functional and behavioral perspective (Section 7.5). Each subsection for an individual assistance defines additional requirements, includes a discussion of related approaches, and describes the design and demonstration of the assistance, including the provision of composition knowledge. Lastly, the insights over all assistance approaches are summarized (Section 7.6).

7.1 Upfront Design Considerations

This section first discusses multiple fundamental approaches to the implementation of a composition assistance and presents a validation-based composition assistance as the most impactful approach when establishing a DSE (Section 7.1.1). Afterwards, the architectural integration of a validation-based assistance into the PD-DSS development environment of Chapter 6 is explained (Section 7.1.2) and the description format for validation rules is presented as a foundation for subsequent sections (Section 7.1.3).

7.1.1 Approaches to Composition Assistance

The *composition assistance* can be any approach that integrates with the *PD-DSS Design* application used by DSS engineers to support them in the design of an effective and efficient PD-DSS. Providing a detailed solution design for all imaginable assistance approaches is out of the scope of this thesis. Therefore, this section discusses the advantages and disadvantages of fundamental composition assistance approaches and selects an approach that is most beneficial for the implementation of a DSE. Approaches are subsequently classified into recommendation-based and validation-based composition assistance.

Recommendation-Based Composition Assistance

The goal of *recommendation-based assistance* is to proactively enforce the design of service compositions that effectively and efficiently address the requirements for decision support of a decision maker. For this purpose, the assistance restricts or suggests changes to the process-based composition of decision support services. This thesis considers three (non-exclusive) approaches to the implementation of a recommendation-based assistance:

Pattern-based Composition Assistance provides DSS engineers with patterns as blueprints for service compositions that can be instantiated and potentially nested, but not extended from within the *PD-DSS Design* application. Thus, patterns constrain service compositions that can be designed to those that are demonstrably effective and efficient. A pattern essentially defines the activities of a (partial) decision process and their execution sequence. As a result, a DSS engineer only needs to select decision support services for the implementation of these activities and define the dataflow between them. For energy distribution network planning, a DSS engineer may for example first select a generic process pattern for the optimization of a network topology that consists of the phases “scenario creation”, “load forecasting”, “topology optimization”, and “investment plan identification”. Next, the DSS engineer refines each phase by selecting a nested pattern, e.g., a topology optimization that already includes the conditional execution of a network topology reduction on the condition that the network topology exceeds a certain size. Finally, the DSS engineer selects the implementing decision support services for activities. The described approach is very similar to the approach implemented by Gottschalk et al. [Got+23] for situational business model development. In the DSE context, patterns can be extracted by composition experts based on the service compositions stored in the *PD-DSS Repository* application.

Arguably, a lightweight variant of the pattern-based composition approach is already contained in the PD-DSS development environment described in Chapter 6: A DSS engineer can obtain an existing composition of decision support services from the *PD-DSS Repository* application and replace the contained decision support services without having to change the dataflow between activities if both the previously selected services and the newly chosen services implement the same computation method. While this approach does not support pattern nesting out-of-the-box, it still provides DSS engineers with an option to build their composition of decision support services based on established practices. However, this approach would still allow DSS engineers to edit the resulting composition of decision support services. Since mistakes can be introduced during these manual edits, there is still a need to validate the designed service composition afterwards. This validation need can be avoided by preventing any manual edits, however, this may restrict a DSS engineer in effectively addressing a decision maker’s requirements for decision support if the use case of the decision

maker is not yet covered by existing compositions. This disadvantage also holds for the originally described pattern-based composition approach if no pattern combination aligns with the use case or requirements of the decision maker. Furthermore, it is still possible to nest and combine patterns that are incompatible or do not address the requirements for decision support documented by a decision maker. Therefore, validation of a PD-DSS process model is also required when using a pattern-based composition approach.

Usage-based Composition Assistance uses historical usage data to provide a DSS engineer with recommendations for decision activities and implementing decision support services to use for the development of a PD-DSS. These recommendations can be described as a conditional probability, i.e., given the defined decision process so far, what activity with which computation method and decision support service will most likely come next? For example, in the domain of energy distribution network planning, the assistance might suggest starting the process with an activity for simulating consumer demands, which is implemented by a decision support service that extrapolates historical consumer demands. The conditional probabilities for computation methods and services can be determined based on their usage in the compositions stored in the *PD-DSS Repository* application described in Chapter 6.

Two variants of this approach are imaginable: The first variant presents multiple recommendations for the next activity to the DSS engineer, who is responsible for the final selection. The second variant iteratively combines the most probable recommendation to complete the whole decision process, thus reducing the involvement of the DSS engineer to the correction of mistakes and restarting the recommendation-based generation from that point onwards. Both variants come with the risk of “error propagation”, i.e., mistakes that are frequently included in existing compositions are most likely also recommended for new compositions. There are consequently high expectations on DSS engineers to catch such mistakes as well as misalignments of the recommendations with the requirements for decision support documented by a decision maker. The workload of DSS engineers can be reduced by introducing an additional validation of the proposed service compositions.

Automation-based Composition Assistance utilizes automated service composition to provide a DSS engineer with a complete composition to address all individual requirements for decision support of a decision maker. This approach is similar to the iterative variant of the previous usage-based composition assistance, however, deterministic approaches for service selection based on domain knowledge and the underlying requirements for decision support are applied, e.g., similar to the service-oriented DSS approaches discussed in Section 3.3.3. The responsibilities of a DSS engineer thus reduce to either selecting the most suitable composition if multiple compositions are presented, or adapting a single recommended composition in case it addresses most but not all documented requirements for decision support.

Although the previous explanations of the automation-based composition assistance suggest that the involvement of the DSS engineer can be reduced and therefore the efficiency during PD-DSS development can be increased, this is not necessarily the case for multiple reasons: First, it is hard to estimate how many compositions are possible to address a specific set of decision support requirements. Especially when the automated composition algorithm has no semantic understanding of the decision support activities in an application domain, the DSS engineer must potentially browse through many unsuitable compositions before finding one which is efficient and effective with respect to the considered requirements for decision support. This also requires the DSS engineer to validate each proposed service composition manually. If only one composition is proposed, or a “close-enough” composition is identified and manually refined by the DSS engineer, the DSS engineer can still introduce mistakes into the service composition. Thus, this approach would also profit from an additional validation of the proposed/refined composition of decision support services.

Validation-Based Composition Assistance

A *validation-based composition assistance* improves the quality of a decision support service composition by reporting any flaws contained in the composition to the DSS engineer via the *PD-DSS Design* application. A service composition can either be statically validated during its design or dynamically validated based on test cases that are executed against the generated PD-DSS. Furthermore, validation can be performed manually or automatically. However, manual validation of the composition by a composition expert is expected to result in inefficient PD-DSS development due to the DSS engineer having to wait for feedback from the composition expert. Instead, a validation-based composition assistance should provide automated feedback. During design, this can for example be achieved via a mapping to formal verification approaches that can be executed before the PD-DSS is generated from the service composition and used by the decision maker. Assuming that validation results can be computed in (near-)real-time such that the DSS engineer can immediately address any identified issues, an automated validation-based assistance counts as *constructive quality assurance* since the quality of the derived artifact (PD-DSS) is enforced during its construction [Hof08, Ch. 3].

Implications for the Design of the DSE Composition Assistance

Since any variant of the recommendation-based composition assistance should be accompanied by a validation of the proposed service composition, this thesis subsequently focuses on the design of an automated validation-based composition assistance that continuously evaluates a service composition during its design. A PD-DSS process model can be validated regarding its

completeness with respect to the modeling conventions of Chapter 6 and its *compliance* with respect to the requirements for decision support documented by a decision maker captured using the description approach of Chapter 5. Since the validation rules for process model completeness can be directly derived from the explanations of Section 6.4 and require limited explanations, they are only provided as supplementary material in Appendix A.

7.1.2 Integration into the DSE Platform

Figure 7.2 shows the architectural integration of the validation-based composition assistance into the DSE platform. The assistance receives the requirements for decision support specified by the decision maker and the associated (intermediate) PD-DSS process model created by the DSS engineer via the *PD-DSS Design* application. In turn, the composition assistance provides feedback regarding the PD-DSS process model, which can be displayed to the DSS engineer either as a list or within the modeling canvas next to the affected elements of the PD-DSS process model. The composition assistance partially derives composition knowledge from the documentation of the application domain and its decision support services.

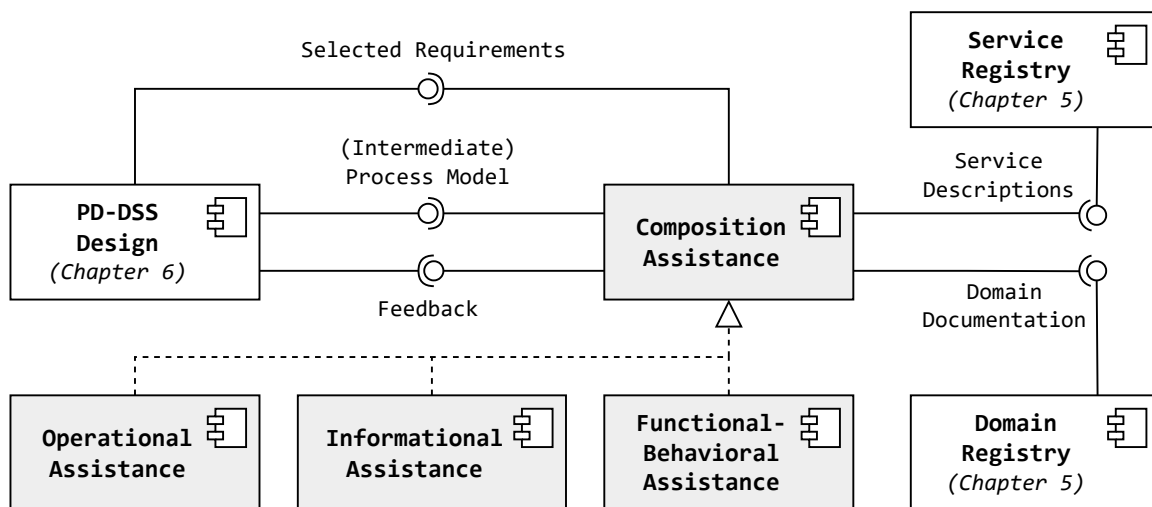


Figure 7.2: Integration of the composition assistance into the DSE platform

The remainder of this chapter describes three complementary approaches to the implementation of the composition assistance with a focus on the validation of the operational perspective (Section 7.3), the informational perspective (Section 7.4), and the functional and behavioral perspective (Section 7.5) with additional architectural considerations to include composition knowledge of composition experts.

7.1.3 Documentation of Validation Rules

Any non-conformity of a designed PD-DSS process model with the PD-DSS modeling conventions and the requirements for decision support of a decision maker is reported to the DSS engineer as a *violation*, i.e., a flaw that must or should be addressed by updating the PD-DSS process model. The explanations of the assistance throughout the remainder of this chapter (and Appendix A) use a tabular format to provide instructions for the detection and communication of a violation (referred to as “*violation rule*”). An example is given with the *Missing Service Violation* to subsequently explain the structure of the documentation format.

Missing Service Violation		Severity: Error
Element:	Task	Phase: Design
Condition:	No decision support service is selected for a task in the process model.	
Message:	“ <i>No decision support service is specified for decision activity [...].</i> ”	

Title The title (“Missing Service Violation”) is a concise label that is presented to the DSS engineer upon detection of a violation. The title is also referenced in textual explanations.

Severity Level Violations define a *severity level* of error or warning. A *composition error* indicates a flaw in the PD-DSS process model that would prevent or interrupt its subsequent enactment, e.g., because no decision support service is documented for the enactment of an activity in case of the *Missing Service Violation* example, or because there is a significant deviation in the requirements for decision support documented by the decision maker, e.g., the use of a prohibited computation method. A *composition warning* is used for violations that do not prevent the enactment of a PD-DSS but are an indicator for a “bad smell”, e.g., the generation of unused data or the use of a computation method that should be avoided.

Element and Condition The element and condition properties of a violation rule describe what process model elements need to be analyzed to detect the violation and what condition must hold for the violation to be present. For the *Missing Service Violation*, each task of the process model must be analyzed to check if a decision support service has been selected for the implementation of the task.

Phase The phase of a violation rule documents when the violation is checked for: during the design of the PD-DSS process model or during instantiation, i.e., after the decision maker has selected data for the data inputs of the process, but before any decision support service has been invoked. For the *Missing Service Violation*, the presence of a service selection can be analyzed during the design of the PD-DSS process model.

Message The message, in addition to the title and the severity level, is returned to the *PD-DSS Design* application for display to the DSS engineer. An ellipsis (*[...]*) in a message is a parameter that includes the name or property of affected elements that motivate the reporting of the violation. For the *Missing Service Violation*, an example for a message is “*No decision support service is specified for decision activity [Optimize network topology]*”, where “*Optimize network topology*” is the label of the task for which no decision support service was selected. It is assumed that DSS engineers can click on a parameter value in the *PD-DSS Design* application to jump to the affected element for efficient bug fixing.

7.2 Cross-Cutting Requirements

The validation-based assistance approaches described in the next three sections focus on the detection of violations in different PD-DSS process model perspectives, i.e., the operational, informational, and functional/behavioral perspective. Although each perspective has individual design requirements, the previous considerations imply several cross-cutting assistance requirements *ARCx* that affect all perspectives and are summarized here to avoid redundancy.

Assistance Requirement ARC1 – Static Validation

The approaches for composition assistance should validate the effectiveness and efficiency of the designed service composition before the resulting PD-DSS is used by decision makers during the enactment of their decision process. This upfront validation ensures that the decision process can be enacted without requiring interruptions to implement fixes in the service composition as this could potentially invalidate any progress made so far. This implies that violations should preferably be detected during the design phase whenever possible.

Assistance Requirement ARC2 – Traceable Feedback

The feedback provided to a DSS engineer should be traceable such that affected process model elements can be quickly identified and corrected. This implies the use of parameters in the message of a violation rule whenever possible.

Assistance Requirement ARC3 – Comprehensive Feedback

The feedback provided to a DSS engineer should be comprehensive, i.e., explain the reason for a violation so that a DSS engineer knows how to correct the flaw in the service composition.

Assistance Requirement ARC4 – Continuous Feedback

A DSS engineer should receive continuous feedback during service composition and not only at the end of the composition activity. This ensures that flaws are detected early and can be addressed before completing the composition on a defective foundation.

7.3 Operational Composition Assistance

The operational composition assistance validates the operational perspective of a PD-DSS process model, which documents the functional decision support services selected by a DSS engineer for the implementation of activities within the decision process (cf. Chapter 6). This section first discusses the potential benefits of the operational composition assistance in the context of a motivational scenario (Section 7.3.1). Based on the discussion, additional requirements specific to the operational composition assistance are defined (Section 7.3.2) and a brief overview of related approaches is given (Section 7.3.3). A solution design is proposed and demonstrated for the example of energy distribution network planning (Section 7.3.4). The section concludes with a discussion of the presented solution design with respect to the requirements for the (operational) composition assistance and an outlook of how it can be used for the implementation of a recommender system (Section 7.3.5).

7.3.1 Motivational Scenario

Ineffectiveness and inefficiency in the operational perspective of a PD-DSS process model can either be introduced due to the selection of an unsuitable decision support service or due to no service being selected at all. Since a missing selection is already detected with the completeness check described in Appendix A.1.1, the subsequent explanations focus on the selection of unsuitable services, i.e., services that do not align with the requirements for decision support of a decision maker. This misalignment may be due to functional characteristics, e.g., because the decision maker wants to minimize the investment costs of an electricity distribution network during its redesign, but the selected decision support service is ineffective as it only supports the minimization of operational costs. Alternatively, the misalignment may also stem from non-functional characteristics, e.g., a decision support service consuming too many resources so that the decision maker is unable to complete the decision process with the PD-DSS. While a PD-DSS that is unable to support the complete decision process is arguably ineffective, the PD-DSS is also inefficient with respect to the resources available to the decision maker. Furthermore, the development of the PD-DSS itself is inefficient when it must be redesigned to account for the lack of resources.

7.3.2 Requirements

In addition to the cross-cutting requirements for composition assistance introduced in Section 7.2, the following requirements ARO_x for the operational composition assistance can be derived from the meta-model for describing decision support requirements (cf. Section 5.4.6).

Assistance Requirement ARO1 – Optimization Characteristics

A computation method can be implemented using multiple decision support services that solve different optimization problems. The alignment of the optimization problem between selected decision support services and specified requirements for decision support should be validated by the operational composition assistance.

Assistance Requirement ARO2 – Non-Functional Characteristics

The descriptions of decision support services also document non-functional characteristics, particularly the resources that are consumed during service execution and quality guarantees in the form of service level objectives. The alignment between provided non-functional characteristics of the selected decision support services and the documented requirements for decision support should be validated by the operational composition assistance.

7.3.3 Background and Related Work

This section discusses existing approaches for the operational validation of business processes to assess their reusability for PD-DSS development. The discussion includes approaches for validating process-driven applications as well as business process models in general.

Schneid et al. [Sch+19] describe an approach for the static analysis of BPMN process models for process-driven applications (PDAs). With respect to the operational perspective, the approach validates that a service (operation) specified for the execution of a service task exists and implements a specific interface potentially required by the utilized BPMN for the enactment of the process model. With respect to the requirements for the operational composition assistance established in Section 7.3.2, this approach neither addresses *ARO1 – Optimization Characteristics* nor *ARO2 – Non-Functional Characteristics* as it solely validates the correctness of the PDA's operational perspective on a technical implementation level.

Schneid et al. [Sch+21c] describe how process analysts can specify regression tests for a PDA by documenting input data combined with an expected process path and output data. These tests can validate the correctness of all process perspectives since they are executed against the derived PDA. However, the operational perspective is essentially excluded if the invoked services are replaced by mocks as suggested in the description of the approach. Furthermore, the derivation of the PDA for test execution requires the process model to be complete before the tests can be executed, which limits *ARC4 – Continuous Feedback*.

In the context of process model validation in general, Awad et al. [Awa+09] extend the BPMN meta-model to capture resource allocation constraints using the *Object Constraint Language (OCL)*. However, they solely focus on the assignment of human resources, which excludes many of the material resources bundled under *ARO2 – Non-Functional Characteristics*.

7.3.4 Design and Demonstration

Since the previously discussed related approaches address the requirements for the operational composition assistance only partially at most, this section presents a design for the operational composition assistance to validate optimization characteristics and non-functional characteristics of a PD-DSS process model.

Validation of Optimization Characteristics

The *optimization characteristics* of an existing decision support service are characterized by optimization objectives, optimization constraints, and the approach to generating decision alternatives (cf. Section 5.4.3). The documentation of a decision maker's requirements for decision support also makes use of these optimization characteristics (cf. *OptimizationRequirement* in Section 5.4.6), but with the addition of a requirement level (e.g., *MUST*, *MUST NOT*, etc.).

Initial violations can be defined with respect to the *MUST* and *MUST NOT* requirement levels, which require the (non-)existence of a service with the specified optimization characteristics (cf. *MUST Optimization Violation* and *MUST NOT Optimization Violation*). Analogously, violations can be defined for the requirement levels *SHOULD* and *SHOULD NOT*. However, the severity level is changed to a warning since *SHOULD (NOT)* allows DSS engineers to disregard a requirement, although they are discouraged to do so (cf. definition of requirement levels in Section 5.4.6). This results in two additional violations (*SHOULD Optimization Violation* and *SHOULD NOT Optimization Violation*). The requirement level *MAY* has no impact on validation due to its optionality.

MUST Optimization Violation		Severity: Error
Element:	Task	Phase: Design
Condition:	For a given optimization requirement with requirement level <i>MUST</i> , no task in the process model is implemented by a decision support service with the optimization characteristics specified by the optimization requirement.	
Message:	“No decision activity addresses optimization characteristic [...].”	

Validation of Non-Functional Characteristics

As described in Section 5.4.4, the invocation of a functional decision support service may consume resources. For each resource, the sum of consumed resource quantities over all selected services cannot exceed the available resource quantity documented by the decision

MUST NOT Optimization Violation		Severity: Error
Element:	Task	Phase: Design
Condition:	For a given optimization requirement with requirement level MUST NOT, a task in the process model is implemented by a decision support service with the optimization characteristics specified by the optimization requirement.	
Message:	“Service [...] selected for decision activity [...] has disallowed optimization characteristic [...].”	
SHOULD Optimization Violation		Severity: Warning
Similar to:	“MUST Optimization Violation”, but with SHOULD requirement level	
SHOULD NOT Optimization Violation		Severity: Warning
Similar to:	“MUST NOT Optimization Violation”, but with SHOULD NOT req. level	
Message:	“Service [...] selected for decision activity [...] has <i>discouraged</i> optimization characteristic [...].”	

maker as part of the requirements for decision support. This is checked as part of the *Resource Violation*. Whether this violation can be checked during the design or instantiation of the composition depends on how resource consumption is specified by selected decision support services. Services can either specify the quantity of consumed resources as an absolute value, or relative to a characteristic of the input data. However, the characteristics of input data are only available after the process model representing the service composition has been instantiated by the decision maker. Thus, this violation is only checked during the design of the decision support service composition for a given resource if all decision support services specify resource consumption for this resource as absolute quantities. If one or more decision support services specify resource consumption as a relative quantity, a violation for the affected resource can only be checked during instantiation.

A service level objective (SLO) defines a provided or desired lower boundary for a quality characteristic of a decision support service, e.g., its availability. The alignment between provided and required SLOs is checked as part of the *SLO Violation*.

Demonstration

A demonstration of the previously described concepts is given with the (partial) case study for energy distribution network planning shown in Fig. 7.3. The top of the figure shows a partial BPMN process model describing the composition of decision support services. Since the

Resource Violation		Severity: Error
Element:	Task	Phase: Design, Instantiation
Condition:	For a given resource, the sum of resource quantities consumed by decision support services exceeds the consumable resource quantities defined in the decision support requirements.	
Message:	“The sum of consumed quantities (...) exceeds the available quantity (...) for resource [...].”	
SLO Violation		Severity: Error
Element:	Task	Phase: Design
Condition:	For a given service level indicator, a selected decision support service guarantees a lower/upper bound that is below/above the threshold specified in the requirements for decision support.	
Message:	“The service for decision activity [...] guarantees a [lower/upper] bound of [...] for SLI [...], but required is a [lower/upper] threshold of [...] or [higher/lower].”	

first user task has no associated decision support service, the completeness check discussed in Appendix A.1.1 reports a *Missing Service Selection Violation* (E1). Characteristics of the functional decision support service selected for the mathematical exact optimization are shown on the right side of the object diagram. The shown information is an excerpt of the service description in the service registry (cf. Chapter 5). Since the service references an optimization target that is disallowed by the required service capturing the decision maker’s requirements for decision support shown on the left side of the object diagram, a *MUST NOT Optimization Violation* is reported (E2). A *Resource Violation* (E3) can be reported during the design of the PD-DSS process model as the resource consumption of the service is specified absolutely and exceeds the available resource quantity. Furthermore, a *SLO Violation* (E4) is reported since the guaranteed availability of the service does not meet the requirements.

7.3.5 Discussion

The demonstration included with the previously presented design of the operational composition assistance demonstrates the fundamental applicability and usefulness of the assistance. This section explains the relation between the presented concepts and the individual and cross-cutting requirements for the operational composition assistance.

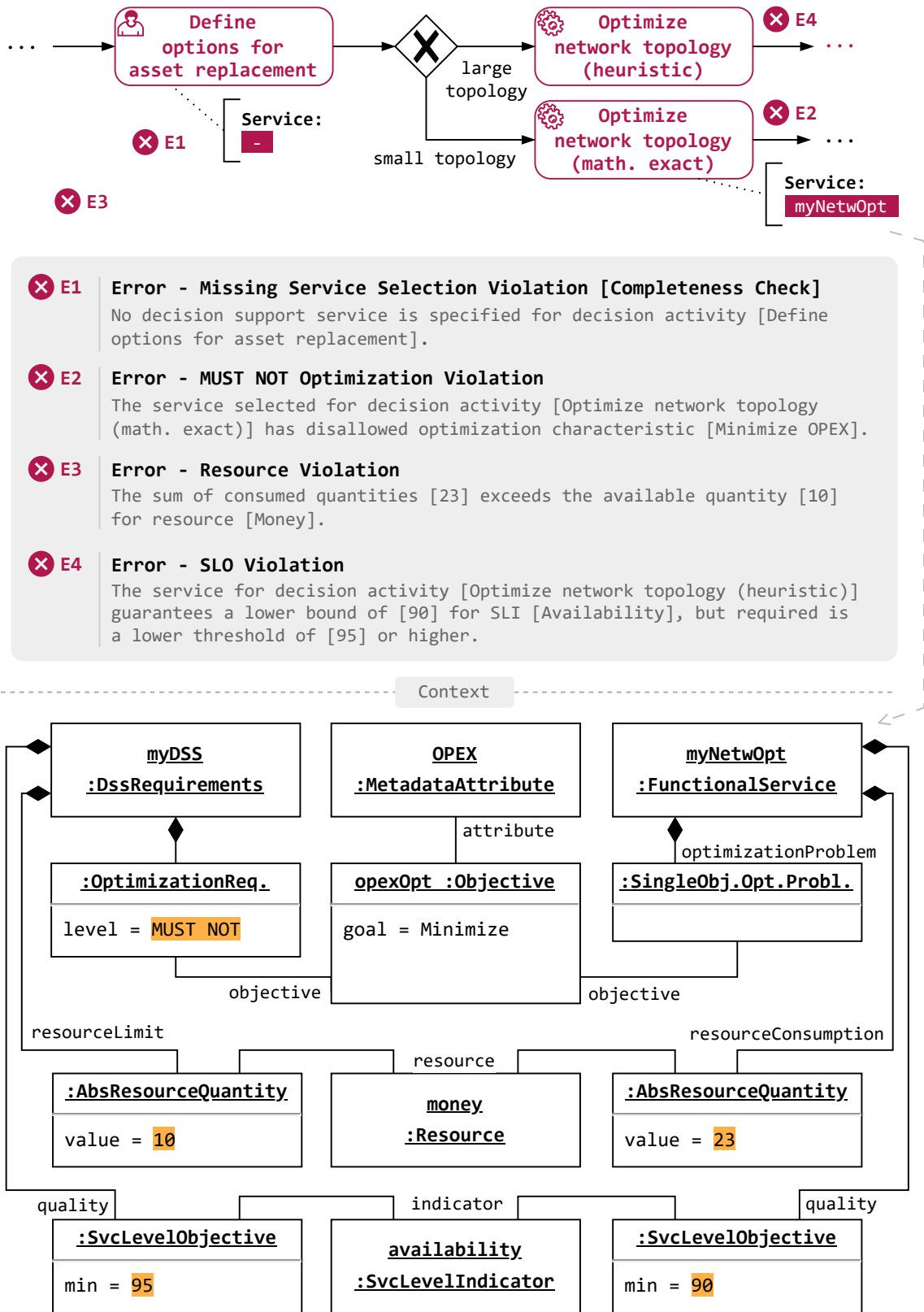


Figure 7.3: Example for selected violations reported by the operational composition assistance

Relation to the Perspective-Specific Requirements

The (partial) case study used for the demonstration includes examples that show the fundamental implementation of requirements for the operational assistance, i.e., *ARO1 – Optimization Characteristics* and *ARO2 – Non-Functional Characteristics*. Nevertheless, the design of the case study revealed some scenarios where assistance is limited.

Regarding *ARO1 – Optimization Characteristics*, a MUST or SHOULD optimization violation is not detected as long as at least one other service selected for a different task addresses the requirement. For example as shown in Fig. 7.3, the process model could contain multiple tasks for the same purpose (e.g., network optimization) that are conditionally executed. Although not explicitly shown in the excerpt of the object diagram due to space constraints, the decision maker intends to minimize investment costs. However, it may be possible that the service selected for one task addresses the requirement (e.g., heuristic optimization minimizes investment costs) while another does not (e.g., mathematical exact optimization minimizes *operational* costs). Without the depicted MUST NOT requirement, this misalignment with requirements would go unnoticed since the condition of the MUST optimization violation only evaluates to true if no service in the service composition exhibits the optimization requirement. In addition to the shown workaround, i.e., explicitly disallowing unwanted optimization characteristics, the shortcoming can be addressed by adapting the description format of Chapter 5 to document all potentially available optimization characteristics for each computation method. It is then possible to adapt the condition of a MUST and SHOULD violation to apply when a selected service does not support the optimization characteristic although it theoretically could support the characteristic based on the associated computation method (e.g., the mathematical exact optimization minimizes operational costs although it could theoretically also minimize investment costs according to the associated “asset optimization” computation method, therefore a violation is present).

Regarding *ARO2 – Non-Functional Characteristics*, the explanation of the resource violation already described how the exceeding of a resource cannot always be detected during the design of a service composition, but may only become obvious when a decision maker selects input data during the instantiation of the underlying process model due to services describing their resource consumption relative to some metadata characteristic of the input data. This may introduce an inefficiency during PD-DSS development if the decision maker must inform the DSS engineer about the limitations of the designed PD-DSS for the selected data, essentially requiring the DSS engineer to partially redesign the service composition. This can be avoided by defining maximum values for affected metadata in the documented requirements for decision support, thus enabling the validation to catch these violations solely during the design of the service composition. This also applies to metadata for data generated

during PD-DSS enactment, e.g., “Define options for asset replacement” in Fig. 7.3. Here, DSS engineers can use their domain expertise to estimate the resources consumed by these tasks.

Also related to *ARO2 – Non-Functional Characteristics*, the specification of service quality requirements based on service level objectives could be extended to also support requirement levels (both in the meta-model and the composition assistance).

Relation to the Cross-Cutting Requirements

Regarding the cross-cutting assistance requirements (Section 7.2), all violations can be detected during process design (*ARC1 – Static Validation*) except for some resource violations that can only be detected during instantiation. However, the identification of resource violations can also be moved solely to the design phase as previously discussed. For violations of optimization characteristics or service level objectives, violation messages specify the activity in the decision process where the violation occurs and information on why a violation occurs, thus addressing *ARC2 – Traceable Feedback* and *ARC3 – Comprehensive Feedback*. Traceability is however limited for resource violations as often no single activity can be identified as a sole reason for exceeding resource constraints. This could potentially be improved by presenting a list of services included in the PD-DSS process model sorted by their resource consumption to support DSS engineers in identifying services whose replacement enables the largest resource savings. Due to the simplicity of violation conditions combined with the potential to use caching, *ARC4 – Continuous Feedback* is addressed since even without caching every task only needs to be evaluated once.

Transition to Recommender System

The operational composition assistance can be transformed into a recommender system that supports DSS engineers in the selection of decision support services for the implementation of an activity in the decision process. For this purpose, the recommender system can simply check for each decision support service listed in the service registry if using the service for the implementation of the task last added to the process model would result in a violation. All services that do not result in a violation can be recommended to the DSS engineer (potentially with some secondary sorting, i.e., frequency of service usage) while all other services causing a violation can be proactively discarded. The feasibility of this approach concerning computation time is however highly dependent on the number of decision support services documented in the service registry. The number of alternatives could be reduced by adding a filter in the form of a preceding selection of the computation method such that only decision support services implementing this computation method are considered.

7.4 Informational Composition Assistance

The informational composition assistance validates the data exchange with and throughout the tailored PD-DSS. This specifically includes a compatibility assessment for data provided by a decision maker before the enactment of the decision process and its subsequent processing by decision support services. This section first describes the potential benefits of the informational composition assistance in supporting a DSS engineer during the design of a PD-DSS process model (Section 7.4.1). Afterwards, requirements for the informational composition assistance are derived (Section 7.4.2) and used to identify shortcomings of existing approaches for the validation of a (PD-DSS) process model's informational perspective (Section 7.4.3). A design for the informational composition assistance is proposed, and its applicability is demonstrated for energy distribution network planning (Section 7.4.4). Insights from a prototypical implementation regarding the technical feasibility of the design are presented (Section 7.4.5) and the design and implementation of the informational composition assistance with respect to the initially defined requirements are discussed (Section 7.4.6).

7.4.1 Motivational Scenario

The benefits of the informational composition assistance are best explained with an illustrative example. Figure 7.4 shows an excerpt from a simulated case study in the domain of energy distribution network planning. The excerpt focuses on the three tasks implemented with automated decision support services, i.e., the simulation of consumer demands across a time interval, which are then used to determine loads of network assets. These loads are subsequently used together with topological information about the network to identify an investment plan that documents cost-minimizing topology investments.

Mistakes in the informational perspective of the decision support service composition can result in ineffectiveness or inefficiency of the derived tailored PD-DSS. An example of an inefficiency is the scenario where a decision support service rejects the provided data during runtime or throws an error due to data incompatibilities. For example, the asset loads computed during “Simulate asset loads” may have a daily resolution, i.e., specify a maximum value for the asset load per day, but the decision support service selected for “Minimize investments” may only support a yearly resolution. If this incompatibility is not caught, the optimization service likely rejects the provided data during runtime and the whole decision process must be restarted after the error is fixed. Even worse, the decision support service may not be able to determine the resolution based on the data format and compute wrong results based on the assumption of a different resolution.

