universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Supporting Behavioral Consistency in Adaptive Case Management

verfasst von / submitted by

Christoph Kaineder

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr.techn.)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dr. Uwe Zdun

Vita

From 2014 to 2018, Christoph Kaineder was a research associate at the Faculty of Computer Science at the University of Vienna, Austria. In December 2013 he received a master's degree in computer science. Christoph has published more than 15 peer-reviewed scientific papers, and has participated in two research projects, CACMTV (Content-Aware Coding for Mobile TV) and CACAO (Consistency Checking, Recommendations, And Visual Modeling For Ad Hoc Changes By Knowledge Workers In Adaptive Case Management).

Acknowledgment

First of all, I would like to thank my advisor Prof. Uwe Zdun for his continuous support, patience, and encouragements during the past years, which shaped my scientific thinking and work. I am also very grateful for the excellent working environment. It has always been a pleasure to spend time with my colleagues, both at work and in my free time. Thank you to all co-authors for their help and contributions. I would also like to acknowledge the work of the anonymous reviewers of my scientific contributions, which helped me to continuously rethink and improve my work. Many thanks to all those who were willing to participate in experiments, to the FFG (Die Österreichische Forschungsförderungsgesellschaft) for the funding of the CACAO project (no. 84346) and to ISIS Papyrus for the professional cooperation and additional resources.

Finally, I would like to mention the two most important people of my life, my son Konstantin born in September 2017 and my wife Gerlinde. Thank you for your patience and support. I am happy that we are a family.

Abstract

Adaptive Case Management (ACM) is part of an ongoing trend in the field of business process management that aims to enable highly flexible process-aware information systems. ACM follows a non-prescriptive paradigm. That is, the business users have the freedom to decide what is to be done in order to achieve a business goal. Consequently, processes in ACM range between semi-structured and completely unstructured, and ad hoc actions can be performed in reaction to unforeseeable events.

This high degree of flexibility comes along with challenges regarding behavioral consistency. Many knowledge-intensive domains are subject to a vast and ever-growing amount of behavioral constraints stemming from sources such as laws, standards, business contracts, and best practices, to name but a few. To preserve the high flexibility of ACM during case execution, such behavioral constraints must be integrated with great care. In particular, it must be avoided to introduce any kind of rigidness that would pose an obstacle for flexibility during case execution since flexibility is the core feature and a major selling point of ACM. Moreover, during the design of case templates, which can be used as basis for case execution, business administrators are challenged not to introduce errors that would lead to behavioral inconsistencies during case execution. While the state of the art provides numerous approaches for behavioral verification of flow-driven business processes, case modeling is a relatively new paradigm with a lack of sufficient support for behavioral verification.

This thesis aims to make formal/technical as well as empirical contributions to support behavioral consistency in ACM. To support business administrators during the creation or amendment of case templates, this thesis proposes an approach for the efficient behavioral verification of case templates based on reduction techniques and model checking. By offering a business-driven approach based on business ontology and high-level behavioral patterns for the creation of behavioral constraints, business users are enabled to specify behavioral constraints using the language and terminology of their business domain. This approach shifts the control over the implemented constraints to the business domain, and the constraints are used during case execution for supporting the business users in staying consistent with desired behaviors. On the technical side, behavioral constraints are often formalized in temporal logics, most notably Linear Temporal Logic (LTL). While business users are shielded from the complexity of temporal logics by

highly abstracted patterns, technical users who implement these patterns are challenged to write correct LTL formulas. To support the creation of correct LTL formulas, plausibility checking of LTL formulas is proposed. The plausibility checking approach makes use of Event Processing Language (EPL) for the creation of plausibility specifications. The results of three controlled experiments on the understandability of behavioral constraint representations (with 70, 146 and 215 participants) suggest that EPL specifications are significantly more understandable than LTL specifications. The same empirical studies also suggest that the highest degree of understandability is provided by using a pattern-based approach for behavioral consistency specifications, which provides evidence that this approach is highly suitable for defining behavioral constraints in ACM. Another empirical study with 116 participants focused on the understandability of textual and graphical pattern-based behavioral constraint representations. The results of this study suggest that there are no significant differences in understandability between graphical and textual pattern-based behavioral constraint representations.

Zusammenfassung

Adaptive Case Management (ACM) ist Teil eines sich abzeichnenden Trends im Bereich des Geschäftsprozessmanagements, der hochflexible, prozess-getriebene Informationssysteme ermöglichen soll. ACM folgt dabei einem nicht-präskriptiven Paradigma, was bedeutet, dass die Geschäftsbenutzer die Freiheit haben zu entscheiden, was zu tun ist, um ein Geschäftsziel zu erreichen. Folglich sind die Prozesse in ACM halbstrukturierten bis vollständig unstrukturiert, und Ad-hoc-Aktionen können als Reaktion auf unvorhersehbare Ereignisse ausgeführt werden.

Diese hohe Flexibilität ist mit Herausforderungen hinsichtlich der Konsistenz mit erwünschtem Verhalten während der Fallausführung verbunden. Viele wissensintensive Bereiche unterliegen einer großen und ständig wachsenden Anzahl von Verhaltensbeschränkungen, die sich aus Quellen wie Gesetzen, Standards, Geschäftsverträgen und Best Practices ergeben, um nur einige zu nennen. Um die hohe Flexibilität von ACM während der Umsetzung von Fällen zu bewahren, müssen solche Verhaltensbeschränkungen mit großer Sorgfalt integriert werden. Insbesondere muss vermieden werden, dass irgendeine Art von Starrheit eingeführt wird, die ein Hindernis für die Flexibilität bei der Behandlung von Fällen darstellen würde, da Flexibilität das Kernmerkmal und ein wichtiges Verkaufsargument von ACM ist. Bei der Entwicklung von Fallvorlagen, die als Grundlage für die Umsetzung von Fällen verwendet werden können, werden Geschäftsadministratoren darüber hinaus aufgefordert, keine Fehler einzuführen, die zu Verhaltensinkonsistenzen während der Fallausführung führen würden. Während der Stand der Technik zahlreiche Ansätze für die Verhaltensverifizierung von flussgetriebenen Geschäftsprozessen bietet, ist die Fallmodellierung ein relativ neues Paradigma mit einem Mangel an ausreichender Unterstützung für die Verhaltensverifizierung.

Diese Arbeit zielt darauf ab, formale/technische sowie empirische Beiträge zur Unterstützung der Verhaltenskonsistenz in ACM zu leisten. Zur Unterstützung von Unternehmensadministratoren bei der Erstellung oder Änderung von Fallvorlagen wird in dieser Arbeit ein Ansatz zur effizienten Verhaltensverifikation von Fallvorlagen vorgeschlagen, der auf Reduktionstechniken und Modellprüfungen basiert. Durch die Bereitstellung eines geschäftsorientierten Ansatzes auf der Grundlage von Geschäftsontologie und hoch abstrahierte Verhaltensmustern zur Schaffung von Verhaltensbeschränkungen können Geschäftsbenutzer Verhaltensbeschränkungen anhand der Sprache und Terminologie ihres Geschäftsbereichs festlegen. Dieser Ansatz

verlagert die Kontrolle über die implementierten Einschränkungen auf die Geschäftsdomäne, und die Einschränkungen werden während der Fallausführung verwendet, um die Geschäftsbenutzer dabei zu unterstützen, mit den gewünschten Verhaltensweisen konsistent zu bleiben. Auf der technischen Seite werden Verhaltensbeschränkungen häufig in zeitlichen Logiken formalisiert, vor allem in Linear Temporal Logic (LTL). Während Geschäftsbenutzer durch hoch abstrahierte Muster von der Komplexität temporaler Logiken abgeschirmt sind, sind technische Benutzer, die diese Muster implementieren, gefordert, korrekte LTL-Formeln zu schreiben. Um die Erstellung korrekter LTL-Formeln zu unterstützen, wird eine Plausibilitätsprüfung von LTL-Formeln vorgeschlagen. Der Plausibilitätsprüfungsansatz verwendet Event Processing Language (EPL) zur Erstellung von Plausibilitätsspezifikationen. Die Ergebnisse von drei kontrollierten Experimenten zur Verständlichkeit von Verhaltensbeschränkungsdarstellungen (mit 70, 146 und 215 Teilnehmern) legen nahe, dass EPL-Spezifikationen wesentlich verständlicher sind als LTL-Spezifikationen. Die gleichen empirischen Studien deuten auch darauf hin, dass der höchste Grad an Verständlichkeit durch die Verwendung eines musterbasierten Ansatzes für Verhaltenskonsistenzspezifikationen gegeben ist, was auch zeigt, dass dieser Ansatz sehr geeignet ist, um Verhaltensbeschränkungen in ACM zu definieren. Eine weitere empirische Studie mit 116 Teilnehmern konzentrierte sich auf die Verständlichkeit von textuellen und graphischen Muster-basierten Verhaltensbeschränkungsdarstellungen. Die Ergebnisse dieser Studie deuten darauf hin, dass es keine signifikanten Unterschiede in der Verständlichkeit von textuellen und graphischen Muster-basierten Verhaltensbeschränkungsdarstellungen gibt.

Contents

Vita i

Acknowledgment iii

Abstract v

Zusammenfassung vii

I. Introduction 1

1. Introduction 3

1.1. Adaptive Case Management 4
1.2. Behavioral Consistency 5
1.3. Motivation 5
1.4. Problem Statement 6
1.5. Research Questions 8
1.6. Research Methodology 12
1.7. Thesis Outline and Publications 13

II. Studies on Supporting Behavioral Consistency in ACM 21

2. Behavioral Consistency Checking of Case Management Models 23

2.1. Introduction 23
2.2. Motivation 24
2.3. Approach Overview 26
2.4. Formalization of Case Templates 28
2.5. Case Element Reduction 29
2.6. Condition Reduction 33
2.7. Model Transformation 35
2.8. Experimental Results 40

2.9. Discussion . . . 41
2.10. Related Work . . . 44
2.11. Conclusion and Future Work . . . 45

3. Business-Driven Behavioral Constraint Authoring 47
3.1. Introduction . . . 47
3.2. Motivating Example . . . 49
3.3. Approach . . . 50
3.4. Practical Scenario . . . 54
3.5. Discussion . . . 60
3.6. Related Work . . . 61
3.7. Implementation . . . 64
3.8. Conclusion and Future Work . . . 64

4. Behavioral Consistency Support Framework 65
4.1. Introduction & Motivation . . . 65
4.2. Framework Overview . . . 66
4.2.1. Recommendation Feedback Loop . . . 69
4.2.2. Enactment Feedback Loop . . . 70
4.2.3. Elicitation Feedback Loop . . . 70
4.3. Framework Components . . . 70
4.3.1. Ontology . . . 70
4.3.2. Constraint Elicitation & Constraint Authoring . . . 71
4.3.3. Case Enactment . . . 75
4.3.4. Constraint Enactment . . . 75
4.3.5. Recommendation of Actions . . . 76
4.4. Implementation . . . 79
4.5. Discussion . . . 81
4.6. Conclusion & Future Work . . . 81

5. Plausibility Checking of Behavioral Constraints Formalized in Linear Temporal Logic 83
5.1. Introduction . . . 83
5.2. Related Work . . . 85
5.3. Preliminaries . . . 86
5.3.1. Linear Temporal Logic (LTL) . . . 86
5.3.2. Event Processing Language (EPL) . . . 88

5.4. Plausibility Checking Approach . . . 89
5.5. Reviewing Existing LTL Patterns . . . 90
5.6. Creation of LTL Formulas . . . 91
5.7. Discussion . . . 93
5.8. Conclusion and Future Work . . . 94

III. Empirical Studies on the Understandability of Behavioral Constraint Representations 95

6. On the Understandability of Behavioral Constraints Formalized in Linear Temporal Logic, Event Processing Language and Property Specification Patterns 97

6.1. Introduction . . . 97
6.1.1. Problem Statement . . . 99
6.1.2. Research Objectives . . . 100
6.1.3. Context . . . 100
6.1.4. Guidelines . . . 101
6.2. Background on Behavioral Constraint Representations . . . 101
6.2.1. Linear Temporal Logic (LTL) . . . 102
6.2.2. Property Specification Patterns (PSP) . . . 103
6.2.3. Event Processing Language (EPL) . . . 105
6.3. Experiment Planning . . . 106
6.3.1. Goals . . . 106
6.3.2. Experimental Units . . . 106
6.3.3. Experimental Material & Tasks . . . 107
6.3.4. Hypotheses, Parameters, and Variables . . . 109
6.3.5. Experiment Design & Execution . . . 110
6.3.6. Procedure . . . 111
6.4. Analysis . . . 112
6.4.1. Data Set Preparation . . . 112
6.4.2. Descriptive Statistics . . . 112
6.5. Statistical Inference . . . 126
6.6. Analysis of Qualitative Data . . . 132
6.7. Discussion . . . 134
6.7.1. Evaluation of Results and Implications . . . 134

6.8. Threats to Validity . . . 138
6.8.1. Threats to Internal Validity . . . 138
6.8.2. Threats to External Validity . . . 140
6.8.3. Threats to Construct Validity . . . 142
6.8.4. Threats to Content Validity . . . 144
6.8.5. Threats to Conclusion Validity . . . 144
6.9. Related Work . . . 145
6.10. Conclusion and Future Work . . . 146
6.10.1. Summary . . . 146
6.10.2. Impact . . . 147
6.10.3. Future Work . . . 147

7. Modeling Compliance Specifications in Linear Temporal Logic, Event Processing Language and Property Specification Patterns 149
7.1. Introduction . . . 149
7.1.1. Problem Statement . . . 150
7.1.2. Research Objectives . . . 151
7.1.3. Guidelines . . . 153
7.2. Background . . . 153
7.3. Experiment Planning . . . 153
7.3.1. Goals . . . 153
7.3.2. Experimental Units . . . 154
7.3.3. Experimental Material & Tasks . . . 154
7.3.4. Hypotheses, Parameters, and Variables . . . 154
7.3.5. Experiment Design & Execution . . . 158
7.3.6. Procedure . . . 159
7.4. Analysis . . . 159
7.4.1. Data Set Preparation . . . 162
7.4.2. Descriptive Statistics . . . 162
7.4.3. Statistical Inference . . . 181
7.5. Discussion . . . 188
7.5.1. Evaluation of Results and Implications . . . 188
7.5.2. Threats to Validity . . . 190
7.6. Related Work . . . 195
7.7. Conclusion and Future Work . . . 197

8. On the Understandability of Graphical and Textual Pattern-Based Behavioral Constraint Representations **199**
8.1. Introduction . . . 199
8.1.1. Problem Statement . . . 201
8.1.2. Research Objectives . . . 202
8.1.3. Guidelines . . . 203
8.2. Background on Pattern-Based Behavioral Constraint Representations . . . 203
8.2.1. Property Specification Patterns . . . 203
8.2.2. Declare . . . 204
8.3. Experiment Planning . . . 209
8.3.1. Goals . . . 209
8.3.2. Experimental Units . . . 210
8.3.3. Experimental Material & Tasks . . . 210
8.3.4. Hypotheses, Parameters, and Variables . . . 212
8.3.5. Experiment Design . . . 215
8.3.6. Procedure . . . 215
8.4. Analysis . . . 215
8.4.1. Data Set Preparation . . . 215
8.4.2. Analysis of Previous Knowledge, Experience and Other Features of Participants . . . 216
8.4.3. Descriptive Statistics of Dependent Variables . . . 220
8.5. Statistical Inference . . . 227
8.6. Analysis of Free Text Answers . . . 231
8.7. Discussion . . . 233
8.7.1. Evaluation of Results and Implications . . . 233
8.7.2. Threats to Validity . . . 234
8.8. Related Work . . . 239
8.8.1. Empirical Studies on the Understandability of Behavioral Constraint Representations in Software Architecture and Software Engineering . . . 240
8.8.2. Empirical Studies on the Understandability of Behavioral Constraint Representations in Business Process Management . . . 241
8.9. Conclusion and Future Work . . . 243
8.9.1. Summary . . . 243
8.9.2. Impact . . . 243
8.9.3. Future Work . . . 244

IV. Conclusions **247**

9. Conclusions and Future Work **249**

9.1. Research Questions Revisited 249
9.2. Limitations 253
9.3. Future Work 255

A. Appendix **257**

A.1. Experiment on Graphical and Textual Behavioral Constraint Representations — Sample Solutions of Experimental Tasks 257
A.2. Experiment on Graphical and Textual Behavioral Constraint Representations — Evaluation of Normality Assumption & Parametric Testing by Welch's t-test . . 261

Part I.

Introduction

1. Introduction

Process-aware information systems, which manage and execute processes on the basis of process models (cf. van der Aalst [1]), are increasingly applied for the automation of processes. Many process-aware information systems as well as other implementations of behavioral models (such as UML activity diagrams [2]) used today are prescriptive, which makes their application difficult in scenarios demanding ad hoc actions and on-the-fly changes of processes.

An emerging trend in supporting semi-structured or unstructured (i.e., ad hoc) processes is concerned with shifting the focus from a prescriptive paradigm to a non-prescriptive paradigm (cf. Swenson [3]). In a prescriptive paradigm, the steps needed to achieve a certain business goal are determined beforehand (cf. Schonenberg et al. [4]). On the contrary, in a non-prescriptive paradigm, the business users have a freedom to decide what is to be done in order to achieve a business goal. The specified business goal will eventually be enacted through operational steps such as performing ad hoc tasks, selecting process fragments, accessing data records, etc. Consequently, in the non-prescriptive paradigm, processes in execution can be changed in any appropriate way such that goals are eventually achieved. Adaptive Case Management (ACM) [5] is a prominent example of a realization of the non-prescriptive paradigm that is addressed in this thesis. ACM is part of an emerging trend in the field of business process management, aimed at supporting highly flexible, knowledge-intensive software systems (cf. Schonenberg et al. [4], van der Aalst & Weske [6], and Swenson [5]). ACM can generally be considered as one of the few techniques enabling flexible and knowledge-intensive business processes that already has seen significant industry adoption (cf. Pucher [7]). ACM aims at providing knowledge, methods, techniques, and infrastructure for supporting business processes that are unstructured and unpredictable in their execution. That is, they can be driven by unknown events or might require the ad hoc inclusion of new actions in reaction to those events. The major difference between ACM and the prescriptive paradigm is that the main focus of ACM are cases, which denote a specific situation or a set of circumstances that requires a set of actions driven by goals to achieve an acceptable outcome or objective. ACM enables the ability to turn technical and production workers into knowledge workers (a recurring term for business users in ACM terminology) and it emphasizes the use of business knowledge to address unpredictable cases (cf. Swenson [3]).

While a non-prescriptive approach, such as ACM, certainly provides additional freedom in adapting processes as they run, it also brings along several challenges. For instance, important behavioral constraints (e.g., stemming from compliance rules) can be overlooked or accidentally disabled by business users, or an ad hoc action can be inconsistent with other parts of the case. Even worse, the business knowledge required for detecting and resolving such problematic incidents is merely kept in the heads of the knowledge workers (as implicit knowledge) instead of being formally documented.

1.1. Adaptive Case Management

Marin et al. [8] aggregate existing definitions of ACM (e.g., Swenson [3] and Motahari-Nezhad & Swenson[9]) and define ACM as *"a practice for knowledge-intensive processes with a case folder as central repository, whereas the course of action for the fulfillment of goals is highly uncertain and the execution gradually emerges according to the available knowledge base and expertise of knowledge workers"*.

According to Ciccio et al. [10], knowledge-intensive processes have the following characteristics:

- *Knowledge-driven*: Available knowledge drives decision making and taken actions.
- *Collaboration-oriented*: Working on a case is a social process, involving knowledge workers with diverse backgrounds or roles.
- *Event-driven*: Actions are taken in reaction to events.
- *Unpredictable*: Not all events and actions comprising a case are known a priori.
- *Emergent*: The actual course of actions gradually emerges during case execution.
- *Goal-oriented*: Knowledge workers try to achieve goals or work towards milestones.
- *Constraint- and rule-driven*: Taken actions (or their absence) must comply with behavioral constraints and rules.
- *Non repeatable*: Each case instance is unique and hardly repeatable.