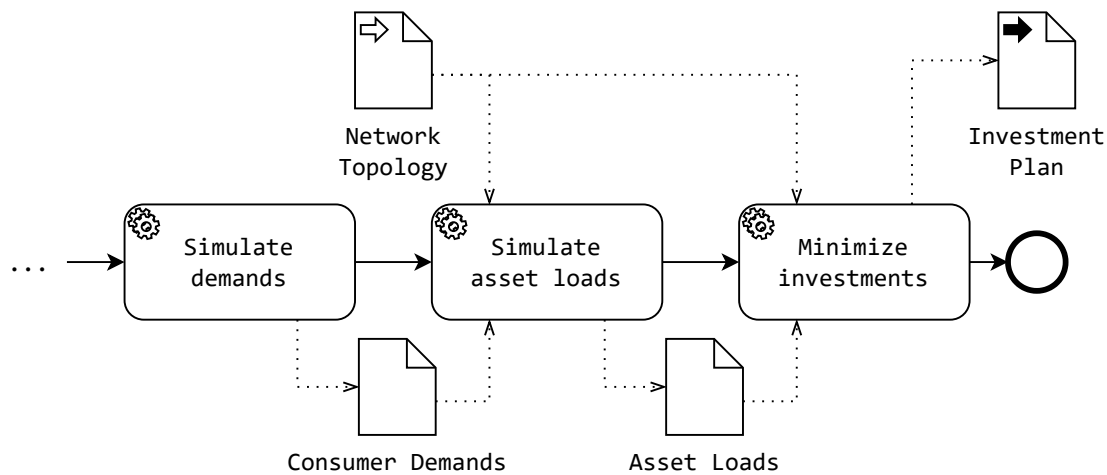


Figure 7.4: Foundation of running example for the informational composition assistance

Since data is not only produced by decision support services but can also be provided by decision makers as input to the decision process, the previously described inefficiency can also occur when a decision maker selects incompatible data during the instantiation of the PD-DSS process model. For example, the network topology selected by the decision maker may be incompatible with the decision support service selected for the “Minimize investments” task if it only supports network topologies up to 1,000 assets.

The informational perspective of a decision support service composition can also provide hints regarding the effectiveness of the resulting PD-DSS. For example, the result of the “Minimize investments” task is a detailed investment plan, but the decision maker may be only interested in some aggregated statistics, e.g., the overall investment costs, which indicates that a task is missing for the computation of these statistics.

7.4.2 Requirements

In addition to the cross-cutting requirements for composition assistance introduced in Section 7.2, this section presents additional requirements *AR_Ix* for the informational composition assistance based on the data requirements of decision support services captured by the service description format presented in Chapter 5.

Assistance Requirement **ARI1** – Process Output

The informational composition assistance should validate that the process model output(s) returned to the decision maker after using the tailored PD-DSS align with the documented requirements for decision support to ensure the effectiveness of the PD-DSS.

Assistance Requirement ARI2 – Input Type

In the DSE context, data is associated with a data type that describes the structure of the data. The informational composition assistance should consequently validate that the data type of the data provided as input to a decision support service matches the data type documented in the description of the decision support service.

Assistance Requirement ARI3 – Input Quantity

An arbitrary minimum and maximum cardinality can be specified for input data of a decision support service. The informational composition assistance should consequently validate that the quantity of the provided data aligns with the quantity required by the decision support service. This includes validation that all required inputs are provided.

Assistance Requirement ARI4 – Input Format

Data of a specific data type can be expressed using different data formats with each decision support service usually only supporting a subset of the available data formats. The informational composition assistance should consequently validate that the data format of the provided data aligns with the data format required by the decision support service.

Assistance Requirement ARI5 – Input Metadata

Decision support services can document constraints on metadata characteristics of input data in the form of assertions, e.g., to describe that the service only supports network topologies with a size below 1000 nodes. The informational composition assistance should consequently validate that the characteristics of the provided input data align with the asserted constraints of the decision support service defined with respect to metadata attributes.

Assistance Requirement ARI6 – Selected Data

The informational composition assistance should apply the previously described input data validations (*ARI2 – ARI5*) to all data that is obtained from a data service. A data service may either be statically selected during the design of the overall service composition, or more frequently, is selected by the decision maker during the instantiation of the process model – either explicitly by selecting a data service from the service registry, or implicitly by making their own data available to the DSS, e.g., in the form of an upload.

Assistance Requirement ARI7 – Processed Data

The informational composition assistance should apply the previously described input data validations (*ARI2 – ARI5*) to the data that is obtained from a previously executed functional service. This ensures that the data generation or processing of a previous service does not produce data that is incompatible with a subsequent decision support service.

7.4.3 Background and Related Work

Since a composition of decision support services representing a tailored PD-DSS is described as a process model (cf. Chapter 6), the validation of a service composition with respect to its informational perspective (also referred to as dataflow validation) could potentially reuse existing approaches that validate the informational correctness of a process model. This subsection discusses such existing approaches with respect to the requirements previously established for the informational composition assistance.

Fundamental Dataflow Errors

Both Sadiq et al. [Sad+04] and Trčka, van der Aalst, and Sidorova [TvS09] define multiple fundamental dataflow errors for process models that can be summarized as follows:

Redundant Data Data is redundant if it is produced (i.e., provided as a process input or generated by an activity) but not consumed. This error is detected as part of the completeness checks described in Appendix A.1.2.

Lost Data Data is lost if it is overwritten before it is consumed, e.g., if two activities are executed in parallel and write to the same data object. The completeness checks described in Appendix A.1.2 issue a warning if data is potentially lost due to writes from multiple activities.

Missing Data Data is missing if it is required for the execution of an activity but not provided. This error is detected as part of *ARI3 – Input Quantity*.

Inconsistent Data Data is inconsistent if it is modified externally after it was produced and before it is consumed. This error is not relevant in the context of a PD-DSS since data is immutable, i.e., services only create new data and do not update existing data (cf. Chapter 6).

In addition to these four errors which are shared between the two publications, Sadiq et al. [Sad+04] furthermore define the following dataflow errors:

Mismatched Data Data provided to an activity has a different data structure than required for the execution of the activity. If both data structures can store the same information, this error is detected as part of *ARI4 – Input Format*, otherwise as part of *ARI2 – Input Type*.

Misdirected Data Data is misdirected if the data-consuming activity is executed before the data-producing activity. This is not covered by a requirement since it also requires consideration of the behavioral perspective.

Insufficient Data Data is insufficient if it does not contain enough information to execute the activity. This error is detected as part of *ARI5 – Input Metadata*.

Trčka, van der Aalst, and Sidorova [TvS09] furthermore define dataflow errors that consider the deletion of data throughout the enactment of a process model. However, from a modeling perspective, data deletion is an engine-specific characteristic of certain BPMN engines (cf. [Sch+21b]) and therefore not further considered throughout this section. Furthermore, intermediate data produced during PD-DSS usage that is not returned to the decision maker is deleted after the enactment of the decision process has completed.

Although the presented dataflow errors largely align with the requirements for the informational composition assistance, there is still a need for an algorithmic approach to detecting the errors in a given process model. Unfortunately, Sadiq et al. [Sad+04] do not provide an implementation for the detection of their described dataflow errors. Trčka, van der Aalst, and Sidorova [TvS09] provide an implementation based on temporal logic. Their implementation is not coupled to BPMN, but an application to BPMN was later contributed by von Stackelberg et al. [vSta+14]. However, as evident from the above enumeration of dataflow errors, the list by Trčka, van der Aalst, and Sidorova [TvS09] is missing errors that are relevant to address all requirements of the informational composition assistance. Similarly, Rachdi, En-Nouaary, and Dahchour [RED17] describe an approach that can only detect a subset of the defined dataflow errors (i.e., *Missing Data*, *Inconsistent Data*, *Redundant Data* and *Lost Data*) in BPMN process models. Thus, the described approaches are not immediately applicable to address all requirements for the informational composition assistance.

Dataflow Validation for Process-Driven Applications

Another attempt to simplify the implementation of the informational composition assistance is to reuse existing approaches for validating the dataflow of process-driven applications (PDAs), i.e., the concept that motivated the definition of the PD-DSS concept in Chapter 6.

Schneid et al. [Sch+19; Sch+21b] describe an approach for detecting data anomalies in the BPMN process model of a PDA. The authors describe three types of anomalies: *undefined read* (data is read without previously being written), *never used* (data is written but never read before it is overwritten or deleted), and *invalid deletion* (data is deleted, although it was never written). In addition to only covering a small subset of the previously discussed fundamental dataflow errors, the approach also relies on an analysis of the source code that is used for the implementation of service tasks. This is not feasible in the DSE context due to the black box characteristic of decision support services (cf. Chapter 4).

Schneid et al. [Sch+21c] describe an approach for process analysts without programming skills to define regression tests for PDAs. Since this approach can fundamentally validate all perspectives of a PDA, it was already discussed in Section 7.3.3. The limitation concerning *ARC4 – Continuous Feedback* also applies to the informational perspective.

Schiffner, Rothschädl, and Meyer [SRM14] describe the necessity to automatically validate the message exchange between services that are utilized for the implementation of a business process. However, they do not provide details on how this validation should work. In addition, they use *Subject-Oriented Business Process Management*, which does not utilize BPMN.

Dataflow Validation for Process Models

Due to the limitations of existing approaches for PDA dataflow validation, the discussion is broadened to dataflow validation of BPMN process models without the generation of an executable application. The subsequently discussed approaches specifically focus on the documentation of data constraints for process tasks similar to *ARI5 – Input Metadata*.

Awad, Decker, and Lohmann [ADL10] and Weber, Hoffmann, and Mendling [WHM10] describe how the state that can be associated with a BPMN data object can be used to define pre- and postconditions for activities. A precondition ensures that input data has a specific state, and a postcondition documents that output data will be in a specific state. However, since the state corresponds to a label, i.e., a single value from an enumeration, it is not suited to specify more complex assertions on metadata as required by *ARI5 – Input Metadata*.

Borrego et al. [Bor+13] describe a similar approach that also utilizes pre- and postconditions to validate the compatibility of data exchanged between activities. However, conditions are not defined with respect to a single label/state, but with respect to quantitative data objects. A condition is defined as a triple of a data object identifier, comparison operator, and value. These condition triples can furthermore be combined using logical AND and OR operators. While this approach does partially address *ARI5 – Input Metadata*, it lacks support for qualitative metadata and data types (*ARI2 – Input Type*). It also does not account for many of the fundamental dataflow errors defined at the beginning of this subsection.

Summary of Related Work

The review of existing approaches for validating the dataflow (i.e., informational perspective) of BPMN process models shows that all approaches only address a subset of the requirements for the informational composition assistance presented in Section 7.4.2. In addition to the previously presented limitations that are specific to each approach, all discussed approaches furthermore focus only on dataflow validation throughout the design of the process model and the data exchange between activities (cf. *ARI7 – Processed Data*), but no approach explicitly considers validation during data selection when the process model is instantiated (cf. *ARI6 – Selected Data*). Consequently, there is still a need for an approach to implement the introduced requirements for the informational composition assistance.

7.4.4 Design and Demonstration

This subsection presents the design of the informational composition assistance to validate the effectiveness and efficiency of the data exchange of and with a tailored PD-DSS before it is used during the enactment of a decision process. Described violations are demonstrated using the example from energy distribution network planning presented in Section 7.4.1.

Validation of Output Data

Validation of output data ensures that decision makers are provided with their desired decision recommendations. In particular, each decision process – and consequently each composition of decision support services – should always return some output data since decision support services are free of side effects and do not modify existing data. Therefore, a *Redundant Process Violation* is reported if no output is returned to the decision maker.

Redundant Process Violation		Severity: Error
Element:	Process: Set of all Data Outputs	Phase: Design
Condition:	The set of all process data outputs is empty.	
Message:	“The process does not return any output data to the decision maker.”	

If output data is returned to the decision maker, it must align with the documented requirements for decision support. As described in Section 5.4.6, the decision maker can specify requirements for output data with respect to the output of a computation method and the associated data format. This implies a *Missing Output Violation* and an *Output Format Violation*. Support for different requirement levels works analogous to the presentation of the operational composition assistance and is omitted here to avoid redundancy.

Additionally, a warning in the form of a *Redundant Output Violation* is issued in case an output is returned that is not documented in the requirements for decision support, as this may be an indicator of a redundant activity in the decision process.

Demonstration In the example shown in Fig. 7.5, a *Missing Output Violation* (E1) is reported because the decision support requirements expect an investment summary to be returned. Since this output is not provided by the “Minimize investments” activity, an additional activity is required. This is an indicator of how the informational validation can also uncover deficits in the functional perspective of the decision support service composition as a side effect. An *Output Format Violation* (E2) is reported since the investment plan computed throughout the “Minimize investments” task is described in JSON format, but a PDF format should be

Missing Output Violation		Severity: Error
Element:	Process: Set of all Data Outputs	Phase: Design
Condition:	Every process output is assigned data from an output data slot that is not associated with the method data specified in the decision support requirements.	
Message:	“No process output returns required service output [...].”	
Output Format Violation		Severity: Error
Element:	(Process) Data Output	Phase: Design
Message:	“Process output [...] provided by activity [...] has data format [...], but required data format is [...].”	
Redundant Output Violation		Severity: Warning
Element:	(Process) Data Output	Phase: Design
Condition:	The process output is assigned data from an output data slot that is not documented in the decision support requirements.	
Message:	“The process output [...] returns data that is not documented in the requirements for decision support.”	

used according to the requirements for decision support of the decision maker. A *Redundant Process Violation* would be reported if the “Investment Plan” process data output were missing in the documented requirements for decision support.

Validation of Input Data

Validation of input data ensures that the data provided as input to a decision support service meets the data requirements of the service for the targeted data slot. The requirements for the informational composition assistance in Section 7.4.2 already hint at four properties that characterize an input data slot of a decision support service: data type and data quantity are defined for the associated computation method, and data format and metadata are specifically defined for the data slot. The violations that can be defined for these data characteristics depend on whether the input data is obtained from a data service selected by a decision maker or a previously executed functional decision support service.

Input Data from a Data Service In the simplest case, the previously listed input data characteristics can be directly obtained from the description of a data service that was selected by the decision maker for a process data input during the instantiation of the PD-DSS process

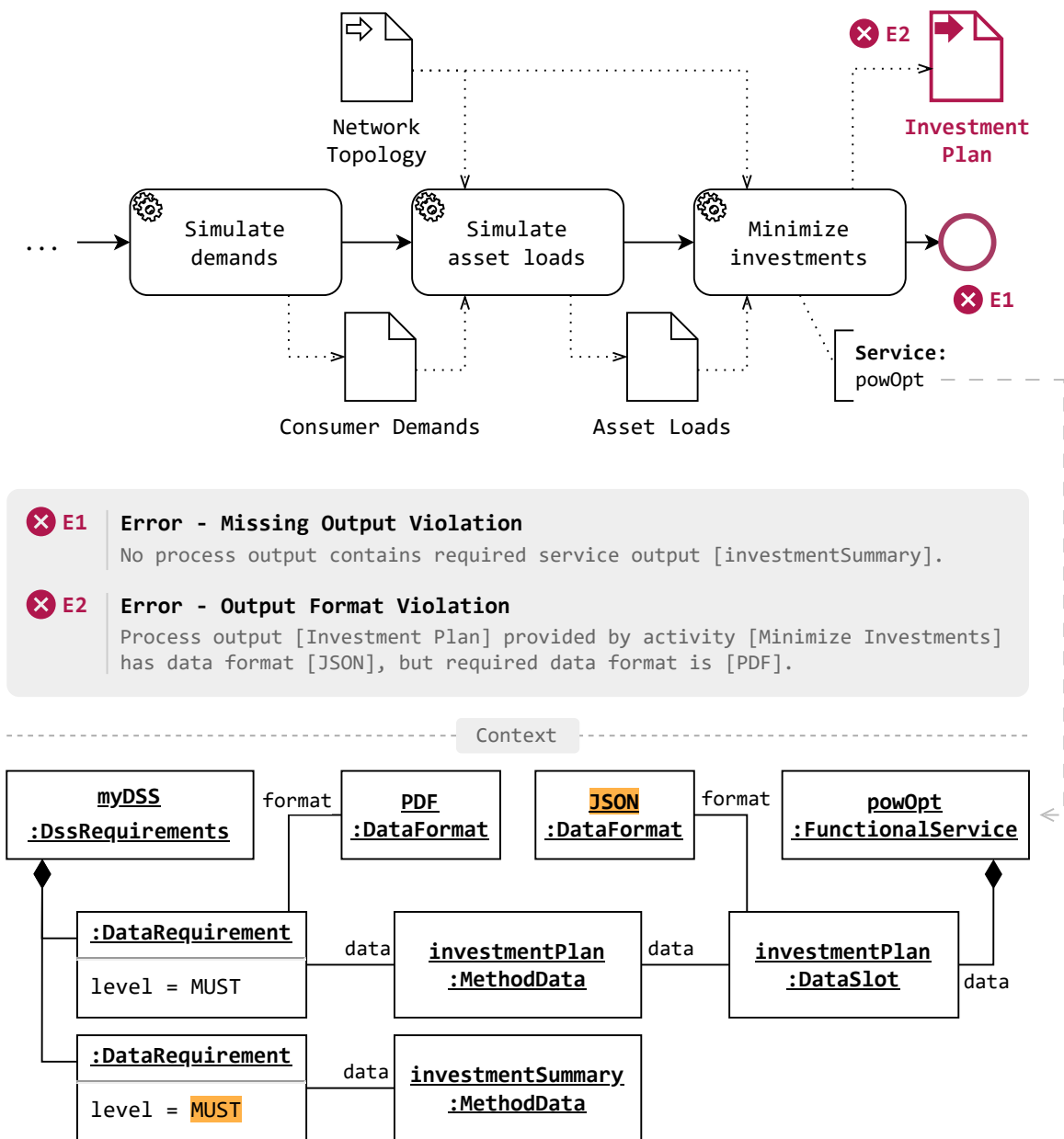


Figure 7.5: Demonstration of informational violations with respect to output data

model. Then, the data type, format and quantity of the data service can be compared to the required data characteristics of the input data slot where the data is consumed, and violations can be reported in case of differences. This is captured with the *Data Format Violation*, *Data Type Violation*, and *Data Quantity Violation*. The *Data Quantity Violation* can also identify missing data since a cardinality of 0 would be below the minimum cardinality of 1 or greater.

The identification of metadata violations exceeds a simple equality check, since the metadata values documented in the data service description must be checked against the

Data Format Violation		Severity: Error
Element:	(Task) Data Input	Phase: Design, Instantiation
Condition:	The input data slot represented by the data input of a task has a different data format than the data assigned by a data association.	
Message:	<i>“Data [...] provided to activity [...] for input [...] has data format [...], but the required data format is [...].”</i>	
Data Type Violation		Severity: Error
Similar to:	Data Format Violation	
Message:	<i>“Data [...] provided to activity [...] for input [...] has data type [...], but required is data type [...].”</i>	
Data Quantity Violation		Severity: Error
Similar to:	Data Format Violation	
Message:	<i>“Data [...] provided to activity [...] for input [...] has a cardinality of [...], which is below the minimum cardinality of [...].”</i> – analogous: <i>“above the maximum cardinality”</i>	

metadata constraints of the data slot. For example in Fig. 7.4, if the service selected for “Minimize investments” requires `topology.size < 1000`, it must be checked if this constraint is fulfilled by the data service selected for the “Network Topology” input that is assigned to the `topology` data slot. Despite this more complicated check, the corresponding *Metadata Violation* can be defined similarly to the previously described violations.

Metadata Violation (Instantiation)		Severity: Error
Element:	Task: Data Input	Phase: Instantiation
Condition:	The input data slot represented by the data input defines assertions on the metadata that are not fulfilled by the associated data.	
Message:	<i>“Data [...] provided to activity [...] for input [...] does not pass constraint [...] because metadata attribute [...] has value [...].”</i>	

Input Data from a Functional Service During the design of a PD-DSS process model by a DSS engineer, no data services have yet been selected by a decision maker from which the input data characteristics can be obtained. In this case, the data characteristics of input

data objects must be derived from the associated output data slots of producing decision support services, and the data characteristics of process data inputs must be derived from the associated input data slots. For the example shown in Fig. 7.4, the data format of the “Network Topology” input can be derived from the data format required by data slots of the decision support services implementing the “Simulate asset loads” and “Minimize investments” task, or the data type of the “Consumer Demands” output is derived from the “Simulate demands” task. Based on these derived data characteristics, the previously defined *Data Format Violation*, *Data Type Violation*, and *Data Quantity Violation* can be identified analogously during design.

Metadata violations again must be handled differently, as only metadata assertions can be derived from an associated service, but no concrete metadata values. Consequently, it is necessary to check if all data that would pass the metadata assertions of the producing decision support service would also pass the assertions of the consuming data service. For example, all network topologies that pass a `size < 500` assertion also pass a `size < 1000` assertion. This results in the *Metadata Violation (Design)* with an updated message to reflect the comparison of metadata assertions.

Metadata Violation (Design)		Severity: Error
Element:	Task: Data Input	Phase: Design
Condition:	The input data slot represented by the data input defines metadata assertions that are incompatible with assertions derived for the associated data.	
Message:	<i>“Metadata incompatibility inferred for data [...]: Output [...] of [...] asserts [...], but input [...] of [...] requires [...].”</i>	

In the context of these metadata violations, it is important to consider that services may define metadata assertions in relation to the metadata of an input data slot. For example in Fig. 7.4, the service implementing “Simulate asset loads” could assert that the resolution of the “Asset Loads” forecast is equal to the resolution of the provided “Consumer Demands” forecast. Thus, whether the resolution of “Asset Loads” meets the resolution requirement of the service implementing the “Minimize investments” task depends on the asserted resolution of the service selected for the “Simulate demands” task.

Lastly, process data inputs can be reused for multiple data slots as indicated in Fig. 7.4 with the “Network Topology” input. Data characteristics of reused input data cannot be uniquely inferred if the associated tasks specify conflicting data characteristics, e.g., if the simulation supports a different data format for the network topology than the optimization. Although this would technically be reported by the previously presented violations, this error is explicitly

checked for to improve traceability. This results in four additional violations in the form of the *Multiple Data Formats Violation*, *Multiple Data Types Violation*, *Multiple Data Quantities Violation*, and *Multiple Metadata Violation*. Only the definition for the *Multiple Data Formats Violation* is provided to avoid redundancy as the other violations can be defined analogously.

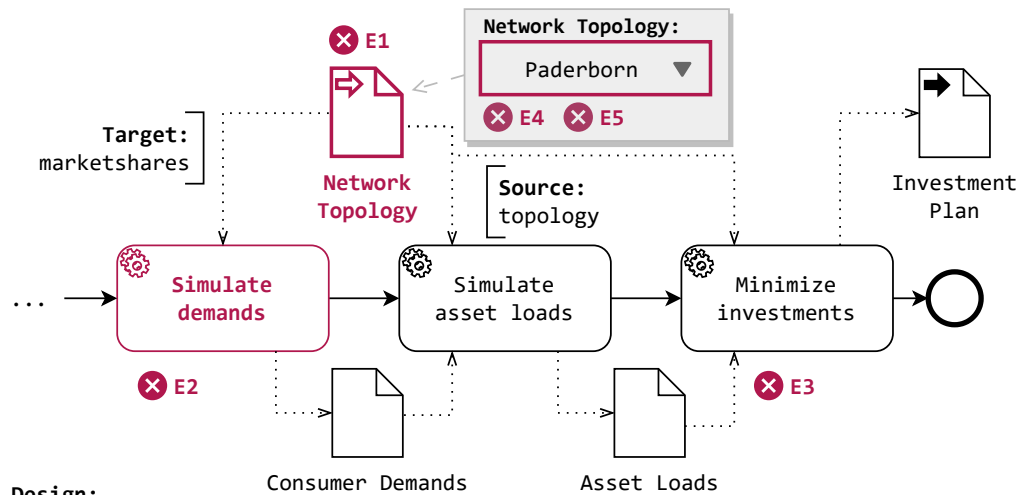
Multiple Data Formats Violation		Severity: Error
Element:	(Process) Data Input, Data Object	Phase: Design
Condition:	A process data input is associated with multiple input data slots or a data object is associated with multiple output data slots, but not all data slots have the same data format.	
Message:	“Data [...] cannot have multiple data formats ([input/output] [...] of activity [...] has data format [...], ... < repeats for all affected data slots >).”	

Demonstration Examples for discussed violations are given in Fig. 7.6. Due to space constraints, the relevant excerpts of the associated service descriptions are shown as tables instead of object diagrams, and task labels are used instead of service IDs for easier lookup. The “Network Topology” process data input is supposed to provide both data for the market shares that are required for simulating consumer demands and the network topology that is needed for the simulation of asset loads and network optimization. Since data cannot simultaneously describe market shares and a network topology, conflicting data types are reported in a *Multiple Data Types Violation* (E1). The service for “Simulate demands” requires the historical consumer demands for the computation of a forecast, but no such data is provided, resulting in a *Data Quantity Violation* (E2). Since the service selected for the “Simulate asset loads” always has a daily resolution but the “Minimize investments” requires a yearly resolution, a *Metadata Violation* can be reported during design time (E3).

Two other errors are only detected during PD-DSS instantiation when a decision maker selects the data service with the “Paderborn” network topology: First, the topology is provided in a .csv format, which is incompatible with the .pnet format required by the service and results in a *Data Format Violation* (E4). Second, the topology has a size of 2345 which exceeds the allowed size of 1000 defined by the decision support service selected for the “Minimize investments” task, which results in another *Metadata Violation* (E5).

7.4.5 Prototypical Implementation

Many of the previously defined violations considering the informational perspective of a PD-DSS process model can be detected with a simple equality check. However, the detection



Design:

- ✘ E1 Error - Multiple Data Types Violation**
Data [Network Topology] cannot have multiple data types (input [marketShares] of [Simulate Demands] has data type [MarketShares], input [topology] of activity [Simulate Asset Loads] has data type [NetworkTopology], ...)
- ✘ E2 Error - Data Quantity Violation**
Data [-] provided to activity [Simulate Asset Loads] for input [historical] has a cardinality of [0] which is below the minimum cardinality of [1].
- ✘ E3 Error - Metadata Violation (Design)**
Metadata incompatibility inferred for data [Asset Loads]: Output [loads] of [Simulate Asset Loads] asserts [resolution = DAILY], but input [loads] of [Minimize Investments] requires [resolution = YEARLY].

Instantiation:

- ✘ E4 Error - Data Format Violation**
Data [Paderborn] provided for input [topology] of [Simulate Asset Loads] has data format [.csv], but the required data format is [.pnet].
- ✘ E5 Error - Metadata Violation (Instantiation)**
Data [Paderborn] provided to activity [Simulate Asset Loads] for input [topology] does not pass constraint [size < 1000] because metadata attribute [size] has value [1234].

Context

Service	Data (In)	Quantity	Type	Format
Simulate Demands	marketshares	1	MarketShares	.csv
	historical	1	ConsumerDemands	.dmnd
Simulate Asset Loads	topology	1	NetworkTopology	.pnet
Paderborn (Data)	-	1	NetworkTopology	.csv
Service	Data (In/Out)	Metadata: Values / Assertions		
Simulate Demands	demands	[V] resolution = "DAILY"		
Simulate Asset Loads	loads	[A] resolution = demands.resolution		
Minimize Investments	topology	[A] size < 1000		
	loads	[A] resolution = "YEARLY"		
Paderborn (Data)	-	[V] size = 2345		

Figure 7.6: Demonstration of informational violations for input data

of metadata violations is more challenging when the compatibility of assertions on metadata attributes must be validated. This challenge is further increased if assertions on output data are defined relative to input data, thereby forming a chain of interdependent assertions across multiple tasks. This section therefore describes a prototypical implementation of the informational composition assistance with a focus on the detection of metadata violations due to the summarized complexity potentially presenting a technical challenge for the realization of the composition assistance. The remainder of this subsection first provides background on *JSON Schema*, a data validation standard used by the prototypical implementation. Afterwards, the insights gathered from the prototypical implementation are summarized.

Background: JSON Schema

The *JSON Schema* specification [IET22] defines an approach to describe the structure of JSON data [Ecm17]. A JSON Schema document is itself a JSON document. An example of a JSON document describing the metadata of a network topology and the associated JSON schema of a consuming service are shown in Listings 7.1 and 7.2 respectively. The schema requires each compatible topology to have a name of type string and size below 1000. The instance represents a topology with the name “Paderborn” and a size of 2345, which exceeds the upper bound of 1000 and is therefore not compatible with the schema/service.

Listing 7.1: Topology JSON

```
{
  // "$schema": ->
  "name": "Paderborn",
  "size": 2345 // ↯
}
```

Listing 7.2: Topology JSON Schema for Service

```
{
  "type": "object", "properties": {
    "name": { "type": "string" },
    "size": { "type": "int", "maximum": 1000 }
  }
}
```

As evident from the previous example, the key-value structure of JSON naturally aligns with the concept of metadata in the DSE context, and JSON Schema aligns with the assertions on metadata attributes specified by decision support services. Thus, it is easy to determine whether a data instance with concrete metadata passes the metadata assertions of the decision support service where it should be processed. In addition to this conceptual alignment, the decision to use JSON Schema for the prototypical implementation is largely motivated by its vast ecosystem and tool support. In particular, there are validators such as *AJV*¹ to validate a JSON document against a JSON Schema, thereby supporting primarily decision makers

¹ Validator website: <https://ajv.js.org/>

in selecting appropriate data services during the instantiation of a decision support service composition. Furthermore, the *is-json-schema-subset* library² can determine whether one JSON Schema is a subset of another JSON Schema, i.e., if documents validating against one schema would also validate against the other. This analysis can be used to determine whether the output of one service is compatible to the input of another service by checking if the output schema of the producing service is a subset of the input schema of the consuming service.

Insights from the Prototypical Implementation

The prototypical implementation demonstrates the validity of the previously described preliminary considerations. In particular, the concepts of metadata and assertions included in the description of decision support services (cf. Chapter 5) can be mapped to JSON Schema and JSON documents. The technical details of this mapping are documented in the paper “Detecting Data Incompatibilities in Process-Driven Decision Support Systems” [KGE22] and Appendix B. The prototypical implementation furthermore shows that existing tooling can in fact be reused for checking the compatibility between data services and functional decision support services. The tooling always completed validation in 100ms on commodity hardware with comprehensive messages stating the source of the validation error.

Nevertheless, the use of JSON Schema for the prototypical implementation exhibits one shortcoming: While it is possible to reference metadata assertions of an input schema from an output schema, the definition of assertions of an input with respect to another input is not supported, e.g., to express that two provided forecasts must have the same resolution. This suggests an approach that directly supports service descriptions without an intermediate translation to JSON (Schema) for a productive DSE implementation.

7.4.6 Discussion

The demonstrations presented throughout the explanations of the informational composition assistance’s solution design (Section 7.4.4) already show the capabilities of the assistance to detect fundamental errors in the informational perspective of a decision support service composition. The prototypical implementation furthermore shows that the most complex validation performed by the informational composition assistance, i.e., the detection of metadata violations throughout the design and instantiation of a decision support service composition, is technically feasible. This section briefly discusses the presented insights with respect to the requirements initially defined for the informational composition assistance

² Library website: <https://www.npmjs.com/package/is-json-schema-subset>

(Section 7.4.2) with a focus on the cross-cutting assistance requirements (Section 7.2) to ensure that all requirements are sufficiently addressed and to potentially identify future extensions.

Requirements Specific to the Informational Composition Assistance

The requirements defined for the informational composition assistance map to corresponding subsections in Section 7.4.4. In particular, *AR11 – Process Output* is addressed in the first subsection, and *AR12 – Input Type* to *AR17 – Processed Data* are addressed in the subsequent subsection. Details of the design are not repeated here to avoid redundancy.

The discussion of related work includes definitions for fundamental dataflow errors in process models. These errors mostly represent a subset of the requirements for the informational composition assistance defined in Section 7.4.2. This suggests two conclusions, namely that the requirements for the informational composition assistance are largely complete, and that all fundamental dataflow errors can be detected by the informational composition assistance since it addresses all requirements. The only dataflow error that cannot be mapped to a requirement is misdirected data, i.e., when the data-consuming activity is executed before the data-producing activity. Detection of this error requires an integrated view of the PD-DSS process model including the behavioral perspective. Detection of this dataflow error could be supported by computing all possible process paths and checking if a path exists where a decision support service depending on data of another decision support service is executed first. However, the absence of this automated detection may not be significant, as misdirected data can be spotted in the process model. This differs from the previously described violations, which are dependent on information that is stored in the associated service descriptions and not depicted in the process model.

Cross-Cutting Requirements for Composition Assistance

As obvious from the “Phase” property in the violation definitions, all violations can be detected during the design or instantiation of the decision support service composition (*ARC1 – Static Validation*). However, meaningful validation requires the descriptions of decision support services to include as many metadata assertions as possible. This is especially important for interactive decision support services where the metadata of the produced data may depend on the decision maker’s interactions with the service when using the tailored DSS. In some cases, these assertions may be dependent on the use case of the tailored DSS. Since specific use cases cannot be foreseen by the service provider who creates the description of the decision support services, the approach for the informational composition assistance might need to be extended such that DSS engineers can add additional metadata assertions based on the usage of the

service within a service composition. Additionally, the requirements for decision support specified by a decision maker could include representative data services. This provides a DSS engineer with additional information for the design of the service composition and could move the detection of some errors from instantiation to design.

The messages in violation templates address requirements *ARC2 – Traceable Feedback* and *ARC3 – Comprehensive Feedback*. Since most violations correspond to simple equality checks that can be done in constant time, the informational composition assistance is suitable for *ARC4 – Continuous Feedback*. Even the detection of metadata violations, which are significantly more complex than the other defined violations, can be detected within milliseconds as demonstrated by the (unoptimized) prototypical implementation. Based on the quantity of data services, it is also possible to precompute the compatibility of data services and functional decision support services to reduce runtime during instantiation.

Threats to Validity

In addition to the global threats to validity discussed in Section 8.6, two additional threats specifically apply to the informational composition assistance. First, the partial case study selected for the demonstration of the solution design in Section 7.3.4 only includes BPMN tasks for activities, but not repeatedly executed subprocesses. Since the quantity of the data provided to the subprocess is reduced to a single data instance within the subprocess, but all other data characteristics are retained, the previously defined violations should nevertheless be immediately applicable to the dataflow to/within/from a subprocess. Second, the described design and selected case study do not consider conditionality. However, the metadata assertions implied by conditions associated with sequence flows from gateways must be considered when checking the compatibility of provided data. For example, assuming a gateway exists that only invokes a mathematical exact optimization service if the size of the network topology is below 1,000 nodes and a heuristic optimization service otherwise, then the assertion of the size implied by the gateway condition must be considered when determining its compatibility with the mathematical exact optimization service. This can be implemented by adding the gateway conditions as assertions to the data objects/inputs.

Transition to Recommender System

Similar to the operational composition assistance, the informational composition assistance can be used for the implementation of a recommender system to suggest suitable decision support services to a DSS engineer. In particular, the recommender system can suggest functional decision support services that are compatible with data objects or process data

inputs that currently have no outgoing data association, since this data would otherwise be reported as redundant. Furthermore, the validation can be used to proactively restrict the list of available data services presented to decision makers when selecting process data inputs during PD-DSS instantiation, thus ensuring that only compatible data services can be selected.

7.5 Functional and Behavioral Composition Assistance

This section presents an approach for a composition assistance that validates the functional and behavioral perspective of a PD-DSS process model. The section first describes the benefits of such composition assistance and introduces a running example (Section 7.5.1). The cross-cutting assistance requirements are extended with requirements specific to the functional-behavioral composition assistance (Section 7.5.2) and subsequently used to evaluate the reusability of existing validation approaches (Section 7.5.3). Afterwards, the design of the composition assistance for the functional and behavioral perspective is presented and demonstrated for energy distribution network planning (Section 7.5.4). Insights from a prototypical implementation are presented (Section 7.5.5), and lastly, the insights are discussed with respect to the assistance requirements (Section 7.5.6).

7.5.1 Motivational Scenario

The goal of the functional and behavioral composition assistance is to ensure that a tailored PD-DSS generated from a composition of decision support services provides the decision support functionality required by a decision maker. The validation of the functional and behavioral perspective is aggregated into a single functional-behavioral composition assistance since DSS functionality is not only characterized by what is done (= functional perspective), but also under what conditions it is done (= behavioral perspective). The benefits of such a functional-behavioral composition assistance are best explained using an example.

Running Example

The running example used to demonstrate the proposed design of the functional-behavioral composition assistance is visualized in Fig. 7.7. The (partial) example assumes that a decision maker wants to minimize network investments using a mathematical exact optimization approach. A network reduction is performed before the optimization if the size of the provided network topology exceeds 1000 assets to ensure that the optimization completes in a reasonable amount of time. When the network optimization encounters an error in the network topology, the topology must be fixed manually by the decision maker. If a network reduction was

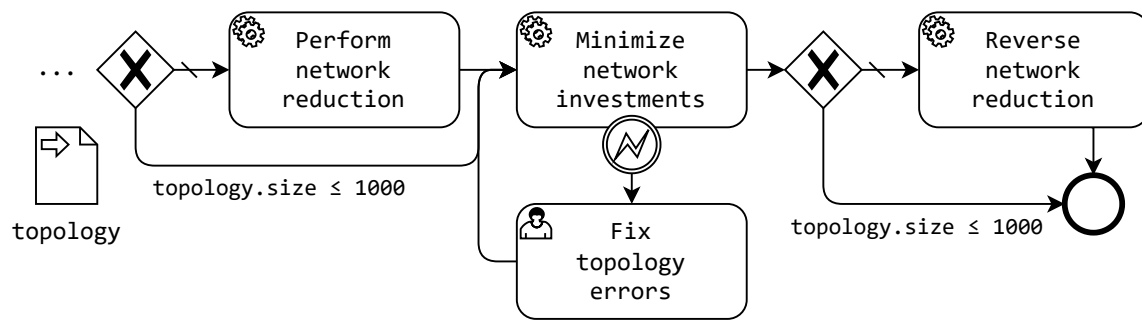


Figure 7.7: Foundation for the running example

performed due to the size of the provided input topology, the reduction is reversed after the optimization such that the investment plan computed by the network optimization is actually applicable to the original network.

Benefits of a Functional-Behavioral Composition Assistance

The functional-behavioral composition assistance ensures the *effectiveness* of a PD-DSS process model with respect to the requirements for decision support documented by a decision maker by validating that all documented decision activities are supported by the PD-DSS service composition. For example in the context of Fig. 7.7, the decision maker could have specified the desire to obtain a graphical visualization of the computed investment plan after the optimization, which is missing in the depicted process model.

The functional-behavioral composition assistance ensures the *efficiency* of a PD-DSS process model by validating that no unnecessary functionality is included in the service composition and that all potential runtime errors are caught and properly handled, to avoid interruptions throughout the enactment of the decision process. For example, if the correction of topology errors would not be included in the example shown in Fig. 7.7, the enactment of the decision process would need to be restarted with the fixed topology, thus invalidating any results computed so far. The efficient enactment of a PD-DSS is furthermore ensured by validating the completeness of the PD-DSS process model with respect to the modeling conventions of Chapter 6 as described in Appendix A.1.3.

Similar to the previously discussed operational and informational composition assistance, many defects can be detected based on the existing structured documentation of the application domain, decision support services, and requirements for decision support (cf. Chapter 5). However, the detection of some defects may require additional, domain-specific (composition) knowledge. For example, the DSS engineer may forget to reverse the previously performed network reduction, thereby rendering the computed investment plan essentially useless since

the contained recommendations are not immediately applicable to the initially provided network topology. This domain-specific defect is also an example of a defect that can only be detected when reviewing the process path from the initial application of the network reduction to the end of the decision process, thereby demonstrating the usefulness of considering the functional and behavioral assistance simultaneously.

7.5.2 Requirements

This section transforms the previous considerations into explicit requirements ARF_x for the design of the functional-behavioral composition assistance. The requirements extend the cross-cutting requirements for composition assistance introduced in Section 7.2. and can be grouped into two categories:

Domain-Parametrized Assistance Functionality

This category aggregates requirements for assistance functionality that can be parametrized based on the domain documentation and service registry described in Chapter 5.

Assistance Requirement ARF1 – Completeness

The functional-behavioral composition assistance should validate that a decision support service composition includes the functionality documented in a decision maker's requirements for decision support to ensure the effectiveness of the resulting tailored PD-DSS.

Assistance Requirement ARF2 – Robustness

The functional-behavioral composition assistance should validate that potential errors documented in the descriptions of decision support services are handled throughout the service composition to ensure efficiency during usage of the resulting tailored PD-DSS.

Domain-Specific Assistance Functionality

This category aggregates requirements for assistance functionality that detects functional and behavioral flaws defined by composition experts specifically for a concrete application domain.

Assistance Requirement ARF3 – Expert Definition

The approach must consider that flaws specific to an application domain are defined by composition experts who are familiar with the characteristics of the application domain but have no programming skills. Furthermore, the definition of benefits should be possible without extensive upfront training in a specific documentation format to encourage the participation of composition experts, thereby increasing the extent of the assistance.

Assistance Requirement ARF4 – Missing Functionality

The functional-behavioral composition assistance should enable composition experts to define flaws based on missing functionality (given the other elements of the decision support service composition). For example, in the context of the motivational scenario of Section 7.5.1, an example of missing functionality would be a missing reversal of network reduction.

Assistance Requirement ARF5 – Unwanted Functionality

The functional-behavioral composition assistance should enable composition experts to define flaws based on unwanted functionality that is either redundant or unwarranted. An example for unwanted functionality in the context of the motivational scenario of Section 7.5.1 would be a second, consecutive execution of the network optimization although subsequent optimization runs cannot improve the results of a previously executed optimization.

Assistance Requirement ARF6 – Missing Branching

The functional-behavioral composition assistance should enable composition experts to define flaws based on a lack of control flow branching introduced by conditionality or parallelism. For the motivational scenario of Section 7.5.1, an example of missing conditionality would be a missing check whether the provided network topology must be handled differently due to its size, or a lack of parallelism for the generation of visual reports for the decision maker.

Assistance Requirement ARF7 – Unwanted Branching

The functional-behavioral composition assistance should enable composition experts to define flaws based on unwanted control flow branching that is either redundant or unwarranted. For example, the parallel execution of a network reduction and network optimization suggests that the optimization will run without the results of the network reduction.

Assistance Requirement ARF8 – Generalization

The functional-behavioral composition assistance should support the generalization of defined flaws such that they can be identified in multiple similar but slightly deviating scenarios. For example, the previously described defect of repeated consecutive network optimization could also apply to other tasks, e.g., network simulation. The reuse enabled by generalization reduces effort during the definition of flaws. Generalization particularly includes the identification of flaws independent of the natural language label assigned to tasks in the process model since these are chosen based on the personal preference of the DSS engineer. Furthermore, it must be possible to define flaws concerning non-consecutive process elements. For example, the previously provided example of a missing network reduction should not only be detected for a single intermediate network optimization as shown in Fig. 7.7, but also for an intermediate network simulation followed by a network optimization, etc.

7.5.3 Background and Related Work

This subsection discusses existing approaches for the functional and behavioral validation of process models to assess their reusability considering the previously established requirements for a functional-behavioral composition assistance. The discussion includes validation approaches specific to process-driven applications and business process models in general.

Functional and Behavioral Validation of Process-Driven Applications

Existing work in the context of process-driven applications (PDAs) is sparse. Again, the approach by Schneid et al. [Sch+21c], which was already discussed for the operational and informational perspective, can also be used to validate the functional and behavioral perspective of a PD-DSS by running regression tests against the derived PD-DSS that were documented using user interface wizards. The previously discussed limitations, in particular considering *ARC4 – Continuous Feedback*, still apply.

The approach described by Schiffner, Rothschädl, and Meyer [SRM14] considers the validation of a process model representing a PDA for multiple process perspectives as a key requirement for a PDA development environment. However, their approach only considers the automated validation of the informational perspective. The validation of other perspectives, particularly the functional and behavioral perspective, are only addressed with (collaborative) manual validation, which is too inefficient for *ARC4 – Continuous Feedback*.

Functional and Behavioral Validation of Process Models

Due to a lack of approaches for the functional and behavioral validation of PDAs, the discussion of related work is extended to the functional and behavioral validation of process models in general without the intention to derive an executable artifact from the process model.

A survey by Morimoto [Mor08] indicates that many approaches for process model validation utilize formal verification, e.g., based on automata, petri-nets, or process algebras. Validation rules must consequently be defined with respect to these formalizations. This requires extensive upfront training and therefore violates *ARF3 – Expert Definition*. A workaround to this limitation can be achieved by providing an abstraction over the formal verification approach, e.g., a user-friendly visual notation that is transparently mapped to the suitable formal representation. This approach is used by Förster et al. [För+07] in which a business process and associated constraints specified similar to a UML activity diagram are internally mapped to a labeled transition system and temporal logic. This mapping enables a model checker to verify that a business process upholds all constraints. While the approach fundamentally aligns with the requirements introduced in Section 7.5.2 for the domain-specific

composition assistance, it also has limitations: First, a violation of a constraint is reported as a single textual counterexample that is not mapped to elements in the original business process, thus preventing *ARC2 – Traceable Feedback*. Second, the explanations mention the need to map natural language labels to common identifiers that are shared between the business process and the constraints but do not explicitly describe how to obtain these common identifiers. Furthermore, the approach does not consider the reuse of constraints for similar types of violations, e.g., to express that the same activity should not be repeated consecutively. Thus, the approach is limited in addressing *ARF8 – Generalization*.

The approach of mapping an abstract, human-readable and -writable artifact to a formal representation is also often applied in the context of *process anti-patterns* or *process weakness patterns*, which “represent typical problems a process may have together with ideas of how to address them” [Bec+12]. Koschmider, Laue, and Fellmann [KLF19] reviewed 48 papers describing process validation approaches based on anti-patterns. The authors find that existing approaches are capable to detect defects in process models for their syntax, control flow, understandability, composition, dataflow, business rules, or overall business process model. The categories “rule-related defects” and the subcategories “need for process improvements” and “compliance” of “process-related defects” are closely related to the goal of the functional-behavioral composition assistance introduced in this section. However, many of the papers in these categories do not focus on the functional and/or behavioral perspective, but instead consider flaws in the data exchange across organizational boundaries [EEB16; KGL13], data security [Ram+18], ambiguity in natural language labels [LKG16], process inefficiencies with respect to complexity or resource consumption [BWW11], or anomalies in process adaptation rules [DH12]. Other approaches require anti-pattern authors to learn and apply query languages such as the *Structured Query Language (SQL)* [Bec+12] or the *Generic Model Query Language (GMQL)* [Ber+15; DH15; Del+15], which does not align with *ARF3 – Expert Definition*. Even other approaches require manual identification of anti-patterns [HB09], which is too inefficient to address *ARC4 – Continuous Feedback*, or do not provide sufficient details for the automated detection of anti-patterns [Bec+10]. Other papers listed by Koschmider, Laue, and Fellmann for the selected categories present collections of anti-patterns but do not discuss their (automated) detection ([KV07; GL07; HD15]).

Summary of Related Work

The discussion of related approaches shows that no approach can immediately address all previously established requirements for the functional-behavioral composition assistance. Nevertheless, the anti-pattern approach used by many contributions demonstrates how process model flaws can be specified in an “expert-friendly” way and subsequently be mapped to

an internal representation that supports the automated detection of flaws. Anti-patterns also conceptually align with the purpose of the functional-behavioral composition assistance, i.e., detect deficits in a process model and provide instructions on how to address them. For this reason, the subsequent design of the functional-behavioral composition assistance is also based on an anti-pattern based approach for the domain-specific part of the assistance, while additional checks are implemented for the domain-parametrized functionality of the assistance using the established violation rule templates.

7.5.4 Design and Demonstration

This section presents a design for the functional-behavioral composition assistance based on the detection of violations and anti-patterns in PD-DSS process models. The first half of this subsection focuses on the design of the domain-parametrized assistance functionality, while the second half focuses on the design of the domain-specific assistance functionality. The applicability of each proposed design is demonstrated using the example introduced in the motivational scenario of Section 7.5.1.

Domain-Parametrized Assistance

The design of the domain-parametrized functionality of the functional-behavioral composition assistance uses violation rule templates as those have already proven useful in the context of the previously discussed operational and informational composition assistance.

Completeness A completeness violation indicates that the PD-DSS process model does not address the functional requirements for decision support documented by a decision maker. The corresponding validation essentially works analogous to the operational composition assistance described in Section 7.3: For each *goal requirement* and *method requirement* documented by the decision maker (cf. Section 5.4.6), the validation checks if an activity with a decision support service implementing the affected computation method/goal is (not) included in the PD-DSS process model as prescribed by the associated requirement level. In case of a method requirement, this check includes the characteristics of the computation method. Due to the similarity of the resulting violation rule templates with the operational composition assistance, the explicit violations are not repeated here to avoid redundancy.

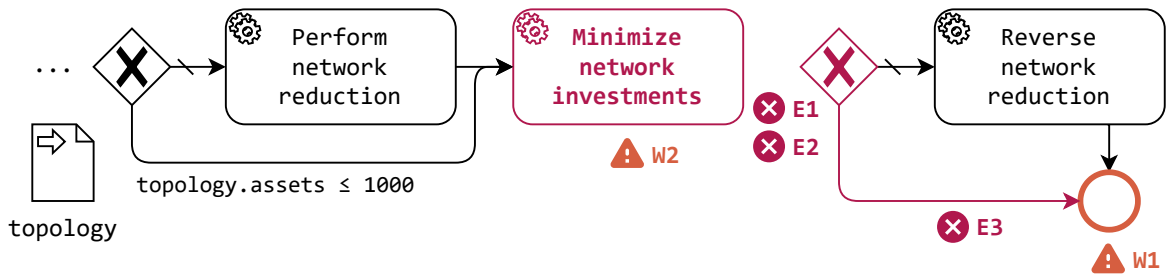
Robustness Robustness violations detect a lack of error handling in the derived PD-DSS. Lack of error handling can be detected with the *Uncaught Error Violation* based on the description of decision support services, which include a list of potentially occurring errors via the associated computation method. As usual, the severity level is reduced to a warning since the enactment of the decision process is fundamentally possible if no errors occur.

Uncaught Error Violation		Severity: Warning
Element: Task		Phase: Design
Condition: The computation method implemented by the decision support service selected for the task specifies an error for which no error boundary event is attached to the task.		
Message: “Potential error [...] of task [...] is not handled.”		

Demonstration Selected examples for the discussed violations are shown in Fig. 7.8 for a variation of the running example introduced in Section 7.5.1. Errors E1 – E3 are reported by the completeness checks described in Appendix A.1.3. A missing sequence flow between the “Minimize network investments” task and the subsequent exclusive gateway results in an *Unreachable Element Violation* (E1) reported for the gateway and a *Dead End Violation* (E2) reported for the task. Since exclusive gateways have no explicit label, they are printed as “Exclusive Gateway” in the error messages. However, the corresponding element can still be identified in the process model due to internally assigned IDs. This also applies to other unlabeled process model elements, e.g., the end event referenced in the *Missing Condition Violation* (E3). An example of a completeness violation is given in the form of a *SHOULD Functionality Violation* (W1) as the decision maker requested a visualization of the computed investment plan that is not part of the process model. An *Uncaught Error Violation* (W2) is reported since the topology error that is potentially thrown by the *PowOpt* service selected for the *Minimize network investments* activity is not caught with an error boundary event.

Domain-Specific Assistance

The detection of the previously described violations automatically adapts to a concrete application domain based on the description of the domain and contained decision support services. As discussed throughout the requirements for the functional-behavioral composition assistance, the assistance must also detect flaws that are specific to a concrete application domain. For this purpose, the functional-behavioral composition assistance requires composition knowledge contributed by composition experts. Based on the discussion of related approaches in Section 7.5.3, anti-patterns provide an abstraction that is suitable for composition experts to document their knowledge and can be mapped to automated approaches for anti-pattern detection. The subsequently described design for the domain-specific functionality of the functional-behavioral composition assistance first describes its architecture, the constituents of an anti-pattern, and an anti-pattern representation that strikes a compromise between human- and machine-readability.



- ✘ **E1** **Error - Unreachable Element Violation [Completeness Check]**
Gateway [Exclusive Gateway] will never be executed because it has no incoming sequence flow.
- ✘ **E2** **Error - Dead End Violation [Completeness Check]**
Process will get stuck in activity [Minimize network investments] because it has no outgoing sequence flow.
- ✘ **E3** **Error - Missing Condition Violation [Completeness Check]**
Sequence flow from [Exclusive Gateway] to [End Event] has no condition.
- ⚠ **W1** **Warning - SHOULD Functionality Violation**
Process should include task for [Visualize Investment Plan].
- ⚠ **W2** **Warning - Uncaught Error Violation**
Error [Topology Error] of task [Minimize network investments] is not handled.

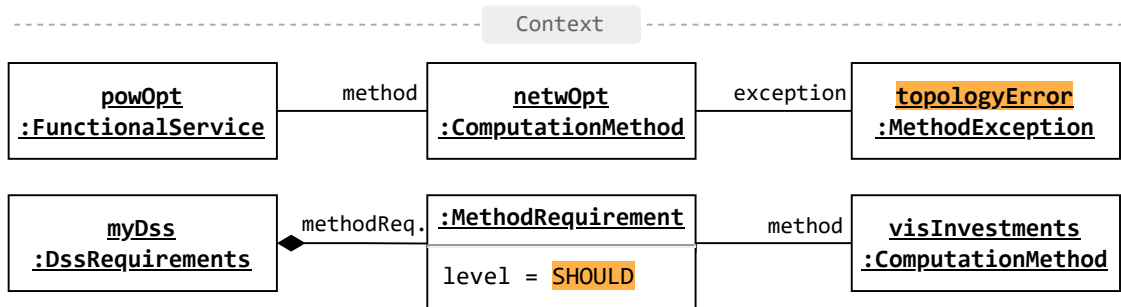


Figure 7.8: Demonstration of selected domain-agnostic and domain-parametrized violations

Architecture of the Domain-Specific Functional-Behavioral Composition Assistance

The architecture of the domain-specific functional-behavioral composition assistance is shown in Fig. 7.9. The *Anti-Pattern Definition & Repository* application stores the anti-patterns defined by composition experts. Composition experts are provided with two sources for anti-pattern derivation: First, they can define anti-patterns based on the structured description of decision support services and associated computation methods and goals using their knowledge of the application domain. The descriptions are provided by the *Domain Registry* and *Service Registry* respectively (cf. Chapter 5). Second, composition experts can also monitor PD-DSS service compositions created by DSS engineers to identify anti-patterns

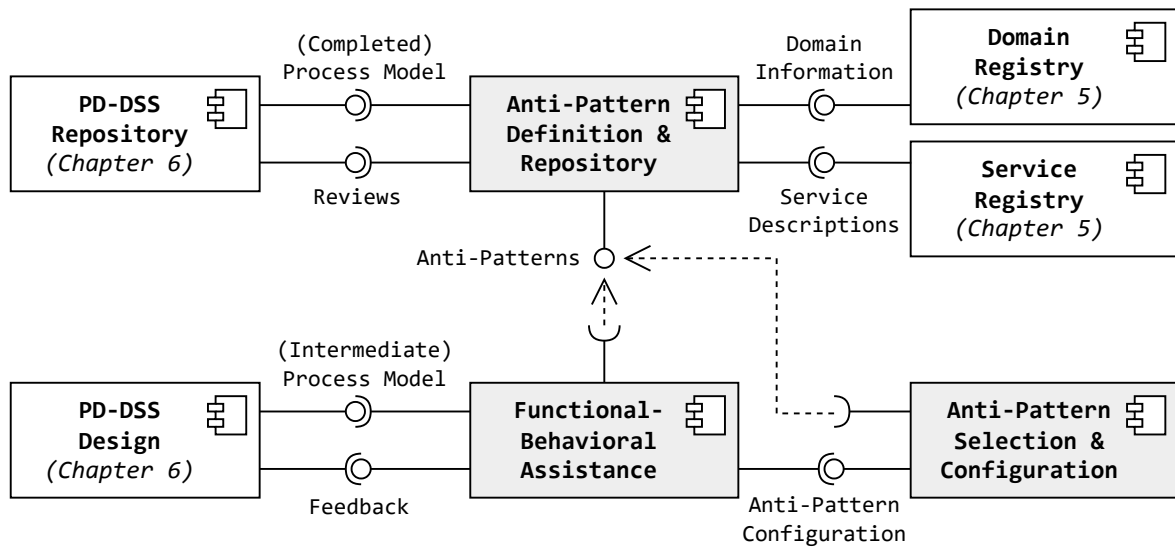


Figure 7.9: Architectural overview of the functional-behavioral composition assistance

based on their practical relevance. For hints regarding which service compositions performed exceptionally well or poorly and are therefore especially suitable for anti-pattern derivation, composition experts can utilize feedback for service compositions provided by decision makers after using the derived tailored PD-DSS. This feedback is subsequently referred to as a *(PD-DSS) review* to avoid confusion with the feedback provided to DSS engineers by a composition assistance. Decision makers should provide reviews at the *PD-DSS Repository* application due to its aggregating nature (cf. Chapter 6).

The functional-behavioral composition assistance then obtains the anti-patterns and integrates with the *PD-DSS Design* application described in Chapter 6 to provide DSS engineers with feedback on the designed PD-DSS process model similar to the messages of violation rule templates. Optionally, a DSS engineer can restrict and configure the detection of anti-patterns using the *Anti-Pattern Selection & Configuration* application.

PD-DSS Reviews Reviews of a PD-DSS process model should be provided by decision makers as a quantitative rating in combination with a natural language explanation. The quantitative rating enables composition experts to quickly identify PD-DSS service compositions that implement the associated requirements for decision support particularly well or poorly. On the other hand, the natural language explanations support an arbitrary level of detail and enable composition experts to gain a more thorough understanding of the advantages and disadvantages of individual service compositions. Reviews should assess both the effectiveness and efficiency of the PD-DSS derived from the process-based service

composition. Effectiveness and efficiency can be rated considering the overall requirements for decision support or for each contained “sub-requirement” regarding specific functional, non-functional or data requirements (cf. Section 5.4.6). Feedback can be quantified using a Likert scale where decision makers rate their agreement with the statements “The PD-DSS efficiently addresses my requirements for decision support” and “The PD-DSS effectively addresses my requirements for decision support” from *strongly disagree* (1) over *disagree* (2), *neither agree nor disagree* (3), *agree* (4) to *strongly agree* (5). Alternatively, the scale from employee performance reviews may be reused due to offering more expressive labels for response options ranging from *unsatisfactory performance* (1) over *improvement desired* (2), *meets expectations* (3), *exceeds expectations* (4) to *outstanding performance* (5) [SDA20].

Anti-Pattern Constituents Based on a literature review, Koschmider, Laue, and Fellmann [KLF19] identify three properties that should be documented for an anti-pattern: (1) a descriptive name, (2) a textual, graphical, or formal definition (or a combination thereof), and (3) a description that documents why the anti-pattern is unfavorable (“problem”) and how and why the anti-pattern should be avoided (“improvement”). An anti-pattern documentation for the functional-behavioral composition assistance furthermore includes (4) a severity level, i.e., error or warning, to distinguish the effects on PD-DSS enactment (cf. Section 7.1.3).

Composition experts can specify a name, a severity level, and a textual description for the problem and improvement of an anti-pattern regardless of their training since these properties only require a textual specification. For the definition of the anti-pattern itself, only a graphical definition remains as an option since a textual definition does not support automated detection of the anti-pattern (cf. *ARC4 – Continuous Feedback*) and a formal definition cannot be expected from composition experts (cf. *ARF3 – Expert Definition*). However, it is unclear if and how a graphical anti-pattern definition approach can be designed that is both readable and writable for composition experts, and also has a machine-readable representation suitable for automated anti-pattern detection.

Graphical Anti-Pattern Definition The necessity to find a compromise between a notation that is both human-readable and machine-readable already emerged during the definition of a PD-DSS representation in Chapter 6 where BPMN was ultimately selected for the description of a PD-DSS as an executable process model. The selection was motivated by the fact that BPMN is widely known and usable without extensive upfront training while also defining a serialization format for the visual notation, thereby enabling automated processing.

Given BPMN’s advantages, the obvious idea of reusing BPMN for the definition of anti-patterns is not unreasonable: Anti-pattern detection is essentially a form of matching

where a subset of process model elements is identified within a PD-DSS process model that meets the prerequisites of the anti-pattern. In a naive approach, composition experts can simply define an anti-pattern as a partial BPMN process model. When the partial anti-pattern process model can be found in the designed PD-DSS process model, the DSS engineer can be informed about the presence of the anti-pattern, its associated problem and improvement descriptions, and affected model elements. Unfortunately, this naive approach fails to address many requirements for the domain-specific assistance described in Section 7.5.2. For example, the approach can only detect the existence of elements associated with *ARF5 – Unwanted Functionality* and *ARF7 – Unwanted Branching*, but the nonexistence of elements needed for *ARF4 – Missing Functionality* and *ARF6 – Missing Branching* cannot be detected. Another shortcoming is that the heterogeneity of task labels and the support for non-consecutive process elements are not addressed (cf. *ARF8 – Generalization*).

The restrictions show that additional selectors and placeholders are needed to control the specificity and fuzziness of the matching. Such control is provided by *BPMN-Q*, an extension of BPMN that supports visual queries against BPMN process models [Awa07]. Although *BPMN-Q* was originally introduced to retrieve process models from a model repository that satisfy certain constraints [Awa07], it has also been successfully applied for the detection of anti-patterns in the work by Ramadan et al. [Ram+18] and Laue and Awad [LA10], albeit with a focus on security-related concerns or domain-agnostic anti-patterns respectively. As a result, it still has to be shown if and how *BPMN-Q* can be applied in the PD-DSS context.

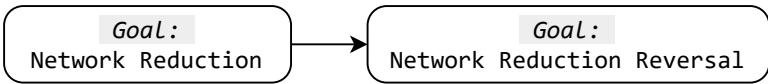
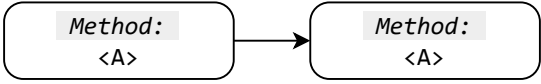
BPMN-Q Elements and Extensions Table 7.1 shows an overview of *BPMN-Q* elements with minor extensions to enable their application to PD-DSS process models. Since *BPMN-Q* extends BPMN, an anti-pattern definition based on *BPMN-Q* can fundamentally include any BPMN element. A collection of elements relevant in the context of a PD-DSS is shown under the *standard elements* group. A BPMN task is not contained in the list of standard elements since tasks are matched via a task selector to support the matching of tasks independent of the chosen natural language label. Instead, a *task selector* interprets the task label as an ID referring to a computation goal, a computation method, or a concrete decision support service as indicated by the preceding prefix. The standard BPMN elements in combination with task selectors support the definition of simple anti-patterns, e.g., *Redundant Network Reduction*.

A *placeholder* can be used instead of a concrete task ID for “fuzzy” matching. A named placeholder corresponds to *BPMN-Q variable*, and an anonymous placeholder is added for scenarios where the ID is irrelevant as it is not matched across multiple process model elements. The decoration of a placeholder is changed from a preceding @-sign to angle brackets for better readability. An example is provided with the *Redundant Activity* anti-pattern.


Table 7.1: (Extended) BPMN-Q elements for the graphical definition of anti-patterns

	<p>Standard Elements: An anti-pattern definition can include any process model element defined by the BPMN specification, including start, end, and error boundary events, parallel and exclusive gateways, and sequence flows.</p>
	<p>Task Selectors: A <i>task selector</i> matches a BPMN (service/user) task with the associated computation goal, computation method, or decision support service of the specified ID as described on the task label.</p>
	<p>Placeholders: A <i>placeholder</i> matches an arbitrary ID for a computation goal, computation method or decision support service. A <i>named placeholder</i> (<A>) matches the same ID across multiple tasks.</p>
	<p>Nonexistent Flow: A <i>nonexistent flow</i> specifies the lack of a sequence flow between two process model elements.</p>
	<p>Nonsequential Flow: A <i>nonsequential flow</i> specifies that the second process model element must eventually (but not necessarily immediately) follow the first element. Optionally, a lower and upper cardinality for intermediate elements can be defined (default: 0..*).</p>
	<p>Nonexistent Nonsequential Flow: A <i>nonexistent nonsequential flow</i> combines the characteristics of a nonexistent flow and a nonsequential flow.</p>
	<p>Conditional Flows: A condition can be used to augment a flow. For nonsequential flows, this means the condition must be present on some sequence flow between elements.</p>

A placeholder only supports fuzziness for a single task, but not for intermediate process model elements. For this purpose, a *nonsequential flow* documents that multiple elements can be contained between two connected tasks. Both the sequential and nonsequential flow

Redundant Network Reduction		Severity: Warning
Definition:	 <pre> graph LR G1[Goal: Network Reduction] --> G2[Goal: Network Reduction Reversal] </pre>	
Problem:	A network reduction is redundant because it is immediately reversed, thus introducing inefficiency with an unnecessary computation.	
Improvement:	Remove the network reduction and its reversal or add activities in between.	
Redundant Activity		Severity: Warning
Definition:	 <pre> graph LR M1[Method: <A>] --> M2[Method: <A>] </pre>	
Problem:	The same computation method is executed twice in direct succession.	
Improvement:	Remove one of the duplicate executions.	

have a counterpart in the *nonexistent flow* and *nonexistent nonsequential flow* to describe the lack of a flow between tasks. A lack of flow between tasks especially includes the situation where the task specified as the target of the nonexistent flow is not included in the process model. For example, the anti-pattern *Missing Reversal of Network Reduction* is detected when a network reduction is performed in the designed PD-DSS process model, but either no task for the reversal of the process model is included in the process model, or at least one scenario exists where the reversal is not performed, e.g., due to conditional execution.