ACM is an agile approach to business process management. Instead of imperative, procedural process models, which prescribe *how* things are to be done, it specifies the desired outcome (the *what*), which may be adapted in the course of case execution. A case can be instantiated on the

basis of a case template, which models predictable aspects of a case as a starting point, or case execution can be completely ad hoc.

Interestingly, when we consider how we do things in our everyday life, it appears to be quite similar to the ACM approach. For example, when we are hungry, the goal is to eat, but the process how to get food usually is different from day to day. Sometimes we go to the supermarket, sometimes to a restaurant. We adapt in case a restaurant is closed or does not have any seats left. We might have to align with colleagues that are interested in joining us for lunch. We might have unforeseeable interactions with a waiter, for example, in the event of a fork falling down to the floor and we ask for a new one. Last but not least, we have to pay before leaving the restaurant, which can be seen as a constraint or rule. Even this trivial example already shows the high complexity involved in trying to model each and every aspect or event that possibly can occur a priori.

1.2. Behavioral Consistency

The high flexibility of ACM does not imply that all behaviors are allowed. Being consistent with desired behaviors means that certain constraints stemming from sources such as laws, regulations, standards, best practices, or internal policies must be respected. Otherwise we might run into problems such as litigation, loss of certification, loss of job, etc.

Which behaviors are desired or forbidden depends largely on the (business) domain. A frequently-used example in the literature is behavioral compliance in the corporate and financial sector, most notably prescribed by the Sarbanes-Oxley Act of 2002 (SOX) [11] in reaction to major corporate accounting scandals in the U.S. (e.g., Enron and WorldCom) and Basel III [12] in response to weaknesses in financial regulation responsible for the financial crisis in 2007/2008. Another example of heavily regulated domains is the construction industry. Compliance rules in this domain are often related to occupational safety and health. For example, certain precautions and safe practices are required if a lead contamination (commonly caused by lead-based paint) is present or to be presumed in buildings built before 1978 that undergo renovation (cf. United States Environmental Protection Agency's Lead-Based Paint Renovation, Repair and Painting Rule [13]). Another example is the health care sector: Processes in hospitals must comply with state-of-the-art medical knowledge and treatment procedures (e.g., Rovani et al. [14]).

1.3. Motivation

Current ACM solutions focus on enabling a high degree of flexibility, but they seem to often fail in providing adequate support for control and consistency with desired behaviors. Therefore,

it is not surprising that the application of ACM, as a non-prescriptive, flexible approach, is still often seen skeptically, especially in domains that are subject to a large amount of behavioral constraints. Despite running ACM solutions, many companies are still reluctant to allow dynamic changes and ad hoc actions due to the low level of control over these changes and the high risk of subsequent behavioral inconsistencies.

For example, customers of a bank might make a phone call to request assistance with tasks that they cannot or are not willing to perform with another contact channel of the bank. This kind of communication procedure is unofficially tolerated for well-known customers who can be recognized by voice and phone number. In other cases, an identity validation protocol must take place on the phone (e.g., asking a specific question with an answer that is only known by the customer). In every case, the customer has to sign a consent form for achieving business compliance. For many interactions, it is accepted by banks for well-known customers to officially sign the consent form at a later time at one of the bank's branches. It is crucial to support such ad hoc actions for the sake of customer satisfaction, even though this sometimes temporarily leads to non-conformance with regards to regulations or internal policies of the bank. Consequently, to enable both flexibility and consistency, the bank could introduce behavioral constraints that must not be violated. The customer's identity must be validated at the time of any ad hoc action, and a consent form must be signed eventually after an ad hoc action or a set of ad hoc actions.

A major challenge in non-prescriptive approaches like ACM is that the consistency of cases after ad hoc actions is at stake in the absence of automatic behavioral consistency checking. In consequence, there is a reluctance of companies and institutions to enable their business users to perform ad hoc changes since they could make mistakes that might lead to inconsistent cases. This thesis aims to provide support for behavioral consistency in ACM. An important additional challenge in this context is that business users are usually not technical experts, meaning that it is necessary to provide understandable modeling concepts and adequate checking support for enabling behavioral consistency in ACM.

1.4. Problem Statement

Rigidness of Processes

Despite the existing demand for business process flexibility, prescriptive process-aware information systems are still often preferred, even in knowledge-intensive environments. Companies seem to value the certainty of predefined structures and procedures, especially in domains where compliance plays an important role (e.g., the banking sector). A potential downside of this rigidness might be that as soon as a demand for activities that the software system does not provide

arises, business users are tempted to perform manual tasks independently from the software system. That is, such activities are not recorded, which is problematic with regards to compliance, and the users are not supported at all by the IT system while performing such manual actions.

Hard-Coded Behavioral Consistency

Hard-coding of mechanisms enforcing behavioral consistency in models and program code leads to consistency problems, and to poor maintainability and traceability between compliance specifications, internal policies, models, and the source code, especially when compliance specifications change frequently. Long maintenance cycles could result in the week- or month-long enforcement of outdated behavioral constraints, while at the same time providing no support for currently valid behavioral constraints.

Lack of Behavioral Consistency Support during Case Modeling

While there exist numerous approaches for the verification of flow-driven business process models (e.g., Eshuis [15], Kherbouche et al. [16], Sbai et al. [17], and Raedts et al. [18]), the verification of case models has not been studied sufficiently. A major reason for that might be the relatively brief existence of case modeling and associated standardization efforts. The Case Management Model and Notation (CMMN) standard (cf. Object Management Group [19]) was released in May 2014 and revised in Version 1.1 in December 2016. Case modeling is in many ways different from flow-driven business process modeling (e.g., BPMN [20]). Case models are usually loosely structured by dependencies and goal-driven. The start or completion of activities and goals (also called milestones) can be guarded by conditions regarding the current data state of a case.

Lack of Behavioral Consistency Support during Case Execution

If the flexibility of ACM is taken advantage of, case execution is either based on a specific case model with the option to deviate from the predesigned model, or can be completely ad hoc. Obviously, in case of deviations from the designed case template (i.e., the fundamental model for case execution) or in ad hoc scenarios, there is a large chance for running into behavioral inconsistencies if there is no adequate support available.

Lack of Support for Business-Driven Behavioral Consistency Modeling

Behavioral consistency is clearly a business issue, yet this responsibility is often shifted to technical/IT departments, which may result in numerous problems (e.g., lack of traceability, poor

maintainability, inaccessibility). A major obstacle for enabling business users to model behavioral constraints is the lack of an understandable approach for the modeling of behavioral constraints.

Lack of Support to Make Implicit Knowledge Explicit

Business users, especially those with many years of experience, are usually a rich source of knowledge, but this knowledge often remains implicit, and thus hidden or not immediately accessible for other, less experienced business users. Implicit knowledge can be related to adhering to existing behavioral constraints which are not explicitly present in the IT system or to the compensation of behavioral constraint violations. Explicitly capturing this implicit knowledge as well as making it available to other business users has the potential to improve the behavioral consistency of the ACM software.

1.5. Research Questions

The main research question of this thesis can be broadly stated as "*How can behavioral consistency be supported in ACM?*". More specifically, there are the following five research questions (and their refinements where appropriate):

Research Question 1 (RQ 1)

How can behavioral consistency during case modeling be supported?

An established and wide-spread technique for model verification is model checking. It explores all possible execution traces of a model described by a state-transition system. Any undesired behavior is detected, provided the specification is correctly defined.

For the refinement of RQ 1, we state the following research questions:

Research Question 1.1 (RQ 1.1)

How can the behavior of case templates be captured as a state-transition system for model checking?

While the transformation of flow-driven business process models to state-transition systems is well-studied (e.g., Eshuis [15], Köhler et al. [21], and Kherbouche et al. [16]), the relatively new case modeling paradigm has not been covered sufficiently. The rich semantics and the strong focus on data make case templates a challenging source for state-transition system transformation.

Research Question 1.2 (RQ 1.2)

Since model checking is known to be computationally expensive, how can case templates be model checked efficiently?

Model checking has a downside, namely its high computational complexity, which grows exponentially with the problem size (cf. Sistla & Clarke [22]). Consequently, a 1:1 mapping of a case template, considering all its elements and possible data values, to a state-transition system would not be a smart choice as it most likely would lead to a state space explosion already for small models. That is, the challenge is to find a suitable level of abstraction that keeps the state space at a feasible size for computation with current hardware.

Research Question 1.3 (RQ 1.3)

Can model checking be applied to real-world case templates that are realistic in size and structure?

The performance evaluation of the behavioral verification approach focuses on existing real-world case templates. We consider it crucial for practical exploitability that the approach can yield results within acceptable time boundaries.

Research Question 2 (RQ 2)

How can business domains and their terminology be considered during behavioral constraint modeling?

On the one hand existing behavioral consistency approaches usually do not consider the business domain and its terminology adequately (cf. Elgammal et al. [23]), on the other hand business rule approaches do not support temporal behavioral aspects sufficiently (cf. OMG [24]). Hence, the goal of this research question is to study the integration of business domain specific aspects with behavioral constraint approaches.

Research Question 3 (RQ 3)

How can behavioral consistency during case execution be supported?

Besides the design time of case templates, there are important runtime aspects that must be taken into account for enabling behavioral consistency. Running case instances may deviate from templates and behavioral constraints focusing on runtime data cannot be checked at design time.

RQ 3 is further refined as follows:

Research Question 3.1 (RQ 3.1)

How can prescriptiveness be avoided while providing support for behavioral consistency?

A key aspect of ACM is its flexibility. Imposing behavioral constraints bears the risk of introducing prescriptiveness. That is, it is challenging to position behavioral constraints as a supportive measure rather than an obstacle for case progression.

Research Question 3.2 (RQ 3.2)

How can implicit knowledge be leveraged for user guidance (i.e., the recommendation of next actions)?

It is a commonly known fact that knowledge often remains implicit. That is, transferring such knowledge is a challenging task. As ACM usually does not provide a step-by-step guidance for users as flow-driven approaches would provide, users might run into problems regarding deciding how to handle a specific situation, especially when the behavioral consistency of a case is at stake. In this context, the challenge is to capture formerly implicit knowledge and to make it available for supportive measures to achieve behavioral consistency.

Research Question 4 (RQ 4)

How can it be supported that a formal behavioral constraint specification matches the intent of its creator?

Formal behavioral constraints are often abstracted by patterns that specify a specific high-level intent (e.g., *something must happen in response to...*) in the attempt to make them more accessible by hiding the complexity of formal/technical languages. Nonetheless, patterns must provide a formal/technical representation to enable automated verification support. Linear Temporal Logic (LTL) is widely used for that purpose (cf. Reichert & Weber [25]), but it could be the case that the LTL representation does not match the intent of its pattern-based representation. That is, the LTL formula could be incorrect, and an incorrect specification would make the behavioral verification of cases a pointless exercise. Consequently, in this research question, we study supportive measures for creating correct LTL specifications.

Research Question 5 (RQ 5)

How understandable are existing representative behavioral constraint languages?

Usually ACM users are not technical or formal language experts. Therefore, easy-to-understand modeling concepts must be provided. Very little is known on how understandable existing major

specification languages for behavioral constraints are.

RQ 5 is further refined as follows:

Research Question 5.1 (RQ 5.1)

How understandable are existing behavioral constraints modeled in different major languages, and are there significant differences?

Despite the long existence of many major behavioral constraint specification approaches (e.g., Linear Temporal Logic [26] was first proposed in 1977, the Property Specification Patterns [27] exist since 1999), the core focus of most researchers has been on the formal/technical perspective of those approaches, whereas studying the usage point of view from an empirical perspective has not drawn much attention from researchers. Indeed, we are not aware of any existing work that provides an empirical study on the understandability of different representative behavioral constraint specification approaches. Gaining more insights into the understandability of behavioral constraint representations is crucial for evaluating their suitability for practical use and finding potential ways for their improvement with regards to understandability.

Research Question 5.2 (RQ 5.2)

How understandable are major behavioral constraint languages when applied for modeling, and are there significant differences?

While RQ 5.1 focuses on evaluating the understandability of existing behavioral constraints, RQ 5.2 covers the understandability of representative behavioral constraint approaches in the context of behavioral constraint authoring.

Research Question 5.3 (RQ 5.3)

Is there a difference in understandability between graphical and textual behavioral constraint approaches?

Comparisons of graphical and textual languages seem to yield contradictory results. The descriptive results of a study by Haisjackl & Zugal [28] could indicate that the graphical declarative business processes are advantageous (compared to textual declarative ones) in terms of perceived understandability, error rate, duration, and mental effort. In a study by Heijstek et al. [29], participants who predominantly used textual architecture descriptions performed significantly better than those using graphical ones. A study by Sharafi et al. [30] on the understandability of graphical and textual software requirement models did not reveal any statistically significant differences. Since existing studies report inconclusive results, we consider it impor-

tant to further study whether there are differences in understandability between graphical and textual behavioral constraint approaches.

1.6. Research Methodology

Design Science Research

The research conducted in the course of this thesis is driven by the *design science research* methodology (cf. March & Smith [31]). In this context, the term *design* has a dual meaning, namely the process of designing, and the design (i.e., artifact or product) as result of this process. According to March & Smith [31], the two processes are *build* (i.e., creating an artifact) and *evaluate* (i.e., evaluation of the design) whereas the artifacts can be *constructs* (i.e., the conceptual vocabulary of a problem domain), *models* (i.e., the relationship between constructs), *methods* (i.e., a solution to a problem – e.g., an algorithm or guideline), and *instantiations* (i.e., implementations of constructs, models, or methods in a working system). Hevner et al. [32] propose seven guidelines for applying the design science methodology in information systems research, namely *Design as an Artifact*, *Problem Relevance*, *Design Evaluation*, *Research Contributions*, *Research Rigor*, and *Design as a Search Process*, *Communication of Research*. Peffers et al. [33] integrate among others the guidelines of Hevner et al. [32] which results in six design science process activities, namely *Problem identification and motivation*, *Objectives of a solution*, *Design and development*, *Demonstration*, *Evaluation*, and *Communication*. They state that "*[...] a design research artifact can be any designed object in which a research contribution is embedded in the design*".

Controlled Experiments

A controlled experiment is an empirical study in which (an) independent variable(s) are manipulated to measure the effect on (a) dependent variable(s) with regard to well-defined testable hypotheses (cf. Wohlin et al. [34]). It is designed to minimize the effects of variables other than the independent variables (i.e., unobserved variables or bias). In our studies, we use a single independent variable consisting of different treatments, namely the behavioral constraint representations, and study the effect of these treatments on the construct understandability, which is comprised of the dependent variables (syntactic/semantic) correctness and response time.

1.7. Thesis Outline and Publications

This dissertation is based on research work which has been either published in or submitted to a research venue (i.e., under review). The following publications contributed to the chapters of this dissertation:

Part II is concerned with contributing to the behavioral consistency in ACM by providing design time and runtime checking support as well as support for behavioral constraint authoring.

Chapter 2 discusses a model checking approach for the behavioral verification of ACM case models. It addresses RQ 1 (*How can behavioral consistency during case modeling be supported?*). To counteract the high computational demands of model checking techniques, the proposed approach includes state space reduction techniques as a preprocessing step before state-transition system generation. Consequently, the problem size is decreased, which decreases the computational demands needed by the subsequent model checking as well. An evaluation of the approach with a large set of LTL specifications on two real-world case models, which are representative for semi-structured and structured process models and realistic in size, shows an acceptable performance of the proposed approach. This chapter is based on the following papers:

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Reduction Techniques for Efficient Behavioral Model Checking in Adaptive Case Management" In: *Proceedings of the Symposium on Applied Computing (SAC '17), Marrakech, Morocco — April 03 - 07, 2017, ISBN: 978-1-4503-4486-9.*
  DOI: 10.1145/3019612.3019617
- Christoph Czepa, Uwe Zdun, "Behavioral Consistency Checking of Case Templates in Adaptive Case Management" (submitted to journal)

In Chapter 3, we discuss an approach that seeks to enable business-driven behavioral constraint authoring by combining business ontology and behavioral constraint patterns. Consequently, the chapter is concerned with RQ 2 (*How can business domains and their terminology be considered during behavioral constraint modeling?*). The resulting language is close to natural language and can be used for automated verification. This chapter is based on the following paper:

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Ontology-Based Behavioral Constraint Authoring" In: *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW), ISBN: 978-1-4673-9933-3.*
  DOI: 10.1109/EDOCW.2016.7584380

Chapter 4 proposes a behavioral consistency support framework. It discusses the necessary components and how they interact to enable behavioral consistency support during case execution. Consequently, this chapter is mainly concerned with RQ 3 (*How can behavioral consistency during case execution be supported?*). This chapter is based on the following paper:

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Towards a Compliance Support Framework for Adaptive Case Management" In: *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW), ISBN: 978-1-4673-9933-3*. DOI: 10.1109/EDOCW.2016.7584390

Chapter 5 proposes a plausibility checking approach for behavioral constraints formalized as Linear Temporal Logic (LTL) formulas. Consequently, this chapter is concerned with RQ 4 (*How can it be supported that a formal behavioral constraint specification matches the intent of its creator?*). Particularly, it is difficult to create a correct LTL formula that really matches the intent of its creator. LTL is a popular way to define compliance rules (cf. Reichert & Weber [25]), and it is a widely used specification language commonly applied in model checking (e.g., Cimatti et al. [35]). Plausibility checking makes use of Event Processing Language (EPL) based plausibility specifications, which are significantly more understandable than LTL formulas (cf. Chapter 6 & Chapter 7). This chapter is based on the following papers:

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Plausibility Checking of Formal Business Process Specifications in Linear Temporal Logic" In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), Ljubljana, Slovenia, June 13-17, 2016, ISSN: 1613-0073*. URN: urn:nbn:de:0074-1612-0

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Plausibility Checking of Formal Business Process Specifications in Linear Temporal Logic (Extended Abstract)" In: *Proceedings of the 7th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA 2016), Vienna, Austria, October 3-4, 2016, ISSN: 1613-0073*. URN: urn:nbn:de:0074-1701-6

In Part III, we will present several empirical studies on the understandability of major behavioral constraint representations. Very little is known on how understandable existing major specification languages for behavioral constraints are. ACM users, who are usually not technical or formal methods experts, should be only exposed to a behavioral constraint representation that is highly understandable. Another objective of these studies is the evaluation of the plausibility checking approach that is based on the hypothesis that plausibility specifications are easier to

create than temporal logic formulas, and to evaluate differences in understandability of graphical and textual pattern-based approaches.