Missing Reversal of Network Reduction		Severity: Error
Definition:	 <pre> graph LR G1[Goal: Network Reduction] -.-X//> G2[Goal: Network Reduction Reversal] </pre>	
Problem:	A network reduction is performed but never reversed. Results computed after the network reduction do not apply to the original topology.	
Improvement:	Add an activity for the reversal of the network reduction.	

Like task selectors, conditional flows are an extension to BPMN-Q. For a conditional sequence flow or a conditional nonexistent flow, it is sufficient to check whether the associated condition is present on a flow in the designed PD-DSS process model. For a nonsequential flow, the condition must exist among some sequence flow contained between the first and second task connected by the nonsequential flow without being reset by a merging exclusive gateway. An example of a conditional flow is included in the *Potential Redundant Network Optimization* anti-pattern, which documents that the necessity of a network optimization

should be checked based on the results of the power flow analysis. The condition exhibits two characteristics that deviate from the default format of conditions introduced throughout the explanation of the behavioral PD-DSS perspective in Chapter 6: First, the left part of the condition cannot refer to a concrete data object or process data input as their names can be arbitrarily chosen by DSS engineers for the PD-DSS process model. Instead, the condition refers to the input of the optimization task by referring to the ID of the computation method (*Asset_Optimization*) and the method data (*loads*) separated by a colon. This assumes that input method data and output method data have unique identifiers among a computation method to avoid confusion about whether the method data refers to input or output method data. The reference to method data implies that conditions cannot be defined for computation goals as these do not specify concrete input and output data. The second derivation is the right side of the condition referring to a (constant) parameter as indicated by the use of curly braces in the definition and the corresponding row in the tabular anti-pattern documentation. The parameter defines the maximum acceptable asset load, which may be specific to the concrete decision situation of a decision maker. The DSS engineer must specify a value for the parameter, e.g., *110%*, before the anti-pattern can be detected in the designed PD-DSS process model. This is done with the *Anti-Pattern Selection & Configuration* application of Fig. 7.9, where DSS engineers can furthermore overwrite the severity level of anti-patterns.

Potential Redundant Network Optimization		Severity: Warning
Definition:	<pre> Goal: Power Flow Simulation -----X//-----> Method: Asset Optimization Asset_Optimization:loads .maxLoad > {t} </pre>	
Problem:	A network optimization is performed without checking if the asset loads computed as part of a power flow analysis actually exceed permitted thresholds. This can introduce an inefficiency due to an unnecessary instantiation of the optimization model.	
Improvement:	Add an exclusive gateway that only invokes the network optimization if the maximum load of an asset exceeds the permitted threshold.	
Parameters:	{t}: Accepted upper threshold for asset loads	

Demonstration The detection of three out of the four previously defined exemplary anti-patterns is shown in Fig. 7.10. Unlike the previously reported violations, an anti-pattern does not include a parametrized message. As a result, the process model elements that are part of the detected anti-pattern are listed instead, and an information icon provides a

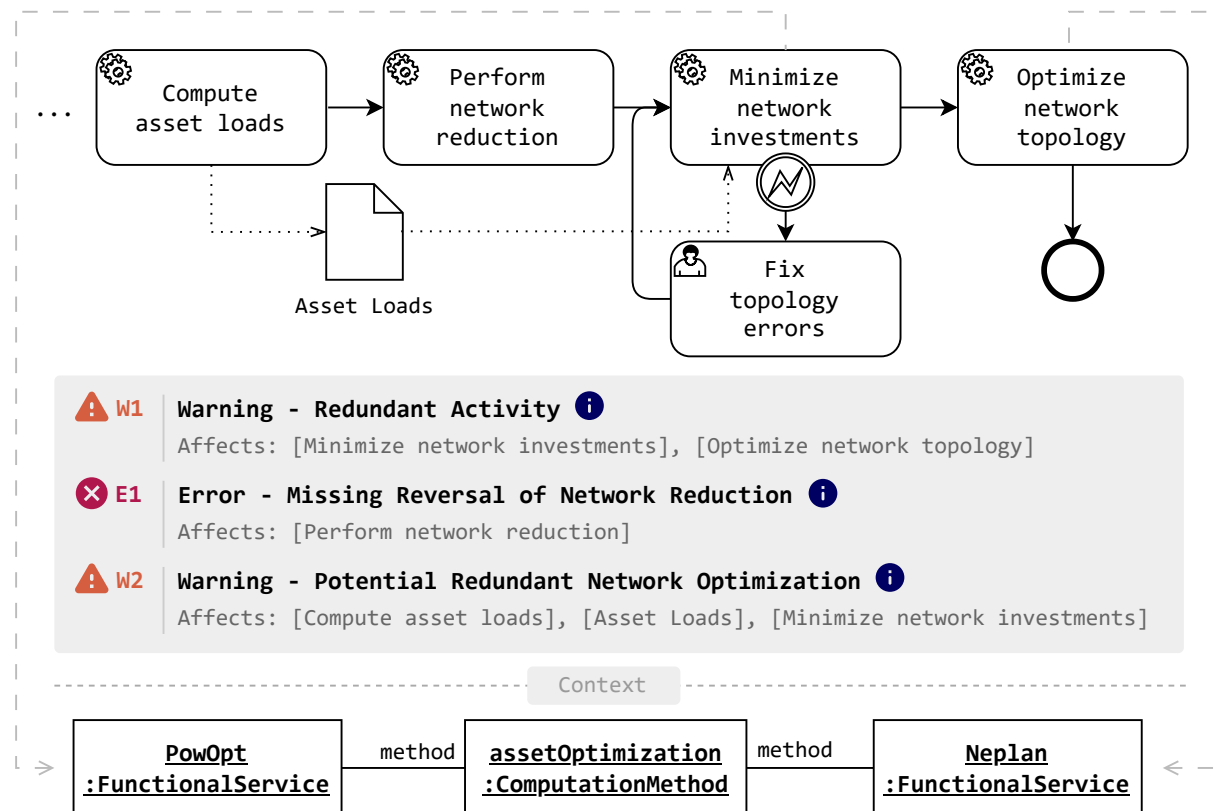


Figure 7.10: Demonstration of selected domain-specific anti-patterns

reference to the complete anti-pattern documentation, including the description of the problem and improvement hints. Since anti-patterns affect multiple process model elements, it is recommended to highlight the affected process model elements only when the DSS engineer specifically requests it, e.g., by clicking on or hovering over the error message.

The example furthermore highlights the importance of task selectors: Although the tasks “Minimize network investments” and “Optimize network topology” use different natural language labels as well as different decision support services, the redundancy of a duplicate optimization can still be detected as part of the *Redundant Activity* anti-pattern due to the “*Method:*” selector prefix used in the anti-pattern. For warning W2, the list of affected elements includes the “Asset Loads” data object as it matches the *Asset_Optimization:loads* identifier of the *Potential Redundant Network Optimization* anti-pattern.

7.5.5 Prototypical Implementation

The implementation of the proposed anti-pattern design for the domain-specific functional-behavioral composition assistance is challenged by the unavailability of a reusable and

extensible tool for the execution of BPMN-Q queries. As a result, the technical feasibility of the proposed design is demonstrated with an implementation based on graph matching as visualized in Fig. 7.11. The intermediate PD-DSS process model designed by a DSS engineer using the *PD-DSS Design* application is first transformed into a graph representation and stored in a graph database. Afterwards, the (configured) anti-patterns are converted into queries that are run against the database. If a non-empty result is returned, the anti-pattern is present in the process model and the result is transformed into the established format for assistance feedback such that it can be displayed to the DSS engineer via the *PD-DSS Design* application. The remainder of this subsection briefly summarizes the necessary transformations and the insights gathered throughout the prototypical implementation.

Transformation Approach

The transformation approach utilizes the fact that a process model already corresponds to a graph: Activities, gateways, and events correspond to *vertices* that are connected via sequence flows corresponding to *edges*. Furthermore, *label functions* can be used to capture additional properties, e.g., the type of a task, the associated computation method, or the condition specified for a sequence flow. Thus, the *converter module* and *matcher module* of the functional-behavioral composition assistance only need to transform the BPMN representation of the process model into the representation supported by the selected *graph database* for graph

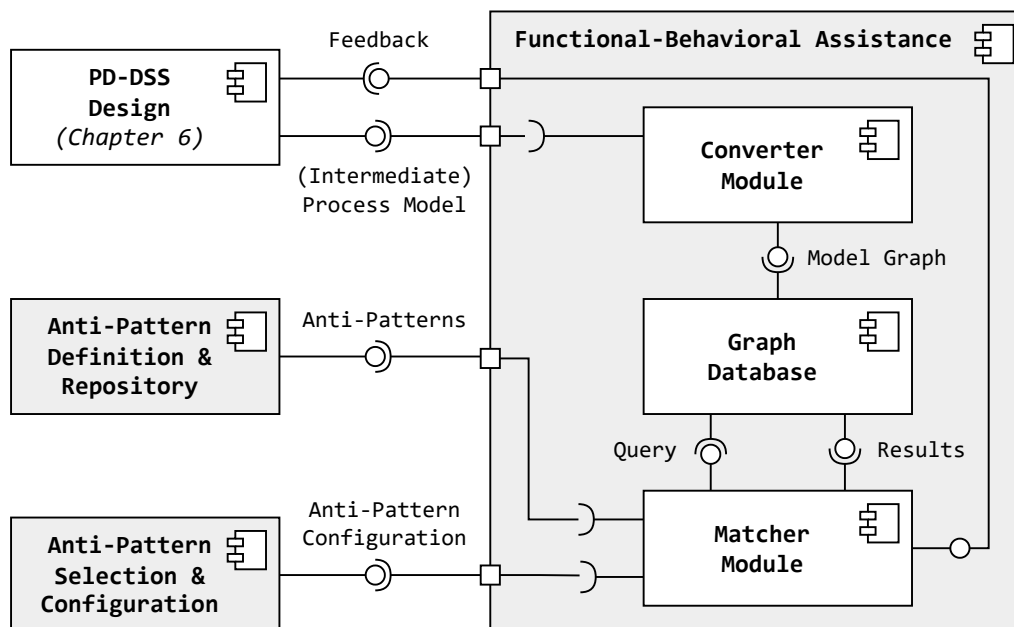


Figure 7.11: Internal architecture of the functional-behavioral composition assistance

storage and querying. For the prototypical implementation, the graph database *Neo4j*³ was chosen. The technical details of the transformation of BPMN process models and BPMN-Q anti-patterns are described in the paper “Anti-pattern Detection in Process-Driven Decision Support Systems” [KE22] with a minor correction and extension published in Appendix A.2. The implementation is furthermore provided as supplementary material (cf. Appendix B).

Insights from the Prototypical Implementation

Using the prototypical implementation with exemplary anti-patterns and process models for energy distribution network planning demonstrated the fundamental applicability and technical feasibility of the anti-pattern approach to provide the domain-specific functionality of the functional-behavioral composition assistance. The application furthermore provided insights regarding the performance of the approach. The first execution of a query took rather long on a 2018 commodity laptop, i.e., even for comparatively simple patterns with a complexity similar to the *Redundant Network Reduction* or *Redundant Activity* anti-patterns, the first query completed between 50ms and 100ms. A query for an anti-pattern with a nonexistent nonsequential flow such as the *Missing Reversal of Network Reduction* anti-pattern could even require up to 200ms for the first query. These execution times would only support checking a handful of anti-patterns before the DSS engineer would no longer perceive the feedback as real-time. Fortunately, subsequent executions of a previously executed anti-pattern query were completed significantly faster in 3ms or less, even when the underlying process model graph changed in between. This is due to the fact how *Neo4j* utilizes caching and transforming queries into an optimized representation referred to as an *execution plan* (cf. [Veg18] for additional technical details). These execution times suggest that even a comprehensive catalog of anti-patterns can be detected using the approach in perceived real-time.

7.5.6 Discussion

The fundamental capabilities of the functional-behavioral composition assistance are demonstrated with the examples provided for illustrative purposes throughout the explanation of its design. The technical feasibility of the domain-specific part of the functional-behavioral composition assistance is fundamentally demonstrated with the prototypical implementation. Therefore, the subsequent discussion focuses on the relation between the proposed design concepts and the requirements for the functional-behavioral composition assistance as well as the cross-cutting requirements for the composition assistance established in Section 7.2.

³ Neo4j website: <https://neo4j.com/>

Requirements Specific to the Functional-Behavioral Composition Assistance

Requirements specific to the functional-behavioral composition assistance established in Section 7.5.2 are grouped into two categories for the domain-parametrized and domain-specific functionality of the assistance. These categories are represented as subsections of Section 7.5.4 detailing the design of the assistance. The design explanations of the domain-parametrized functionality are further structured to provide an immediate mapping to *ARF1 – Completeness* and *ARF2 – Robustness* and are not repeated here to avoid redundancy.

The requirements for the domain-specific part of the functional-behavioral composition assistance are largely addressed by the expressiveness of BPMN-Q. The graphical anti-pattern definition approach enabled by BPMN-Q (with minor extensions for the DSE context) significantly contributes towards *ARF3 – Expert Definition*. In particular, the reuse of BPMN standard elements including gateways and sequence flows supports the definition of *ARF5 – Unwanted Functionality* and *ARF7 – Unwanted Branching*. The use of nonexistent flows in a BPMN-Q anti-pattern definition supports *ARF4 – Missing Functionality* and *ARF6 – Missing Branching*. The selectors, placeholders, and actual parametrization that can be used in an anti-pattern definition address *ARF8 – Generalization*, although the use of a graph database for the detection of anti-patterns shows the potential of using filters that could be specified for placeholders to restrict computation methods, goals and services that can be affected by the anti-pattern. Filtered placeholders could also be useful outside of tasks, e.g., as shown in the *Network Reduction Condition Mismatch* anti-pattern for conditions.

Network Reduction Condition Mismatch		Severity: Error
Definition:		
Filter:	c1 != c2	
Problem:	The condition that determines the execution of a network reduction is different from the condition which determines whether the network reduction is reversed. This can result in a mismatch where a reduction is not reversed, or the reversal of a never executed reduction is attempted.	
Improvement:	Unify the conditions before the network reduction and its reversal.	

Cross-Cutting Requirements for the Composition Assistance

In addition to the requirements specific to the functional-behavioral composition assistance, the assistance also addresses the cross-cutting requirements established in Section 7.2.

Requirement *ARC1 – Static Validation* is addressed since all (anti-pattern) violations can be detected during the design of the decision support service composition. For the domain-parametrized functionality of the functional-behavioral composition assistance, requirements *ARC2 – Traceable Feedback* and *ARC3 – Comprehensive Feedback* are addressed by using the proven violation rule templates. For the domain-specific assistance functionality, these requirements are addressed by reporting affected process model elements and the description of an identified anti-pattern. In theory, the anti-pattern problem description could be further customized to the process model under consideration by numbering the elements in the BPMN-Q anti-pattern definition such that they can be referenced in the problem description, e.g., for the *Redundant Activity* anti-pattern the error message could be parametrized as “*Activity [#1] and activity [#2] are redundant since they use the same computation method [<A>]*” (where #1 and #2 get replaced by the natural language labels of the matched tasks, and <A> gets replaced by the name of the computation method). However, fortifying the messages with references adds some complexity to the definition of anti-patterns and may discourage some potential composition experts from contributing their composition knowledge. The prototypical implementation demonstrated that the approach of anti-pattern detection using a graph database is fast enough for *ARC4 – Continuous Feedback*.

Threats to Validity

In addition to the cross-cutting threats to validity discussed in Section 8.6, the proposed design for the functional-behavioral composition assistance has not explicitly been evaluated with subprocesses. Although their inclusion in a BPMN-Q anti-pattern definition is trivial, they essentially correspond to a sub-graph in the process model that has not been considered for the transformation of a process model into a graph representation. The examples furthermore do not consider the parallel execution of tasks that can be included in a BPMN-Q anti-pattern definition. In case of dependencies between tasks of different parallel flows it might be necessary to transform the process model into a labeled transition system before inserting it into the graph database (similar to the approach by Förster et al. [För+07]).

The prototypical implementation furthermore focuses on the transformation of the process model and anti-pattern definitions. The *anti-pattern definition application* was not explicitly implemented, although this is a minor limitation since its modeling capabilities are fundamentally similar to the *PD-DSS Design* application whose technical feasibility was demonstrated in Chapter 6. The prototypical implementation also does not support task selectors, but they can be easily implemented by adding properties for the IDs of computation method/goal and decision support service to the nodes in the graph representing BPMN tasks.

Transition to Recommender System

The functional-behavioral composition assistance can be used to implement a recommender system that utilizes composition knowledge for proactive error prevention. In particular, whenever the DSS engineer adds an element to a process model that is the source of a nonexistent (nonsequential) flow, the recommender system could automatically add the process element that is the target of the flow if it is not already included in the PD-DSS process model. For example, whenever a network reduction is added to the process model, a task for the reversal of the network reduction could be automatically added due to the *Missing Reversal of Network Reduction* anti-pattern. This proactively prevents DSS engineers to create PD-DSS process models containing the anti-pattern. Furthermore, the description of anti-pattern improvements could be switched from the current textual to a more structured definition to support the automated correction of flaws. For example, a redundant task could be automatically removed for the *Redundant Activity* anti-pattern. This transformation could potentially be described as a graph transformation to enable its automated application.

7.6 Key Takeways

The proposed composition assistance can support DSS engineers in composing decision support services into PD-DSS process models that effectively and efficiently address a decision maker's requirements for decision support. The composition assistance is integrated into the *PD-DSS Design* application used by the DSS engineer and continuously provides actionable feedback for the designed composition regarding its operational, informational, functional, and behavioral perspective. The assistance automatically derives composition knowledge from the documentation of the application domain and the contained decision support services. In addition to this domain-parametrized support, the assistance can furthermore provide domain-specific support for designing the functional and behavioral perspective of a PD-DSS service composition by detecting anti-patterns contributed by composition experts based on feedback provided by decision makers and DSS engineers. Although the assistance is primarily intended to validate a designed PD-DSS process model, it can also support the implementation of a recommender system that actively proposes extensions to a PD-DSS process model to reduce the workload of DSS engineers.

The composition assistance concludes the series of contributions that together support all phases and activities of the DSE lifecycle. The remainder of the thesis focuses on the discussion of the proposed solution designs with respect to the requirements for tailored DSS development established in Chapter 3 and the research questions established in Chapter 1.

PART III

Solution Discussion

Case-Study Evaluation

By combining the partial solution designs proposed through Chapters 5 to 7, all phases of the DSE lifecycle are addressed (cf. Fig. 8.1). Each chapter already demonstrates the fundamental applicability and technical feasibility of its described solution design in isolation considering its individual design requirements derived from the DSE concept introduced in Chapter 4. This chapter provides a coherent evaluation of the partial solution designs – and, by extension, an evaluation of the overall DSE concept – with respect to the requirements for tailored DSS development established in Chapter 3. For this purpose, the DSE lifecycle activities of Fig. 8.1 are enacted based on practical insights from a research project with industry partners to demonstrate the applicability of the DSE concept. The results of this case study are subsequently used to show that the DSE concept addresses all previously established requirements for providing each decision maker with a tailored DSS.

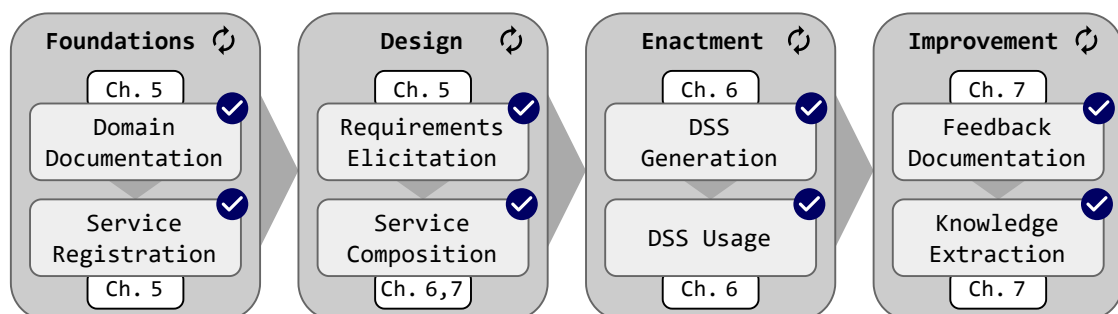


Figure 8.1: All phases of the DSE lifecycle are addressed with the presented solution design

After an overview of the research project that serves as a foundation for the case study (Section 8.1), case study insights from enacting the DSE lifecycle phases are described for the *Foundation* phase (Section 8.2), the *Design* phase (Section 8.3), the *Enactment* phase (Section 8.4), and the *Improvement* phase (Section 8.5). Each section also describes the contributions of the associated solution design (which is visually summarized in advance with the refinement of the DSE platform reference architecture in Fig. 8.2) towards the requirements for tailored DSS development. Lastly, threats to validity that could impede the generalization of the solution design and evaluation results are presented (Section 8.6) and the case study insights are summarized (Section 8.7).

8.1 Context: Research Project “FlexiEnergy”

The subsequently described case study is based on practical insights that were gathered throughout the author’s participation in the *FlexiEnergy* research project¹ from September 2018 to December 2021. The German research project received funding from the European Union to design and develop a decision support system for energy distribution network planning under uncertainty across multiple energy sectors. Nine partners from academia and industry collaborated in the research project, including two local distribution network operators and an agency specialized in energy distribution network planning.

Throughout the research project, multiple alternative activities for the fundamental decision process phases of energy distribution network planning presented in Section 2.3 were documented for selection based on different situational factors. While the duration of the project limited implementation efforts to a few selected activities, other alternative activities were nevertheless defined on a conceptual level, thus providing a sufficient amount of selection alternatives for a DSE case study. However, the focus on a conceptual specification of decision activities in combination with the confidentiality of some contributions made by project partners requires the subsequent case study to be “simulated”, i.e., it is closely based on real-world experience gathered throughout the research project, but it does not describe actual project contributions. Furthermore, due to the extent of real-world planning processes, the subsequently presented case study only focuses on the initial forecasting phase during which one or multiple forecasts are created to anticipate customers’ energy demands. Since the previous chapters already demonstrated the solution design on a fine-grained (technical) level, the subsequent demonstration in the context of the coherent case study focuses on a textual description of integrating the partial solution designs across the DSE lifecycle phases.

¹ Project website: <https://www.flexi-energy.de/en/>

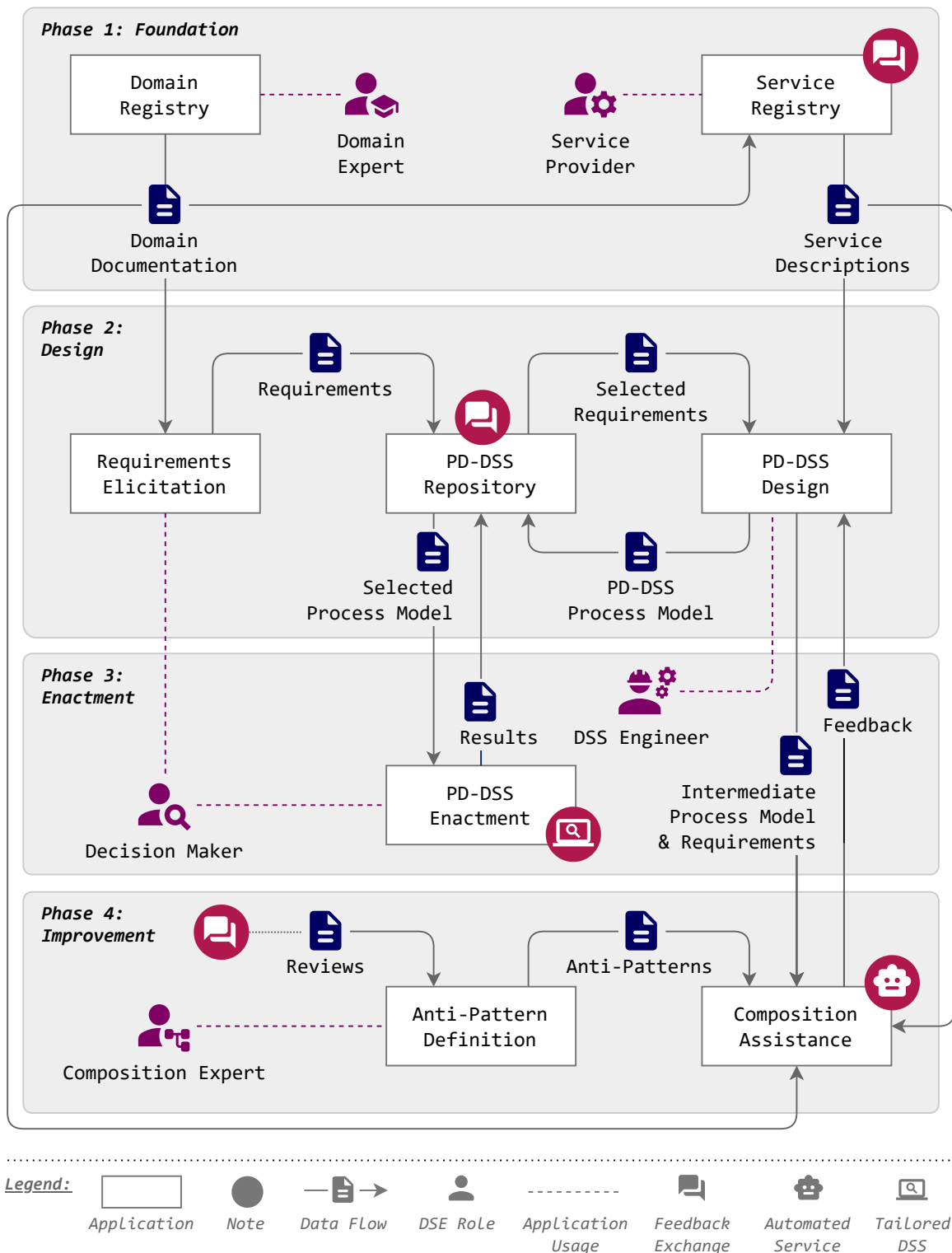


Figure 8.2: Refinement of the DSE platform reference architecture based on the solution design

8.2 Phase 1 – Foundation

This section describes the artifacts produced during the enactment of the DSE lifecycle phase *Foundation* in the context of the *FlexiEnergy* case study. During the initial *Domain Documentation* lifecycle activity, domain experts use the *Domain Registry* application to create a documentation of decision making in an application domain. For better readability, the explanation of this domain documentation is subsequently split into data characteristics (Section 8.2.1) and computation methods (Section 8.2.2). During the subsequent *Service Registration* lifecycle activity, service providers use the *Service Registry* to describe functional and data decision support services (Section 8.2.3). Based on the case study insights, the contributions of the *Foundation* phase towards the requirements for tailored DSS development established in Section 3.1 are discussed (Section 8.2.4).

8.2.1 Domain Documentation: Data Types and Metadata Attributes

The data type that is most relevant for forecasting the demands of energy customers connected to an energy distribution network is *Customer Demands*. This data type corresponds to a list of quadruples describing which customer (identified by an ID) at what point in time (documented as an ISO-string) consumes or produces how much energy (in kW) of what energy type, e.g., (`#1234`, `2023-01-01T01:00:00Z`, `0.345`, `electricity`). This structure can be used to describe historical, current, and expected future energy demands with an arbitrary temporal resolution for multiple energy sectors. This suggests metadata attributes *Start* and *End* to document the first and last timestamp for which data is provided respectively. Furthermore, a *Resolution* attribute indicates whether the data contains “hourly”, “daily”, “weekly”, “monthly”, or “yearly” values. The *Sectors* attribute indicates whether the data describes “electricity”, “gas” or “heat” demand for one or multiple of the listed sectors (i.e., the *Sectors* attribute has a minimum cardinality of one and a maximum cardinality of three). All metadata attributes may be relevant to determine whether the demand data is compatible with a service, e.g., if it is within a certain time interval, has the correct temporal resolution, or includes (only) the supported energy sector(s). The *Customer Demands* data type can be represented using a data format suitable for storing a list of quadruples. Naturally, this can be the tabular CSV format, or JSON, either as an array of arrays (*JSON-Array*) or as an array of JSON objects (*JSON-Objects*). Additional proprietary data formats are possible.

The *Customer Demands* data type illustrates that the definition of data types and metadata attributes may depend on the personal preference and requirements of domain experts and other ecosystem participants. For example, instead of defining the *Resolution* attribute as a

qualitative attribute with a set of selected values determining its range, it could also be defined as a quantitative attribute that specifies an arbitrary number of minutes between timestamps. This supports more resolutions than the qualitative variant presented before, but in direct comparison, a “yearly” resolution is easier to grasp than 525,600 minutes. An additional example for situational modeling is the size of a dataset, which can be computed based on the attribute values specified for *Start*, *End*, and *Resolution*. However, ecosystem participants might require an explicit *Size* attribute to define service constraints or conditions on sequence flows in the BPMN process model. Additional data types and metadata attributes are briefly explained when needed throughout the remainder of this section to avoid repetitiveness.

8.2.2 Domain Documentation: Computation Methods

This section presents multiple computation methods available for generating forecasts of customers’ energy demands that were conceptualized during the *FlexiEnergy* research project and are part of the domain documentation created by domain experts. The methods are grouped under their associated computation goal.

Immediate Demand Forecasting

The *Immediate Demand Forecasting* goal aggregates computation methods that compute the expected energy demands of customers within a single activity in the decision process.

The **Demand Extrapolation** computation method uses a statistical approach to project the future energy demands of individual customers based on their historical energy demands. The method requires the historical energy demands of customers as input and produces the extrapolated energy demands as output. Thus, it only utilizes the previously described *Customer Demands* data type. An overview of potentially applicable approaches to implement this computation method is presented by Ghalekhondabi et al. [Gha+17]. Method characteristics can be used to document details of the chosen approach, e.g., whether the extrapolation is performed using regression or a neural network.

The **Demand Scaling** computation method scales the current energy demands of individual customers by multiplying them with a scaling factor. This method requires two inputs, the current energy demands of customers and the scaling factors to apply, and produces the scaled expected future energy demands as output. The *Scaling Factors* data type documents the scaling factor as a list of triples consisting of the energy sector, the scaling factor (relative to the first documented energy demand of customers), and the time after which the scaling factor should be applied, e.g., (‘electricity’, ‘2025-01-01’, 1.1). Scaling factors can be provided as input to the decision process by decision makers. Otherwise, they are determined

throughout a preceding activity of the decision process, either manually via user selection or automatically based on statistical analysis (with both approaches corresponding to individual computation methods of a *Scaling Factor Specification* goal).

Technology Adoption Simulation

The *Technology Adoption Simulation* goal is the first part of a two-stage demand forecasting approach where the adoption of consumer technologies such as electric vehicles or heat pumps by customers is first simulated and the energy demands of customers are subsequently updated based on the simulation results.

The **Strategy-Based Technology Adoption Simulation** computation method assigns technologies to customers based on some pre-defined internal strategy. Naive strategies include a geographically uniform or random assignment of technologies to customers. More advanced strategies compute the best- or worst-case assignment of technologies with respect to the effects on network asset loads. A method characteristic enables associated decision support services to document the implemented strategy. The computation method requires the expected development of consumer technology market shares and the network topology with customer locations as input and produces the simulated technology adoptions of customers as output. Market shares of consumer technologies are captured with the *Technology Market Shares* data type. It consists of triples documenting the consumer technology, the date, and the expected market share, e.g., ('BEV', '2023-01-01', 0.04). The simulation results are captured with the *Technology Assignment* data type consisting of events in the form of quadruples that document the ID of the customer, the date, the consumer technology, and whether the technology was adopted or disposed of in case of outdated technology, e.g. ('C-1234', '2023-06-03', 'BEV', 'adoption').

The **Rule-Based Technology Adoption Simulation** computation method works similarly to the previously described *Strategy-Based Technology Adoption Simulation* method. However, instead of following a pre-defined strategy for the assignment of technologies to customers, this method can be parametrized with assignment rules to define the circumstances that favor or hinder customers from adopting or disposing of certain technologies. Assignment rules are defined based on characteristics of customers. For example, customers in prosperous neighborhoods are currently more inclined to adopt a PV system. Compared to the *Strategy-Based Technology Adoption Simulation*, the *Rule-Based Technology Adoption Simulation* requires two additional inputs, i.e., the characteristics of customers and the assignment rules. The assignment rules are optional under the assumption that implementing decision support services use sensible default rules otherwise. Details regarding the associated data types *Customer Characteristics* and *Assignment Rules* are available in *FlexiEnergy* publications.

Energy Demand Updating

The *Energy Demand Updating* computation goal is the second part of the two-stage demand forecasting approach that calculates the effects of the previously simulated consumer technology adoptions on the energy demands of customers.

The **Peak-Based Energy Demand Updating** computation method directly updates the demands of customers based on the assigned consumer technologies. For this purpose, the computation method requires the current energy demands of customers, the computed technology assignments, and a mapping from each consumer technology and time step of the planning interval to the peak load (in kW) caused by the technology. The latter mapping is documented using the *Peak Technology Loads* data type, e.g., ('BEV', '2023-01-01', 11). The computation method outputs the updated (i.e., expected) energy demands of customers. Depending on whether a technology was adopted or disposed of at a given point in time, the peak energy demands of the affected technology are simply added or subtracted from the current energy demands of customers.

The **Profile-Based Energy Demand Updating** computation method works similarly to the *Peak-Load Energy Demand Updating* method. However, instead of working with the peak demand of each consumer technology, this method utilizes individual load profiles per technology and customer. As a result, the method supports a more fine-grained description of technology demands that may be influenced by *demand side management (DSM)* where the distribution network operator actively controls the energy demand of consumer technologies based on the available network capacity. The effects of DSM must be computed outside the *Profile-Based Energy Demand Updating* method. For example, the **Managed EV Charging Simulation** computation method was considered throughout the *FlexiEnergy* research project to simulate DSM influences on the charging behavior of electric vehicles. The method was intended to minimize network loads while achieving a certain level of battery charge for each vehicle. However, under the interface prescribed by the computation method, it is also possible to implement an algorithm that aims for a different optimization target under the same (or additional) constraints, e.g., to optimize electric vehicle charging such that consumer costs are minimized (cf. [TDA16]). While the latter optimization target is less useful for distribution network planners, it can benefit municipal energy suppliers in setting energy prices.

Peak Shaving

In addition to the previously discussed demand side management applied during energy demand updating, the *Peak Shaving* computation goal bundles additional approaches that can reduce peaks in customers' energy demands. The computation methods associated with

this goal can be fundamentally applied to each customer demand forecast, but require a high temporal resolution to compute reliable results.

The **Sector Coupling Offset** computation method reduces energy peaks in one energy sector by using sector coupling technologies to transform energy from one energy sector to another, e.g., via a Power-to-Gas or Power-to-Heat conversion. For this purpose, the method requires the energy demands of customers and the network topologies for all energy sectors as input and returns the attenuated energy demands as output. This peak-shaving alternative is only available if the distribution network operator has access to distribution networks for multiple energy sectors in the same region and sector coupling technology already exists.

The **Network Traffic Light Simulation** computation method is another peak-shaving approach that was discussed throughout the *FlexiEnergy* research project but was only partially conceptualized. The approach tries to influence customers' energy demands by using financial incentives based on the current network load (i.e., increase network fees when the load is high and reduce network fees if the load is low).

8.2.3 Descriptions of Decision Support Services

This section discusses exemplary functional and data decision support services in the context of the *FlexiEnergy* case study to demonstrate what and why different implementations for the previously described computation methods are available.

Functional Services

Many of the previously documented computation methods can be implemented by decision support services built on existing research or commercially available software. For example, the *Demand Extrapolation* computation method can essentially be implemented using any of the approaches summarized by Ghalekhondabi et al. [Gha+17]. Similarly, Theodoropoulos, Damousis, and Amditis [TDA16] summarize multiple approaches that can be used for the implementation of *Managed EV Charging Simulation*. Since an exhaustive list of potential decision support services provides limited benefits, the following explanations focus on examples that illustrate the motivations behind the development of multiple services for the implementation of the same computation method.

Effectiveness One decision support service may be more effective than another service. For example, for the *Strategy-Based Technology Adoption Simulation* computation method, one decision support service may be more effective in identifying network bottlenecks than another service when computing a worst-case geographic distribution of consumer technologies. Such a difference likely only becomes obvious when using the two services on the same data

and comparing results, or by relying on feedback provided for the services in the service registry. Alternatively, the effectiveness of a service can be derived from value assignments for method characteristics documented by the service. For example, approaches for *Demand Extrapolation* using a neural network have been trained on energy demand data of customers specific to a single country or region. Consequently, including the origin of training data in the description of decision support services indicates whether the service is applicable in another region based on the similarity of distribution networks between regions.

Efficiency One decision support service may be more efficient than another. For example, two implementations for the *Rule-Based Technology Adoption Simulation* computation method were developed throughout the *FlexiEnergy* research project. The first implementation does not use parallelization while the second does. The sequential implementation has a linear runtime in the number of customers to consider while the parallel implementation has a fraction of the runtime (depending on the degree of parallelization). The parallel implementation achieves the speedup by using multiple cores of a virtual machine, but additional parallelization can be achieved by hosting the service on a cluster of multiple virtual machines. However, benchmarks indicated that the coordination of parallelization adds some management overhead. Thus, there is an intersection point when plotting the runtime of the implementations concerning the number of customers to assign consumer technologies for. The sequential implementation is more efficient up to this intersection point, i.e., the number of customers to consider, afterwards the parallel implementation is more efficient. The intersection point can be used to define data assertions, i.e., that the sequential service should only be used up to a certain number of customers, and the parallel service should only be used after exceeding a certain number.

Other Qualities For reasons discussed in Section 4.4.1, automated functional decision support services are tightly coupled to the infrastructure they are executed on. Thus, it may also be desirable to host services on different infrastructure to improve service quality such as performance or availability (e.g., switch to a more powerful virtual machine with more cores to improve the performance of the parallel implementation of the *Rule-Based Technology Adoption Simulation* computation method). These quality advantages (and potentially also increased effectiveness and efficiency) are likely accompanied by increased usage fees. Consequently, alternative services also enable DSS engineers to allocate resources to the most important activities in the decision process.

Data Services

In the previously described context of forecasting the energy demands of customers, all data that is independent of a concrete distribution network is fundamentally suitable for the

establishment of a reusable data service. For example, throughout the *FlexiEnergy* research project, multiple publicly available studies were analyzed for expected future developments regarding overall energy demands and consumer technology characteristics. The gathered insights can be used to define one or multiple reusable datasets with scaling factors or market shares that can be used with the *Demand Scaling* and *Demand Simulation* computation methods, respectively. While this research work provides a starting ground for the decision support ecosystem, authors of (future) studies can directly make their insights available as a data service using the same format, while potentially charging an additional fee.

The aforementioned examples describe data services that are publicly accessible, either immediately or after paying a usage fee. Additionally, other data services are also fundamentally reusable, but cannot be publicly shared due to privacy concerns. For example, the historical energy demands of customers should be available to every distribution network planner within a single company, but not to planners outside the company. This can be achieved using the access controls associated with decision support services.

8.2.4 Relation to Requirements for Tailored DSS Development

This section relates the demonstrated solution design associated with the *Foundation* lifecycle phase to the requirements for tailored DSS development established in Section 3.1.

The decision support services documented at the service registry provide DSS engineers with alternative decision support functionality to select and combine into a tailored DSS to address the situational requirements for decision support of a decision maker. Thus, the *Foundation* lifecycle phase contributes towards requirement **R1 – Situativity** by providing the basis for service composition throughout the subsequent *Design* lifecycle phase. Furthermore, the documentation of decision making in an application domain created during the *Foundation* lifecycle phase provides a basis for decision makers to document their situational requirements for decision support, which is also integral to address **R1 – Situativity**. Additionally, the composability of computation methods and decision support services documented throughout the *Foundation* lifecycle phase addresses requirement **R2.1 – Modularity**.

The computation methods identified and described during the *Foundation* lifecycle phase define interfaces that can be implemented differently by individual decision support services, e.g., as indicated with the references to the literature for the *Demand Extrapolation* and *Managed EV Charging* computation methods during the description of the case study. Furthermore, the proposed HTTP-based data exchange between services enables each decision support service to be developed using arbitrary technologies on an arbitrary platform. This addresses requirement **R3.1 – Reusability** by supporting the reuse of existing implementations.

Tool support for the development of wrappers to make existing applications available with a different communication protocol has already been proposed by Wolters, Kirchhoff, and Engels [WKE20] and can also be applied in the DSE context to increase service variety.

Since the *Foundation* phase – like any DSE lifecycle phase – is executed continuously, the domain description and service descriptions can always be extended by domain experts and service providers using the associated content management system. Consequently, considering service advancements such as the parallelized implementation of the *Rule-Based Technology Adoption Simulation* method mentioned throughout the case study is not a challenge. This also applies to data types and data formats as well as computation goals and computation methods, thus addressing requirement **R3.2 – Extensibility**. The domain documentation and service registry also implement **R3.3 – Discoverability** for decision support functionality concerning interfaces (i.e., computation methods) and concrete implementations (i.e., decision support services). For example, a DSS engineer may not have known about all available approaches to forecasting the energy demands of customers without the corresponding documentation.

Since the service descriptions are later analyzed by the composition assistance to identify flaws in a PD-DSS process model, the *Foundation* phase also contributes towards addressing requirement **R4.3 – Error Prevention**. This was demonstrated in the FlexiEnergy case study with data assertions to document the tradeoff between parallelized and non-parallelized implementations of the *Rule-Based Technology Adoption Simulation* computation method, or the documentation of optimization characteristics for decision support services implementing the *Managed EV Charging Simulation* computation method to later prevent functional errors.

The domain documentation addresses requirement **R5.1 – Common Terminology** by assigning names to and defining the structure of data types and computation methods provided by DSE participants. The *Foundation* lifecycle phase also supports addressing requirement **R5.2 – Artifact Sharing** since it enables the advertisement of shared decision support services across organizational boundaries (subject to access restrictions). The *Foundation* lifecycle phase also contributes to addressing requirement **R5.3 – Experience Sharing** since the service registry supports reviews for decision support services based on feedback provided by decision makers and DSS engineers during the later *Improvement* lifecycle phase. The *Foundation* phase furthermore addresses requirement **R5.4 – Organizational Scalability** since the domain and service descriptions can be extended and added by any DSE participant.

The service descriptions contribute towards addressing requirement **R6.1 – Transparency** when decision makers want to obtain additional information about the modules that comprise their tailored PD-DSS. Lastly, the *Foundation* phase has the most impact on requirement **R7 – Domain-Portability** since it enables domain experts to capture decision-making characteristics of the application domain as a foundation for all other lifecycle phases.

8.3 Phase 2 – Design

This section describes the artifacts produced during the enactment of the DSE lifecycle phase *Design* in the context of the *FlexiEnergy* case study. During the initial *Requirements Documentation* lifecycle activity, decision makers use the *Requirements Elicitation* application to document their requirements for decision support. During the subsequent *Service Composition* activity, DSS engineers use the *PD-DSS Design* application to describe a tailored DSS as a BPMN process model. The requirements and PD-DSS process models are stored in the *PD-DSS Repository* for further processing in subsequent lifecycle phases.

This section discusses three PD-DSS process models and underlying situational requirements for decision support for cross-sector distribution network planning (Section 8.3.1), electricity distribution network planning with demand side management (Section 8.3.2), and meta decision making (Section 8.3.3). Based on the case study insights, the contributions of the *Design* lifecycle phase towards the requirements for tailored DSS development established in Section 3.1 are discussed (Section 8.3.4).

8.3.1 Cross-Sector Distribution Network Planning

The first case study example assumes that a distribution network planner works for a distribution network operator that manages distribution networks for each energy sector in a given region. The partial process model describing the composition of decision support services for forecasting customers' energy demands within the tailored PD-DSS is shown in Fig. 8.3.

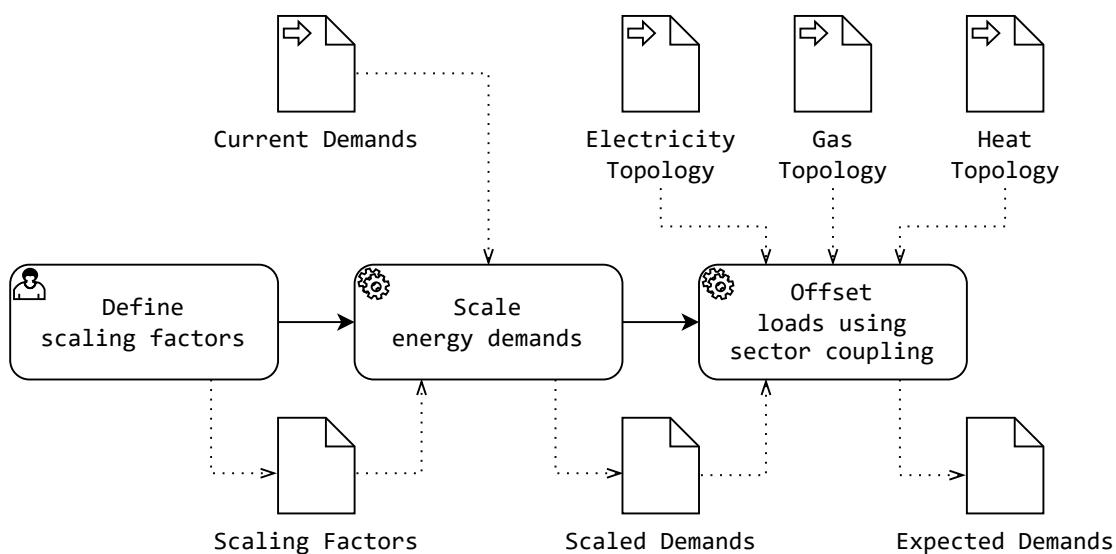


Figure 8.3: Process model excerpt for a PD-DSS supporting sector coupling

The region under consideration includes a large share of industrial customers and trade buildings. Therefore, the forecasting is built around the *Demand Scaling* computation method instead of *Technology Adoption Simulation* and *Energy Demand Updating*, as those methods are more suitable for regions with primarily residential customers. Since the distribution network planner does not have access to scaling factors (or does not want to use published ones), the scaling factors are defined during the “Define scaling factors” task instead of being provided as another data input to the process. In the third task, peaks in energy demands are attenuated by using the *Sector Coupling Offset* computation method. This method can be utilized by the distribution network planner since the distribution network operator manages the energy distribution networks for all energy sectors in the region.

8.3.2 Electricity Distribution Network Planning with DSM

The second example assumes that a distribution network operator manages only the electricity distribution network for a region, which is equipped with demand side management (DSM) for electric vehicle charging. The partial process model of the tailored PD-DSS for demand forecasting is shown in Fig. 8.4.

The region under consideration includes a large share of residential customers. As a result, demand forecasting is built around the combination of methods associated with the *Technology Adoption Simulation* and *Energy Demand Updating* computation goals. Since the distribution network planner can only utilize DSM to influence electric vehicle charging, the PD-DSS process model is split into two parallelly executed flows: The top flow computes the energy demands accrued by electric vehicles, and the bottom flow computes the energy demands accrued by other consumer technologies. Both flows utilize the *Rule-Based Technology Adoption Simulation* computation method (without the optional assignment rules) for the simulation of technology adoptions. The top flow subsequently uses the *Managed EV Charging Simulation* and *Profile-Based Energy Demand Updating* computation methods to account for DSM. The bottom flow instead uses the *Peak-Based Energy Demand Updating* computation method as it is more resource-efficient while computing results that are closely as effective as the *Profile-Based Energy Demand Updating* with default load profiles. After both flows have been executed, the computed energy demands for the assigned technologies are added to the current energy demands of customers to obtain their expected future energy demands.

8.3.3 DSM Benchmarking / Meta Decision Making

The previous two examples show the demand forecasting functionality of two tailored PD-DSS with the intent to further process the created demand forecast, e.g., to compute an investment

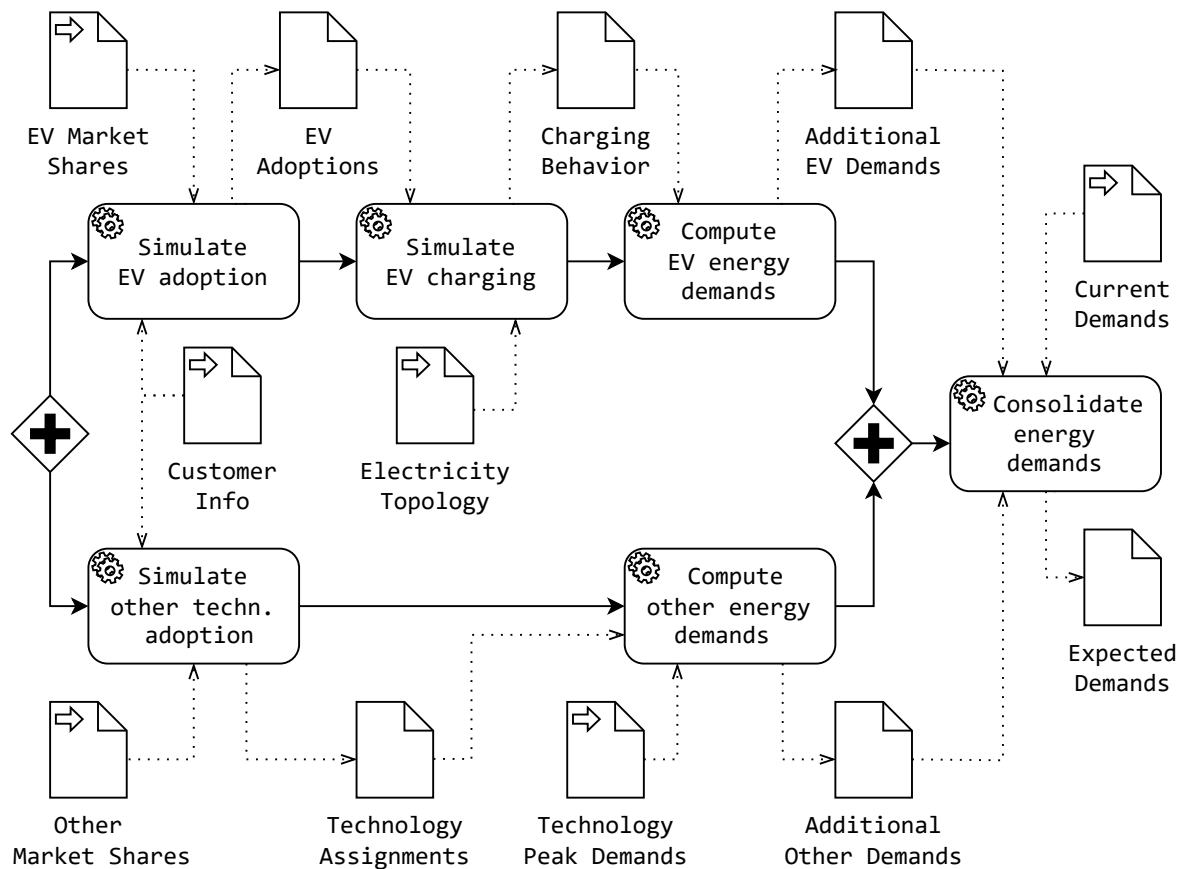


Figure 8.4: Process model excerpt of a PD-DSS supporting electricity networks with DSM

plan describing a replacement strategy for network assets given the expected energy demands of customers. The process model shown in Fig. 8.5 can be used for two different purposes, which are discussed after the fundamental explanation of the depicted process model.

The process model shown in Fig. 8.5 first computes a worst-case distribution of electric vehicles among an electricity distribution network using the *Strategy-Based Technology Adoption Simulation* computation method. Similar to the process model discussed in the previous subsection, the effects of the technology adoptions subsequently are once simulated using the *Profile-Based Energy Demand Updating* computation method to compute the demands of electric vehicle charging with demand side management (DSM), and once using the *Peak-Based Energy Demand Updating* computation method to compute the demands of electric vehicle charging without DSM. Afterwards, the computed demands are manually compared using an appropriate user interface.

The decision support provided by the presented process model can serve two purposes. First, the gathered insights on the effects of DSM during electric vehicle charging can support a decision maker in the decision of whether an energy distribution network should be upgraded

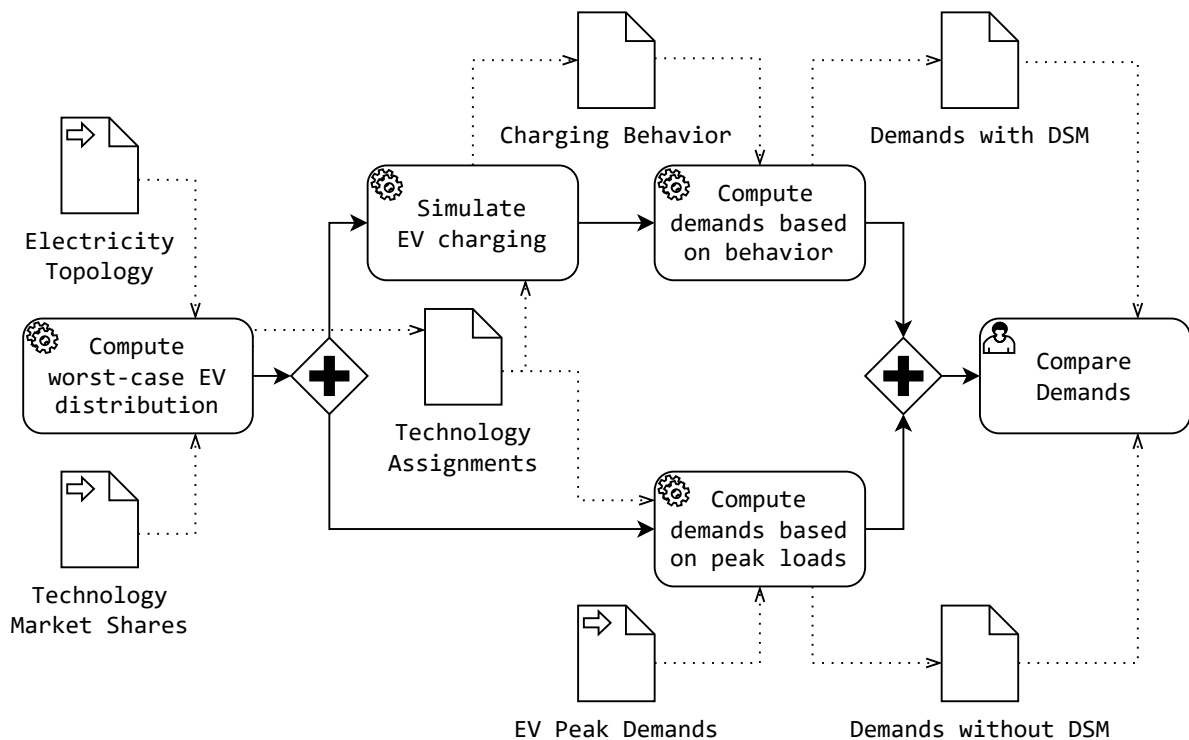


Figure 8.5: Process model excerpt for benchmarking the effects of DSM for vehicle charging

with DSM technology or not. Consequently, the process model demonstrates how the decision support functionality described in the previous Section 8.2 can be used for different decision problems (i.e., investment planning for traditional network assets such as transformers and cables, and DSM investment planning). Second, assuming a distribution network operator has already made the DSM investment, a DSS engineer can use the comparison to decide whether the added complexity, runtime, and other resource consumption required for DSM simulation is reasonable when doing long-term investment planning, which is usually based on a temporal resolution where DSM effects are not as obvious. Thus, the process model can be used for meta decision making, i.e., deciding on the constitution of the decision process that is supported by the tailored PD-DSS (cf. Section 2.2.4).

8.3.4 Relation to Requirements for Tailored DSS Development

This section relates the demonstrated solution design associated with the *Design* lifecycle phase to the requirements for tailored DSS development established in Section 3.1.

By enabling decision makers to document their situational requirements for decision support and by enabling DSS engineers to design a tailored DSS as a process-based composition of decision support services, the *Design* lifecycle phase addresses requirement *R1 – Situativity*.

For example, concerning **R1.1 – Situational Decision Problems**, decision makers can require and DSS engineers can select as well as compose computation methods and implementing decision support services with different optimization characteristics as well as different approaches to the computation of decision recommendations. In the previously described case study, different optimization characteristics can for example be defined for the *Managed EV Charging Simulation* computation method, which furthermore can only be used in the decision process if the distribution network operator can actively control the charging behavior of electric vehicles. Adapting the decision support to the resources available to the decision maker as required by **R1.2 – Situational Resources** can also be considered. This is for example demonstrated by the “Define scaling factors” task in Fig. 8.3 where the otherwise missing scaling factors required for the subsequent scaling of energy demands are created, or the alternative *FlexiEnergy* implementations of the *Rule-Based Technology Adoption Simulation* computation method that should be chosen based on the extent of the provided input data due to different runtime characteristics. The lack of scaling factors can also be seen as an example for **R1.3 – Situational Competences** if scaling factors are already published as data services, but the decision maker does not agree with the developments implied by the scaling factors.

Although requirement **R2.1 – Modularity** is primarily addressed by the *Foundation* lifecycle phase, the *Design* lifecycle phase nevertheless contributes to this requirement by ensuring the modular decision support services can be assembled into a holistic DSS. Similarly, the behavioral perspective of the process-based service composition representing a DSS enables the foundation for addressing requirement **R2.2 – Navigation**, although the implementation of this navigation is ultimately up to the subsequent *Enactment* lifecycle phase.

By supporting the design of service compositions that mimic an existing off-the-shelf DSS and by advertising these service compositions via the *PD-DSS Repository*, the *Design* lifecycle phase also contributes to addressing requirements **R3.1 – Reusability** and **R3.3 – Discoverability**. In particular, rebuilding existing off-the-shelf systems enables a gradual adoption of the DSE. The discovery of complete PD-DSS service compositions is also beneficial for benchmarking or meta decision making as discussed in Section 8.3.3.

The *Design* lifecycle phase is essential to addressing requirement **R4 – Suitability for Non-Programmers**. Concerning the associated requirement **R4.1 – Abstraction**, the designed process model describes the functional, behavioral, informational, and operational perspective of the PD-DSS using a visual notation. DSS engineers do not need to consider the technical implementation of these perspectives. For example in Fig. 8.4, parallel execution is achieved by simply adding a parallel gateway and the data exchange between decision support services is described by the data associations in the process model. BPMN, which is used to specify the process model representing a PD-DSS, was chosen during solution design because it is

widespread and usable without extensive upfront training, thus addressing requirement **R4.2 – Learnability**. The composition assistance integrated into the *PD-DSS Design* application can also support learnability by highlighting syntactical and semantic mistakes such that they can be avoided in the future. The composition assistance furthermore addresses requirement **R4.3 – Error Prevention** by reporting flaws in any of the process model perspectives. For example, the composition assistance can issue a warning if the parallel implementation of the *Rule-Based Technology Adoption Simulation* computation method is used, although the number of customers is below the intersection point where the performance of the decision support service with the parallel implementation outperforms the sequential implementation.

The *Design* lifecycle phase furthermore supports addressing requirement **R5.2 – Artifact Sharing** since the cross-organizational use of decision support artifacts simply requires the selection of decision support services from different service providers during the design of the PD-DSS process model. For example, a decision support service for the *Profile-Based Energy Demand Updating* computation method developed during the *FlexiEnergy* research project can be combined with previously published approaches for the *Managed EV Charging Simulation* computation method during the design of the PD-DSS. In addition to the experience-based suggestions of the composition assistance during the design of a PD-DSS, the sharing of designed (best-practice) PD-DSS process models via the *PD-DSS Repository* application also contributes to requirement **R5.3 – Experience Sharing**.

The previously discussed learnability of BPMN also enables decision makers to use the process model to understand the inner workings of the tailored PD-DSS. Thus, the PD-DSS process model works as a means of documentation, thereby addressing requirement **R6.1 – Transparency**. Contrary to some automated service composition approaches, the manual service composition by DSS engineers ensures the similarity of PD-DSS designs given the same and sufficiently precise documentation of a decision maker's situational requirements for decision support, thereby addressing requirement **R6.2 – Determinism**. The domain-agnostic nature of the *Design* lifecycle phase contributes towards **R7 – Domain-Portability**.

8.4 Phase 3 – Enactment

This section describes the artifacts produced during the application of the DSE lifecycle phase *Enactment* in the context of the *FlexiEnergy* case study. During the initial *DSS Generation* lifecycle activity, a designed PD-DSS process model is automatically transformed for usage with an existing BPMN engine. Since this activity happens transparent to DSE participants, this section focuses on the *DSS Usage* lifecycle activity where decision makers instantiate (Section 8.4.1) and interact (Section 8.4.2) with a PD-DSS using the *PD-DSS Enactment*

application. For conciseness, the case study description focuses on the enactment of the “Cross-Sector Distribution Network Planning” PD-DSS presented in Section 8.3.1. Based on the case study insights, the contributions of the *Enactment* phase towards the requirements for tailored DSS development established in Section 3.1 are discussed (Section 8.4.3).

8.4.1 PD-DSS Instantiation

As indicated in Fig. 8.2, the instantiation of a PD-DSS process model is triggered via the *PD-DSS Repository* application, which forwards the selected process model to the *PD-DSS Enactment* application. During the instantiation of the PD-DSS process model, the decision maker selects data for each data input provided to the decision process. In the “Cross-Sector Distribution Network Planning” example process introduced in Section 8.3.1, the inputs consist of *Current Demands*, *Electricity Topology*, *Gas Topology*, and *Heat Topology*.

For each process data input, the decision maker can either select a data service from the service registry or create a (temporary) data service by selecting a local file. In case the decision maker selects data that is incompatible with a decision support service selected for the implementation of a decision activity, a corresponding error is reported by the informational composition assistance. In the example shown in Fig. 8.6, the decision maker has chosen a private data service that holds the energy demand data of customers in a specific city. Since the chosen demand data has a yearly temporal resolution, the informational composition assistance reports an incompatibility to the daily resolution required by the decision support service selected for the implementation of the *Sector Coupling Offset* computation method associated with the “Offset demands using sector coupling” task. Once all data selection errors are addressed, the decision maker can trigger the enactment of the PD-DSS and potentially interact with it as demonstrated throughout the next subsection.

8.4.2 PD-DSS Usage

Figure 8.7 showcases the enactment of the “Define scaling factors” task as highlighted in the progress indicator on the left. Since the task is implemented by an integrated interactive decision support service and requires input from the decision maker, the right side of the user interface is defined by the implementing decision support service. Here, the decision maker can interact with the graphs to define the development of scaling factors over time for the different energy sectors. The chosen decision support service requires the decision maker to directly manipulate the development of the electricity, gas, and heat demand over time. A different decision support service could instead require the decision maker to define the expected demand developments of individual consumer technologies, which are then

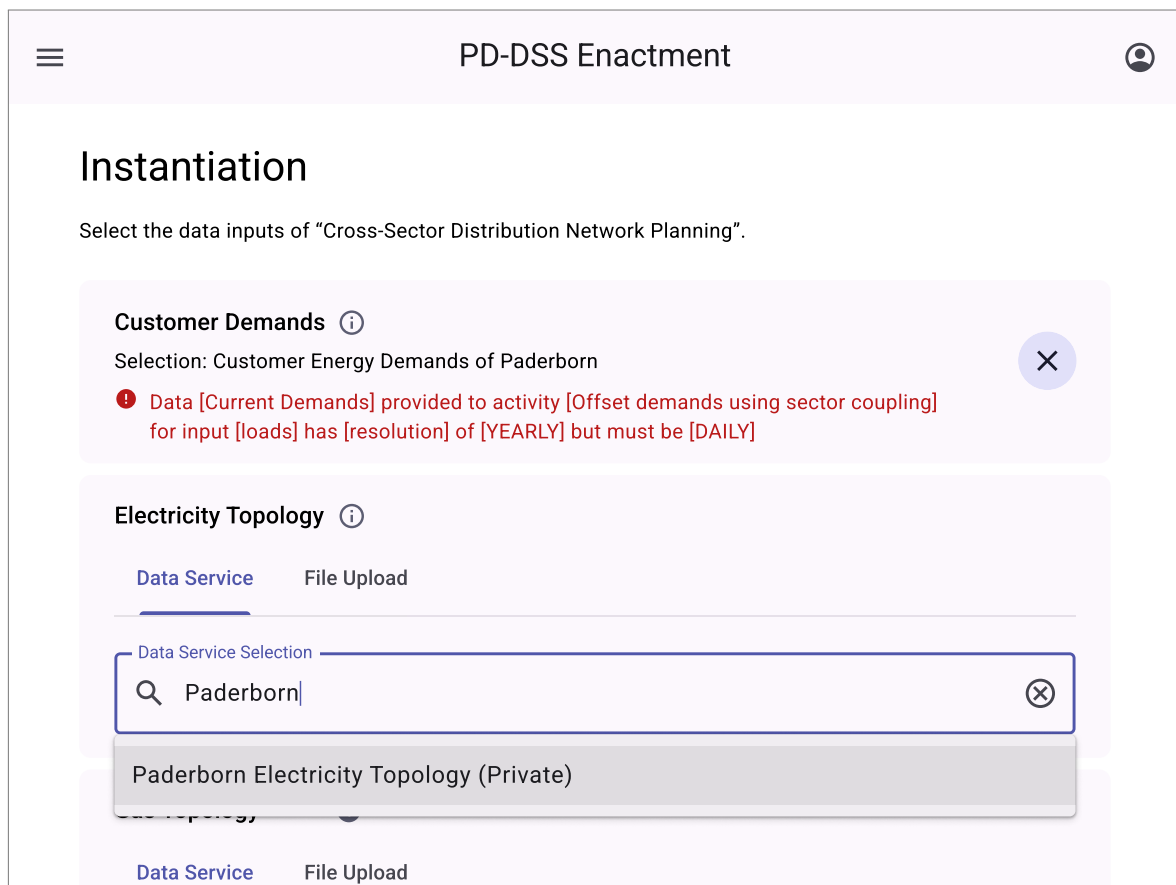


Figure 8.6: Mockup of PD-DSS instantiation in the case study context

internally added by the service to determine the scaling factor per energy sector. After the decision maker confirms the completion of scaling factor definition via the “Next” button, the scaling factors are transparently forwarded to the subsequent “Scale energy demands” task. Since this task and the final task are implemented using automated decision support services, no further interaction from the decision maker is required. For these tasks, the UI of the *PD-DSS Enactment* application merely acts as a progress indicator that shows the current state of process enactment. After the tasks have been executed, the overall process output (not depicted in the process model excerpt of Fig. 8.3) is stored at the *PD-DSS Repository*.