Chapter 6, we study the understandability of three major temporal property representations: (1) Linear Temporal Logic (LTL) is a formal and well-established logic that offers temporal operators to describe temporal properties; (2) Property Specification Patterns (PSP) are a collection of recurring temporal properties that abstract underlying formal and technical representations; (3) Event Processing Language (EPL) can be used for runtime monitoring of event streams using Complex Event Processing. We conducted two controlled experiments with 216 participants in total to study the understandability of those approaches using a completely randomized design with one alternative per experimental unit. We hypothesized that PSP, as a highly abstracting pattern language, is easier to understand than LTL and EPL, and that EPL, due to separation of concerns (as one or more queries can be used to explicitly define the truth value change that an observed event pattern causes), is easier to understand than LTL. We found evidence supporting our hypotheses which was statistically significant and reproducible. This empirical study is concerned with RQ 5.1 (*How understandable are existing behavioral constraints modeled in different major languages, and are there significant differences?*). This chapter is based on the following paper:

- Christoph Czepa, Uwe Zdun, "On the Understandability of Temporal Properties Formalized in Linear Temporal Logic, Property Specification Patterns and Event Processing Language" In: *IEEE Transactions on Software Engineering (TSE), ISSN: 0098-5589*. DOI: 10.1109/TSE.2018.2859926

Chapter 7 is concerned with the understandability of representative behavioral constraint languages with regards to behavioral constraint authoring. Mature verification and monitoring approaches, such as model checking and complex event processing, which can be applied for ensuring compliance at design time and runtime, exist already for a long time. However, so far little is known about the understandability of the different languages used for modeling compliance specifications in those approaches. This leads to uncertainty about the approaches which might be a major obstacle for their broad practical adoption. We conducted a controlled experiment with 215 participants on the understandability of modeling compliance specifications in representative modeling languages, namely Linear Temporal Logic (LTL), the Complex Event Processing based Event Processing Language (EPL), and Property Specification Patterns (PSP). The results of the study show that the formalizations in PSP were overall more correct which indicates that the pattern-based approach provides a higher level of understandability than EPL and LTL. More advanced users, however, seemingly are able to cope equally well with PSP and EPL in modeling compliance specifications. This study is concerned with RQ 5.2 (*How under-*

standable are major behavioral constraint languages when applied for modeling, and are there significant differences?). This chapter is based on the following paper:

- Christoph Czepa, Amirali Amiri, Evangelos Ntentos, Uwe Zdun, "Modeling Compliance Specifications in Linear Temporal Logic, Event Processing Language and Property Specification Patterns: A Controlled Experiment on Understandability" In: *Software & Systems Modeling (SoSyM), ISSN: 1619-1366*. DOI: 10.1007/s10270-019-00721-4 (accepted for publication)

Chapter 8 reports a controlled experiment with 116 participants on the understandability of representative graphical and textual behavioral constraint representations. Particularly, graphical and textual behavioral constraint patterns present in the declarative business process language *Declare* and textual behavioral constraints based on the *Property Specification Patterns* are subject of this study. The main goal of this study is finding out whether there is a difference in understandability between graphical and textual behavioral constraint representations on the basis of established behavioral constraint approaches. Therefore, it is concerned with RQ 3.3 (*Is there a difference in understandability between graphical and textual behavioral constraint approaches?*). In addition to measuring the understandability construct, this study assesses subjective aspects like the perceived learning difficulty and the perceived potential for further improvements of the tested approaches, and it provides an analysis of free text answers given by the participants regarding perceived positive and negative aspects as well as suggestions for improvement of the representations. The participants were not allowed to use the learning material during the experiment session. This step was taken to ensure undisturbed testing of the participants' understanding of the textual terms and graphical shapes of a notation under the exclusion of potential effects resulting from looking up graphical shapes or textual terms. An interesting finding of this study is the overall low achieved correctness in the experimental tasks which seems to indicate that pattern-based behavioral constraint representations are hard to understand for novice software designer in the absence of additional supportive measures (e.g., tutorials or tool support). The results of the analysis of free text answers and the descriptive statistics regarding understandability are slightly in favor of the textual representations, but the inference statistics do not indicate any significant differences in terms of understandability between graphical and textual behavioral constraint representations. Moreover, the evaluation of subjective aspects does not show any significant differences between the approaches. This chapter is based on the following paper:

- Christoph Czepa, Uwe Zdun, "On the Understandability of Graphical and Textual Pattern-Based Behavioral Constraint Representations" In: *ACM Transactions on Software Engi-*

neering and Methodology (TOSEM), ISSN: 1049-331X. DOI: 10.1145/3306608 (accepted for publication)

The following publications are highly related to this dissertation, but its contents have not been included in the thesis. As the works are related to the thesis contents, we summarize each of those publications in the following:

- Thanh Tran, Erhard Weiss, Christoph Ruhsam, Christoph Czepa, Huy Tran, Uwe Zdun, "Embracing Process Compliance and Flexibility through Behavioral Consistency Checking in ACM: A Repair Service Management Case" In: *Business Process Management Workshops, BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers, ISBN: 978-3-319-42886-4*.
  DOI: 10.1007/978-3-319-42887-1

  This paper discusses behavioral constraint authoring and execution on basis of a practical repair service management case in our prototypical extension of the ISIS Papyrus ACM software.

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Towards Structural Consistency Checking in Adaptive Case Management" In: *Business Process Management Workshops, BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers, ISBN: 978-3-319-42886-4*.
  DOI: 10.1007/978-3-319-42887-1

  Before going into studying behavioral consistency of case models (e.g., the verification of a model based on compliance rules), we saw the necessity to provide adequate support for structural consistency, which is concerned with modeling errors that could lead to inconsistencies such as inaccessible elements or potentially contradictory guard conditions. Consequently, this work is a first step towards enabling structural consistency support for case models.

- Thanh Tran, Erhard Weiss, Christoph Ruhsam, Christoph Czepa, Huy Tran, Uwe Zdun, "Enabling Flexibility of Business Processes by Compliance Rules: A Case Study from the Insurance Industry" In: *Proceedings of the Industry Track at the 13th International Conference on Business Process Management 2015 co-located with 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2015, ISSN: 1613-0073*. URN: urn:nbn:de:0074-1439-9

  This paper reports a case study on one of ISIS Papyrus' customers, called *Die Mobiliar*, a major Swiss insurance company. We recognized that a huge quantity of similar business

process models existed, which were created by business administrators for necessary small deviations of business processes. This could be problematic because of several reasons, including poor maintainability and agility. By enabling support for behavioral consistency, we were able to decrease the number of process templates in the library and to support flexibility while handling unpredictable insurance cases.

- Christoph Czepa, Huy Tran, Uwe Zdun, Stefanie Rinderle-Ma, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Supporting Structural Consistency Checking in Adaptive Case Management" In: *On the Move to Meaningful Internet Systems: OTM 2015 Conferences. OTM 2015. Lecture Notes in Computer Science, vol 9415. Springer, Cham, ISBN: 978-3-319-26147-8.* DOI: 10.1007/978-3-319-26148-5

  In this work, we discuss potential structural inconsistencies that can be present in case models and how to identify them by applying graph-based checking and model checking. To support a meaningful evaluation of behavioral constraints (e.g., compliance rules), the identification of structural inconsistencies should take place before attempting behavioral consistency checking.

- Thanh Tran, Erhard Weiss, Alexander Adensamer, Christoph Ruhsam, Christoph Czepa, Huy Tran, Uwe Zdun, "An Ontology-Based Approach for Defining Compliance Rules by Knowledge Workers in Adaptive Case Management" In: *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW), ISBN: 978-1-4673-9933-3.* DOI: 10.1109/EDOCW.2016.7584347

  This paper is concerned with the implementation of business-driven, ontology-based behavioral constraint authoring in a prototypical extension of the ISIS Papyrus ACM software.

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "On the Understandability of Semantic Constraints for Behavioral Software Architecture Compliance: A Controlled Experiment" In: *2017 IEEE International Conference on Software Architecture (ICSA), ISBN: 978-1-5090-5729-0.* DOI: 10.1109/ICSA.2017.10

  This empirical paper applies and studies behavioral consistency approaches in a different context, namely in the context of software architecture compliance. In particular, we compared the understandability of software architecture descriptions in a natural language (English) and high-level structured natural languages (CEP-based and pattern-based) that can be used for automated verification. Interestingly, the statistical inference of this study suggests that there is no difference in understandability of the tested languages. This could indicate that the high-level abstractions employed bring those structured languages

closer to the understandability of unstructured natural language architecture descriptions. Moreover, it might also suggest that natural language leaves more room for ambiguity, which is detrimental for its understanding. Overall, the understandability of all three approaches is at a high level. However, the results must be interpreted with caution. Potential limitations of that study are that its tasks are based on common architectural patterns/styles (i.e., a participant possibly recognizes the meaning of a constraint more easily by having knowledge of the related architectural pattern) and the rather small set of involved behavioral constraint patterns (i.e., only very few behavioral constraint patterns were necessary to represent the architecture descriptions).

- Christoph Czepa, Huy Tran, Uwe Zdun, Thanh Tran, Erhard Weiss, Christoph Ruhsam, "Lightweight Approach for Seamless Modeling of Process Flows in Case Management Models" In: *Proceedings of the Symposium on Applied Computing (SAC '17), Marrakech, Morocco — April 03 - 07, 2017, ISBN: 978-1-4503-4486-9.*
  DOI: 10.1145/3019612.3019616

  Case management models are business process models that allow a great degree of flexibility at runtime by design. In contrast to flow-driven business processes (e.g., BPMN, EPC, UML activity diagrams), case management models primarily describe a business process by tasks, goals (i.e., milestones), stages, and dependencies between them. However, flow-driven processes are still often required and relevant in practice. In the recent case management standard CMMN (Case Management Model and Notation), support for process flows is offered by referencing BPMN processes. This results in a conceptual break between case elements and those in such subprocesses, so that dependencies from and to elements contained in flow-driven processes are unsupported. Moreover, case designers and other involved stakeholders are required to have substantial knowledge of not only case modeling but also of flow-driven business process modeling, which makes it overly complex. To counteract these current limitations, this paper proposes a lightweight and seamless integration of process flows in case management modeling as a first class citizen. Just a single new element, the Flow Dependency, in combination with existing case elements, enables support for fully integrated process flows in case models. Although the approach is that lightweight, an evaluation based on workflow patterns shows its high degree of expressiveness.

The following two book chapters were also published in the course of this dissertation:

- Christoph Czepa, Uwe Zdun, Christoph Ruhsam, "Transforming Compliance Regulations into User Experience" In: *Intelligent Adaptability, Future Strategies Inc., in association with the Workflow Management Coalition (WfMC) (2017), ISBN: 978-0-986321443.*

- Thanh Tran, Erhard Weiss, Christoph Ruhsam, Christoph Czepa, Huy Tran, Uwe Zdun, "Enabling Flexibility of Business Processes Using Compliance Rules: The Case of Mobiliar" In: *Business Process Management. Cases, Springer (2017), ISBN: 978-3-319-58306-8.* DOI: 10.1007/978-3-319-58307-5

Part II.

Studies on Supporting Behavioral Consistency in ACM

2. Behavioral Consistency Checking of Case Management Models

Case templates in Adaptive Case Management (ACM) are business process models that may range from unstructured over semi-structured to structured. Due to that versatility, both industry and academia show a growing interest in the case modeling paradigm. In this chapter, we discuss a model checking approach for the behavioral verification of ACM case templates. To counteract the high computational efforts involved with model checking techniques, the proposed approach includes state space reduction techniques as a preprocessing step before state-transition system generation. Consequently, the problem size is decreased, which decreases the computational efforts of the subsequent model checking as well. The evaluation of the approach on the basis of a large set of Linear Temporal Logic (LTL) specifications, which are based on recurring property specification patterns, and two real-world case templates, which are representative for semi-structured and structured process models and realistic in size, shows an overall good performance of the proposed approach.

2.1. Introduction

Many software vendors offer Adaptive Case Management (ACM) as their business process management solution (cf. Forrester Research [36]). A case template in ACM is a business process model that describes the basic structure and behavior of the case instances (aka business process instances) that originate from it. Consequently, a case template must contain neither structural errors (e.g., inaccessible elements) nor undesired behavior (e.g., non-compliance with laws, standards or best practices) since all case instances derived from this template would be affected.

Case modeling is emerging as a major approach for business process modeling (cf. Forrester Research [36], Kurz et al. [37], and Marin et al. [8]). While there exist ways to detect structural inconsistencies in case templates (cf. Czepa et al. [38]), the behavioral verification of case templates has not yet been sufficiently studied. A powerful, yet computationally expensive approach for detecting undesired behavior is model checking (cf. Clarke [39]). Model checking is a verification technique that explores all possible execution traces of a (business

process) model. Any undesired behavior is detected, provided the specification is correctly defined. However, model checking has a downside as well, namely its computational complexity, which is *PSPACE-complete* (cf. Sistla & Clarke [22]). That is, the runtime of model checking grows exponential with the problem size. Consequently, the work presented in this chapter addresses the following research questions: How can the behavior of case templates be captured as a state-transition system for model checking? Since model checking is known to be computationally expensive, how can case templates be model checked efficiently? Can model checking be applied to real-world case templates that are realistic in size and structure?

This chapter discusses a model checking approach for detecting undesired behavior in case templates. The presented approach comprises four steps: (1) Case elements that are not required for the detection of undesired behavior are removed (*Case Element Reduction*). (2) Conditions are abstracted by pre-computing all possible outcomes (*Condition Reduction*). (3) A case template is transformed to a state-transition system for model checking (*Model Transformation*). (4) Model checking is performed to find out whether the system meets a specified desired behavior (*Verification by Model Checking*). Steps 1 & 2 aim at improving the performance of model checking by reducing the state space that is to be considered in model checking of a case template. The approach is applied to two real-world case templates that are representative for different degrees of structuredness, and both are realistic in size.[1] The applied reduction techniques and the overall approach enable the verification of those real-world case templates within reasonable response times.

2.2. Motivation

Let us consider an example from health care regarding the treatment of a fracture, shown in Figure 2.1. The case template is described in Case Management Model and Notation (CMMN) [19] which can be used to model the essential structures of ACM case templates (cf. Czepa et al. [38], Kurz et al. [37], and Marin et al. [8]). In total, this case template has ten tasks, seven dependencies (dotted lines) and seven entry criteria (diamond symbols). We will use this simple example to discuss our approach. Please note that the approach is able to handle complex case templates (e.g., containing nested stages) as well. *Examine Patient*, *Prescribe Analgesics* and *Establish Venous Access* are not dependent on any other element and can be started as decided by the business users. Other elements are dependent: For example, *Perform Surgery* is dependent on the completion of *Perform X Ray* and can only be started if the condition "*diagnosis == 'compound fracture'* " (note: displaying conditions of criteria is omitted in the diagram for reasons

[1] Completely unstructured case templates are not considered since a design time verification of those would be pointless due to the unconstrained order of execution of the elements of such a process.

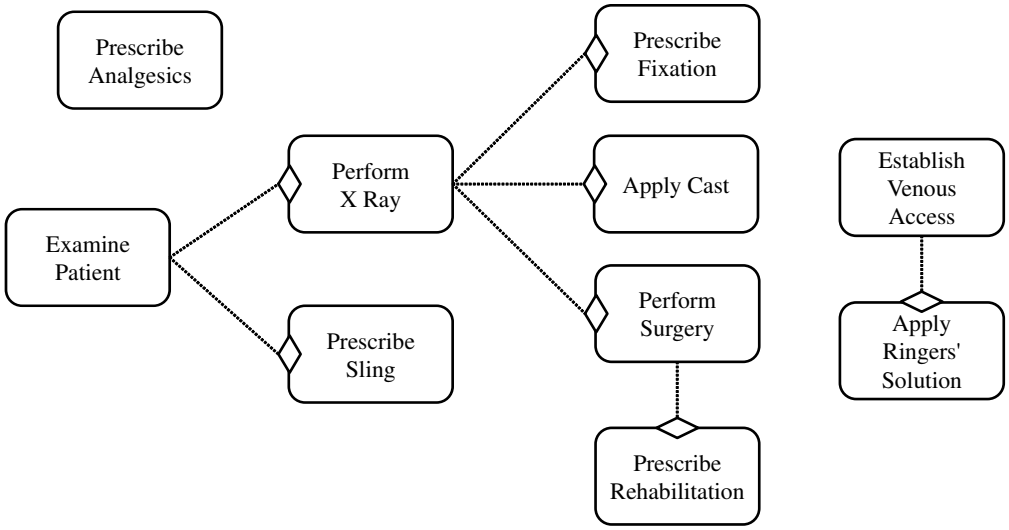


Figure 2.1.: Case template for the treatment of a suspected fracture

of clarity) of the entry criterion is met, and *Prescribe Fixation* is dependent on the completion of *Perform X Ray* and can only be started if the condition "*diagnosis == 'contusion'* " of the entry criterion is met.

Formal verification of models is a recurring research interest. While flow-driven business process models have already been studied extensively (cf. Eshuis [15], Kherbouche et al. [16], Sbai et al. [17], and Raedts et al. [18]), the verification of case templates to detect undesired behavior has not yet been investigated to a similar extent. In this work, we employ a well-established verification technique, namely model checking (cf. Clarke [39]), which requires the definition of the semantics of case templates as a state-transition system (cf. Cimatti et al. [40]). This state-transition system is then checked against a specification in a formal temporal logic such as Linear Temporal Logic (LTL; cf. Pnueli [26]) or Computation Tree Logic (CTL; cf. Clarke & Emerson [41]). This chapter is concerned with finding an adequate state-transition system for case templates that enables their formal verification within acceptable response times.

Regarding this subject, we identify the following core challenge: As execution times in model checking grow exponentially with problem size, the problem must be kept small in size, which is challenging due to the rich semantics of case templates. Consequently, the generated state-transition system must be a highly abstracted model of a case template that still captures the rich semantics of the original template to allow a meaningful verification.

We will use the example given in Figure 2.1 to discuss possible ways for reducing the problem size as a preprocessing step before model checking. Let us consider specifications that describe desired behavior. Here, we consider well-established patterns from software verification (cf.

Dwyer et al. [27]) and business process compliance (cf. Elgammal et al. [23]). For instance, the *Exclusive* pattern "*P EXCLUSIVE Q*" (where the presence of P mandates the absence of Q) can be applied to express that *Prescribe Fixation* demands the absence of *Prescribe Rehabilitation*. This specification could help to avoid unnecessary costs when a rehabilitation is medically not indicated. For the verification of the specification "*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*", it is not necessary to consider each and every element of the case. Only those elements that may have an influence on the outcome of the model checking must be retained. By removing *Prescribe Analgesics*, *Prescribe Sling*, *Establish Venous Access*, *Apply Ringers' Solution*, and *Apply Cast*, the case template can be reduced in size considerably. Moreover, since *Examine Patient* must inevitably lead to *Perform X Ray* before *Prescribe Rehabilitation* and *Prescribe Fixation*, they can be joined to a single activity, which further reduces the problem size. What remains as a possible way to reduce the problem size for model checking are the conditions of the criteria. Since conditions are evaluated on basis of data, it would require the inclusion of data in the state-transition system for model checking of the specification, which would introduce new variables and increase the state space. To avoid this, we propose to divide the problem into smaller pieces by precomputing the behavior of conditions. For example, once *Perform X Ray* is complete, the outgoing dependencies to the entry criteria of *Prescribe Fixation* and *Perform Surgery* get triggered. Since the condition "*diagnosis == 'contusion'* " of the entry criterion of *Prescribe Fixation* is contradictory to the condition "*diagnosis == 'compound fracture'* " of the entry criterion of *Perform Surgery*, only either *Prescribe Fixation* or *Perform Surgery* is possible, but not both of them. By applying these reductions, the model is reduced in size and complexity, which improves model checking performance, while the necessary semantics are preserved to properly verify the given specification. The given example will be revisited in Section 2.5 & 2.6, in which the reduction techniques are discussed in detail.

2.3. Approach Overview

Figure 2.2 shows an overview of the approach. *Case Element Reduction* uses the provided *Case Template* (which must be free of structural inconsistencies; cf. Czepa et al. [38]) and *Specification* to create a *Reduced Model*. Tasks, goals, stages, criteria and dependencies that are not needed to model check the given specification have been removed from this model. *Condition Reduction* preprocesses all the possible combinations of criteria that can be activated at once. By this, the approach circumvents the explicit modeling of these conditions, which would also require the explicit consideration of all data attributes that are referenced in the conditions of criteria. *Model Transformation* uses the *Reduced Model* and the *Possible Satisfiable Combinations of Criteria* to create a *State-Transition System* for model checking. The *Verification by Model*

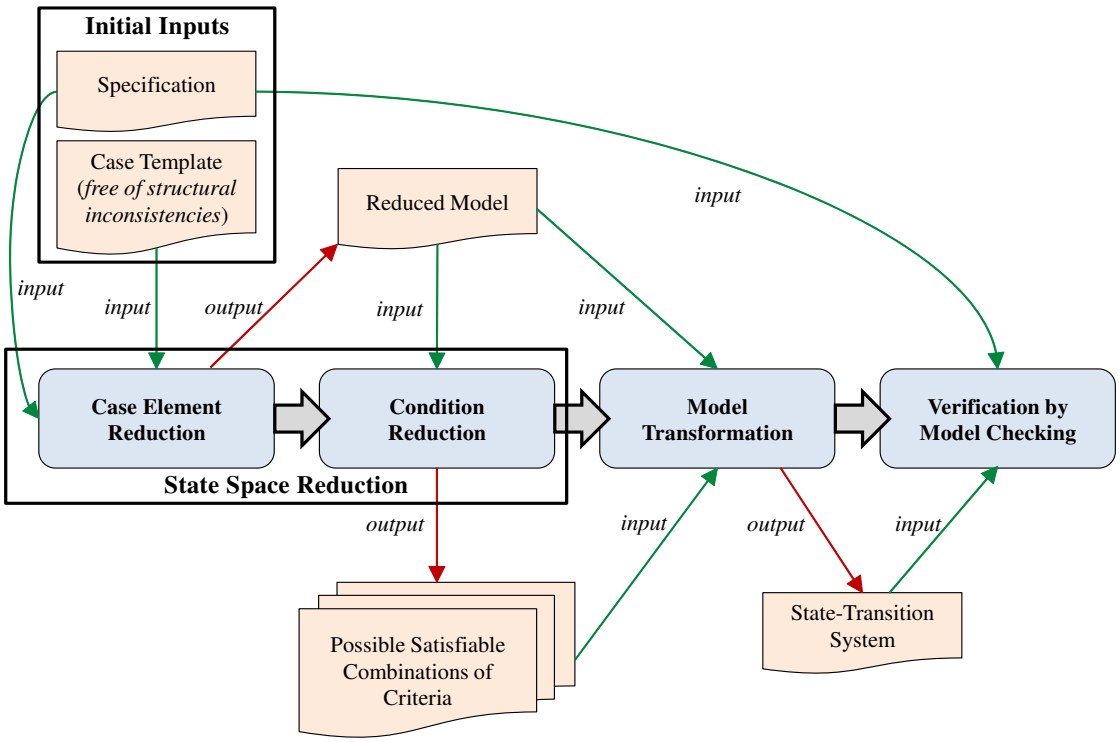


Figure 2.2.: Behavioral consistency checking approach overview

Checking uses this *State-Transition System* and evaluates it against the provided *Specification*.