8.4.3 Relation to Requirements for Tailored DSS Development

This section relates the demonstrated solution design associated with the *Enactment* lifecycle phase to the requirements for tailored DSS development established in Section 3.1.

The *Enactment* lifecycle phase has limited contribution towards *RI – Situativity* as it only implements the situational design decisions made during the *Design* lifecycle phase. Never-

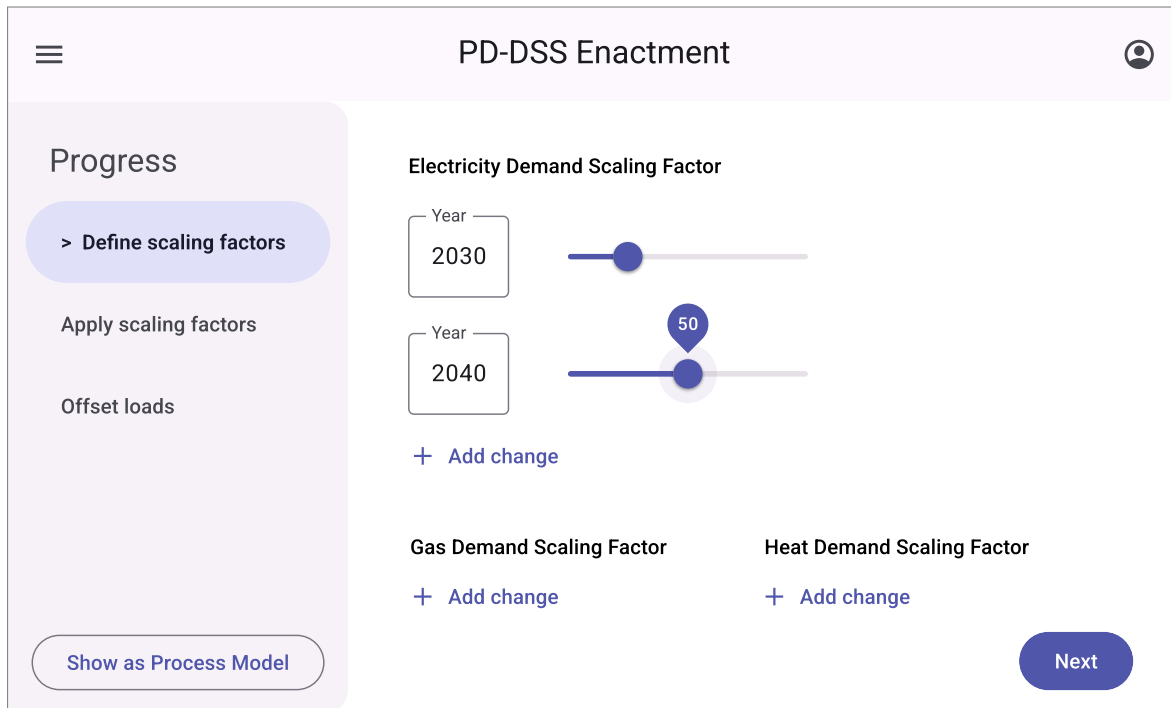


Figure 8.7: Mockup of interactive PD-DSS enactment in the case study context

theless, the *Enactment* phase contributes towards addressing requirement **R1.2 – Situational Resources** as the use of conditional sequence flows in the process model can result in the invocation of different decision support services during enactment based on the characteristics of data provided by the decision maker during instantiation or generated by preceding decision support services. For example, a process model can conditionally execute the sequential and parallel *FlexiEnergy* implementations of the *Rule-Based Technology Adoption Simulation* computation method depending on the number of customers connected to the distribution network that was selected by the decision maker during instantiation.

Similar to the *Design* lifecycle phase, the *Enactment* lifecycle phase supports addressing requirement **R2.1 – Modularity** by supporting the assembly of individual decision support artifacts into a holistic DSS. However, the focus of the *Enactment* phase lies on the implementation of requirement **R2.2 – Navigation** by invoking decision support services according to the (conditional) sequence flow defined during the *Design* phase. For the previously demonstrated example, the decision maker is prompted for input during the enactment of an interactive task, and the subsequent automated tasks are orchestrated by the *PD-DSS Enactment* application without requiring any further input from the decision maker. By embedding the user interfaces of manual tasks into the *PD-DSS Enactment* application and potentially interleaving them with automated tasks, the *Enactment* phase does not require decision makers to jump between

different applications, thus addressing requirement **R2.3 – Unified Execution Environment**.

The *Enactment* lifecycle phase also contributes towards addressing requirement **R4 – Suitability for Non-Programmers**. Without its transformation of a PD-DSS process model into an artifact suitable for the execution with an existing BPMN engine it would not be possible to visually model a tailored PD-DSS during the *Design* phase to address requirements **R4.1 – Abstraction** and **R4.2 – Learnability**. The integration of the informational composition assistance during the instantiation of a PD-DSS process model furthermore contributes towards addressing requirement **R4.3 – Error Prevention** as demonstrated by reporting the selection of an unsuitable data service in Fig. 8.6.

By orchestrating the invocation of functional decision support services across organizational boundaries during the enactment of the PD-DSS process model, the *Enactment* phase primarily addresses **R5.2 – Artifact Sharing** and contributes to **R5.4 – Organizational Scalability**.

Since the *PD-DSS Enactment* application informs decision makers about the status of process enactment as shown in Fig. 8.7, and optionally also about the underlying process model, the *Enactment* lifecycle phase contributes towards addressing requirement **R6.1 – Transparency**. The repeated instantiation of a PD-DSS process model to apply the same decision process to different data inputs furthermore contributes towards addressing requirement **R6.2 – Determinism**. Similar to the *Design* lifecycle phase, the domain-agnostic design of the *Enactment* lifecycle phase contributes to addressing requirement **R7 – Domain-Portability**.

8.5 Phase 4 – Improvement

This section describes the artifacts produced during the enactment of the DSE lifecycle phase *Improvement* in the context of the *FlexiEnergy* case study. During the initial *Feedback Documentation* lifecycle activity, decision makers and DSS engineers use the *PD-DSS Repository* and *Service Registry* applications to provide reviews for PD-DSS process models and decision support services (Section 8.5.1). During the subsequent *Knowledge Extraction* lifecycle activity, composition experts use the *Anti-Pattern Definition* application to capture composition knowledge based on the feedback of other DSE participants (Section 8.5.2). Based on the case study insights, the contributions of the *Improvement* phase towards the requirements for tailored DSS development established in Section 3.1 are discussed (Section 8.5.3).

8.5.1 Feedback: PD-DSS and Service Reviews

A decision maker primarily provides feedback for a PD-DSS if there is a discrepancy between the required and provided decision support functionality. If this discrepancy can be attributed

to an ambiguity in the requirements for decision support documented at the beginning of the *Design* lifecycle phase, the decision maker can usually update the requirements to be more specific. For example, the case-study description of the previous enactment phase discusses two alternative decision support services for the interactive definition of scaling factors, i.e., definition per energy sector or per consumer technology. If the decision maker had preferred the alternative decision support service for the implementation of the task, the decision maker can update the requirements to include the corresponding method characteristic. If no such method characteristic is available, this feedback can be added as a PD-DSS review via the *PD-DSS Repository* application. This also allows domain experts, who monitor the repository, to update the domain documentation accordingly. Furthermore, if no such decision support service has been implemented yet, service providers are alerted to the demand for such a service. Furthermore, the violation of promised service quality levels can be sanctioned by DSS engineers by leaving corresponding reviews for the service at the service registry.

Another example that demonstrates the usefulness of PD-DSS reviews outside of extensibility can be found in the context of the “benchmark” process model described in Section 8.3.3. Here, decision makers can point out that the benchmark is actually an apples-to-oranges comparison since DSM is not evaluated in isolation, but instead, a comparison between the profile-based and peak-based energy demand updating computation methods is performed. For a more expressive comparison, an activity for the *Profile-Based Energy Demand Updating* computation method should be added that uses default load profiles for electric vehicle charging. Furthermore, decision makers can use the feedback mechanism for benchmark processes to document the (anonymized) benchmark results for energy distribution networks under their management to provide composition experts with insights that supports the derivation of new composition knowledge as described in the next subsection.

8.5.2 Composition Knowledge: Anti-Patterns

The composition assistance largely relies on composition knowledge that can be derived from the artifacts created throughout the *Foundation* lifecycle phase. Nevertheless, the functional-behavioral features of the assistance also consider domain-specific composition knowledge that is contributed by composition experts in the form of anti-patterns. One source for these anti-patterns is the general domain expertise of composition experts. For example, the *Demand Scaling* computation method should not be executed after a *Peak Shaving* computation since the peak demand offsets are specific to the energy demands provided as an input and cannot be arbitrarily generalized, which is assumed by a subsequent scaling. This may result in the definition of an “Demand Scaling after Peak Shaving” anti-pattern.

The PD-DSS reviews of decision makers are another source for anti-pattern derivation. For example, for the “benchmarking” PD-DSS introduced in Section 8.3.3, composition experts can analyze the benchmark results reported by decision makers and may find that the use of demand side management during charging of electric vehicles does not yield a benefit proportionate to the (runtime) complexity added during the enactment of the decision process. Or a “Profile-Based Energy Demand Updating without DSM” anti-pattern may warn DSS engineers during the design of a PD-DSS process model that the use of the *Profile-Based Energy Demand Updating* computation method has no benefits without individual load profiles computed by a preceding *Managed EV Charging Simulation*.

8.5.3 Relation to Requirements for Tailored DSS Development

This section relates the demonstrated solution design associated with the *Improvement* lifecycle phase to the requirements for tailored DSS development established in Section 3.1.

As demonstrated in the context of the previously described case study with different decision support services for the implementation of the “Define scaling factors” task, the feedback provided by decision makers and DSS engineers can alert service providers to missing decision support services, or alert domain experts to a lack of precision in the domain documentation. Thus, the *Improvement* phase encourages the continuous development of novel decision support artifacts, thereby contributing to requirement **R3.2 – Extensibility**.

The comprehensive and traceable validation results reported by the composition assistance may prevent repeated errors and thereby improve the learnability of the modeling approach. Since the composition assistance also utilizes anti-patterns for validation, e.g., the “Demand Scaling after Peak Shaving” anti-pattern, the *Improvement* phase also contributes towards addressing requirement **R4.2 – Learnability**. The composition knowledge is also used by the composition assistance to implement requirement **R4.3 – Error Prevention**.

The *Improvement* lifecycle phase addresses requirement **R5.3 – Experience Sharing** by enabling composition experts to define anti-patterns that enforce best practices regarding specific partial decision support service compositions, e.g., disregarding demand side management during long-term energy demand forecasting. Best practices can be extracted from experiences documented by decision makers for PD-DSS process models at the *PD-DSS Repository* or by DSS engineers for individual decision support services at the *Service Registry*.

The best practices established throughout the *Improvement* lifecycle phase can also contribute towards addressing **R6.2 – Determinism** by ensuring similar compositions for the same requirements for decision support. However, the similarity between compositions ultimately depends on the choices made by a DSS engineer during service composition.

Lastly, the anti-patterns enable domain-specific composition assistance, and thus, the *Improvement* phase contributes towards addressing requirement **R7 – Domain-Portability**.

8.6 Threats to Validity

Each previous chapter already discussed threats that could potentially affect the validity of the proposed partial solution design and its evaluation if applicable. This section discusses cross-cutting threats to validity for the overall solution design and research approach.

Single Application Domain All examples used for (partial) requirement elicitation and solution design demonstration throughout the thesis are taken from the domain of energy distribution planning due to the author’s involvement in the *FlexiEnergy* research project. While the use of a single application domain establishes connections between solution requirements and solution design, it also results in a lack of heterogeneity that would be introduced by considering multiple application domains. This heterogeneity could potentially add additional diversification to requirements, and by extension, solution design, which is recommended considering requirement **R7 – Domain-Portability**. Thus, the focus on a single application domain potentially limits the *external validity* of the solution design, i.e., its generalizability (cf. [RH09]). However, the author’s anecdotal experience indicates that the proposed solution design should be applicable to decision support for business model development as well (cf. [GKE21]). Furthermore, many of the requirements for tailored DSS development can also be found in published literature and other domains as evident from the discussion of related work in Section 3.3, which suggests a sufficient variety.

Simulated Demonstration Although insights from the *FlexiEnergy* research project motivated this thesis and were used to demonstrate the proposed solution design, the research project ended before the solution concepts could be applied within the project. Thus, all applications of solution concepts in the domain of energy distribution network planning were simulated by the thesis author. In particular, for the coherent case-study evaluation presented in previous sections, the author assumed all roles of DSE participants, i.e., the role of a domain expert, service provider, DSS engineer, and composition expert. Different researchers and stakeholders without the *FlexiEnergy* experience or familiarity with BPMN could potentially arrive at different conclusions, which limits the *reliability* of the evaluation (cf. [RH09]).

Isolated Implementations Each partial solution design proposed in Part II of the thesis is accompanied by a prototypical implementation to fundamentally demonstrate the technical feasibility of the solution design. However, contrary to the DSE reference architecture proposed in Section 4.5, the implementations themselves are not integrated into a single prototypical

implementation for a holistic DSE since they only focus on technically challenging aspects of the solution design. Thus, a prototypical implementation for a holistic DSE to demonstrate the technical feasibility of the proposed reference architecture remains to be shown. Nevertheless, integrating the individual implementations should be straightforward due to the described contractual interfaces and user interface mockups presented throughout the thesis.

8.7 Key Takeaways

The solution design presented in Part II of the thesis can be applied to forecasting the energy demands of customers during (cross-sectoral) energy distribution network planning. This insight is based on a coherent (simulated) case study derived from practical experience gathered throughout the *FlexiEnergy* research project with industry partners. The case-study findings in combination with informed arguments indicate that the solution design addresses all requirements for tailored DSS development established in Section 3.1.

The contributions of each DSE lifecycle phase towards the requirements for tailored DSS development are visually summarized in Table 8.1. A filled circle symbolizes that a requirement is primarily addressed by the solution concepts associated with a lifecycle phase. A half-filled circle symbolizes that the solution design associated with a lifecycle phase supports the addressing of a requirement, but is not as pivotal as the lifecycle phase that addresses the requirement. An outlined circle is used when a phase does not contribute to addressing a requirement, or the contribution is too limited to warrant an explicit mention.

As evident from the tabular representation of contributions and the final “Coverage” column, each requirement for tailored DSS development is addressed by a phase of the DSE lifecycle. In fact, each requirement is usually implemented with contributions from multiple DSE lifecycle phases. The tabular summary also shows that the *Improvement* lifecycle phase has the least amount of contributions to tailored DSS development. While this does not make the phase redundant as it provides many contributions that enable the addressing of requirements in other lifecycle phases, the observation nevertheless suggests that the realization of the *Improvement* can be postponed when implementing a decision support ecosystem.

Although it can be concluded from the explanations of previous sections and the summary in Table 8.1 that all established requirements for tailored DSS development are addressed by the proposed DSE solution design, the discussed threats to validity as well as previous discussions of partial solution designs present some indicators for future work. These are discussed in the next chapter after a summary of thesis contributions with respect to the initially formulated research questions.

Requirement	Foundation	Design	Enactment	Improvement	Coverage
<i>R1 – Situativity</i>					
<i>R1.1 – Situational Decision Problems</i>	●	●	○	○	✓
<i>R1.2 – Situational Resources</i>	●	●	●	○	✓
<i>R1.3 – Situational Competences</i>	●	●	○	○	✓
<i>R2 – Process Orientation</i>					
<i>R2.1 – Modularity</i>	●	●	●	○	✓
<i>R2.2 – Navigation</i>	○	●	●	○	✓
<i>R2.3 – Unified Execution Environment</i>	○	○	●	○	✓
<i>R3 – Variety</i>					
<i>R3.1 – Reusability</i>	●	●	○	○	✓
<i>R3.2 – Extensibility</i>	●	○	○	●	✓
<i>R3.3 – Discoverability</i>	●	●	○	○	✓
<i>R4 – Suitability for Non-Programmers</i>					
<i>R4.1 – Abstraction</i>	○	●	●	○	✓
<i>R4.2 – Learnability</i>	○	●	●	●	✓
<i>R4.3 – Error Prevention</i>	●	●	●	●	✓
<i>R5 – Collaboration</i>					
<i>R5.1 – Common Terminology</i>	●	○	○	○	✓
<i>R5.2 – Artifact Sharing</i>	●	●	●	○	✓
<i>R5.3 – Experience Sharing</i>	●	●	○	●	✓
<i>R5.4 – Organizational Scalability</i>	●	○	●	○	✓
<i>R6 – Compliance</i>					
<i>R6.1 – Transparency</i>	●	●	●	○	✓
<i>R6.2 – Determinism</i>	○	●	●	●	✓
<i>R7 – Domain-Portability</i>					
	●	●	●	●	✓

●: phase addresses requirement, ●: phase supports addressing of requirement, ○: no contribution

Table 8.1: Contribution of DSE lifecycle phases towards requirements of Section 3.1

Summary and Future Work

This thesis introduces the concept of a decision support ecosystem as a collaborative low-code environment for the assisted process-driven development of tailored decision support systems to address the individual requirements for decision support of decision makers, thus enabling efficient and effective decision making. This chapter summarizes the contributions of the thesis with respect to the initially formulated research questions (Section 9.1) and presents suggestions for future work (Section 9.2).

9.1 Summary of Contributions

Decision making is among the most important but also among the most complex activities in a business. As a result, decision makers often rely on decision support systems to assist them in their decision-making process. However, to optimally assist an individual decision maker, the decision-making process supported by a DSS must align with the decision-making process followed by the decision maker. The limited customization capabilities of existing off-the-shelf decision support systems are often insufficient for an optimal alignment with a decision maker's individual decision-making process. This introduces the danger of recommending and implementing suboptimal decisions. After presenting this motivation as well as foundations and requirements for tailored DSS development in Chapters 1 to 3, this thesis describes multiple contributions to enable the development of tailored decision support systems that address the individual decision support requirements of each decision maker. The contributions are subsequently summarized for the research questions defined in Section 1.4.

Contribution 1: Concept of a Decision Support Ecosystem

Chapter 4 introduces the concept of a decision support ecosystem (DSE) to address the requirements for tailored DSS development previously established in Chapter 3. A DSE is a network of individuals and organizations providing technical and non-technical contributions for the efficient and effective co-development of tailored decision support systems via a shared digital platform. The DSE concept can be viewed as an extension of modern low-code development platforms with a specific focus on DSS development. DSS development in a DSE is based on the composition of reusable decision support services using model-driven development supported by an assistance system and domain knowledge. To support the implementation of a DSE, Chapter 4 describes a lifecycle for DSS development within a DSE, presents a reference architecture for the shared digital platform of a DSE, and describes the roles and responsibilities of DSE participants. A high-level overview of these contributions is given in Fig. 9.1. The visualization shows the four DSE lifecycle phases *Foundation*, *Design*, *Enactment*, and *Improvement*, the associated software applications that were conceptualized in subsequent chapters as a reference architecture for a DSE platform (cf. Fig. 8.2), and the roles that primarily use an application. Each software application and role is subsequently explained in the context of the other thesis contributions. Nevertheless, it can already be concluded that the DSE concept presented in Chapter 4 with its shared digital platform uniting heterogeneous DSE participants addresses research question RQ₄ on how the coordination and continuous exchange between stakeholders can be supported from a technical perspective.

Contribution 2: Decision Support Description Language

Chapter 5 introduces a description language to capture decision-making characteristics in a given application domain. The description language is initially used by domain experts via the *Domain Registry* application to document data characteristics and interfaces for decision support functionality with associated optimization characteristics. Based on the domain documentation, service providers create descriptions for their offered decision support services using the *Service Registry* application. Descriptions can be created for functional decision support services as well as data decision support services and include additional non-functional characteristics regarding service quality or resource consumption. Furthermore, each service description points to an internet address where the service is made available. While the service descriptions provided by service providers describe existing decision support artifacts, the description language can also be used by decision makers to describe desired decision support artifacts via the *Requirements Elicitation* application. Thus, the description language introduced in Chapter 5 answers RQ₁ regarding the description of decision support services.

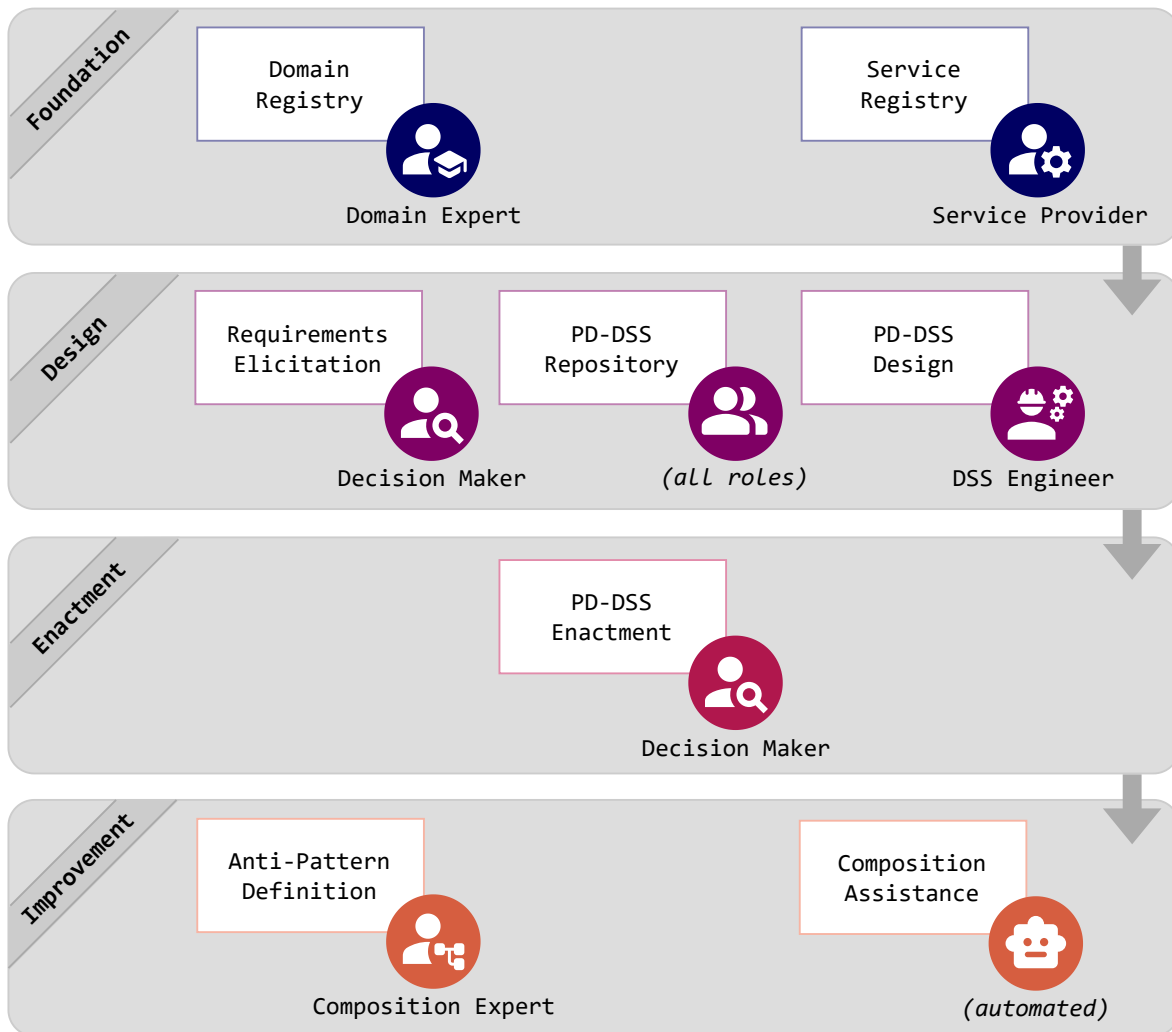


Figure 9.1: Overview of thesis contributions (cf. Fig. 8.2 for data exchange)

Contribution 3: Concept of a Process-Driven Decision Support System

Chapter 6 introduces the concept of a process-driven decision support system (PD-DSS), which corresponds to a holistic (tailored) DSS composed of decision support services to address the individual requirements for decision support documented by a decision maker. For PD-DSS development, a DSS engineer uses the *PD-DSS Design* application to create a BPMN process model documenting the activities within the assisted decision process and their (conditional) execution sequence, the decision support services used for the manual or automated realization of a process activity, and the data exchange between activities. The process model is subsequently forwarded to the *PD-DSS Enactment* application where the invocation of decision support services is coordinated using a BPMN engine. The

application prompts the decision maker for input data and interaction with interactive decision support services, and otherwise invokes automated decision support services with provided or generated data as prescribed by the process model. BPMN was selected as a notation due to being widespread and applicable without extensive upfront training. Consequently, the role of the DSS engineer does not require any programming skills and can essentially be assumed by any domain expert, including decision makers themselves. The PD-DSS concept introduced in Chapter 6 thus answers RQ₂ regarding a suitable modeling approach for the composition of a tailored DSS by non-programmers.

Contribution 4: Composition Assistance

Chapter 7 describes a composition assistance that supports DSS engineers during the creation of a PD-DSS by automatically validating the alignment of the designed PD-DSS process model with the requirements for decision support of a decision maker, decision-making best practices, and modeling conventions. Errors are detected in real-time and reported to DSS engineers in the *PD-DSS Design* application so that they can be addressed immediately during process model design. The assistance detects flaws in the selection of activities or implementing decision support services, the order of their execution, and the data exchange between them. While most of the composition knowledge applied by the composition assistance can be derived from the description of the application domain and its decision support services, composition experts can also provide domain-specific composition knowledge in the form of anti-patterns using the *Anti-Pattern Definition* application. Composition experts themselves obtain the composition knowledge from feedback provided by decision makers for a PD-DSS at the *PD-DSS Repository* application, or feedback provided by DSS engineers for individual decision support services at the *Service Registry*. Thus, the composition assistance described in Chapter 7 answers RQ₃ regarding the assistance of non-programmers during the composition of decision support services into a tailored DSS.

Demonstration of Applicability and Technical Feasibility

Following a design science research approach, the applicability of each previous contribution was demonstrated and discussed in the context of energy distribution network planning, using isolated examples as well as a coherent case study based on insights gathered during a research project with industry partners. The insights from the case study demonstration in Chapter 8 support the implementation of the DSE concept in other application domains. In addition to applicability, the technical feasibility of the contributions was fundamentally demonstrated with the prototypical implementation provided for each contribution. Thus, by aggregating

contributions 1 to 4 and considering their applicability as well as technical feasibility, the thesis consequently answers the initially formulated overall research question RQ_O regarding the feasibility of an assisted model-driven approach to DSS development by decision makers and other non-programmers using service orientation in a multi-stakeholder context.

9.2 Future Work

The research presented in this thesis is complete and self-contained as it answers the initially formulated research questions, and the proposed solution design addresses the established requirements for tailored DSS development. Nevertheless, some minor improvements were already presented throughout the discussion of the solution design that can be implemented throughout subsequent cycles of design science research. The solution design also enables additional research questions as a basis for future research projects, potentially using a different research approach. This section briefly describes potential future research questions.

Practical Assessment of Generalizability

The discussion in Section 8.6 presented the focus on a single application domain and a simulated case study as a potential threat to the generalizability of the solution design and evaluation results. Consequently, the research question “*Are there domain characteristics that limit the generalization of the solution approach?*” should be evaluated through multiple real-world field studies in other application domains. In particular, the application of the solution design in a big data could reveal additional requirements that result in extensions to the solution design. Additional requirements likely can be derived by translating the approach to automated decision making where decisions are automatically implemented without any further input from decision makers. The prototypical implementation of the DSE concept should also be further advanced to answer the research question “*How do DSE participants assess the usability and learnability of the approach?*” in the context of the field studies.

Improving the Efficiency of Manual Service Composition

Since the process-based composition of decision support services does not require programming skills, the DSE concept increases efficiency during tailored DSS development by removing the dependency on trained software developers. Nevertheless, the efficiency during service composition could be further increased, for example, by allowing the composition assistance to suggest modifications or actively edit the composition of decision support services implementing a PD-DSS. In this context, the fundamental approaches discussed in Chapter 7

provide a starting ground to answer the research question “*How can a recommendation-based composition assistance be implemented to reduce the manual work performed by DSS engineers?*”. Furthermore, as demonstrated in the context of low-code development in general, domain experts may still require some additional procedural guidance to ensure the effective and efficient development of software applications in the form of a software development method [Kir+22]. This implies the research question “*Does a development method as an extension of the DSE lifecycle provide a procedural framework that further increases the efficiency and effectiveness of DSS engineers during service composition?*”.

Automated Service Composition with Dialog-Based Artificial Intelligence

Another approach to potentially improve the efficiency during DSS development is to avoid the role of the DSS engineer completely. This implies an automated system that composes decision support services according to the requirements for decision support specified by a decision maker. The existing approaches using automated service composition discussed in Chapter 3 have severe limitations with respect to the established requirements for tailored DSS development. Most importantly, it is hard to implement multiple iterations in which the system improves the suggested composition based on feedback provided by the decision maker. However, recent innovations in the domain of large language models (LLMs) show that this form of artificial intelligence (AI) is capable of engaging in a stateful dialog to reason about generated artifacts and to adapt them based on human feedback. This is for example demonstrated in academic work by Austin et al. [Aus+21], or more recently on a commercial level by services such as *ChatGPT*¹. As discussed in Section 7.1, the prediction-based approach of LLMs aligns with the sequential creation of process models. Thus, decision makers could initially specify their requirements for decision support in natural language. The AI could clarify ambiguities in the requirements and propose an initial service composition. The proposed service composition can then be improved based on natural language feedback provided by the decision maker. This concept results in the research question “*How can a stateful, dialog-based composition of decision support services using natural language be implemented using artificial intelligence?*”.

DSE Governance and Health

After the fundamental derivation of the decision support ecosystem concept, the thesis focused on advancing the (architectural) design of the shared DSE platform as the primary enabler of a DSE. However, the long-term success of a DSE not only depends on its technical capabilities

¹ ChatGPT website: <https://openai.com/blog/chatgpt>

but also on the quantity of DSE participants and the quality of their contributions. The value of a DSE can be expected to increase exponentially with more users contributing (compositions of) decision support services and domain expertise (cf. [JC12]). Maintaining *ecosystem health*, i.e., “its overall performance and sustainable well-functioning” [SE20], requires governance tools for DSE participants and the DSE platform provider in particular [JC12; SE20]. These governance tools include “procedures and processes by which a company controls, changes or maintains its current and future position” in the ecosystem [BJ12]. In particular, future work may consider the research questions “*What metrics characterize the health of a decision support ecosystem?*” and “*What governance tools are required to ensure the health of a decision support ecosystem?*”. The answers to both research questions likely can build upon the existing research for maintaining the health of digital business ecosystems, in particular software ecosystems and data ecosystems. In this context, it may also be worthwhile to consider the research question “*How can a decision support ecosystem be established?*”, referring to organizational changes or financial incentives that are required to transition from the current status quo with off-the-shelf decision support systems to a DSE for tailored DSS development. A starting ground for answering this research question is the work by Bosch [Bos09] in the context of software ecosystems.

References

- [AAE16] Nuha Alshuqayran, Nour Ali, and Roger Evans. “A Systematic Mapping Study in Microservice Architecture”. In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.
- [ADL10] Ahmed Awad, Gero Decker, and Niels Lohmann. “Diagnosing and Repairing Data Anomalies in Process Models”. In: *Business Process Management Workshops*. LNBIP 43. Springer, Berlin, Heidelberg, 2010, pp. 5–16.
- [AK19] Ali A. Alwasouf and Deepak Kumar. “Research Challenges of Web Service Composition”. In: *Software Engineering. Advances in Intelligent Systems and Computing 731*. Springer, Singapore, 2019, pp. 681–689.
- [Al +21] Md Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Bin Haider, and Anindya Iqbal. “An Empirical Study of Developer Discussions on Low-Code Software Development Challenges”. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 46–57.
- [All+14] J. M. Allwood, V. Bosetti, N. K. Dubash, L. Gómez-Echeverri, and C. von Stechow. “Glossary”. In: *Climate Change 2014: Mitigation of Climate Change*. Cambridge University Press, 2014.
- [Ami18] Mohammad Javad Amiri. “Object-Aware Identification of Microservices”. In: *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 2018, pp. 253–256.
- [AS06] Faisal Al-Qaed and Alistair Sutcliffe. “Adaptive Decision Support System (ADSS) for B2C E-Commerce”. In: *Proceedings of the Eighth International Conference on Electronic Commerce*. ACM, 2006, p. 492.
- [ASL18] Ali Arefi, Farhad Shahnian, and Gerard Ledwich. *Electric Distribution Network Management and Control*. Power Systems. Springer, Singapore, 2018.

- [ASP10] Daud M. Ahmed, David Sundaram, and Selwyn Piramuthu. “Knowledge-based scenario management — Process and support”. In: *Decision Support Systems* 49.4 (2010), pp. 507–520.
- [Aus+21] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Le Quoc, and Charles Sutton. *Program Synthesis with Large Language Models*. 2021. URL: <https://arxiv.org/pdf/2108.07732> (visited on June 18, 2023).
- [Awa+09] Ahmed Awad, Alexander Grosskopf, Andreas Meyer, and Mathias Weske. *Enabling resource assignment constraints in BPMN*. 2009. URL: https://www.researchgate.net/publication/228594159_Enabling_Resource_Assignment_Constraints_in_BPMN (visited on May 6, 2023).
- [Awa07] Ahmed Awad. “BPMN-Q: A Language to Query Business Processes”. In: *Enterprise modelling and information systems architectures – concepts and applications*. Gesellschaft für Informatik e. V., 2007, pp. 115–128.
- [Axe+17] Jakob Axelsson, Ulrik Franke, Jan Carlson, Severine Sentilles, and Antonio Cicchetti. “Towards the architecture of a decision support ecosystem for system component selection”. In: *2017 Annual IEEE International Systems Conference (SysCon)*. IEEE, 2017, pp. 1–7.
- [Bec+08] Christoph Becker, Hannes Kulovits, Andreas Rauber, and Hans Hofman. “Plato: A Service Oriented Decision Support System for Preservation Planning”. In: *JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2008, pp. 367–370.
- [Bec+10] Joerg Becker, P. Bergener, Michael Räckers, Burkhard Weiß, and Axel Winkelmann. “Pattern-Based Semi-Automatic Analysis of Weaknesses in Semantic Business Process Models in the Banking Sector”. In: *ECIS 2010 Proceedings* (2010).
- [Bec+12] Joerg Becker, Philipp Bergener, Dominic Breuker, and Michael Raeckers. “An Empirical Assessment of the Usefulness of Weakness Patterns in Business Process Redesign”. In: *ECIS 2012 Proceedings* (2012).
- [Ben+18] Saloua Bennani, Mahmoud El Hamlaoui, Mahmoud Nassar, Sophie Ebersold, and Bernard Coulette. “Collaborative model-based matching of heterogeneous models”. In: *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design*. IEEE, 2018, pp. 443–448.

- [Ber+15] Philipp Bergener, Patrick Delfmann, Burkhard Weiss, and Axel Winkelmann. “Detecting potential weaknesses in business processes”. In: *Business Process Management Journal* 21.1 (2015), pp. 25–54.
- [Bey+16] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering*. 1st edition. O’Reilly, 2016.
- [BF21] Alexander C. Bock and Ulrich Frank. “Low-Code Platform”. In: *Business & Information Systems Engineering* 63.6 (2021), pp. 733–740.
- [BJ12] Alfred Baars and Slinger Jansen. “A Framework for Software Ecosystem Governance”. In: LNBIP 114. Springer, Berlin, Heidelberg, 2012, pp. 168–180.
- [BL14] Nathan Bennett and G. James Lemoine. “What a difference a word makes: Understanding threats to performance in a VUCA world”. In: *Business Horizons* 57.3 (2014), pp. 311–317.
- [BL17] Redouane Blal and Abderrahmane Leshob. “A Model-Driven Service Specification Approach from BPMN Models”. In: *14th IEEE International Conference on E-Business Engineering*. IEEE, 2017, pp. 126–133.
- [BL18] Eric J. Bolland and Carlos J. Lopes. *Decision Making and Business Performance*. New Horizons in Management. Edward Elgar Pub., Inc, 2018.
- [Bla+18] Redouane Blal, Abderrahmane Leshob, Javier Gonzalez-Huerta, Hafedh Mili, and Anis Boubaker. “From inter-organizational business process models to service-oriented architecture models”. In: *Service Oriented Computing and Applications* 12 (2018), pp. 227–245.
- [Blo14] Eva Blomqvist. “The Use of Semantic Web Technologies for Decision Support – A Survey”. In: *Semantic Web* 5.3 (2014), pp. 177–201.
- [BM21] Travis Breaux and Jennifer Moritz. “The 2021 Software Developer Shortage is Coming”. In: *Commun. ACM* 64.7 (2021), pp. 39–41.
- [Bor+13] Diana Borrego, Rik Eshuis, María Teresa Gómez-López, and Rafael M. Gasca. “Diagnosing correctness of semantic workflow models”. In: *Data & Knowledge Engineering* 87 (2013), pp. 167–184.
- [Bos09] Jan Bosch. “From Software Product Lines to Software Ecosystems”. In: *Proceedings of the 13th International Software Product Line Conference*. SPLC ’09. Carnegie Mellon University, 2009, pp. 111–119.

- [Bou+15] Ricarda B. Bouncken, Johanna Gast, Sascha Kraus, and Marcel Bogers. “Coope-
tition: a systematic review, synthesis, and future research directions”. In: *Review
of Managerial Science* 9.3 (2015), pp. 577–601.
- [BS23] Sascha Christian Burmeister and Guido Schryen. “Distribution network opti-
mization: predicting computation times to design scenario analysis for network
operators”. In: *Energy Systems* (2023), pp. 1–28.
- [BSH99] H. K. Bhargava, S. Sridhar, and C. Herrick. “Beyond spreadsheets: tools for
building decision support systems”. In: *Computer* 32.3 (1999), pp. 31–39.
- [BW08] Thadthong Bhrammanee and Vilas Wuwongse. “ODDM: A framework for
modelbases”. In: *Decision Support Systems* 44.3 (2008), pp. 689–709.
- [BWR11] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. “Requirements are
slipping through the gaps — A case study on causes & effects of communication
gaps in large-scale software development”. In: *2011 IEEE 19th International
Requirements Engineering Conference*. IEEE, 2011, pp. 37–46.
- [BWW11] Jörg Becker, Burkhard Weiß, and Axel Winkelmann. “Automatic Identification
of Structural Process Weaknesses – Experiences with Semantic Business Process
Modeling in the Financial Sector”. In: *Wirtschaftsinformatik Proceedings 2011*
(2011), pp. 15–26.
- [Cab20] Jordi Cabot. “Positioning of the low-code movement within the field of model-
driven engineering”. In: *Proceedings of the 23rd ACM/IEEE International
Conference on Model Driven Engineering Languages and Systems: Companion
Proceedings*. ACM Digital Library. ACM, 2020, pp. 1–3.
- [Cal+98] Ma. Laura Caliusco, Pablo Villarreal, Alejandro Toffolo, Ma Laura Taverna,
and Omar Chiotti. “Decision support systems generator for industrial companies
Module IV: Forecasting support system”. In: *Computers & Industrial Engineering*
35.1-2 (1998), pp. 315–318.
- [Cam22] Camunda. *Variables — Camunda Platform 8 Docs*. 2022. URL: [https://
docs.camunda.io/docs/8.1/components/concepts/variables/
#inputoutput-variable-mappings](https://docs.camunda.io/docs/8.1/components/concepts/variables/#inputoutput-variable-mappings) (visited on July 2, 2023).
- [Cas+98] Griselda Castro, Leonardo Gregoret, Walter Delfabro, and Omar Chiotti. “De-
cision support systems generator for industrial companies Module II: System
to support the material management”. In: *Computers & Industrial Engineering*
35.1-2 (1998), pp. 307–310.

- [CD20] Massimiliano Caramia and Paolo Dell’Olmo. “Multi-objective Optimization”. In: *Multi-objective Management in Freight Logistics*. 2nd edition. Springer, Cham, 2020, pp. 21–51.
- [CDT17] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. “Contextual understanding of microservice architecture”. In: *ACM SIGAPP Applied Computing Review* 17.4 (2017), pp. 29–45.
- [CL10] Junyi Chai and James N.K. Liu. “An ontology-driven framework for supporting complex decision process”. In: *2010 World Automation Congress*. 2010, pp. 1–6.
- [Cor80] Alexander H. Cornell. *The Decision-maker’s Handbook*. Prentice-Hall, 1980.
- [CY98] Ta-Tao Chuang and Surya B. Yadav. “The development of an adaptive decision support system”. In: *Decision Support Systems* 24.2 (1998), pp. 73–87.
- [CZH02] Xiao-hong Chen, Yan-ju Zhou, and Dong-bin Hu. “A problem solving framework for group decision support system”. In: *Journal of Central South University of Technology* 9.4 (2002), pp. 279–284.
- [DD13a] Dursun Delen and Haluk Demirkan. “Data, information and analytics as services”. In: *Decision Support Systems* 55.1 (2013), pp. 359–363.
- [DD13b] Haluk Demirkan and Dursun Delen. “Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud”. In: *Decision Support Systems* 55.1 (2013), pp. 412–421.
- [DE13] Amit V. Deokar and Omar F. El-Gayar. “On semantic annotation of decision models”. In: *Information Systems and e-Business Management* 11.1 (2013), pp. 93–117.
- [Del+15] Patrick Delfmann, Matthias Steinhorst, Hanns-Alexander Dietrich, and Jörg Becker. “The generic model query language GMQL – Conceptual specification, implementation, and runtime evaluation”. In: *Information Systems* 47 (2015), pp. 129–177.
- [Dem21] David J. Deming. *The Growing Importance of Decision-Making on the Job: Working Paper*. 2021. URL: <http://www.nber.org/papers/w28733> (visited on September 22, 2022).
- [DH12] Markus Döhring and Steffen Heublein. “Anomalies in Rule-Adapted Workflows - A Taxonomy and Solutions for vBPMN”. In: *2012 16th European Conference on Software Maintenance and Reengineering (CSMR 2012)*. IEEE, 2012, pp. 117–126.

- [DH15] Patrick Delfmann and Michael Hübers. “Towards Supporting Business Process Compliance Checking with Compliance Pattern Catalogues - A Financial Industry Case Study”. In: *Enterprise Modelling and Information Systems Architectures* 10.1 (2015), pp. 67–88.
- [Di +22] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. “Low-code development and model-driven engineering: Two sides of the same coin?” In: *Software and Systems Modeling* 21.2 (2022), pp. 437–446.
- [DL01] C.-S. J. Dong and G. S. L. Loo. “Flexible web-based decision support system generator (FWDSSG) utilising software agents”. In: *12th International Workshop on Database and Expert Systems Applications*. 2001, pp. 892–897.
- [DN03] Ulrich Derigs and Nils-H. Nickel. “Meta-heuristic based decision support for portfolio optimization with a case study on tracking error minimization in passive portfolio management”. In: *OR Spectrum* 25.3 (2003), pp. 345–378.
- [dOli+21] Ruan de Oliveira, Tiago Massoni, Narallynne Araújo, Camila Sarmiento, and Francielle Santos. “Ants Doing Legwork: Investigating Motivators for Software Development Career Abandonment”. In: *Proceedings of the XXXV Brazilian Symposium on Software Engineering*. SBES '21. ACM, 2021, pp. 353–362.
- [Dra+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. “Microservices: Yesterday, Today, and Tomorrow”. In: *Present and Ulterior Software Engineering*. Springer, Cham, 2017, pp. 195–216.
- [DS13] Ching-Shen James Dong and Ananth Srinivasan. “Agent-enabled service-oriented decision support systems”. In: *Decision Support Systems* 55.1 (2013), pp. 364–373.
- [DS96] James W. Dean and Mark P. Sharfman. “Does Decision Process Matter? A Study of Strategic Decision-Making Effectiveness”. In: *Academy of Management Journal* 39.2 (1996), pp. 368–392.
- [dSJW19] Aaron de Smet, Gregor Jost, and Leigh Weiss. *Three keys to faster, better decisions*. 2019. URL: <https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/three-keys-to-faster-better-decisions> (visited on October 6, 2022).

- [EC07] Said Elbanna and John Child. “Influences on strategic decision effectiveness: Development and test of an integrative model”. In: *Strategic Management Journal* 28.4 (2007), pp. 431–453.
- [Ecm17] Ecma International. *ECMA-404 - The JSON Data Interchange Syntax*. 2017. URL: <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>.
- [ED13] Omar El-Gayar and Amit Deokar. “A semantic service-oriented architecture for distributed model management systems”. In: *Decision Support Systems* 55.1 (2013), pp. 374–384.
- [EEB16] Iliada Eleftheriou, Suzanne M. Embury, and Andrew Brass. “Data Journey Modelling: Predicting Risk for IT Developments”. In: *The practice of enterprise modeling*. LNBIP 267. Springer, Cham, 2016, pp. 72–86.
- [EY19] Ali Ehsan and Qiang Yang. “State-of-the-art techniques for modelling of uncertainties in active distribution network planning: A review”. In: *Applied Energy* 239 (2019), pp. 1509–1523.
- [FE16] Masud Fazal-Baqaie and Gregor Engels. “Software Processes Management by Method Engineering with MESP”. In: *Managing Software Process Evolution*. Springer, Cham, 2016, pp. 185–209.
- [Fie00] Roy T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis. University of California, 2000.
- [Fie99] Cristina Fierbinteanu. “A decision support systems generator for transportation demand forecasting implemented by constraint logic programming”. In: *Decision Support Systems* 26.3 (1999), pp. 179–194.
- [FJF21] Siamak Farshidi, Slinger Jansen, and Sven Fortuin. “Model-driven development platform selection: four industry case studies”. In: *Software and Systems Modeling* 20.5 (2021), pp. 1525–1551.
- [För+07] Alexander Förster, Gregor Engels, Tim Schattkowsky, and Ragnhild van der Straeten. “Verification of Business Process Quality Constraints Based on Visual Process Patterns”. In: *TASE 2007*. IEEE, 2007, pp. 197–208.
- [FPV97] Bijan Fazlollahi, Mihir A. Parikh, and Sameer Verma. “Adaptive decision support systems”. In: *Decision Support Systems* 20.4 (1997), pp. 297–315.

- [Gar+16] Martin Garriga, Cristian Mateos, Andres Flores, Alejandra Cechich, and Alejandro Zunino. “RESTful service composition at a glance: A survey”. In: *Journal of Network and Computer Applications* 60 (2016), pp. 32–53.
- [Geo92] Arthur M. Geoffrion. “The SML Language for Structured Modeling: Levels 1 and 2”. In: *Operations Research* 40.1 (1992), pp. 38–57.
- [Gha+17] Iman Ghalekhondabi, Ehsan Ardjmand, Gary R. Weckman, and William A. Young. “An overview of energy demand forecasting methods published in 2005–2015”. In: *Energy Systems* 8.2 (2017), pp. 411–447.
- [GKE21] Sebastian Gottschalk, Jonas Kirchhoff, and Gregor Engels. “Extending Business Model Development Tools with Consolidated Expert Knowledge”. In: *Business Modeling and Software Design*. LNBIP 422. Springer International Publishing, 2021, pp. 3–21.
- [GL07] Volker Gruhn and Ralf Laue. *Good and bad excuses for unstructured business process models*. 2007. URL: https://www.researchgate.net/publication/221034631_Good_and_Bad_Excuses_for_Unstructured_Business_Process_Models (visited on May 16, 2023).
- [Got+23] Sebastian Gottschalk, Enes Yigitbas, Alexander Nowosad, and Gregor Engels. “Continuous situation-specific development of business models: knowledge provision, method composition, and method enactment”. In: *Software and Systems Modeling* 22.1 (2023), pp. 47–73.
- [Got22] Sebastian Gottschalk. “Situation-specific development of business models within software ecosystems”. PhD thesis. 2022.
- [Gra87] Paul Gray. “Group decision support systems”. In: *Decision Support Systems* 3.3 (1987), pp. 233–242.
- [GRR17] Antonio Manuel Gutiérrez–Fernández, Manuel Resinas, and Antonio Ruiz–Cortés. “Redefining a Process Engine as a Microservice Platform”. In: *Business Process Management Workshops*. LNBIP 281. Springer, Cham, 2017, pp. 252–263.
- [Har96] E. Frank Harrison. “A process perspective on strategic decision making”. In: *Management Decision* 34.1 (1996), pp. 46–53.
- [HB09] Markus Held and Wolfgang Blochinger. “Structured collaborative workflow design”. In: *Future Generation Computer Systems* 25.6 (2009), pp. 638–653.

- [HD15] Steffen Höhenberger and Patrick Delfmann. “Supporting Business Process Improvement through Business Process Weakness Pattern Collections”. In: *Wirtschaftsinformatik Proceedings 2015* (2015).
- [Hev+04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. “Design Science in Information Systems Research”. In: *MIS Quarterly* 28.1 (2004), p. 75.
- [Hof08] Dirk W. Hoffmann. *Software-Qualität*. Springer Berlin Heidelberg, 2008.
- [Hol+08] Clyde W. Holsapple, Varghese S. Jacob, Ramakrishnan Pakath, and Jigish S. Zaveri. “Adaptive Decision Support Systems via Problem Processor Learning”. In: *Handbook on Decision Support Systems 1*. Springer, Berlin, Heidelberg, 2008, pp. 659–696.
- [Hol+14] Magnus Holm, Aimar Cordero Garcia, Göran Adamson, and Lihui Wang. “Adaptive Decision Support for Shop-floor Operators in Automotive Industry”. In: *Procedia CIRP* 17 (2014), pp. 440–445.
- [HT06] B. Hailpern and P. Tarr. “Model-driven development: The good, the bad, and the ugly”. In: *IBM Systems Journal* 45.3 (2006), pp. 451–461.
- [IET05] IETF. *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*. 2005. URL: <https://datatracker.ietf.org/doc/html/rfc3986>.
- [IET22] IETF. *JSON Schema: A Media Type for Describing JSON Documents*. 2022. URL: <https://json-schema.org/specification.html>.
- [IET97] IETF. *Key words for use in RFCs to Indicate Requirement Levels*. 1997. URL: <https://www.ietf.org/rfc/rfc2119.txt>.
- [JC12] Slinger Jansen and Michael Cusumano. “Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance”. In: *IWSECO 2012 Workshop on Software Ecosystems*. CEUR-WS, 2012, pp. 41–58.
- [JC19] Christina Terese Joseph and K. Chandrasekaran. “Straddling the crevasse: A review of microservice software architecture foundations and recent advancements”. In: *Software: Practice and Experience* 49.10 (2019), pp. 1448–1484.
- [JDD90] Marius A. Janson, L. Douglas Smith, and Ronald Dattero. “Communicative action and decision support system development: an integrative approach”. In: *Behaviour & Information Technology* 9.6 (1990), pp. 503–516.
- [Jes20] Barnim G. Jeschke. *Entscheidungsorientiertes Management: Einführung in eine konzeptionell fundierte, pragmatische Entscheidungsfindung*. 2nd edition. De Gruyter, 2020.

- [Kar+20] Holger Karl, Dennis Kundisch, Meyer auf der Heide, Friedhelm, and Heike Wehrheim. “A Case for a New IT Ecosystem: On-The-Fly Computing”. In: *Business & Information Systems Engineering* 62.6 (2020), pp. 467–481.
- [Kar20] A. V. Karkanitsa. “Models, Algorithms, and Architecture for Generating Adaptive Decision Support Systems”. In: *Pattern Recognition and Image Analysis* 30.2 (2020), pp. 174–183.
- [KE22] Jonas Kirchhoff and Gregor Engels. “Anti-pattern Detection in Process-Driven Decision Support Systems”. In: *Software Business*. LNBIP 463. Springer, Cham, 2022, pp. 227–243.
- [KGE22] Jonas Kirchhoff, Sebastian Gottschalk, and Gregor Engels. “Detecting Data Incompatibilities in Process-Driven Decision Support Systems”. In: *Business Modeling and Software Design*. LNBIP 453. Springer, Cham, 2022, pp. 89–103.
- [KGL13] Tri Astoto Kurniawan, Aditya K. Ghose, and Lam-Son Lê. “Resolving Violations in Inter-process Relationships in Business Process Ecosystems”. In: *Service-Oriented Computing - ICSOC Workshops 2012*. LNCS 7759. Springer, Berlin, Heidelberg, 2013, pp. 332–343.
- [Kir+21] Jonas Kirchhoff, Sascha Christian Burmeister, Christoph Weskamp, and Gregor Engels. “Towards a Decision Support System for Cross-Sectoral Energy Distribution Network Planning”. In: *Energy Informatics and Electro Mobility ICT*. BIS-Verlag, 2021, pp. 40–46.
- [Kir+22] Jonas Kirchhoff, Nils Weidmann, Stefan Sauer, and Gregor Engels. “Situational Development of Low-Code Applications in Manufacturing Companies”. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. ACM, 2022, pp. 816–825.
- [Kir21] Jonas Kirchhoff. “Providing Decision Makers with Tailored Decision Support Systems”. In: *The 1st Early Career Researchers Workshop Colocated with ECSS 2021*. Informatics Europe, 2021.
- [KLF19] Agnes Koschmider, Ralf Laue, and Michael Fellmann. “Business Process Model Anti-Patterns: A Bibliography and Taxonomy of Published Work”. In: *Proceedings of the 27th European Conference on Information Systems (ECIS)*. 2019.
- [Kow17] Martin Kowalczyk. “Introduction”. In: *The Support of Decision Processes with Business Intelligence and Analytics*. Springer Vieweg, Wiesbaden, 2017, pp. 1–14.

- [KQ08] Mustafa Karakul and Hassan Qudrat-Ullah. “How to Improve Dynamic Decision Making? Practice and Promise”. In: *Complex Decision Making*. Springer, Berlin, Heidelberg, 2008, pp. 3–24.
- [KR14] Supha Khankaew and Stephen Riddle. “A review of practice and problems in requirements engineering in small and medium software enterprises in Thailand”. In: *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE, 2014, pp. 1–8.
- [KV07] Jana Koehler and Jussi Vanhatalo. *Process Anti-Patterns: How to Avoid the Common Traps of Business Process Modeling*. 2007. URL: <https://dominoweb.draco.res.ibm.com/reports/rz3678.pdf> (visited on May 16, 2023).
- [KV11] Diana Kalibatiene and Olegas Vasilecas. “Survey on Ontology Languages”. In: *Perspectives in Business Informatics Research*. Springer, Berlin, Heidelberg, 2011, pp. 124–141.
- [KWE22a] Jonas Kirchhoff, Christoph Weskamp, and Gregor Engels. “Decision Support Ecosystems: Definition and Platform Architecture”. In: *Decision Support Systems XII: Decision Support Addressing Modern Industry, Business, and Societal Needs*. LNBI 447. Springer, Cham, 2022, pp. 97–110.
- [KWE22b] Jonas Kirchhoff, Christoph Weskamp, and Gregor Engels. “Requirements-Based Composition of Tailored Decision Support Systems”. In: *Human-Centered Software Engineering*. LNCS 13482. Springer, Cham, 2022, pp. 150–162.
- [LA10] Ralf Laue and Ahmed Awad. “Visualization of Business Process Modeling Anti Patterns”. In: *Electronic Communications of the EASST 25* (2010).
- [Lar+22] Felix Larrinaga, William Ochoa, Alain Perez, Javier Cuenca, Jon Legaristi, and Miren Illarramendi. “Node-RED Workflow Manager for Edge Service Orchestration”. In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–6.
- [Law13] Kirk Lawrence. *Developing Leaders in a VUCA environment*. 2013. URL: <https://emergingrnleader.com/wp-content/uploads/2013/02/developing-leaders-in-a-vuca-environment.pdf> (visited on September 8, 2022).
- [LBB12] Wenbin Li, Youakim Badr, and Frédérique Biennier. “Digital ecosystems”. In: *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. ACM, 2012, pp. 117–122.

- [LCC13] Ee-Peng Lim, Hsinchun Chen, and Guoqing Chen. “Business Intelligence and Analytics”. In: *ACM Transactions on Management Information Systems* 3.4 (2013), pp. 1–10.
- [LDB16] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. “Web Service Composition”. In: *ACM Computing Surveys* 48.3 (2016), pp. 1–41.
- [Len+18] Jörg Lenhard, Vincenzo Ferme, Simon Harrer, Matthias Geiger, and Cesare Pautasso. “Lessons Learned from Evaluating Workflow Management Systems”. In: *Service-Oriented Computing – ICSSOC 2017 Workshops*. LNCS 10797. Springer, Cham, 2018, pp. 215–227.
- [Len16] Jörg Lenhard. “Portability of Process-Aware and Service-Oriented Software: Evidence and Metrics”. PhD thesis. University of Bamberg Press, 2016.
- [Let21] Timothy C. Lethbridge. “Low-Code Is Often High-Code, So We Must Design Low-Code Platforms to Enable Proper Software Engineering”. In: *Leveraging Applications of Formal Methods, Verification and Validation*. LNCS 13036. Springer, Cham, 2021, pp. 202–212.
- [Ley10] Frank Leymann. “BPEL vs. BPMN 2.0: Should You Care?”. In: *Business Process Modeling Notation: Second International Workshop*. LNBIP 67. Springer, Berlin, Heidelberg, 2010, pp. 8–13.
- [Li+19] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. “Service Mesh: Challenges, State of the Art, and Future Research Opportunities”. In: *13th IEEE International Conference on Service-Oriented System Engineering*. IEEE, 2019, pp. 122–1225.
- [Lie+06] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. “End-User Development: An Emerging Paradigm”. In: *End User Development*. Human-Computer Interaction Series 9. Springer, Dordrecht, 2006, pp. 1–8.
- [Lie+18] Grischa Liebel, Matthias Tichy, Eric Knauss, Oscar Ljungkrantz, and Gerald Stieglbauer. “Organisation and communication problems in automotive requirements engineering”. In: *Requirements Engineering* 23.1 (2018), pp. 145–167.
- [LJ14] Zhengli Liang and Xiaofeng Jia. “The Technical System Establishment and Application of Decision Support System Generator Based on Component Coordination”. In: *2014 Seventh International Symposium on Computational Intelligence and Design*. IEEE, 2014, pp. 229–233.

- [LKG16] Ralf Laue, Wilhelm Koop, and Volker Gruhn. “Indicators for Open Issues in Business Process Models”. In: *Requirements engineering*. LNCS 9619. Springer, Cham, 2016, pp. 102–116.
- [LL09] Kathryn B. Laskey and Kenneth Laskey. “Service oriented architecture”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.1 (2009), pp. 101–105.
- [LN12] Niels Lohmann and Martin Nyolt. “Artifact-Centric Modeling Using BPMN”. In: *Service-Oriented Computing - ICSOC 2011 Workshops*. LNCS 7221. Springer, Berlin, Heidelberg, 2012, pp. 54–65.
- [Luo+21] Yajing Luo, Peng Liang, Chong Wang, Mojtaba Shahin, and Jing Zhan. “Characteristics and Challenges of Low-Code Development”. In: *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2021, pp. 1–11.
- [Mac+16] Oliver Mack, Anshuman Khare, Andreas Krämer, and Thomas Burgartz. *Managing in a VUCA World*. Springer International Publishing, 2016.
- [Man16] Konstantinos Manikas. “Revisiting software ecosystems Research: A longitudinal literature study”. In: *Journal of Systems and Software* 117 (2016), pp. 84–103.
- [Mar+20] Ricardo Martins, Filipe Caldeira, Filipe Sa, Maryam Abbasi, and Pedro Martins. “An overview on how to develop a low-code application using OutSystems”. In: *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. IEEE, 2020, pp. 395–401.
- [MBD20] Mihaela Minciu, Florin-Aurel Berar, and Razvan Catalin Dobrea. “New decision systems in the VUCA world”. In: *Management & Marketing. Challenges for the Knowledge Society* 15.2 (2020), pp. 236–254.
- [MFB04] Sergio Maturana, Juan-Carlos Ferrer, and Francisco Barañao. “Design and implementation of an optimization-based decision support system generator”. In: *European Journal of Operational Research* 154.1 (2004), pp. 170–183.
- [MGK07] Shah J. Miah, John Gammack, and Don Kerr. “Ontology Development for Context-Sensitive Decision Support”. In: *Third International Conference on Semantics, Knowledge and Grid (SKG 2007)*. IEEE, 2007, pp. 475–478.
- [MGM19] Júlio Menezes, Cristine Gusmão, and Hermano Moura. “Risk factors in software development projects: a systematic literature review”. In: *Software Quality Journal* 27.3 (2019), pp. 1149–1174.

- [MGO20] Frederik Möller, Tobias Moritz Guggenberger, and Boris Otto. “Towards a Method for Design Principle Development in Information Systems”. In: *Designing for Digital Transformation. Co-Creating Services with Citizens and Industry*. LNCS 12388. Springer, Cham, 2020, pp. 208–220.
- [MHR22] Nuria Mollá, Ciara Heavin, and Alejandro Rabasa. “Data-driven decision making: new opportunities for DSS in data stream contexts”. In: *Journal of Decision Systems* 31.sup1 (2022), pp. 255–269.
- [MKP21] Nikolay Mustafin, Pavel Kopylov, and Andrew Ponomarev. “Knowledge-Based Automated Service Composition for Decision Support Systems Configuration”. In: *Data Science and Intelligent Systems*. LNNS 231. Springer, Cham, 2021, pp. 780–788.
- [MM18] Mohsen Mohammadi and Muriati Mukhtar. “Service-Oriented Architecture and Process Modeling”. In: *2018 International Conference on Information Technologies (InfoTech)*. IEEE, 2018, pp. 1–4.
- [Mor08] Shoichi Morimoto. “A Survey of Formal Verification for Business Process Modeling”. In: *Computational Science - ICCS 2008*. LNCS 5102. Springer, Berlin, Heidelberg, 2008, pp. 514–522.
- [MP14] Andreas Metzger and Klaus Pohl. “Software Product Line Engineering and Variability Management: Achievements and Challenges”. In: *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 70–84.
- [MRT76] Henry Mintzberg, Duru Raisinghani, and Andre Theoret. “The Structure of ”Unstructured” Decision Processes”. In: *Administrative Science Quarterly* 21.2 (1976), pp. 246–275.
- [Mue18] Felix Mueller. *Custom Tasklist examples*. 2018. URL: <https://camunda.com/blog/2018/02/custom-tasklist-examples/> (visited on July 2, 2023).
- [Nik+20] Naghmeh Niknejad, Waidah Ismail, Imran Ghani, Behzad Nazari, Mahadi Bahari, and Hussin, Ab Razak Bin Che. “Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation”. In: *Information Systems* 91 (2020), p. 101491.
- [NMJ19] Falak Nawaz, Ahmad Mohsin, and Naeem Khalid Janjua. “Service description languages in cloud computing: state-of-the-art and research issues”. In: *Service Oriented Computing and Applications* 13.2 (2019), pp. 109–125.
- [Nut02] Paul C. Nutt. *Why Decisions Fail: Avoiding the Blunders and Traps That Lead to Debacles*. Berrett-Koehler Publishers Incorporated, 2002.

- [OAS04] OASIS. *UDDI Version 3.0.2*. 2004. URL: http://www.uddi.org/pubs/uddi_v3.htm.
- [OAS07] OASIS. *Web Services Business Process Execution Language: Version 2.0*. 2007. URL: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
- [OAS10] OASIS. *Web Services – Human Task (WS-HumanTask) Specification: Version 1.1*. 2010. URL: <http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.pdf>.
- [OBF19] Marcelo Iury S. Oliveira, Glória de Fátima Barros Lima, and Bernadette Farias Lóscio. “Investigations into Data Ecosystems: a systematic mapping study”. In: *Knowledge and Information Systems* 61.2 (2019), pp. 589–630.
- [Obj13] Object Management Group. *Business Process Model and Notation (BPMN): Version 2.0.2*. 2013. URL: <https://www.omg.org/spec/BPMN/2.0.2/PDF>.
- [Obj21] Object Management Group. *Decision Model and Notation: Version 1.4*. 2021.
- [OL18] Marcelo Iury S. Oliveira and Bernadette Farias Lóscio. “What is a data ecosystem?” In: *Proceedings of the 19th Annual International Conference on Digital Government Research Governance in the Data Age*. ACM, 2018, pp. 1–9.
- [OMG17] OMG. *OMG Unified Modeling Language (OMG UML): Version 2.5.1*. 2017. URL: <https://www.omg.org/spec/UML/2.5.1/PDF>.
- [Ope21] OpenAPI Initiative. *OpenAPI Specification v3.1.0*. 2021. URL: <https://spec.openapis.org/oas/v3.1.0>.
- [Pao+03] Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia Sycara. “The DAML-S Virtual Machine”. In: LNCS 2870. Springer, Berlin, Heidelberg, 2003, pp. 290–305.
- [Pap+07] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. “Service-Oriented Computing: State of the Art and Research Challenges”. In: *Computer* 40.11 (2007), pp. 38–45.
- [Par+13] Gregory S. Parnell, Terry A. Bresnick, Steven N. Tani, and Eric R. Johnson. *Handbook of Decision Analysis*. John Wiley & Sons, Inc, 2013.
- [Pau08] Cesare Pautasso. “BPEL for REST”. In: *Business Process Management*. LNCS 5240. Springer, Berlin, Heidelberg, 2008, pp. 278–293.