2.4. Formalization of Case Templates

This section defines the elements of which case templates are composed that are relevant to describe the proposed reduction techniques and model transformation in later sections of this chapter.

Definition 2.1. A case template $\mathcal{M}$ is a tuple $(\mathcal{T}, \mathcal{G}, \mathcal{S}, \mathcal{E}, \mathcal{X}, \mathcal{C}, \mathcal{D}, \zeta_{\mathcal{E}}, \zeta_{\mathcal{X}}, \delta, s_m, \mathcal{ED}, \rho, \mathcal{CE}, \sigma)$ where

- $\mathcal{T}$ is a set of case tasks,
- $\mathcal{G}$ is a set of goals (aka milestones),
- $\mathcal{S}$ is a set of stages,
- $\mathcal{E}$ is a set of entry criteria,
- $\mathcal{X}$ is a set of exit criteria,
- $\mathcal{C} = \mathcal{E} \cup \mathcal{X}$ is a set of criteria,
- $\mathcal{D}$ is a set of dependencies,
- $\zeta_{\mathcal{E}} : \mathcal{E} \mapsto \mathcal{T} \cup \mathcal{G} \cup \mathcal{S}$ is a total non-injective function which maps an entry criterion to a task, goal[2], or stage,
- $\zeta_{\mathcal{X}} : \mathcal{X} \mapsto \mathcal{T} \cup \mathcal{S}$ is a total non-injective function which maps an exit criterion to a task or stage,
- $\delta : \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \mapsto \mathcal{S}$ is a partial non-injective function which maps a task, goal, or stage to a parent stage,
- $s_m \in \mathcal{S}$ is the main stage of the case,
- $\mathcal{ED} = \{mandatory, optional\}$ is a set of execution directives for tasks, goals, and stages,
- $\rho : \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \mapsto \mathcal{ED}$ is a total non-injective surjective function which maps a task, goal, or stage to an execution directive,

[2]The entry criterion of a goal is also called *completion criterion*.

- $\mathcal{CE} = \{immediate, listening\}$ is a set of evaluation modes for entry criteria where immediate is possible for $e \in \mathcal{E}$ iff $\exists d \mid d = (d_s, d_t) \wedge d_t = e \wedge (\zeta_{\mathcal{E}}(e) = t \mid t \in \mathcal{T})$,
- $\sigma : \mathcal{E} \mapsto \mathcal{CE}$ is a total non-injective surjective function which maps an entry criterion to an evaluation mode.

2.5. Case Element Reduction

Since a specification usually contains merely a small amount of case elements (cf. Elgammal et al. [23], Dwyer et al. [27], van der Aalst and Pešić [42]), those elements that are not contained in a specification are candidates for removal to reduce model complexity. However, since case element reductions might have an disturbing impact on the evaluation of formulas containing next operators, this reduction is only save when applied to next-free temporal logic formulas. The resulting reduced model must preserve the behavior of the original model, so not every case element which is not part of the specification can be removed. In this section we will discuss the *Case Element Reduction* approach which is taken as the first of two reduction steps (the second reduction step will be discussed in Section 2.6).

As a first reduction of case elements, those connected structures that do not contain any of the case elements of the specification formula can be removed. For this reason, a graph called *Flattened Case Graph* (cf. Definition 2.2) is created in a first step.

Definition 2.2. *(Flattened Case Graph)* $G_{\mathcal{M}f} = (V, E)$ is a directed graph representation of $\mathcal{M}$, where $V = \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}$ is a set of vertices. $E = E_e \cup E_x \cup E_{df} \cup E_{dt}$ is a set of edges where $E_e = \{(e, \zeta_{\mathcal{E}}(e)) \mid e \in \mathcal{E}\}$, $E_x = \{(\zeta_{\mathcal{X}}(x), x) \mid x \in \mathcal{X}\}$, $E_{df} = \{(f, d) \mid d = (f, t) \in \mathcal{D}\}$ and $E_{dt} = \{(d, t) \mid d = (f, t) \in \mathcal{D}\}$.

By this, structures of a case model that are connected through dependencies are identified. In a next step, those structures that do not contain elements of the specification formula and are just contained in a stage but not dependent in any other form on a stage can be removed from the case model because they do not have any influence on the verification of the given specification (cf. Definition 2.3).

Definition 2.3. *(Reduction of Structures)*
All elements V_s of a connected component s of $G_{\mathcal{M}f}$ are removed from $\mathcal{M}$ iff $\nexists v \mid v \in V_s \wedge (v \in specification \vee v \in \mathcal{S})$.

In Figure 2.3, *Reduction of Structures* is applied to the motivational example. After the identification of connected structures, all elements contained in structures that do not contain the elements of the specification are removed.

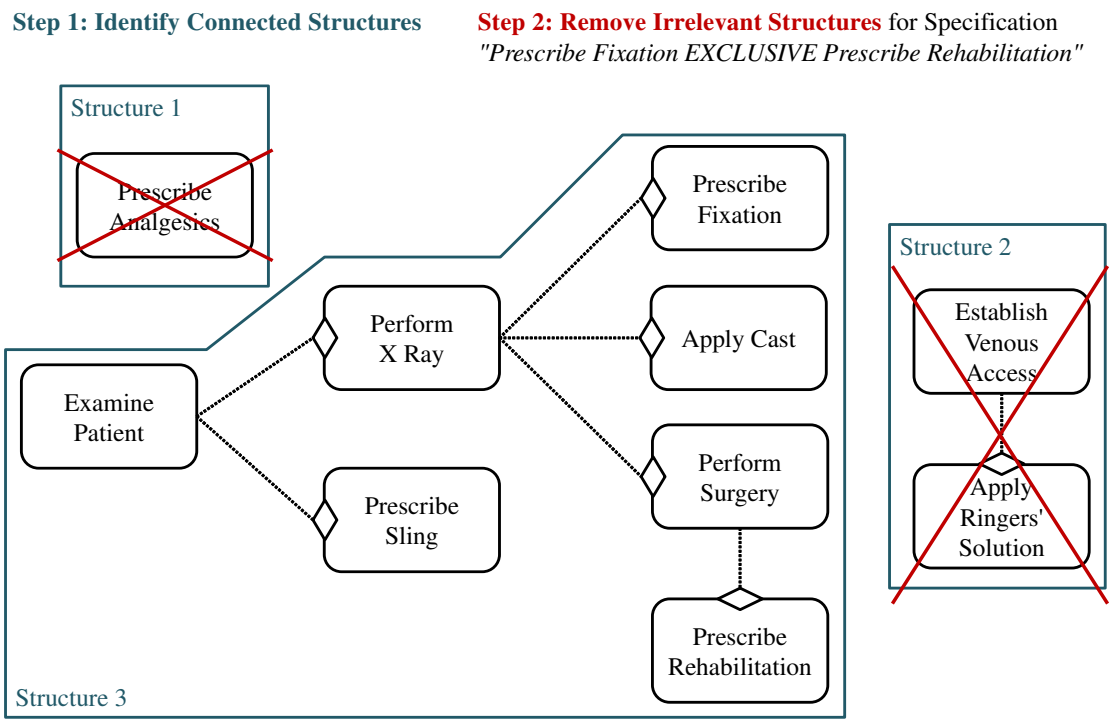


Figure 2.3.: Example for *Reduction of Structures*

Reduction of Structures can be considered as a *macro reduction* because it is able to remove larger structures of a case model. After this first reduction step, it makes sense to perform *micro reductions* that try to decrease the number of case elements in remaining structures. We propose two micro reduction techniques, namely *Rear Reduction* and *Melting Reduction*.

Definition 2.4. *(Rear Reduction)*
A rear reduction of a task, stage or goal $tgs \in \mathcal{T} \cup \mathcal{G} \cup \mathcal{S}$ is defined as follows:

- A task $t \in \mathcal{T}$ is removed from $\mathcal{M}$ iff $t \notin \mathit{specification} \wedge \rho(t) \neq \mathit{mandatory} \wedge (\nexists x \mid (x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = t \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x))) \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = t)$.
- A goal $g \in \mathcal{G}$ is removed from $\mathcal{M}$ iff $g \notin \mathit{specification} \wedge \rho(t) \neq \mathit{mandatory} \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = g)$.
- A stage $s \in \mathcal{S}$ is removed from $\mathcal{M}$ iff $s \notin \mathit{specification} \wedge \rho(t) \neq \mathit{mandatory} \wedge (\nexists x \mid (x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = s \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x))) \wedge (\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = s) \wedge (\nexists tgs \mid tgs \in \mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \wedge \delta(tgs) = s)$.

A rear reduction by removal of tgs also causes the removal of all $d \mid d = (d_s, d_t) \in \mathcal{D} \wedge \zeta_{\mathcal{E}}(d_t) = tgs$, $c \mid c \in \mathcal{C} \wedge (\zeta_{\mathcal{X}}(c) = tgs \vee \zeta_{\mathcal{E}}(c) = tgs)$ and $d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = tgs$ from the case model $\mathcal{M}$.

That means, if no case element is dependent on a specific task or goal, and if this specific task or goal is not part of the specification and not mandatory, then it can be removed. Stages are treated similarly, with the additional condition that the stage must not contain any elements (i.e., it may happen that the contained elements have already been removed by the reduction described in Definition 2.3).

In Figure 2.4, *Rear Reduction* is applied to the case model from Figure 2.3. Tasks *Prescribe Sling* and *Apply Cast* which are not contained in the specification "*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*" and do not have a successor are removed since they do not have any influence on the result of the model checking.

Definition 2.5. *(Melting Reduction)*
A melting reduction between two case elements e_1 and e_2 is performed iff

- $\neg(e_1 \in \mathit{specification} \wedge e_2 \in \mathit{specification})$, and
- $\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = e_1 \wedge (d_t \neq e_2 \vee \zeta_{\mathcal{E}}(d_t) \neq e_2)$, and
- $\nexists x \mid x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = e_1 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x \wedge (d_t \neq e_2 \vee \zeta_{\mathcal{E}}(d_t) \neq e_2))$,
- $\nexists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = e_2 \wedge (d_s \neq e_1 \vee \zeta_{\mathcal{X}}(d_s) \neq e_1)$, and

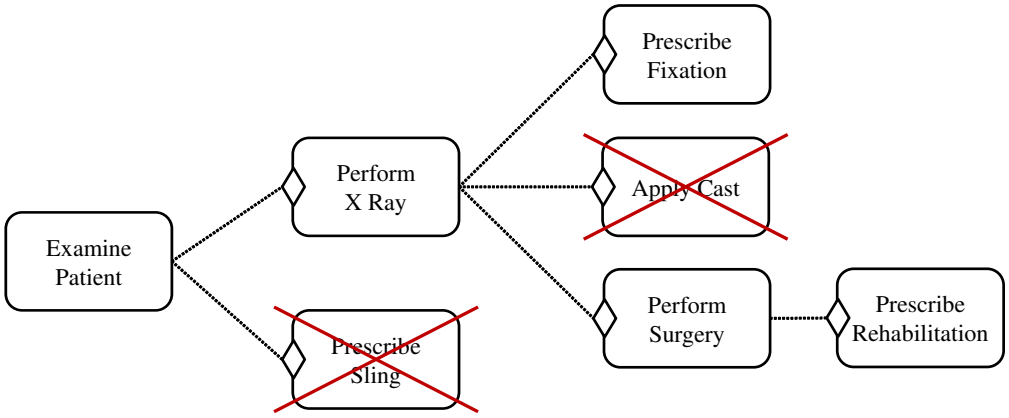


Figure 2.4.: Example for *Rear Reduction*

- $\nexists e \mid e \in \mathcal{E} \wedge \zeta_{\mathcal{E}}(e) = e_2 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_t = e \wedge (d_s \neq e_1 \vee \zeta_{\mathcal{X}}(d_s) \neq e_1))$,

and

- $\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = e_1 \wedge (d_t = e_2 \vee \zeta_{\mathcal{E}}(d_t) = e_2)$, or
- $\exists x \mid x \in \mathcal{X} \wedge \zeta_{\mathcal{X}}(x) = e_1 \wedge (\exists d \mid d = (d_s, d_t) \in \mathcal{D} \wedge d_s = x \wedge (d_t = e_2 \vee \zeta_{\mathcal{E}}(d_t) = e_2))$,

and

- $(e_1 \in \mathcal{T} \wedge e_2 \in \mathcal{T}) \vee (e_1 \in \mathcal{G} \wedge e_2 \in \mathcal{G}) \vee ((e_1 \in \mathcal{S} \wedge e_2 \in \mathcal{S}) \ \wedge \ (\nexists s \mid \delta(s) = e_1 \vee \delta(s) = e_2))$,

and $\rho(e_1) = \rho(e_2)$. Then, the melting reduction can be realized as follows:

1. All $rd \mid rd = (rd_s, rd_t) \in \mathcal{D} \ \wedge \ rd_s = e_1$ are to be removed from $\mathcal{M}$ and substituted by all $d \mid d = (d_s, d_t) \in \mathcal{D} \ \wedge \ d_s = e_2$, so that $d_s = e_1$, and
2. all $rx \mid rx \in \mathcal{X} \ \wedge \ \zeta_{\mathcal{X}}(e1)$ are to be removed from $\mathcal{M}$ and substituted by all $x \mid x \in \mathcal{X} \ \wedge \ \zeta_{\mathcal{X}}(x) = e_2$, so that $\zeta_{\mathcal{X}}(x) = e_1$, and
3. $e_1.label$ &= $e_2.label$, and e_2 is removed from $\mathcal{M}$.

The idea behind the melting reduction is the aggregation of elements, which we illustrate in Figure 2.5 by applying *Melting Reduction* to the case template from Figure 2.4. For the verification of '*Prescribe Fixation EXCLUSIVE Prescribe Rehabilitation*, *Examine Patient* and *Perform X Ray* as well as *Perform Surgery* and *Prescribe Rehabilitation* can be aggregated to joint activities.

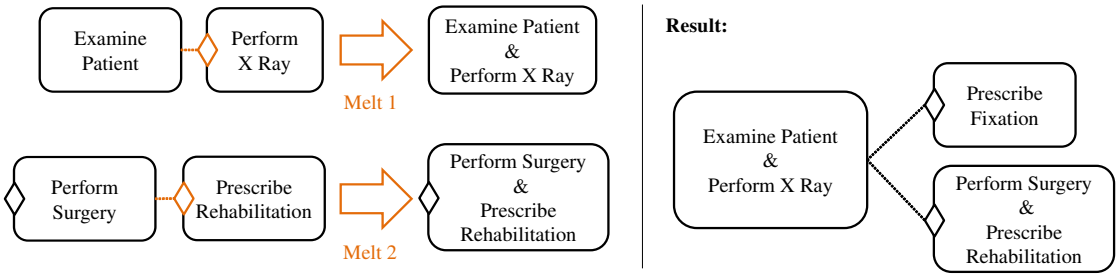


Figure 2.5.: Example for *Melting Reduction*

2.6. Condition Reduction

To avoid the explicit consideration of conditions and their data attributes, we propose to precompute the possible combinations of criteria that are satisfiable altogether at the completion of a case element. When a case element (i.e., a task, goal or stage) is completed, it depends on the current state of data which criteria are fulfilled. In a first step, all the possible combinations of exit criteria that are satisfiable at once are computed (Algorithm 1). In Line 5 of Algorithm 1 the power set of all exit criteria is created. Line 6 defines a loop to iterate over this power set. Line 9 checks whether this combination of criteria is satisfiable at once (Algorithm 3). If this is the case, then the combination is added to a set, which is returned by this function (Line 9). Algorithm 2 is similar to Algorithm 1 as it computes which other combinations that include dependent criteria are satisfiable with the already identified exit combinations, so power sets are created and Algorithm 3 is called again to find all possible criteria combinations. Please note that power sets grow exponential with the size of their sets. However, the created sets of interdependent criteria are rather small (i.e., in case models very few criteria are interdependent). Consequently, the computation times remain acceptable, and there is a large performance improvement compared to the explicit consideration of conditions in model checking.

Algorithm 3 creates a model (in Lines 2 & 3) of the criteria combination (Line 3) and checks whether this model meets the CTL formula in Line 4. For the model, it is necessary to analyze the conditions of the criteria to build up a proper enumeration set which is an abstraction of the possible values of the variables used in conditions. For example, for integer attributes it is sufficient to add the preceding and the succeeding integer values to the enumeration set. Attributes of other data types may require a more comprehensive treatment. For example, the treatment of strings is dependent on the string functions that a condition may contain.