- [PBA98] P. Paranagama, F. Burstein, and D. Arnott. “ADAPTOR: A Personality-Based Adaptive DSS Generator”. In: *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*. IEEE, 1998, 54–63 vol.5.
- [Pef+07] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. “A Design Science Research Methodology for Information Systems Research”. In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77.
- [Phi13] Gloria Phillips-Wren. “Intelligent Decision Support Systems”. In: *Multicriteria decision aid and artificial intelligence*. Wiley-Blackwell, 2013, pp. 25–44.
- [PI18] Daniel Power and Lakshmi Iyer. “Decision Support Systems Research –Most Cited Articles and Books”. In: *Proceedings of the 2018 Pre-ICIS SIGDSA Symposium* (2018).
- [PL03] Randall Perrey and Mark Lycett. “Service-oriented architecture”. In: *2003 symposium on applications and the Internet (SAINT) workshops*. IEEE, 2003, pp. 116–119.
- [Pow09] Daniel J. Power. *Decision Support Basics*. 1st edition. Information systems collection. Business Expert Press, 2009.
- [PS09] Selwyn Piramuthu and Michael J. Shaw. “Learning-enhanced adaptive DSS: a Design Science perspective”. In: *Information Technology and Management* 10.1 (2009), pp. 41–54.
- [Ram+18] Qusai Ramadan, Daniel Strüber, Mattia Salnitri, Volker Riediger, and Jan Jürjens. “Detecting Conflicts Between Data-Minimization and Security Requirements in Business Process Models”. In: *Modelling Foundations and Applications*. LNCS 10890. Springer, Cham, 2018, pp. 179–198.
- [Ram+98] Juan Ramos, Lorena Bearzotti, Enrique Milani, Federico Woscoff, Mariana Gorsky, Carlos M. Carlett, Ma.R. Galli, and Omar Chiotti. “Decision support systems generator for industrial companies Module I: Product design support system”. In: *Computers & Industrial Engineering* 35.1-2 (1998), pp. 303–306.
- [Rec08] Jan Recker. “BPMN Modeling – Who, Where, How and Why”. In: *BPTrends* (2008).
- [RED17] Anass Rachdi, Abdeslam En-Nouaary, and Mohamed Dahchour. “DataFlow Analysis in BPMN Models”. In: *Proceedings of the 19th International Conference on Enterprise Information Systems*. SCITEPRESS, 2017, pp. 229–237.

- [RH09] Per Runeson and Martin Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164.
- [Ric+97] M. Rico, O. Yuschak, M.L Taverna, J.C Ramos, M.R Galli, and O. Chiotti. “Decision support systems generator for industrial companies”. In: *Computers & Industrial Engineering* 33.1-2 (1997), pp. 357–360.
- [Roc+09] J. Rockwell, I. R. Grosse, S. Krishnamurty, and J. C. Wileden. “A Decision Support Ontology for collaborative decision making in engineering design”. In: *2009 International Symposium on Collaborative Technologies and Systems*. IEEE, 2009, pp. 1–9.
- [Rol21] Mike Rollings. *How to Make Better Business Decisions*. 2021. URL: <https://www.gartner.com/smarterwithgartner/how-to-make-better-business-decisions> (visited on September 22, 2022).
- [Ros+98] Pablo Rossi, Clara Díaz, Pablo Fruttero, César Vittori, Mariela Rico, and Omar Chiotti. “Decision support systems generator for industrial companies Module III: Scheduling support system”. In: *Computers & Industrial Engineering* 35.1-2 (1998), pp. 311–314.
- [RS13] Marco Rospocher and Luciano Serafini. “An Ontological Framework for Decision Support”. In: *Semantic Technology*. LNCS 7774. Springer, Berlin, Heidelberg, 2013, pp. 239–254.
- [Sad+04] Shazia Sadiq, Maria Orłowska, Wasim Sadiq, and Cameron Foulger. “Data flow and validation in workflow modelling”. In: *Proceedings of the 15th Australasian database conference - Volume 27*. ACM, 2004, pp. 207–214.
- [Sah+20] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 171–178.
- [SAL18] Farhad Shahnia, Ali Arefi, and Gerard Lich. *Electric Distribution Network Planning*. Power Systems. Springer, Singapore, 2018.
- [Sàn22] Miquel Sànchez-Marrè. “Decisions”. In: *Intelligent Decision Support Systems*. Springer, Cham, 2022, pp. 9–55.
- [SBM11] Dragan A. Savić, Josef Bicik, and Mark S. Morley. “A DSS generator for multiobjective optimisation of spreadsheet-based models”. In: *Environmental Modelling & Software* 26.5 (2011), pp. 551–561.

- [Sch+05] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. “The Enterprise Service Bus: Making service-oriented architecture real”. In: *IBM Systems Journal* 44.4 (2005), pp. 781–797.
- [Sch+19] Konrad Schneid, Claus A. Usener, Sebastian Thöne, Herbert Kuchen, and Christian Tophinke. “Static analysis of BPMN-based process-driven applications”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2019, pp. 66–74.
- [Sch+21a] Eike Schäffer, Marvin Schobert, Tobias Reichenstein, Andreas Selmaier, Volker Stiehl, Markus Herhoffer, Matus Mala, and Jörg Franke. “Reference Architecture and Agile Development Method for a Process-Driven Web Platform based on the BPMN-Standard and Process Engines”. In: *Procedia CIRP* 103 (2021), pp. 146–151.
- [Sch+21b] Konrad Schneid, Herbert Kuchen, Sebastian Thöne, and Sascha Di Bernardo. “Uncovering data-flow anomalies in BPMN-based process-driven applications”. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. ACM Digital Library. ACM, 2021, pp. 1504–1512.
- [Sch+21c] Konrad Schneid, Leon Stapper, Sebastian Thöne, and Herbert Kuchen. “Automated Regression Tests: A No-Code Approach for BPMN-based Process-Driven Applications”. In: *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2021, pp. 31–40.
- [SDA20] Riska Septifani, Panji Deoranto, and Tiyas Widya Armanda. “Employee Performance Assessment using Analytical Network Process and Rating Scale”. In: *Jurnal Teknik Industri* 21.1 (2020), pp. 70–79.
- [SE20] Bahar Schwichtenberg and Gregor Engels. “SecoArc: A Framework for Architecting Healthy Software Ecosystems”. In: *Software Architecture*. CCIS 1269. Springer, Cham, 2020, pp. 95–106.
- [She+14] Quan Z. Sheng, Xiaoqiang Qiao, Athanasios V. Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. “Web services composition: A decade’s overview”. In: *Information Sciences* 280 (2014), pp. 218–238.
- [Sil11] Bruce Silver. *BPMN Method and Style, 2nd Edition, with BPMN Implementer’s Guide: A Structured Approach for Business Process Modeling and Implementation Using BPMN 2*. 2nd edition. Cody-Cassidy Press, 2011.
- [Sil16] Alfons Sillaber. *Leitfaden zur Verteilnetzplanung und Systemgestaltung: Entwicklung dezentraler Elektrizitätssysteme*. 1st edition. Springer Vieweg, 2016.

- [SK86] K.B.C. Saxena and Mohan Kaul. “A conceptual architecture for DSS generators”. In: *Information & Management* 10.3 (1986), pp. 149–157.
- [SLE19] Prince Kwame Senyo, Kecheng Liu, and John Effah. “Digital business ecosystem: Literature review and a framework for future research”. In: *International Journal of Information Management* 47 (2019), pp. 52–64.
- [SRM14] Stephan Schiffner, Thomas Rothschädl, and Nils Meyer. “Towards a Subject-Oriented Evolutionary Business Information System”. In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*. IEEE, 2014, pp. 381–388.
- [SRP17] Todd Sedano, Paul Ralph, and Cecile Peraire. “Software Development Waste”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 130–140.
- [SŞF15] Ioana Andreea Stănescu, Antoniu Ştefan, and Florin Gheorghe Filip. “Cloud-Based Decision Support Ecosystem for Renewable Energy Providers”. In: *Technological Innovation for Cloud-Based Engineering Systems*. Springer, Cham, 2015, pp. 405–412.
- [SSP12] Farzad Shafiei, David Sundaram, and Selwyn Piramuthu. “Multi-enterprise collaborative decision support system”. In: *Expert Systems with Applications* 39.9 (2012), pp. 7637–7651.
- [Sta+13] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. “DataFu: A Language and an Interpreter for Interaction with Read/Write Linked Data”. In: *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 1225–1236.
- [Sta16] Florian Stapel. “Ontology-based representation of abstract optimization models for model formulation and system generation”. PhD thesis. 2016.
- [Sta22] Statista. *Anzahl der Stromnetzbetreiber in Deutschland in den Jahren 2012 bis 2022*. 2022. URL: <https://de.statista.com/statistik/daten/studie/152937/umfrage/anzahl-der-stromnetzbetreiber-in-deutschland-seit-2006/> (visited on January 7, 2023).
- [Sti14] Volker Stiehl. *Process-driven applications with BPMN*. Springer, Cham, 2014.
- [Sun+06] Sherry X. Sun, J. Leon Zhao, Jay F. Nunamaker, and Olivia R. Liu Sheng. “Formulating the Data-Flow Perspective for Business Process Management”. In: *Information Systems Research* 17.4 (2006), pp. 374–391.

- [SWM16] Carl Spetzler, Hannah Winter, and Jennifer Meyer. *Decision Quality: Value Creation from Better Business Decisions*. 1st edition. John Wiley & Sons, 2016.
- [TDA16] Theodoros V. Theodoropoulos, Ioannis G. Damousis, and Angelos J. Amditis. “Demand-Side Management ICT for Dynamic Wireless EV Charging”. In: *IEEE Transactions on Industrial Electronics* 63.10 (2016), pp. 6623–6630.
- [TvS09] Nikola Trčka, Wil M. P. van der Aalst, and Natalia Sidorova. “Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows”. In: *Advanced Information Systems Engineering*. LNCS 5565. Springer, Berlin, Heidelberg, 2009, pp. 425–439.
- [Val+09] Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki, and Negin Daneshpour. “A brief survey of software architecture concepts and service oriented architecture”. In: *2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 2009, pp. 34–38.
- [Veg18] Kees Vegter. *Cypher Query Optimisations - Neo4j Developer Blog*. 2018. URL: <https://medium.com/neo4j/cypher-query-optimisations-fe0539ce2e5c> (visited on May 29, 2023).
- [VKG17] Hulya Vural, Murat Koyuncu, and Sinem Guney. “A Systematic Literature Review on Microservices”. In: *Computational Science and Its Applications – ICCSA 2017*. LNCS 10409. Springer, Cham, 2017, pp. 203–217.
- [vSta+14] Silvia von Stackelberg, Susanne Putze, Jutta Mülle, and Klemens Böhm. “Detecting Data-Flow Errors in BPMN 2.0”. In: *Open Journal of Information Systems (OJIS)* 1.2 (2014), pp. 1–19.
- [VTP20] Pedro Valderas, Victoria Torres, and Vicente Pelechano. “A microservice composition approach based on the choreography of BPMN fragments”. In: *Information and Software Technology* 127 (2020), p. 106370.
- [W3C04] W3C. *OWL-S: Semantic Markup for Web Services*. 2004. URL: <https://www.w3.org/Submission/OWL-S/>.
- [W3C07a] W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. URL: <https://www.w3.org/TR/soap12/>.
- [W3C07b] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2007. URL: <https://www.w3.org/TR/wsdl20/>.
- [W3C12] W3C. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. 2012. URL: <https://www.w3.org/TR/owl2-overview/>.

- [W3C14] W3C. *RDF 1.1 Concepts and Abstract Syntax*. 2014. URL: <https://www.w3.org/TR/rdf11-concepts/>.
- [Wan08] Lihui Wang. “Wise-ShopFloor: An Integrated Approach for Web-Based Collaborative Manufacturing”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.4 (2008), pp. 562–573.
- [Was19] Robert Waszkowski. “Low-code platform for automating business processes in manufacturing”. In: *IFAC-PapersOnLine* 52.10 (2019), pp. 376–381.
- [Wel+15] Tobias Weller, Maria Maleshkova, Keno März, and Lena Maier-Hein. “A RESTful Approach for Developing Medical Decision Support Systems”. In: *The Semantic Web: ESWC 2015 Satellite Events*. LNCS 9341. Springer, Cham, 2015, pp. 376–384.
- [WHM10] Ingo Weber, Jörg Hoffmann, and Jan Mendling. “Beyond soundness: on the verification of semantic business process models”. In: *Distributed and Parallel Databases* 27.3 (2010), pp. 271–343.
- [WKE20] Dennis Wolters, Jonas Kirchhoff, and Gregor Engels. “Specifying Web Interfaces for Command-Line Applications Based on OpenAPI”. In: *Service-Oriented Computing – ICSOC 2019 Workshops*. LNCS 12019. Springer, Cham, 2020, pp. 30–41.
- [Xia+16] Yue Xiang, Junyong Liu, Furong Li, Yong Liu, Youbo Liu, Rui Xu, Yunche Su, and Lei Ding. “Optimal Active Distribution Network Planning: A Review”. In: *Electric Power Components and Systems* 44.10 (2016), pp. 1075–1094.
- [Yan+18] Weihong Yang, Lin Cheng, Ning Qi, Yanru Liu, and Xuyang Wang. “Review on distribution network planning methods considering large-scale access of flexible load”. In: *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*. IEEE, 2018, pp. 1–6.
- [YC06] Yi Yang and Jacques Calmet. “From the OntoBayes Model to a Service Oriented Decision Support System”. In: *2006 International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*. IEEE, 2006, p. 127.
- [YN92] Gee Kin Yeo and Fui Hoon Nah. “A Participants’ DSS for a Management Game with a DSS Generator”. In: *Simulation & Gaming* 23.3 (1992), pp. 341–353.

- [YSL08] Qiongwei Ye, Guangxing Song, and Ting Li. “Service-Oriented Decision Support System for Crisis Management in E-Government”. In: *2008 IEEE Symposium on Advanced Management of Information for Globalized Enterprises (AMIGE)*. IEEE, 2008, pp. 1–5.
- [Zaf+19] Iqra Zafar, Farooque Azam, Muhammad Waseem Anwar, Bilal Maqbool, Wasi Haider Butt, and Aiman Nazir. “A Novel Framework to Automatically Generate Executable Web Services From BPMN Models”. In: *IEEE Access* 7 (2019), pp. 93653–93677.
- [Zar13] P. Zaraté. *Tools for collaborative decision-making*. Focus series in computer engineering and IT. ISTE and John Wiley and Sons Inc, 2013.
- [ZSJ08] Fatemeh “Mariam” Zahedi, Jaeki Song, and Suprasith Jarupathirun. “Web-Based Decision Support”. In: *Handbook on Decision Support Systems 1*. Springer, Berlin, Heidelberg, 2008, pp. 315–338.
- [ZZ10] Yury Zagorulko and Galina Zagorulko. “Ontology-Based Approach to Development of the Decision Support System for Oil-and-Gas Production Enterprise”. In: *New Trends in Software Methodologies, Tools and Techniques*. Frontiers in Artificial Intelligence and Applications 217. IOS Press, 2010.

Index of Definitions

- (n-1)-reliability, 26
- activity, 129
- adaptive decision support system (ADSS), 52
- ambiguity, 35
- assistance, 91
- atomic service, 45
- automated service, 83

- behavioral perspective, 123
- black-box transparency, 43
- BPMN-I, 146
- BPMN-Q, 195
- business analytics, 67
- business intelligence, 67
- Business Process Execution Language (BPEL), 125
- Business Process Model and Notation (BPMN), 67, 126

- canonical data model, 46
- CAPEX, 26
- citizen developer, 47
- code, 47
- collaborative DSS, 66
- completeness, 156
- complexity, 35
- compliance, 156

- composition assistance, 152
- composition error, 157
- composition expert, 86
- Composition Knowledge Base, 92
- Composition Knowledge Extraction, 92
- composition warning, 157
- constructive quality assurance, 155
- content management system (CMS), 115

- data association, 135
- data ecosystem, 75
- data flow, 123, 134
- data input, 134
- data input association, 135
- data object, 135
- data output, 134
- data output association, 135
- data provider, 86
- data service, 83
- database management system (DBMS), 50
- decision, 24
- decision activity, 29
- decision alternative, 24
- decision analyst, 27
- decision constraint, 26
- decision criteria, 26
- decision maker, 27

- decision making, 24
- Decision Model and Notation (DMN), 67, 141
- decision objective, 26
- decision optimality, 26
- decision problem, 24
- decision process, 27
- decision rationality, 27
- decision situation, 30
- decision support artifact, 38
- decision support ecosystem (DSE), 15, 76, 94
- decision support service, 11, 83
- decision support service provider, 86
- decision support system (DSS), 4
- decision support system generator (DSSG), 55
- decision validity, 26
- delegated service, 83
- demand forecasting, 30
- demand side management (DSM), 23, 213
- design principle, 79
- design science research (DSR), 14
- design science research methodology, 14
- digital business ecosystem (DBE), 72
- distribution network, 22
- distribution network operator (DNO), 23
- distribution network planning (DNP), 24
- Domain Documentation, 89
- domain expert, 28, 85
- Domain Knowledge Base, 91
- DSS Editor, 91
- DSS engineer, 86
- DSS Generator, 91
- DSS Implementor, 56
- DSS model, 86
- ecosystem health, 239
- ecosystem service, 72
- end event, 142
- end user, 73
- end-user development (EUD), 47
- end-user programming (EUP), 48
- energy system, 22
- enterprise service bus (ESB), 46
- event, 141
- exact optimization, 33
- exclusive gateway, 139
- external validity, 230
- FlexiEnergy, 7, 21
- Friendly Enough Expression Language (FEEL), 141
- functional perspective, 122
- functional service, 83
- gateway, 139
- group DSS, 66
- heuristic optimization, 33
- high-code development, 47
- higher-level service, 45
- holistic, 8
- inclusive gateway, 141
- influencing factor, 34
- informational perspective, 123
- infrastructure provider, 86
- input-output specification, 134
- input-processing-output (IPO), 83
- input-processing-output (IPO) principle, 96
- integrated service, 84
- intelligent DSS (IDSS), 66
- interactive service, 83
- invalid decision, 26
- investment plan, 24
- JavaScript Object Notation (JSON), 115

- JSON Schema, 180
- low-code application, 49
- low-code development (LCD), 47
- low-code development platform (LCDP), 11, 49
- meta decision making, 30
- model, 48
- model management system (MMS), 66
- Model Repository, 91
- model-driven development (MDD), 48
- monolith, 43
- named placeholder, 196
- network asset, 23
- network reduction, 106
- network simulation, 32
- network topology, 23
- no-code development, 47
- nonexistent flow, 196
- nonexistent nonsequential flow, 196
- nonsequential flow, 196
- Object Constraint Language (OCL), 160
- off-the-shelf DSS, 8
- ontology, 100
- OpenAPI, 46
- operational perspective, 122
- operational/programmed decision problem, 24
- OPEX, 26
- optimal decision, 6, 26
- optimization characteristics, 161
- OWL-S, 102
- parallel gateway, 139
- PD-DSS Design, 128
- PD-DSS Enactment, 128
- PD-DSS Repository, 128
- PD-DSS review, 193
- placeholder, 196
- platform provider, 72
- platform-as-a-service (PaaS), 49
- power flow, 23
- Power-to-Gas (P2G), 23
- Power-to-Heat (P2H), 23
- process anti-patterns, 189
- process weakness patterns, 189
- process-driven application (PDA), 126
- process-driven decision support system (PD-DSS), 16, 127
- product, 73
- rapid application development (RAD), 48
- recommendation-based assistance, 153
- reliability, 230
- rendering, 132
- Representational State Transfer (REST), 46
- Requirements Elicitation, 91
- Resource Description Framework (RDF), 103
- sector coupling, 23
- self-contained system, 46
- service, 43
- service broker, 73
- service choreography, 45
- service composition, 44
- service description, 44
- service description language, 102
- service exception, 99
- service interface, 43
- service level indicators, 99
- service level objectives, 99
- service orchestration, 45
- service provider, 72, 86

- Service Registration, 91
- Service Registry, 91
- service registry, 44
- service task, 131
- service visibility, 44
- service-oriented architecture (SOA), 11, 43
- service-oriented computing, 42
- service-oriented DSS (SO-DSS), 62
- severity level, 157
- shared digital platform, 73
- situational factor, 30
- situational method engineering (SME), 67
- smart grid, 25
- SOAP, 46
- software ecosystem, 74
- software provider, 86
- software-based service, 83
- stakeholder, 27
- start event, 142
- strategic/non-programmed decision problem, 25
- subject matter expert, 27
- suboptimal decision, 26
- subprocess, 129
- substation, 22

- tactical/semi-programmed decision problem, 25
- tailored DSS, 7
- task, 129
- task selector, 195, 196
- technical diversity, 43
- transmission network, 22
- triple stores, 103

- uncertainty, 26, 35
- uncontrolled service, 84

- Universal Description Discovery & Integration (UDDI), 46
- user task, 131

- valid decision, 26
- validation-based composition assistance, 155
- value, 72
- violation, 157
- violation rule, 157
- volatility, 35
- VUCA, 4

- Web Service Description Language (WSDL), 46
- web-based DSS, 66
- WS-* technology stack, 46

Composition Assistance Supplement

This appendix describes two extensions to the composition assistance of Chapter 7. Section A.1 explains how a PD-DSS process model can be validated for completeness to ensure that it specifies all information needed for its subsequent enactment. Section A.2 presents two changes to the transformation of a BPMN-Q anti-pattern to a graph database query.

A.1 Completeness Validation of PD-DSS Process Models

The design of the operational, informational, and functional-behavioral composition assistance in Sections 7.3 to 7.5 focuses on validating the effectiveness and efficiency of a PD-DSS model with respect to the requirements for decision support documented by a decision maker. A PD-DSS process model can additionally be validated to check if it contains all information required for its subsequent enactment, i.e., if it adheres to the PD-DSS modeling conventions established in Chapter 6. For example, validating that an implementing decision support service is selected for each activity in the PD-DSS process model ensures that the enactment of the PD-DSS is not interrupted because the underlying process engine does not know which decision support service to invoke next. This validation is referred to as the *completeness validation* of a PD-DSS process model.

The upcoming subsections explain potential completeness violations, which are documented using the tabular documentation format for violation rules introduced in Section 7.1.3. The explanations are grouped according to the perspective of the underlying process model which they affect, i.e., Section A.1.1 describes checks for the operational perspective, Sec-

tion A.1.2 for the informational perspective, and Section A.1.3 for the functional and behavioral perspective. A demonstration of the described checks is included with the demonstration of the composition assistance described in Chapter 7.

A.1.1 Completeness of the Operational Perspective

The operational perspective of a PD-DSS process model is incomplete if no decision support service is selected for the implementation of a task. Although this mistake can easily occur due to negligence, e.g., because the whole decision process is first defined with respect to the other process perspectives, it would nevertheless prevent the BPMN engine from properly enacting the PD-DSS process model. In particular, if the mistake is only caught during runtime, it could potentially render all work performed so far useless if the fixed decision process must be restarted from the first activity. This results in the definition of the *Missing Service Violation*.

Missing Service Violation		Severity: Error
Element:	Task	Phase: Design
Condition:	No decision support service is selected for a task in the process model.	
Message:	“No decision support service is specified for decision activity [...].”	

This violation can also be defined analogously for the case where a service ID is documented for the implementation of a task, but no associated service description is listed in the *Service Repository*. This violation is not further elaborated here, as the *PD-DSS Design* application is expected to proactively prevent such a mistake, e.g., by implementing service selection as a dropdown control instead of a free text input where the service ID is entered.

A.1.2 Completeness of the Informational Perspective

Validating the completeness of a PD-DSS process model with respect to its informational perspective includes checking that all data associations specify a valid source and target where data is copied from/to. Otherwise, the BPMN engine will not be able to facilitate the data exchange between services. These violations are not further elaborated here as they can be avoided if the *PD-DSS Design* application proactively prevents the specification of invalid source/target method data IDs, e.g., by using a dropdown instead of free text input. Furthermore, checking that each data association is connected to one task and one data object may be omitted if this is already prevented by the implementation of the visual modeling editor used by the *PD-DSS Design* application.

A data object or (process) data input without an outgoing data association is redundant since its value is never used. Although this redundancy does not impact the executability of the process model, it is nevertheless reported as a warning since it could be an indication of another issue, e.g., a data object which should be marked as a data output of the process.

Redundant Data Violation	Severity: Warning
Element: Data Object, Data Input	Phase: Design
Condition: (see “Message”)	
Message: “Data [...] is redundant (has no outgoing data association).”	

Data objects and process data outputs can be written from multiple tasks to simplify the visual appearance of the process model (cf. reuse of the “Error Info” data object in Fig. 6.8). This can result in data loss if the tasks are executed in parallel. Since the reuse of data objects only has a cosmetic benefit but no additional semantic meaning, the informational composition assistance reports data object reuse as an error to avoid consideration of the behavioral perspective in order to determine the possible parallel execution of tasks, which results in increased validation performance.

Potential Data Loss Violation	Severity: Error
Element: Data Object, (Process) Data Output	Phase: Design
Condition: A data object or a process data output has multiple incoming data input associations.	
Message: “Data [...] is written from multiple tasks ([..., ...]), which can lead to data loss when tasks are executed in parallel.”	

A.1.3 Completeness of the Functional and Behavioral Perspectives

With respect to the behavioral perspective, multiple domain-agnostic completeness violations can be defined based on missing sequence flows, i.e., the *Unreachable Element Violation* for missing incoming sequence flows and the *Dead End Violation* for missing outgoing sequence flows. These violations also require the process model to include explicit start and end events.

To ensure that the first activity to execute is unambiguous, the *Start Event Violation* is defined. It does not apply to start events used within a subprocess. Multiple end events do not negatively impact process enactment and are therefore allowed.

Unreachable Element Violation		Severity: Error
Element:	Activity, Gateway	Phase: Design
Condition:	(see “Message”)	
Message:	“{Activity / Gateway} [...] will never be executed because it has no incoming sequence flow.”	
Dead End Violation		Severity: Error
Element:	Activity, Gateway	Phase: Design
Condition:	(see “Message”)	
Message:	“Process will get stuck in {Activity / Gateway} [...] because it has no outgoing sequence flow.”	
Start Event Violation		Severity: Error
Element:	Process	Phase: Design
Condition:	(see “Message”)	
Message:	“A process must have exactly one start event.”	

For exclusive gateways controlling the conditional flow through the process, the existence of outgoing sequence flows on its own is not sufficient. Instead, these sequence flows must be associated with proper conditions to indicate when this sequence flow is taken. The existence of a condition is checked as part of the *Missing Condition Violation*. If not proactively prevented by the *PD-DSS Design* application, the well-formedness of the condition should be validated, e.g., whether it contains a valid data object and metadata attribute identifier. Additionally, it is possible to specify a comparison value that has the wrong data type (e.g., "1000" instead of 1000 when comparing the size of a network topology) or an incompatible comparison operator, (e.g., includes for a single-value quantitative attribute).

Missing Condition Violation		Severity: Error
Element:	Sequence Flow	Phase: Design
Condition:	(see “Message”)	
Message:	“Sequence flow from [...] to [...] specifies no condition.”	

Gateways with a single outgoing sequence flow are redundant, which could be explicitly detected by a violation with a severity level of “warning”. However, since the evaluation

of a condition is assumed to take constant time and therefore does not introduce significant inefficiency, such a violation is not explicitly documented here. Another potential extension is validating that the conditions specified for outgoing sequence flows of an exclusive gateway are mutually exclusive. However, such a check is expectedly complex and resource-consuming due to the potential concatenation of conditions with logical AND and OR operations. Therefore, it is not further described here.

A.2 Transformation Changes

The transformation of BPMN process models and BPMN-Q anti-patterns into graph database queries is already described in [KE22]. This section describes two limitations of the described transformations and explains how these limitations can be addressed.

A.2.1 Nonexistent Flows

The transformation of a nonexistent flow in [KE22] does not consider the conditional execution of a task. This is demonstrated for the *Missing Reversal of Network Reduction* anti-pattern introduced on page 197 with the example shown in Fig. A.1. Here, a path from the “Perform network reduction” task to the “Reverse network reduction” task exists for the top path in the process model, therefore no violation is reported. However, due to the conditionality introduced by the exclusive gateway, the “Reverse network reduction” task is not executed if the bottom path through the process model is taken. Therefore, the anti-pattern should be reported. This can be achieved by changing the partial *Cypher* query for nonexistent flows to `NONE (v IN nodes(p) WHERE v=m)`.

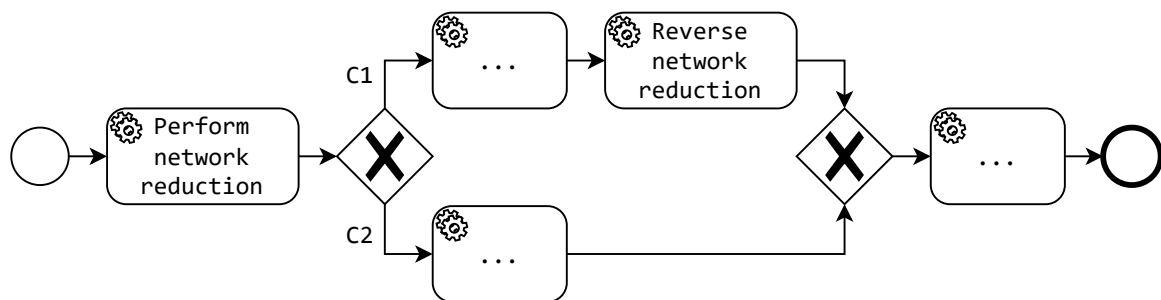


Figure A.1: Exemplary process model with *Missing Reversal of Network Reduction* anti-pattern

The target of a nonexistent flow could also not be included in the process model at all. In the prototypical implementation associated with [KE22], this was accounted for with an

OPTIONAL MATCH for the target node as the graph database would otherwise report an error when trying to match a missing node. However, the overall nonexistence of the target in the process model already indicates the presence of the anti-pattern. For this reason, the prototypical implementation was updated to run two graph database queries for a nonexistent flow. The first query checks if all target nodes of nonexistent flows are present in the process model and the presence of the anti-pattern is reported if a node is missing. Only if the first query identifies the presence of all affected nodes, the second query checks if a path exists where the target node of the nonexistent flow does not follow the source node.

Both issues are corrected in the prototypical implementation described in Appendix B. They are partially responsible for the performance differences reported in Section 7.5.5 and the original paper ([KE22]).

A.2.2 Conditions on Nonsequential Flows

The (extended) BPMN-Q elements for anti-pattern definition summarized in Table 7.1 support the definition of conditions for nonsequential flows. The underlying paper ([KE22]) only considered conditions for singular sequence flows.

To support conditions for nonsequential flows, it is not sufficient to check if a singular sequence flow along the path from the source to the target node specifies the condition or not, as the conditional path could already have been ended by a merging exclusive gateway. This is also demonstrated with the example shown in Fig. A.1 if the task before the end event should only be executed if the C1 condition holds. Since the conditional paths are merged with the exclusive gateway in front of the last task, the condition no longer holds and the anti-pattern should be reported. However, since a path exists from the start of the process to the last task that is annotated with the C1 condition, the anti-pattern would not be reported using the previously described naive check. This issue can be fixed by annotating the active conditions for all sequence flows when transforming the process model into a graph. It is then possible to check whether all sequence flows across all paths from the source node to the target node of a nonsequential flow specify the condition or not.

Prototypical Implementation

Each chapter of the solution design in Part II of the thesis describes insights from a (partial) prototypical implementation to demonstrate the technical feasibility of the solution design. While the prototypical implementation of Chapters 5 and 6 has not been published before, the prototypical implementation of Chapter 7 has already been fundamentally described and published in the referenced papers ([KGE22; KE22]). For central access, all implementations are aggregated into a single code repository, which is available at:

`https://github.com/krchf/dse-poc`

The central `README.md` file describes the structure of the repository and how to execute the software artifacts, including the documentation of any required software prerequisites.