In Figure 2.6, *Condition Reduction* is applied to the model from Figure 2.5. To find out, what the possible combinations of criteria are at the completion of *Examine Patient & Perform X Ray*,

Algorithm 1 Compute all possible satisfiable combinations of exit criteria at the completion of a case element

function COMPUTEPOSSIBLEEXITCRITERIACOMBINATIONS($ce \in \mathcal{T} \cup \mathcal{S}$)
    initialize satisfiableExitCriteriaCombinations
    if $\exists \zeta_{\mathcal{X}}(x) = ce$ **then**
        allExitCriteria = $\{x \mid x \in \zeta_{\mathcal{X}}(x) = ce\}$
        powerSet = $\mathfrak{P}$(allExitCriteria)
        for all cs in powerSet **do**
            if $|cs| > 0$ **then**
                complement := allExitCriteria $\setminus$ cs
                if IsSatisfiableCombination(cs, complement, allExitCriteria) **then**
                    satisfiableExitCriteriaCombinations.add(cs)
    return satisfiableExitCriteriaCombinations

Algorithm 2 Compute all possible satisfiable combinations of exit criteria and dependent criteria at the completion of a case element

function COMPUTEPOSSIBLEEXITANDDEPENDENTCRITERIACOMBINATIONS($ce \in \mathcal{T} \cup \mathcal{S}$)
    initialize satisfiableExitAndDependentCriteriaCombinations
    satisfiableExitCriteriaCombinations := ComputePossibleExitCriteriaCombinations(ce)
    for all satisfiableExitCriteria in satisfiableExitCriteriaCombinations **do**
        initialize criteriaDependentOnCeOrExitCombination
        criteriaDependentOnCeOrExitCombination.addAll($\forall c \mid c \in \mathcal{C} \wedge \exists (ce, c) \in \mathcal{D}$);
        for all x in satisfiableExitCriteria **do**
            for all $d = (d_s, d_t) \mid d \in \mathcal{D} \wedge d_s = x \wedge d_t \in \mathcal{C}$ **do**
                criteriaDependentOnCeOrExitCombination.add(d_t)
        for all $d = (d_s, d_t) \mid d \in \mathcal{D} \wedge d_s = ce \wedge d_t \in \mathcal{C}$ **do**
            criteriaDependentOnCeOrExitCombination.add(d_t)
        powerSet = $\mathfrak{P}$(criteriaDependentOnCeOrExitCombination)
        for all cs in powerSet **do**
            initialize satisfiableSet
            satisfiableSet.addAll(satisfiableExitCriteria)
            satisfiableSet.addAll(cs)
            initialize allSet
            allSet.addAll(satisfiableExitCriteria)
            allSet.addAll(criteriaDependentOnCeOrExitCombination)
            unsatisfiableSet := criteriaDependentOnCeOrExitCombination $\setminus$ cs
            if IsSatisfiableCombination(satisfiableSet, unsatisfiableSet, allSet) **then**
                satisfiableExitAndDependentCriteriaCombinations.add(cs)
    return satisfiableExitAndDependentCriteriaCombinations

Algorithm 3 Compute whether a combination of criteria is satisfiable

function ISSATISFIABLECOMBINATION($satisfiableSet \subsetneq \mathcal{C}, unsatisfiableSet \subsetneq \mathcal{C}, set \subsetneq \mathcal{C}$)
    dataEnumerationMap := CreateEnumerationValues(set)
    dataModel := CreateDataModel(dataEnumerationMap)
    specification := $EF(\bigwedge_{c \in satisfiableSet}(c.bf) \wedge \bigwedge_{c \in unsatisfiableSet}(\neg c.bf))$
    return performVerification(dataModel, specification)

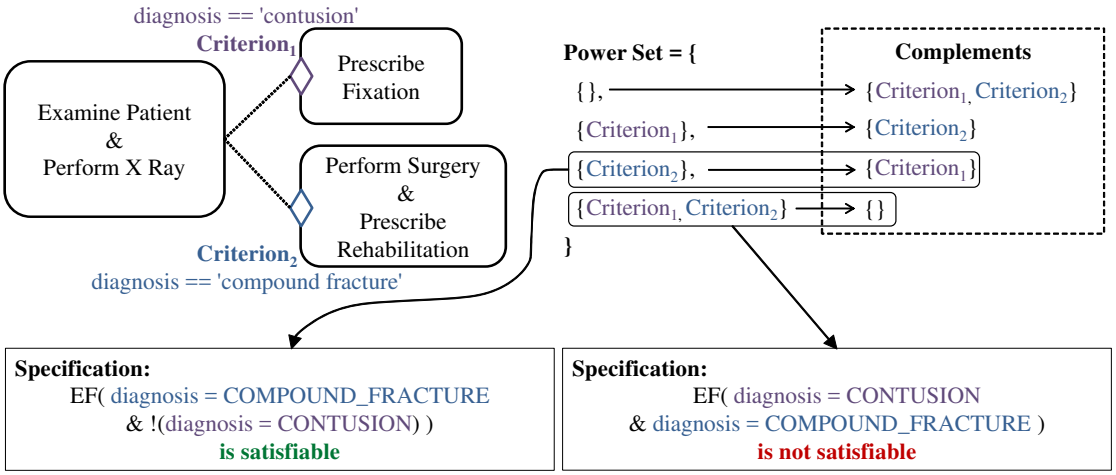


Figure 2.6.: Example for *Condition Reduction*

the power set of $\{\text{Criterion}_1, \text{Criterion}_2\}$ and the complement of each set of the power set is created. For each element of the power set and its complement, Algorithm 3 is called. The data model contains just a single attribute, namely *diagnosis* which has an enumeration set {COMPOUND_FRACTURE, CONTUSION, OTHER}, where OTHER is added representatively for values different from those present in conditions. Figure 2.6 illustrates the evaluation of two criteria combinations in greater detail. The question is whether it is possible at the completion of *Examine Patient & Perform X Ray* that (1) Criterion_2 is satisfied and Criterion_1 is not satisfied, (2) both Criterion_1 and Criterion_2 are satisfied. Since the CTL formula for (1) is satisfiable, it is a possibility that Criterion_1 is met while Criterion_2 is not met at the completion of *Examine Patient & Perform X Ray*. Consequently, this is a satisfiable combination. The CTL specification for (2) is not satisfiable since the conditions of the two entry criteria are contradictory. Consequently, Criterion_1 and Criterion_2 are not satisfiable at the same time, so this is not a satisfiable combination.

2.7. Model Transformation

Model Transformation is concerned with the creation of a state-transition system for model checking. For this purpose, it uses the *Reduced Model* and the *Possible Satisfiable Combination of Criteria*. The state-transition system is encoded in the SMV language. We refer the interested reader to McMillan [43] and Cimatti et al. [40] for more information on the SMV language.

Algorithm 4 transforms incoming dependencies of a task, goal, stage or criterion to a boolean

formula. If the source of the dependency is a task, goal or stage, then its completion state is added to the formula by a conjunction. For an exit criterion, both the condition of the criterion must be satisfied and the termination event of the case element the criterion is attached to must be satisfied. For an entry criterion, it is the activation event instead.

Algorithm 4 Create a boolean formula that represents incoming dependencies

1: **function** HANDLEINCOMINGDEPENDENCIES(incomingDependencies $\subsetneq \mathcal{D}$)
2: initialize booleanFormula
3: **for all** $d = (d_s, d_t)$ in incomingDependencies **do**
4: **if** $d_s \in \mathcal{T} \cup \mathcal{G} \cup \mathcal{S}$ **then**
5: booleanFormula &= d_s#COMPLETED
6: **else if** $d_s \in \mathcal{X}$ **then**
7: booleanFormula &= ((d_s#CONDITION_SATISFIED_EVENT & $\zeta_{\mathcal{E}}(d_s)$#TERMINATION_EVENT) | d_s#CONDITION_SATISFIED_STATE)
8: **else if** $d_s \in \mathcal{E}$ **then**
9: booleanFormula &= ((d_s#CONDITION_SATISFIED_EVENT & $\zeta_{\mathcal{X}}(d_s)$#ACTIVATION_EVENT) | d_s#CONDITION_SATISFIED_STATE)
10: **return** booleanFormula

Listing 2.1 contains the transformation of tasks. A task becomes enabled if the parent stage is active, all incoming dependencies are satisfied, and at least one entry criterion is satisfied which has all of its incoming dependencies satisfied as well. Algorithm 2.2 transforms incoming dependencies of a task, goal, stage or criterion to a boolean formula. If the source of the dependency is a task, goal or stage, then its completion state is added to the formula by a conjunction. For an exit criterion, both the condition of the criterion must be satisfied and the termination event of the case element the criterion is attached to must be satisfied. For an entry criterion, it is the activation event instead. The activation event is non-deterministically set to *true* or *false* if the task is enabled but not yet active or completed. Once the activation event has occurred, the task becomes active. Then it is again chosen non-deterministically, whether the task terminates or not. If the termination event has occurred, the task is completed.

```
for every task ∈ 𝒯
    for every se ∈ {ENABLED, ACTIVATION_EVENT, ACTIVE, TERMINATION_EVENT,
    COMPLETED}
        VAR
            task#se : boolean;
        INIT
            task#se = FALSE;
    ASSIGN
        next(task#ENABLED) := case
            δ(task)#ACTIVE & HandleIncomingDependencies({d | d = (d_s, d_t) ∈ 𝒟 ∧ d_t = task})
                & |_{c|c∈𝒞∧ζ_ℰ(c)=task} (c#CONDITION_SATISFIED_EVENT &
                HandleIncomingDependencies({d | d = (d_s, d_t) ∈ 𝒟 ∧ d_t = c}) : TRUE;
            TRUE : task#ENABLED; esac;
```

```
next(task#ACTIVATION_EVENT) := case
    task#ENABLED & !task#ACTIVE & !task#COMPLETED : {TRUE, FALSE};
    TRUE : FALSE; esac;
next(task#ACTIVE) := case
    task#ACTIVATION_EVENT : TRUE;
    task#TERMINATION_EVENT : FALSE;
    TRUE : task#ACTIVE; esac;
next(task#TERMINATION_EVENT) := case
    task#ACTIVE : {TRUE, FALSE};
    TRUE : FALSE; esac;
next(task#COMPLETED) := case
    task#TERMINATION_EVENT : TRUE;
    TRUE : task#COMPLETED; esac;
```

Listing 2.1: Transformation of tasks

Listing 2.2 contains the transformation of goals. A goal is completed if the parent stage is active and at least one entry criterion is satisfied which has all of its incoming dependencies satisfied as well.

```
for every goal ∈ 𝒢
    VAR
        goal#COMPLETED : boolean;
    INIT
        goal#COMPLETED = FALSE;

    ASSIGN
        next(goal#COMPLETED) := case
            δ(goal)#ACTIVE & | c|c∈𝒞∧ζ_ε(c)=goal (c#CONDITION_SATISFIED_EVENT : TRUE;
            TRUE : goal#COMPLETED; esac;
```

Listing 2.2: Transformation of goals

Listing 2.2 contains the transformation of stages. The main stage of a case is initially enabled, all other stages are not. For all stages except the main stage, a state transition to enabled occurs if the parent stage is active, all incoming dependencies are satisfied, and at least one entry criterion is satisfied which has all of its incoming dependencies satisfied as well. The activation event is non-deterministically set to *true* or *false* if the stage is enabled but not yet active or completed. Once the activation event has occurred, the stage becomes active. If all mandatory elements that are contained by the stage have completed, it is again chosen non-deterministically whether the stage terminates or not. If the termination event has occurred, the stage is completed.

```
for every stage ∈ 𝒮
    for every se ∈ {ENABLED,ACTIVATION_EVENT,ACTIVE,TERMINATION_EVENT,
    COMPLETED}
        VAR
            stage#se : boolean;
        INIT
```

```
        if se = ENABLED ∧ stage = s_m
            stage#se = TRUE;
        else
            stage#se = FALSE;
ASSIGN
if stage ≠ s_m
        next(stage#ENABLED) := case
            δ(stage)#ACTIVE &
                HandleIncomingDependencies({d | d = (d_s, d_t) ∈ D ∧ d_t = stage}) &
                |_{c|c∈C∧ζ_E(c)=stage}(c#CONDITION_SATISFIED_EVENT &
                HandleIncomingDependencies({d | d = (d_s, d_t) ∈ D ∧ d_t = c}) : TRUE;
            TRUE : stage#ENABLED; esac;

next(stage#ACTIVATION_EVENT) := case
    stage#ENABLED & !stage#ACTIVE & !stage#COMPLETED : {TRUE, FALSE};
    TRUE : FALSE; esac;
next(stage#ACTIVE) := case
    stage#ACTIVATION_EVENT : TRUE;
    stage#TERMINATION_EVENT : FALSE;
    TRUE : stage#ACTIVE; esac;
next(stage#TERMINATION_EVENT) := case
    stage#ACTIVE & &_{tgs|tgs∈T∪G∪S∧ρ(tgs)=mandatory∧δ(tgs)=stage}(tgs#COMPLETED): {
        TRUE, FALSE};
    TRUE : FALSE; esac;
next(stage#COMPLETED) := case
    stage#TERMINATION_EVENT : TRUE;
    TRUE : stage#COMPLETED; esac;
```

Listing 2.3: Transformation of stages

Listing 2.4 contains the first part of the transformation of criteria. Each criterion has a 'condition satisfied event'. If it is an exit criterion of a task or stage, or an entry criterion of a stage, then there is additionally a 'condition satisfied state' to remember whether the 'condition satisfied event' has already occurred with the termination or activation event of the element to which the criterion is attached to. It is important to model this state as well to retain which criteria were satisfied at the completion of a case element for later use (i.e., when dependencies are evaluated).

```
for every criterion ∈ C
    VAR
        criterion#CONDITION_SATISFIED_EVENT : boolean;
    INIT
        criterion#CONDITION_SATISFIED_EVENT = FALSE;
    if criterion ∈ X ∧ ζ_X(c) ∈ T ∪ S ∨ criterion ∈ E ∧ ζ_E(c) ∈ S
        VAR
            criterion#CONDITION_SATISFIED_STATE : boolean;
        INIT
            criterion#CONDITION_SATISFIED_STATE = FALSE;
        ASSIGN
            next(criterion#CONDITION_SATISFIED_STATE) := case
```

```
if criterion ∈ 𝒳 ∧ ζ_𝒳(c) ∈ 𝒯 ∪ 𝒮
    criterion#CONDITION_SATISFIED_EVENT & ζ_𝒳(c)#TERMINATION_EVENT
        : TRUE;
else if criterion ∈ ℰ ∧ ζ_ℰ(c) ∈ 𝒮
    criterion#CONDITION_SATISFIED_EVENT & ζ_ℰ(c)#ACTIVATION_EVENT :
        TRUE;
TRUE : criterion#CONDITION_SATISFIED_STATE; esac;
```

Listing 2.4: Transformation of criteria

Listing 2.5 contains the second part of the transformation of criteria. First, combinations of satisfiable criteria are defined. Then, a transition constraint is created. This transition constraint enforces that one of the combinations is actually happening at the termination of the case element.

```
for every task ∈ 𝒯
    combinations := ComputePossibleExitAndDependentCriteriaCombinations(task)
for every combination ∈ combinations with index i
    DEFINE
    task#criteriaCombination#i := task#ACTIVE = TRUE & next(task#ACTIVE) =
        FALSE & &_{c|c∈𝒞∧c∉combination} (next(c#CONDITION_SATISFIED_EVENT) = FALSE)
         & &_{c|c∈𝒞∧c∈combination} (next(c#CONDITION_SATISFIED_EVENT) = TRUE)
    TRANS
    (task#ACTIVE = TRUE & next(task#ACTIVE) = FALSE) -> (
        |_{combination ∈ combinations with index i} (task#criteriaCombination#i));
```

Listing 2.5: Transformation of possible satisfiable exit and dependent criteria combinations

Entry criteria either evaluate immediately (i.e., once all dependencies are fulfilled) or listen continuously for data changes and reevaluate accordingly. The transformation in Listing 2.6 allows state changes of immediately evaluating entry criteria to *true* only at the moment of termination of elements the criterion is dependent on. That is, if the criterion does not evaluate to *true* once all dependencies are fulfilled and enables the attached element, it remains *false* indefinitely (i.e., the element can no longer be enabled by this criterion).

```
for every e | e ∈ ℰ ∧ σ(e) = immediate
    TRANS |_{d|d=(d_s,d_t)∈𝒟∧d_s∈𝒯∪𝒢∪𝒮} (d_s#COMPLETED = FALSE & next(d_s#COMPLETED =
        TRUE))
    |_{d|d=(d_s,d_t)∈𝒟∧d_s∈𝒳} (ζ_𝒳(d_s)#COMPLETED = FALSE & next(ζ_𝒳(d_s)#COMPLETED = TRUE)
        )
     | next(e#CONDITION_SATISFIED_EVENT) = FALSE;
```

Listing 2.6: Transformation of immediate (non-listening) evaluation behavior of criteria

To prevent side effects, there is a limit of one activation or termination event at an instant in time (Listing 2.7).

```
for every (ts_i, ts_j) ∈ (T ∪ S) × (T ∪ S) such that (ts_j, ts_i) was not yet encountered
    and ts_i ≠ ts_j
   TRANS !( next(ts_1#ACTIVATION_EVENT) & next(ts_2#ACTIVATION_EVENT) );
   TRANS !( next(ts_1#TERMINATION_EVENT) & next(ts_2#TERMINATION_EVENT) );
for every (ts_i, ts_j) ∈ (T ∪ S) × (T ∪ S) such that ts_i ≠ ts_j
   TRANS !( next(ts_1#ACTIVATION_EVENT) & next(ts_2#TERMINATION_EVENT) );
```

Listing 2.7: Limit to a single activation or termination event per state

State changes of exit criteria are only possible at the termination of the case element to which an exit criterion is attached to (Listing 2.8).

```
for every x | criterion ∈ X ∧ x ζ_X(x) ∈ T
   TRANS
      x#CONDITION_SATISFIED_EVENT = next(x#CONDITION_SATISFIED_EVENT) |
         ζ_X(x)#TERMINATION_EVENT = TRUE;
```

Listing 2.8: Allow state changes of exit criteria events of tasks only in combination with a termination event of the task

2.8. Experimental Results

For the evaluation of the proposed approach, a representative set of LTL (Linear Temporal Logic) patterns and case templates is taken into account. As sources for LTL patterns, we make use of the property specification patterns by Dwyer et al. [27], [44] and the compliance patterns by Elgammal et al. [23]. Sources for case templates are real-world process models that are either available from ISIS Papyrus' customers or from public sources. Here it is important to consider the degree of structuredness and size of such a model. Interestingly, many available case templates are small in size. A possible reason for this might be that case templates in ACM represent only the framework of a case execution which is then augmented at runtime through case instance-specific ad hoc actions. For the evaluation of the approach, we select the largest to us available case templates, one from a sales company (cf. [45]) with a high degree of structuredness ($\frac{|\mathcal{D}|}{|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S}|} \approx 1.07$) and a total size of $|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}| = 80$ (where $|\mathcal{T}| = 23$, $|\mathcal{G}| = 3$, $|\mathcal{S}| = 3$, $|\mathcal{C}| = 20$, $|\mathcal{D}| = 31$) which we will refer to as *highly-structured case*, and another from health care (cf. Herzberg et al. [46][3]) with a medium degree of structuredness ($\frac{|\mathcal{D}|}{|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S}|} = 0.5$) and total size of $|\mathcal{T} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{D}| = 75$ (where $|\mathcal{T}| = 26$, $|\mathcal{G}| = 6$, $|\mathcal{S}| = 4$, $|\mathcal{C}| = 21$, $|\mathcal{D}| = 18$) which we will refer to as *semi-structured case*.

The prototype written in Java. It uses JGraphT[4] for graph-based parts of the approach, and

[3] We slightly had to adapt the model from [46] because some goals did not have any completion criteria, so we added criteria to those goals.

[4] http://jgrapht.org

it invokes NuSMV[5] (version 2.5.4) for model checking. The experiment was carried out on a common notebook computer with 8 GB RAM, Intel i5-4200U CPU (up to 2.6 GHz) and SATA II SSD on Windows 7, as we wanted to test our approach in the usual setting of a software developer or knowledge worker. The data of this evaluation was collected from 30000 model checks on the *semi-structured case* and *structured case*, which had a total computation time of about 40 hours. LTL specifications for those model checks are based on 15 distinct temporal logic patterns [23], [27]). For each combination of pattern and case template, 1000 LTL formulas are generated from the states of tasks, goals and stages of the case template to create a huge set of specifications. By this, verification runs normally carried out by users who perform behavioral consistency checking on these models are simulated in a large quantity.

Figure 2.7 shows the size reduction that is achieved by the *Case Element Reduction* step. The approach performs better on the semi-structured model than on the highly structured model. The computation of the *Case Element Reduction* step is finished between 0.1 and 0.5 milliseconds. *Condition Reduction* takes several orders of magnitude longer for the highly-structured case (about 500 milliseconds) than for the semi-structured case (0.005 milliseconds). The majority of the overall computation time is spent on model checking (Figure 2.10). Here, the semi-structured case template is verified within few seconds (in most cases even in a fraction of a second). Verifying the highly-structured case takes longer. Most of the verification runs terminate within 10 seconds, but there are also runs that take up to about 1000 seconds.

The computation of the *Case Element Reduction* step (shown in Figure 2.8) is finished between 0.1 and 0.5 milliseconds. *Condition Reduction* takes several orders of magnitude longer for the highly-structured case (about 500 milliseconds) than for the semi-structured case (0.005 milliseconds).

We deliberately do not compare against the situation in which the proposed reductions are not applied because without the reduction techniques, the state space explodes and results are not to be expected within acceptable response times.

2.9. Discussion

Although the proposed approach has a strong focus on ACM case templates, it is to a large extent applicable to CMMN (Case Management Model and Notation [19]) models. However, not all parts of the CMMN standard are yet considered. For example, repetitions of tasks, stages or goals are not yet part of the proposed approach. Whether such explicitly modeled repetitions make sense in the context of ACM is questionable since ad hoc actions can be performed at any time at runtime. Furthermore, flow-driven processes are better suited to model such kind

[5] `http://nusmv.fbk.eu/`

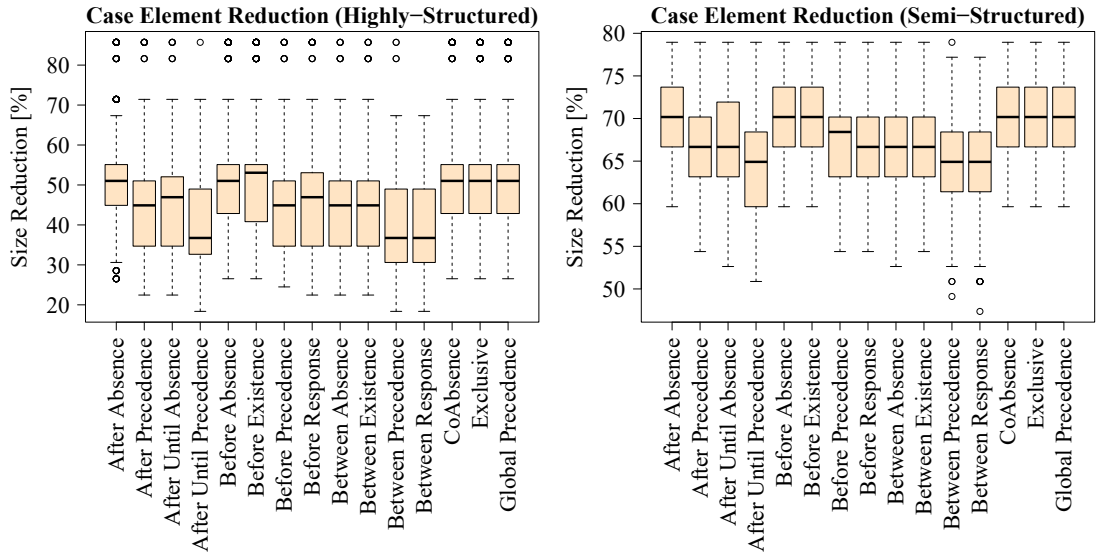


Figure 2.7.: Achieved size reduction after applying *Case Element Reduction* to the *Highly-Structured Case* (left) and *Semi-Structured Case* (right)

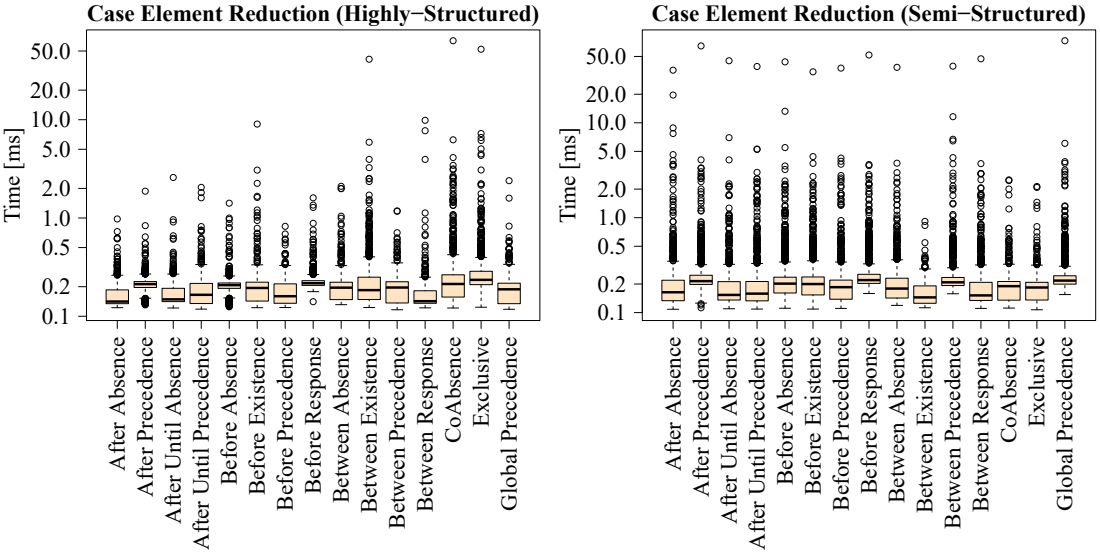


Figure 2.8.: Computation time of *Case Element Reduction* for *Highly-Structured Case* (left) and *Semi-Structured Case* (right)

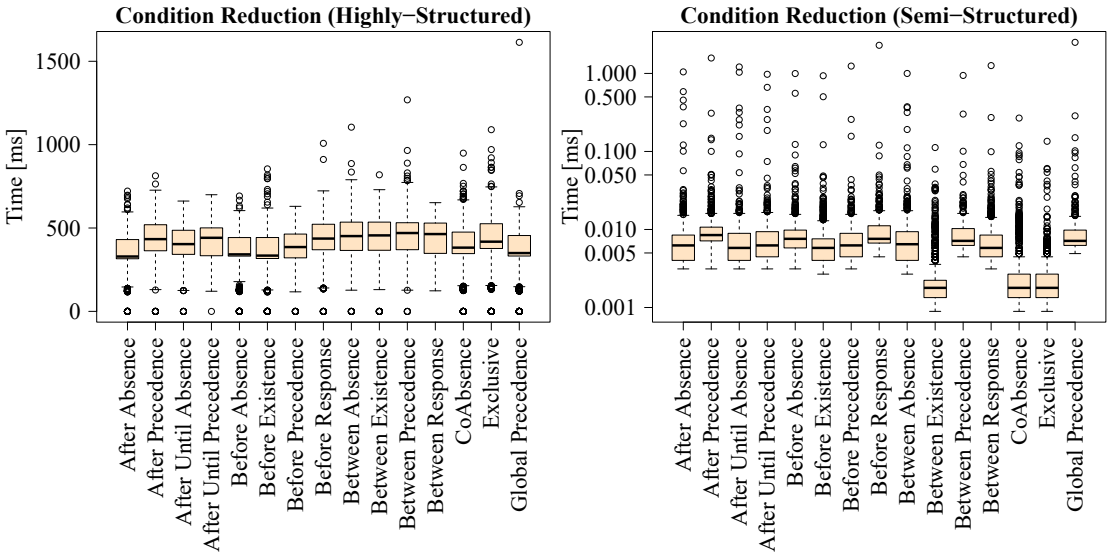


Figure 2.9.: Computation time of *Condition Reduction* for *Highly-Structured Case* (left) and *Semi-Structured Case* (right)

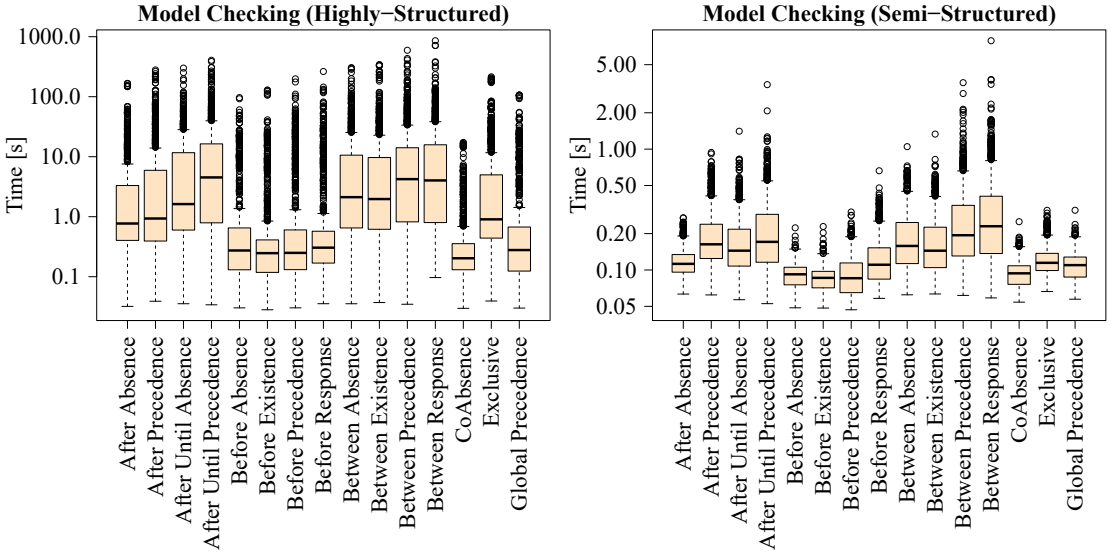


Figure 2.10.: Computation time of *Model Checking* for *Highly-Structured Case* (left) and *Semi-Structured Case* (right)

of looping behavior in business processes. This work has a strong focus on case management modeling elements and their semantics. Flow-driven processes (e.g., BPMN processes) of case templates are out of scope of this work. Nevertheless, the approach could be extended to include flow-driven subprocesses as well. This involves applying reduction techniques for flow-driven processes (cf. Awad et al. [47]), and conditions, such as present as guard conditions after exclusive and inclusive gateways in BPMN, can be precomputed and reduced in similar manner as described in Section 2.6.

By applying the approach to case templates, it becomes possible to verify their behavior at design time. Nonetheless, it is impossible to guarantee compliance of a case instance by model checking of case templates at runtime because runtime-specific circumstances, such as ad hoc actions and data adaptations, might introduce non-compliance. Consequently, it is not sufficient to provide only a tool support for design time verification. To keep cases compliant, runtime monitoring (cf. Ly et al. [48]) of case instances is required. In particular, adequate tools must support business users in staying compliant (cf. Czepa et al. [49]).

A well-known downside of model checking is scalability. If a problem grows too large in size, then model checking will become infeasible due to long computation times. In this work, we use reduction techniques to decrease the size of the problem, which makes it possible to verify real-world case templates that have a realistic size. However, as long as verification is based on model checking, there will always be scalability issues. Consequently, working on a computationally less expensive approach for verifying or testing the behavior of case templates is an interesting and challenging topic for future research.

2.10. Related Work

Many related studies on the verification of business processes focus on flow-driven business processes, such as UML activity diagrams and BPMN models. Eshuis proposes a model checking approach using the NuSMV model checker for the verification of UML activity diagrams [15]. Kherbouche et al. use the model checker SPIN to find structural errors in BPMN models [16]. Sbai et al. use SPIN for the verification of workflow nets, which are Petri nets representing a workflow [17]. Köhler et al. describe a process by means of an automaton and check this automaton by NuSMV [21]. An approach presented by Awad et al. aims at checking compliance of flow-driven business process models using the visual query language BPMN-Q to describe constraints and performing model checking to assure constraints are met [47]. This approach reduces the complexity of BPMN models by analyzing LTL specifications before state space generation. Aforementioned approaches use model checking for verification of business processes, but there also exist alternative approaches. For example, Raedts et al. propose the transformation

of models such as UML activity diagrams and BPMN2 models to Petri nets for verification with Petri net analyzers [18].

In 2014, the CMMN (Case Management Model and Notation) standard was released in version 1.0 by the OMG (Object Management Group) as "a common meta-model and notation for modeling and graphically expressing a Case, as well as an interchange format for exchanging Case models among different tools". Recent research indicates that CMMN is suitable for modeling knowledge-intensive processes and that the essential structural concepts of ACM cases are covered or can be realized by CMMN elements (cf. [8], [37], [38]). CMMN draws many influences, such as case handling [6], business artifacts [50], and the Guard-Stage-Milestone (GSM) language [51] for programming artifact-centric systems. Gonzalez et al. propose a specialized model checker for the GSM language [52]. Our approach enables the verification of case templates by non-specialized model checkers by state space reduction techniques.

In summary, the verification of ACM case templates has not yet been sufficiently addressed in existing studies. Due to the increasing industry adoption of Adaptive Case Management, this topic is highly relevant, not only from a purely academic but also from a practical point of view.

2.11. Conclusion and Future Work

This chapter presented a model checking approach for the behavioral consistency checking of ACM case templates. In particular, it discusses several techniques that aim at reducing the state space required for efficient model checking of case templates, and a transformation to the SMV specification language for describing a case template as state-transition system for model checking. Reductions are achieved by on the one hand using a given specification to remove elements from a case template that are not required for the verification run, and on the other hand, considering the conditions present in a case template in an abstracted manner in the actual verification run. The experimental evaluation based on 30000 model checking runs on two real case templates that are realistic in size and representative for different degrees of structuredness shows an overall good performance of the approach. In general, the computational effort for model checking grows exponentially with the problem size. Therefore, an objective for future work is finding an alternative, less computationally expensive approach for testing or verifying case templates. A major challenge in this regard is that all execution traces that are relevant for checking or testing a specific behavioral property must be considered. This is especially challenging during the creation of test cases, since the set of test cases should cover every relevant execution scenario. Furthermore, alternative techniques such as graph-based checking (cf. Tran et al. [53]) often have limitations concerning the adequate representation of semantically rich models with complex branching and parallel execution behaviors that must be overcome first. A promising

approach to adequately capture semantically rich models through a graph-based approach could be LURCH (a “Lightweight Alternative to Model Checking”; cf. Owen & Menzies [54]). While LURCH can be applied to larger problem sizes than traditional model checking can handle, its results based on a randomized search algorithm are merely approximate and not necessarily complete (cf. Owen et al. [55]).

3. Business-Driven Behavioral Constraint Authoring

In the previous chapter, we have discussed the verification of case templates, which has a strong focus on the transformation of a case template into a state-transition system. Another important input for model checking are the specifications of the model under investigation, namely the behavioral constraints, which were specified by formal Linear Temporal Logic formulas in the previous chapter. In this chapter, we propose an approach for the authoring of behavioral constraints that aims at being business user-friendly by applying pattern abstractions which hide the underlying technical complexity of languages such as LTL, and by making use of an ontology approach to speak the language of the business domain.

3.1. Introduction

Many business domains are subject to a large number of compliance requirements stemming from sources such as regulatory laws (e.g., Sarbanes-Oxley), standards (e.g., ISO 45001 - Occupational health and safety) or best practices (e.g. ITIL). The classical approach for integrating such compliance requirements in the enactment of business processes are flow-driven, predefined business process models that become instantiated and executed. Assuming the business process model is correctly defined, and the business users follow it exactly, we can be sure that business process instances are compliant. In reality, however, deviations from those predefined business processes become frequently necessary. This has led to flexible business process management approaches such as *Adaptive Case Management* (cf. Swenson [3] and Pucher [7]) that enable business users to actively shape the enactment of business processes by skipping activities, performing ad-hoc activities, and defining and changing goals. Several researchers propose approaches that we can broadly collected under the umbrella term *Behavioral Constraints* of business processes. Pešić & van der Aalst [56] propose a declarative approach called *Declare* for flexible business process management that offers a graphical specification language to loosely define business processes by relations between activities such as a 'response relation' or 'precedence relation' with underlying formal representations in Linear Temporal Logic (LTL) or

Event Calculus (Montali et al. [57], [58]). Ly et al. propose a different graphical language called *Compliance Rule Graphs* with elements such as 'antecedent occurrence' and 'consequence occurrence' [59]. Hildebrandt et al. propose *Dynamic Condition Response Graphs* for trustworthy Adaptive Case Management which is another graphical language comprising relations such as a 'condition relation' and a 'response relation' [60]. Schönig & Zeisig propose *DPIL (Declarative Process Intermediate Language)*, a scripting language which allows the definition of macros such as 'sequence(a, b)' that become enacted by a rule engine [61].

What all these approaches have in common is a recurring set of *behavioral constraint patterns*, most often with a strong *temporal* focus. Dwyer et al. identified a collection of fundamental temporal patterns such as 'response' and 'precedence' [27]. It is not surprising that such temporal aspects play a major role in business process management, since the order of execution of specific activities due to dependencies between activities is often of importance.

Above-mentioned behavioral constraint approaches aim at being user-friendly by offering graphical notations or textual specification languages that are based on patterns that have their origin in temporal logics (cf. [23], [27], [42]). However, supporting the user by integrating domain knowledge has not yet been considered, which poses an obstacle for end-user acceptance. A more general approach that does not particularly focus on temporal aspects of business processes is the Semantics of Business Vocabulary and Business Rules (SBVR) standard. It is an adopted standard of the Object Management Group (OMG) which enables the use of domain knowledge for the definition of rules. While this standard enables the definition of a vast amount of different rules, their automated verification, especially in a non-static, temporal context, is not yet fully solved: The SBVR standard document in the latest version 1.3 (May 2015) states that "[...] *capturing the formal semantics in an appropriate logic (e.g., a temporal or dynamic logic) is a harder task. One possibility is to provide a temporal package that may be imported into a domain model, in order to provide a first-order logic solution. Another possibility is to adopt a temporal modal logic (e.g., treat a possible world as a sequence of accessible states of the fact model). It may well be reasonable to defer decisions on formal semantics for dynamic rules to a later version of the SBVR standard.*" Consequently, there seems to be a link missing between behavioral constraint approaches—which have a strong focus on the dynamics of business processes (i.e., temporal relationships)—and ontology-based constraint approaches—which have a strong focus on the static rules of business processes.

In this chapter, we propose an *ontology-based behavioral constraint authoring* approach, which links the ontology of a domain with the application ontology of the business process management solution (which provides the functionality of specific tasks) and allows for using well-known domain concepts in the definition of behavioral constraints. Based on this explicitly defined domain knowledge, business users can define behavioral constraints on the basis of the

well-known concepts and relations of their business domain, supported by an ontology-fetching auto-completion and suggestion feature. The ontology can be adapted by creating derived concepts from existing ones that are used in existing behavioral constraints. Any such adaptation has a direct effect on the enactment of the constraint, as the derived concept is considered in the enactment of the constraint. By deriving domain-specific concepts from more generic concepts such as an activity or goal, the enactment of business processes becomes interfaced with the ontology. Consequently, business process elements, such as activities, goals and data objects, have in the background an organized structure, the ontology, which can be leveraged to enact ontology-based behavioral constraints while business processes are executed. The core benefits of the approach are (1) the possibility to modify behavioral constraints indirectly by modifying the ontology, which can have an impact on existing rules, (2) the authoring of behavioral constraints in the language of the business domain, and (3) the underlying temporal logic pattern-based approach that enables an automated verification of defined behavioral constraints.

3.2. Motivating Example

The building and construction industry is subject to a vast amount of compliance rules stemming from sources such as regulatory laws and standards. Let us consider a compliance document published by the EPA (United States Environmental Protection Agency) in 2011 regarding dealing with potentially lead-contaminated paint during renovations, repairs and maintenance of buildings (cf. EPA [13]). The compliance guide applies to all activities that disturb painted surfaces (short DPS) in residential houses, apartments and child-occupied facilities such as schools and day-care centers built before 1978 in the US. Let us further consider a process management application of a company that manages construction, maintenance, repair and renovation work in cooperation with many and changing subcontractors from diverse professions such as plumbers, carpenters and electricians. The company seeks to avoid non-compliance (to maintain its good reputation and avoid litigation) by assisting the subcontractors in staying compliant. Consequently, they try to implement EPA's compliance guide by defining several behavioral constraints that become automatically enacted by the process management software.

One problem the company is facing now is determining which tasks, that are potentially performed by subcontractors, are DPS activities. They recognize that the subcontractors know better which of their activities fall into this category. Since the pool of existing subcontractors is already large and assumed to be growing, and new DPS activities might evolve over time, the management demands a viable solution that reliably captures all future DPS activities in order to ensure compliance with EPA rules.

The approach enables the company to specify behavioral constraints based on the more gen-

eral and rather abstract concept of “disturbing painted surfaces”, whereas the subcontractors are enabled to relate their concrete, specific DPS activities to this more general and abstract concept. Consequently, the company provides the framework for the subcontractors to support them in achieving compliance with EPA rules, and the subcontractors are encouraged to properly document their domain knowledge in an explicitly defined ontology.

3.3. Approach

The general idea of ontology-based behavioral constraints is the integration of behavioral constraint enactment in business process management with domain-specific ontologies to directly leverage domain knowledge in behavioral constraints. Case instance elements that occur during runtime of business processes, such as instances of tasks, goals and data, are at the same time instances of ontology concepts. Consequently, for every such instance the underlying concept is known. Based on the structured information of the ontology, users can define ontology-based behavioral constraints, and instances of behavioral constraints can be evaluated by considering state changes over time and the background knowledge given by the ontology.

The approach comprises the following core components:

- The *Ontology* consists of the ontology of the application, which is the interface to the business process enactment engine, and a domain-specific ontology, which refines the application ontology of the business process enactment engine, and can be used to capture additional domain-specific knowledge explicitly.
- *Behavioral Constraints* are derived from compliance rules or best practices and defined on the basis of domain knowledge stemming from the ontology.
- *Constraint Patterns* with sound underlying formal representations are the foundation for the creation of behavioral constraints.
- The *Constraint Editor* is a tool for authoring of behavioral constraints. It is based on constraint patterns and the ontology to support the business user during constraint authoring.
- *Case Enactment* is realized as flexible business process enactment approach (e.g., Adaptive Case Management). There exists a tight connection between case enactment and the ontology of the domain, so that case elements (such as activities, goals and data) are conceptually defined in, and instances stemming from, an ontology.
- *Constraint Enactment* supports the business user by evaluating concrete instances of behavioral constraints during case enactment.

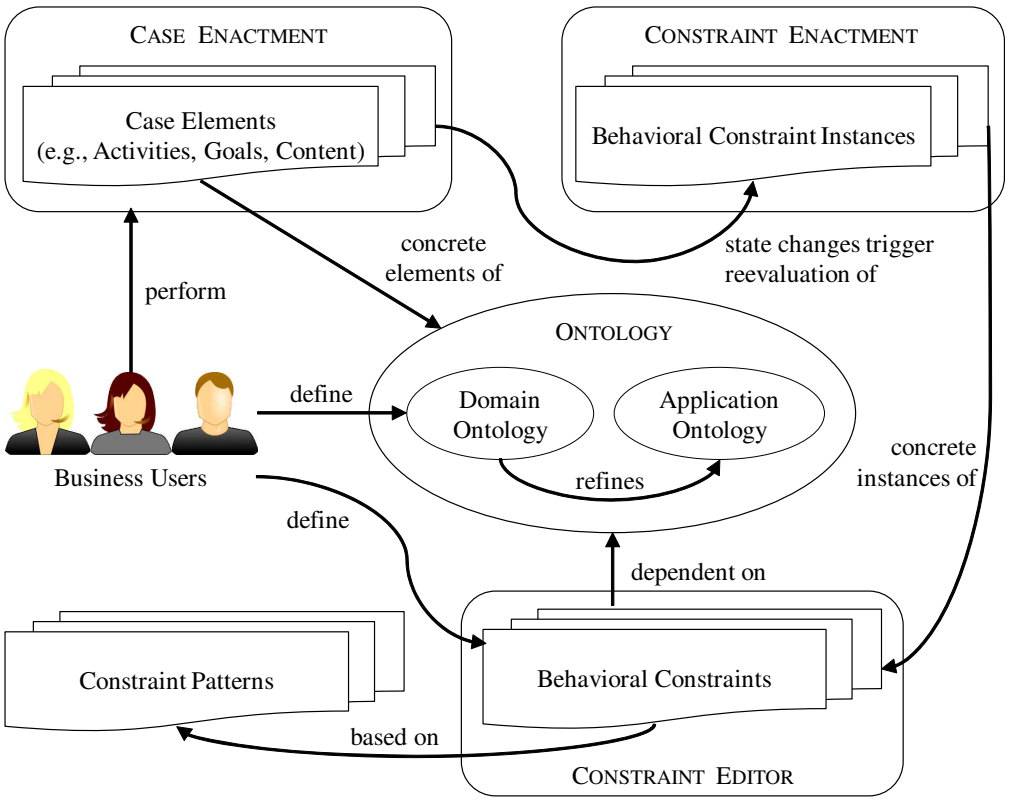


Figure 3.1.: Approach Overview

- *Business Users* define the knowledge of their domain, handle cases, and are enabled to define behavioral constraints by our approach.

Figure 3.1 contains an overview on the proposed approach. *Business Users* build up and maintain the business ontology of their business domain (*Domain Ontology*) which refines business process management concepts (defined in the *Application Ontology*) by domain-specific concepts, and they create *Behavioral Constraints* by leveraging the *Domain Ontology* and a set of *Constraint Patterns*. While the domain ontology is completely adaptable, the set of constraint patterns is dependent on underlying formal checking techniques and must be defined by users with appropriate technical background. During *Case Enactment*, business users perform activities to work towards goals. Of course, this involves also the creation and modification of contents (data objects of a case). All these concrete case elements (aka instances) stem from ontology concepts. Every time case elements change state, the *Constraint Enactment* component reevaluates the instances of behavioral constraints. Since case elements are instances of domain ontology concepts, behavioral constraint instances can be evaluated by querying domain knowledge (taxonomy, relationships) from the ontology.

The creation of ontology-based behavioral constraints by business users using the constraint editor is shown schematically in a sequence diagram in Figure 3.2. The constraint editor makes extensive use of the ontology while the business user types a constraint. In the beginning, the business user starts typing and the constraint editor requests ontology concepts matching the (incomplete) user inputs (e.g., by matching the starting string of concepts or by computing the Levenshtein distance). The response is a set of concepts which are proposed to the business user. The user either selects a proposed concept or continues typing. Eventually a proposed concept must be selected. Once the business user has selected a concept, it becomes possible to use the *context* of the specific domain concept. Thus, further auto-completions can be based on the set of *relations* of the concept. If appropriate, the constraint editor makes not only proposals based on the ontology, but also proposes the possible elements of the constraint grammar to the user. When the user selects a specific relation, the context for suggesting further inputs is narrowed down to concepts that are potential targets of this relation originating from the already selected concept. Figure 3.2 continues with processing user inputs to update the set of possible target concepts in the current relation context. Once the user has selected a target concept, the next part of the constraint could be either an element of the constraint grammar or a relation. For concepts that are derived from the application concepts *Goal* and *Activity*, the specific *runtime state* of these concepts can be directly specified after the name of each concept. Standard runtime states of goals are *initiated* and *completed*, and standard runtime states of activities are *started* and *finished*.

Constraint patterns stemming from different sources (such as Dwyer et al. [27], Elgammal et

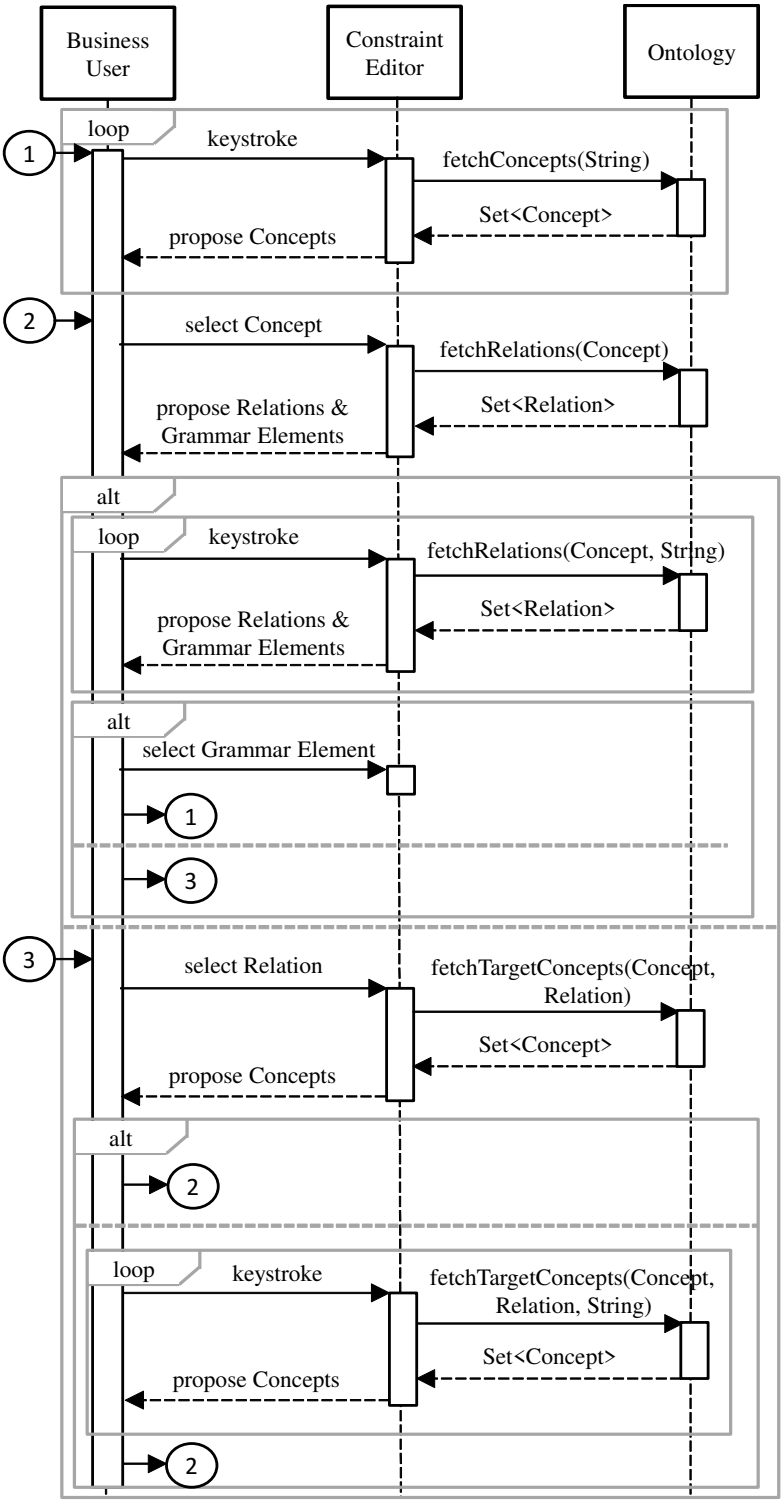


Figure 3.2.: Sequence diagram for constraint authoring

al. [62] and van der Aalst & Pešić [42]) can be integrated. For example, *Response* describes a temporal cause-effect constraint which can be expressed as

```
Expression I leads_to Expression II
```

where *Expression I* is the cause and *Expression II* is the effect. Precedence describes a temporal precondition and can be expressed as

```
Expression I precedes Expression II
```

where *Expression I* is the precondition that must be met before *Expression II* is allowed to happen. An *Expression* is defined based on the ontology and allows leveraging *domain-specific knowledge* for the creation of behavioral constraints. The following ontology elements can be used to define such an ontology-based expression:

- *Concepts*: Concepts are the anchor for defining ontology expressions, so every ontology-based expression must start with a concept. If a concept is derived from a *Goal* or *Activity*, then the *runtime state* can be specified directly after the name of the concept.
- *Relations*: Once the context of a concept is defined, the relations of this concept to other concepts become obvious and usable.
- *Constraint Concepts*: Constraint concepts are specialized child concepts that allow defining specific constraints that determine whether an instance of the parent concept is also an instance of the constraint concept. These constraints may be related to attributes and/or relations of the parent concept.

During case enactment, ontology expressions are evaluated based on the actual instances and relations that are present in a case instance. An ontology expression consisting of a (constraint) concept $\mathcal{C}$ is matched by $\mathcal{E}$ iff $\exists i \mid i \in \mathcal{I} \wedge i : \mathcal{C}$ where $\mathcal{I}$ is the set of case elements of a case instance $\mathcal{E}$ that are instances of ontology elements, and $\mathcal{I}$ comprises states of activities and goals as well as states of data objects by representing a snapshot of the case instance at the instant in time in which an event occurs (i.e., state change of activity, goal or data). An ontology expression consisting of a (constraint) concept $\mathcal{C}$ with a runtime state $\mathcal{C}.s$ is matched by $\mathcal{E}$ iff $\exists i \mid i \in \mathcal{I} \wedge i : \mathcal{C} \wedge i.s = C.s$. An ontology expression consisting of a relation $\mathcal{R}$ is matched by $\mathcal{E}$ iff $\exists (i_1, i_2) : \mathcal{R} \mid i_1 \in \mathcal{I} \wedge i_2 \in \mathcal{I}$.

3.4. Practical Scenario

This section extends the motivating example (cf. Section 3.2) and discusses how the proposed approach can be applied to realize automated support for enacting EPA rules (cf. EPA [13]).

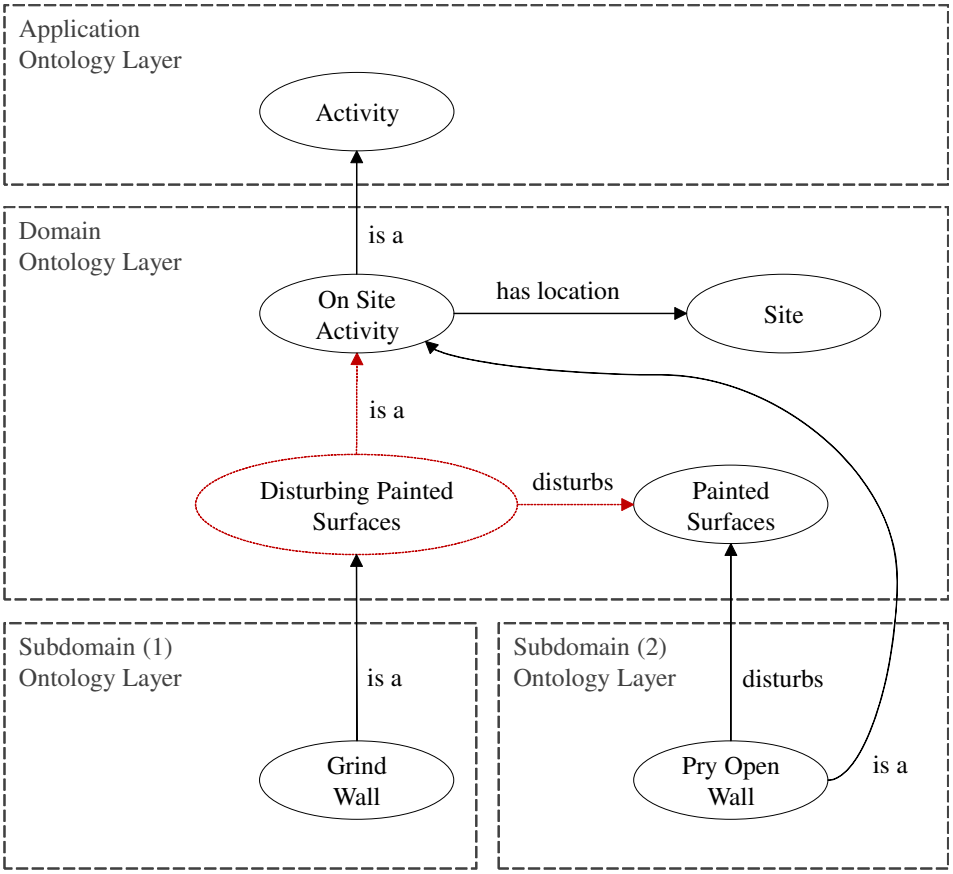


Figure 3.3.: Layered ontology for “disturbing painted surfaces” activities

Figure 3.3 contains an ontology that covers both the abstract concept of "disturbing painted surfaces" (DPS) and specific DPS activities of subcontractors. The concept *Disturbing Painted Surfaces* plays the special role of a *constraint concept* and represents some *On Site Activity* that disturbs *Painted Surfaces*. Consequently, all concepts derived from *Disturbing Painted Surface* have implicitly a relation *disturbs* to the concept *Painted Surfaces*. Moreover, concepts that have a relation *disturbs* to *Painted Surface* and are at the same time derived from an *On Site Activity* concept (that includes children of the *Activity* concept) represent at the same time a *Disturbing Painted Surfaces* concept. Child activities like *Grind Wall* can be derived from this concept. Alternatively, the concept *Pry Open Wall* has the relation *disturbs* and is derived from the more general concept *On Site Activity*. By having the *disturbs* relation and by being at the same time an *On Site Activity*, it is also a *Disturbing Painted Surfaces* concept.

The sample ontology in Figure 3.3 is organized in layers. Subcontractors are enabled to create and maintain their own *Subdomain Ontology Layer* that builds upon the *Domain Ontology Layer*. This layer is maintained by the umbrella organization that employs the subcontractors. The *Application Ontology Layer* is the interface to the business process management software. By having the freedom to modify the ontology of the subdomain, subcontractors can add new activities to the ontology any time and benefit from the behavioral constraint support that has been defined on the basis of existing concepts.

For that reason, ontology-based behavioral constraints are derived from the EPA compliance guide. EPA rules are only applicable to "*activities that disturb painted surfaces in a home or child-occupied facility built before 1978*". Consequently, it must be checked whether the building is a home or child-occupied facility built before 1978 prior to disturbing its surfaces, so as to take precautions due to potential lead contamination. A business user can create such behavioral constraints by using the already existing, more general domain concepts *House* and *Child-Occupied Facility*, and derive the new constraint concepts *Potentially Lead-Contaminated Home* and *Potentially Lead-Contaminated Child-Occupied Facility* that become effective whenever the construction date attribute of the parent concept is smaller than 1978 or not yet defined (Figure 3.4).

Flexible business process approaches like ACM are *goal-driven*. Consequently, several goals related to preparing the work area must be reached in advance of disturbing potentially lead-contaminated surfaces. These goals are (list contains examples and is not meant to be exhaustive):

- Warning signs posted at entrance to work area
- Floors in the work area covered with taped-down plastic
- Doors in the work area closed and sealed

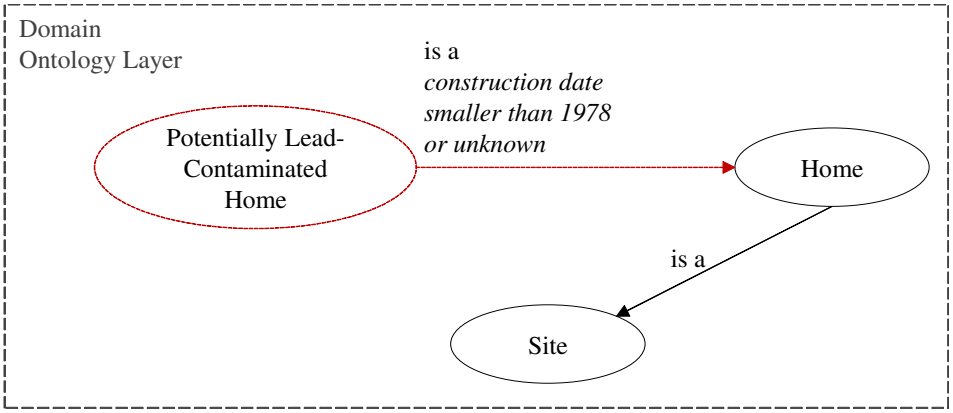


Figure 3.4.: Constraint concept with constraint defined on attribute of the parent concept

Business users can create such goals by deriving child concepts from the more general *Goal* concept (Figure 3.5). To make sure that these goals are reached before any potentially lead-contaminated surface is disturbed, the business user makes use of the constraint editor to create behavioral constraints. Listing 3.1 contains four *Precedence* constraints. Important goals related to the preparations for lead-contaminated work areas must be reached first and only then is it allowed to perform DPS activities. As can be seen for these sample constraints, the controlled natural language grammar of the behavioral constraint language ("operand1 requires operand2 earlier") becomes intertwined with expressions originating from the domain ontology. The operator "has location" stems from the relation with the same designation between the concepts *On Site Activity* and *Site*. The behavioral constraints are close to natural language, which aims at making them comprehensible for business users, but they still follow a structured, well-defined scheme comprising the constraint grammar and the ontology, which is the basis for automated tool support for constraint enactment.

```
Warning signs posted at entrance to work completed precedes
    Disturbing Painted Surfaces started has location Potentially
    Lead-Contaminated Home or Potentially Lead-Contaminated Child-
    Occupied Facility

Floors in the work area covered with taped-down plastic completed
    precedes Disturbing Painted Surfaces started has location
    Potentially Lead-Contaminated Home or Potentially Lead-
    Contaminated Child-Occupied Facility

Doors in the work area closed and sealed completed precedes
    Disturbing Painted Surfaces started has location Potentially
    Lead-Contaminated Home or Potentially Lead-Contaminated Child-
```

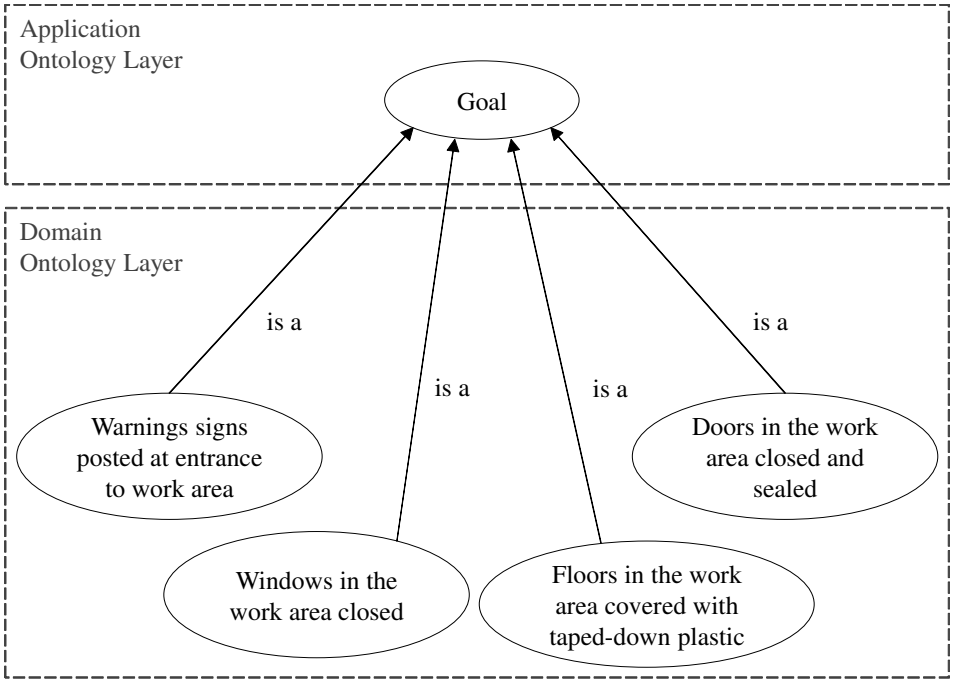


Figure 3.5.: Creating specific goals for EPA rules

```
Occupied Facility
```

Listing 3.1: Behavioral constraints related to preparations

Automated support for enacting behavioral constraints can be provided by translating the constraints to a formal verification language. Possible formalisms and techniques include Computation Tree Logic (CTL), LTL (Linear Temporal Logic), EPL (Event Processing Language) and EC (Event Calculus). For the proposed ontology-based behavioral constraint approach, we do not intend to prescribe a specific underlying formalism or verification technique. Nevertheless, we would like to showcase how the enactment of ontology-based behavioral constraints works from beginning to end. For this purpose, we use LTL, since it is an established formal language for the definition of desired properties of hardware and software systems (cf. Pnueli [26]). LTL has become a *de facto* standard in business process verification due to the possibility of translating LTL formulas to nondeterminstic finite automata (NFAs) for *runtime verification* (cf. [56], [63]) of business processes, and the extensive use of LTL as a specification language in *model checking* (cf. [15], [17]) of business processes and for *compliance* and *security* modeling (cf. [62], [64]). LTL representations of the behavioral constraints are automatically generated by applying the mappings proposed by Dwyer et al. [27] and by properly resolving propositional variables based on the domain ontology. We will now discuss an exemplary enactment of the first

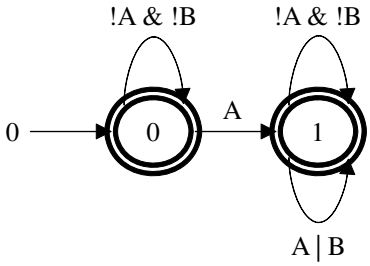


Figure 3.6.: NFA for *precedence* constraint

ontology-based behavioral constraint in Listing 3.1, which is defined as

```
Warning signs posted at entrance to work completed precedes
    Disturbing Painted Surfaces started has location Potentially
    Lead-Contaminated Home or Potentially Lead-Contaminated Child-
    Occupied Facility
```

For automatic enactment as LTL specification, the operands

```
Disturbing Painted Surfaces started has location Potentially Lead-
    Contaminated Home or Potentially Lead-Contaminated Child-
    Occupied Facility
```

denoted as B, and

```
Warning signs posted at entrance to work completed
```

denoted as A, must be resolved to propositional variables to be able to enact the *Precedence* constraint (cf. Dwyer et al. [27]) as LTL formula $\neg B \; \mathcal{W} \; A \equiv \Box \neg B \vee \neg B \; \mathcal{U} \; A$ by an NFA (Figure 3.6) that represents the formula (cf. De Giacomo et al. [63]).

As a concrete case let us consider the renovation of a flat in a building which was constructed in 1971. A new case is opened in the business process software for this renovation. The renovator adds *Home* as the site of the renovation and enters the *construction date* 1971. From this moment on, the case instance has an instance of the concept *Home* which is at the same time an instance of *Potentially Lead-Contaminated Home* because of the construction date, which is smaller than 1978. During the enactment of this case, the renovator can perform various ad-hoc activities. For example, when the renovator intends to start wall grinding (by instantiating the *Grind Wall* concept and starting the activity) on this site (*has location* relation), the propositional variable B becomes true. Since the *Grind Wall* instance has the *is a* relation to *Disturbing Painted Surfaces* and the *has location* relation to the *Home* instance, which is also a *Potentially Lead-Contaminated Home* because of the construction date, B is resolved to *true*. Consequently, $B = true$ is the input to the automaton which is in state 0, but the automaton does not accept such an input. Thus, the *Grind Wall* activity would violate the behavioral constraint. Once the

renovator enters that warning signs are posted at entrance to work, $A = true$ is sent to the NFA. This causes a state transition of the automaton from state 0 to state 1. From that moment on *Disturbing Painted Surfaces* activities can no longer violate the behavioral constraint since all future inputs are accepted by this automaton.

3.5. Discussion

The proposed ontology-based behavioral constraint approach enables the use of domain knowledge in behavioral constraints of business processes. As a result of this integration, the behavioral constraints can be formulated in a structured natural language that offers both the elements stemming from behavioral constraint patterns and the elements of the business domain. Since the resulting ontology-based behavioral constraints are formally defined on the basis of the ontology and constraint patterns (with one or more underlying formal verification techniques), automated verification support for the enactment of these constraints becomes possible during the handling of cases by business users. Since the general idea of ontology-based behavioral constraints can be used independently from a specific grammar and notation of behavioral constraints, it might be integrable with existing approaches such as Declare [56] and DPIL [65]. Declare provides a graphical front-end for the definition of behavioral constraints which become directly enacted as declarative workflows (by automata that represent Linear Temporal Logic specifications or by Event Calculus run as Prolog programs). By using the graphical notation of different arrow connectors, the user can define behavioral constraints among activities which are symbolized by boxes with labels. These approaches can integrate the proposed approach by enabling the user to specify ontology expressions instead of those labels. DPIL (Declarative Process Intermediate Language) is a textual language for the specification of behavioral constraints that become enacted by the JBoss Drools rule engine. Instead of defining business process elements such as data objects and activities in DPIL scripts, a user could select such elements from the business ontology. Integrating domain ontologies into those existing behavioral constraint approaches might improve usability for business users.

Making behavioral constraint authoring accessible to business users might be a controversial topic since the actions of business users are at the same time subject to these behavioral constraints. Classically business processes are defined as flow charts (e.g., BPMN models) and business users are supposed to follow exactly the steps as prescribed by the process model. This enables a strong control but tempts the business user to perform undocumented actions (i.e., actions that are not disclosed to the IT system) if it should become necessary to leave the predefined flow of actions. Knowledge-intensive business domains require more flexibility which has led to constraint-based business processes. Those do not longer require the business user to stick

to predefined, step-by-step business processes. The main objective of constraint-based business processes is supporting the business user to stay compliant with necessary rules (e.g., regulatory laws, standards) while enabling largest possible flexibility. Thus, behavioral constraints must be seen as a support feature for business users to stay compliant. Behavioral constraints are not meant to hinder the work of the business user; if business users were excluded from having control over behavioral constraints, it would become likely that they would again perform undocumented actions just to meet the rules of the IT system.

Ontology-based behavioral constraints create new challenges which are not covered by the work presented in this chapter. It is an open research challenge how to keep the behavioral constraints consistent with an evolving ontology. Concepts of the ontology and relations between them might change over time, which can lead to disappearance of ontology elements that are part of behavioral constraints. This might have different possible implications for the behavioral constraint: The existence of the behavioral constraint might no longer be necessary, so it may be deleted. Alternatively, the need for this behavioral constraint might still exist, but require an adaptation to the current ontology. If and how business users can be supported to make such an adaptation (e.g., by keeping track of ontology changes and leveraging change histories of ontology elements of behavioral constraints) must be further investigated. In general, the maintainability of behavioral constraints by business users must scale evenly with a growing number of behavioral constraints. These challenges lead to many new interesting possibilities for future research.

Further studies are required on the understandability of the behavioral constraint patterns by business users. It could be of interest to find out whether the total count of possible behavioral constraint patterns offered has an influence on understandability. Of interest is also the influence of a specific grammar of such constraint patterns on the understandability.

3.6. Related Work

Yu et al. propose a verification approach for service compositions that are described by BPEL models [66]. Their approach implements a constraint grammar, based on the constraint patterns by Dwyer et al. [27], directly in an ontology. The focus of their work seems to be the realization of these constraint patterns by an ontology, but the connection to domain-specific ontologies is not discussed. Since the set of constraint patterns is maintained by technical users, in our case we do not see the requirement for representing the grammar as part of an ontology. Our approach links the constraint pattern-based grammar with expressions stemming from a domain ontology.

While the previously mentioned approach is pattern-based, an approach by Yan et al. seeks to directly abstract LTL formulas from natural language [67]. In particular, expressions in struc-

tured natural language are directly transformed into LTL. However, the controlled grammar of the language still contains LTL operators (“globally”, “always”, “eventually”) that might be hard to comprehend for non-technical business users as language elements. An integration of domain knowledge by leveraging ontologies is not part of their approach.

SBVR (Semantics of Business Vocabulary and Business Rules) is an adopted standard released and maintained by the OMG (Object Management Group) [24]. The intention behind SBVR is to offer a structured natural language approach for the authoring of business rules. However, automated verification support for SBVR is challenging, especially for non-static rules with temporal aspects. Elgammal & Butler propose a manual translation of SBVR rules to an LTL-based graphical compliance rule language [68]. Solomakhin et al. propose a formalization of SBVR rules by first-order deontic-alethic logic (FODAL) to support automated reasoning based on an ontology. Manaf et al. [69] propose SBVR as a specification language for service choreographies. They show with an example that the same rule can be expressed by DecSerFlow patterns [42] (LTL pattern-based specification language for service flows) and SBVR. The DecSerFlow language is also the foundation for the declarative workflow approach Declare [56]. Declarative workflow approaches predominantly enable the definition of behavioral constraints with a temporal context which specify business processes in a declarative manner, but they do not focus on the integration of domain ontologies. Our approach aims at bridging these worlds by enabling business users to define behavioral constraints that have a temporal focus, in an editor that makes use of the domain ontology to allow for defining rules in the well-known business terms of the specific business domain.

The work presented in this chapter is also related to the huge body of compliance research. Van der Werf et al. propose “Context-Aware Compliance Checking” [70] by deriving ontologies from audit logs and checking rules in Semantic Web Rule Language (SWRL) [71]. Yip et al. propose a similar approach [72]. Despite using ontologies, their approaches are different from ours since they do not focus on temporal behavioral constraints and constraint authoring in structured natural language. Elgammal et al. propose a Compliance Request Language (CRL) that is formally grounded on temporal logic and mapped to LTL formulas [23]. They identify a set of compliance patterns comprising atomic patterns (extension of the order and occurrence patterns proposed by Dwyer et al. [27]), composite patterns (e.g., mutual exclusion and coexistence), resource patterns (i.e., rules related to roles) and timed patterns (i.e., rules related to quantitative time). CRL does not yet consider ontologies. Awad et al. propose a graphical language called BPMN-Q for the specification of compliance requirements which can be transformed to temporal logic formulas [47], but ontologies are not considered. Ly et al. propose a Compliance Monitoring Functionality Framework (CMFF) which contains ten criteria, and they analyze existing compliance monitoring approaches based on their framework [48]. The framework con-

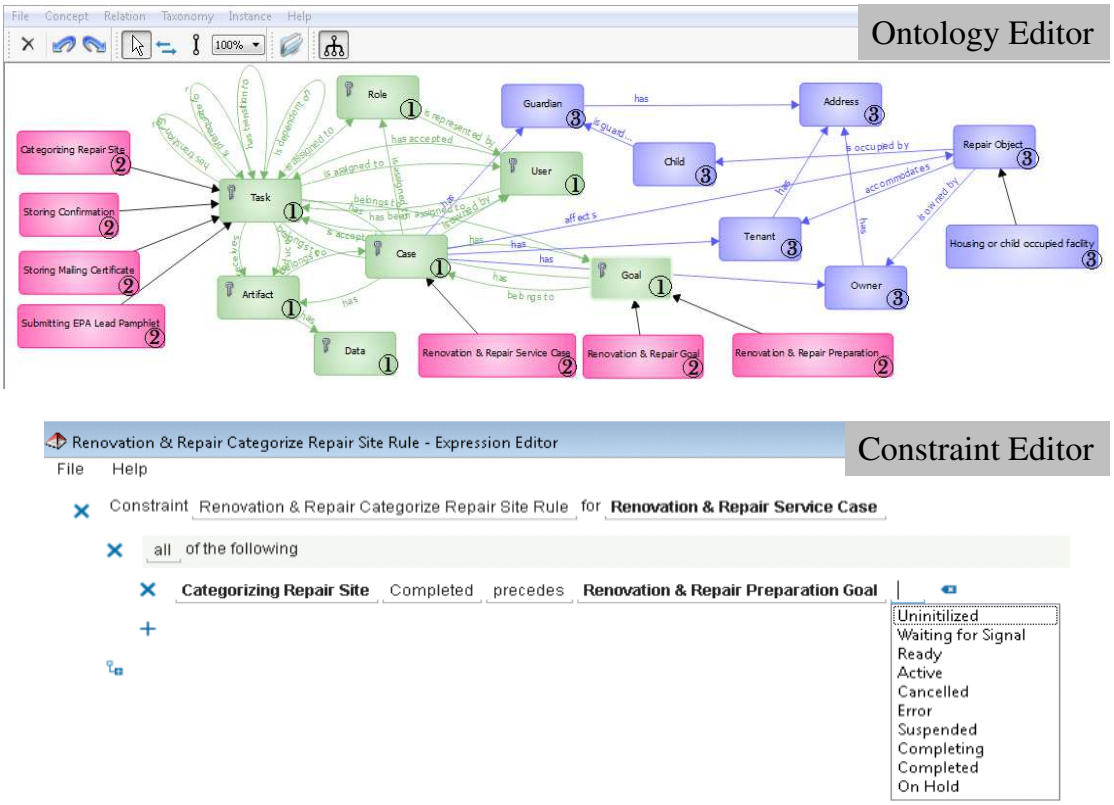


Figure 3.7.: Ontology editor and constraint editor in ISIS Papyrus

siders many important compliance aspects such as activity lifecycles, non-atomic activities, data, roles, reactive and proactive support, but it does not specifically focus on specification languages or ontologies.

In summary, it can be said that on the one hand there exists a large body of work on compliance of business processes which does not explicitly consider the integration of ontologies, and on the other hand there are efforts to create business rules on basis of an ontology. Our work is positioned in between. It is grounded on well-established temporal logic patterns that allow an automated verification (like existing compliance and declarative workflow approaches [23], [56]), and it allows to define parts of constraints on basis of knowledge stemming from an ontology (as it is also the goal of standardization efforts in SBVR [24]).

3.7. Implementation

The approach has been implemented as a prototype extension of the Papyrus Platform[1]. Figure 3.7 shows a screenshot of the ontology editor and the constraint editor.

3.8. Conclusion and Future Work

The presented approach aims at supporting behavioral constraint authoring that is approachable to business users by making use of ontology-based domain knowledge. The approach is applied in the context of a realistic scenario (Section 3.4), but further evaluations are necessary. There exist many assumptions that must be further evaluated: by allowing the business user to actively shape the behavioral constraints, the approach may allow a faster integration and adaption of behavioral constraints, eliminating long delays caused by static maintenance cycles for integrating and modifying behavioral constraints. Consequently, it may also help avoid the enacting of obsolete, faulty and unnecessary constraints that might drive business users into cheating the process management software by performing undisclosed actions. Thus, the approach may also contribute to a more complete documentation (i.e., audit trails). Moreover, it may encourage business users to capture their (formerly implicit) knowledge as explicit (non-mandatory) behavioral constraints. The evaluation of those assumptions by qualitative and quantitative studies are interesting opportunities for future research. Moreover, open challenges regarding consistency and maintainability (as outlined in Section 3.5) must be resolved.

[1] `http://www.isis-papyrus.com`

4. Behavioral Consistency Support Framework

Enabling knowledge workers in creating behavioral constraints, as proposed in the previous section, as well as the automated enactment of behavioral constraints, is the basis for the behavioral consistency support framework that will be presented in this chapter. This chapter presents a framework on how to enable support for behavioral consistency in the context of ACM by means of behavioral constraints. Since ACM applications undergo constant change, there must be ways to introduce behavioral constraints on the fly. Currently, constraints (and similar alternative solutions) are predominately maintained by technical users, which results in long maintenance cycles. The present framework aims at enabling faster adoption of changing behavioral consistency requirements, both explicitly—by enabling non-technical users (knowledge workers) to define and adapt constraints—and implicitly—by learning from the decisions taken by other knowledge workers during case enactments. The former is achieved by supporting domain knowledge maintained in an ontology. The latter is supported by a recommendation approach that enables an automated knowledge transfer between knowledge workers by propagating knowledge, best practices, and the handling of constraints and their violations.

4.1. Introduction & Motivation

Knowledge-intensive processes are challenging: on the one hand, knowledge workers must cope with rapidly changing environments and handle very specific situations which can not be fully designed before the process is executed and require runtime flexibility. On the other hand, the domains in which knowledge work is necessary are often heavily regulated (e.g., the insurance or health sector) and staying compliant is difficult if the IT system does not provide the necessary support for following the imposed compliance requirements.

Adaptive Case Management (ACM) is a paradigm for flexible, goal- and knowledge-driven business process management (cf. Swenson [3]). ACM enables knowledge workers to actively shape the way case instances are executed while documenting their actions and providing the needed IT support. ACM solutions must support knowledge workers in staying on the right

track instead of hindering their work. Long maintenance cycles might lead to the enactment of outdated compliance rules and new, useful compliance rules might be introduced with month-long delays.

This chapter presents a framework that aims to enable faster adoption of changing compliance requirements, both explicitly, by enabling non-technical users (i.e., the so-called knowledge workers) to define and adapt constraints, and implicitly, by learning from the decisions made by other knowledge workers during case enactments. The former is achieved by supporting domain knowledge, realized as an ontology which links domain-specific concepts (defined by knowledge workers) to technical concepts of the ACM ontology (defined by technical users). The latter is supported by a recommendation approach that enables an automated knowledge transfer between knowledge workers by propagating knowledge, best practices, and the handling of constraints and their violations. This recommendation approach is a machine learning problem that can be described as a decision learning (classification) problem. The framework describes the steps that are necessary to prepare data from past case enactments for the learning process, namely the selection of those past case enactments that must be considered for achieving a specific purpose (e.g., to overcome a specific constraint violation), and the preprocessing that uses ontology knowledge to prepare information for the learning process, such as temporal relationships (e.g., relative time between the completion of one task and another).

4.2. Framework Overview

Figure 4.1 shows an overview of the proposed behavioral consistency support framework for ACM. The framework comprises one central concept, the ontology, one targeted type of stakeholder, the knowledge worker, and five supported functionalities:

(Central Concept) The *Ontology* comprises the ontology of the ACM domain and the ontology of a business domain (or the ontologies of several business domains). Business domain-specific concepts (e.g., a domain-specific activity) are put in relation to ACM concepts (e.g., a generic user task that is supported by the ACM application). While the ACM ontology is maintained by technical users (e.g., the vendor), knowledge workers can adapt the ontology of their domain. Constraints can be formulated on the basis of the ontology of the domain, which means that knowledge from the domain supports the creation of constraints. During constraint enactment, it is known which concept a generalization of which other concept is and how concepts are related. Consequently, constraints can be evaluated based on existing case elements, such as tasks, goals, and data objects, which are concrete instances of those concepts. For example, instead of recommending very specialized next actions to knowledge workers during case enactment, it probably makes sense to propose a more general approach (i.e., a parent concept) that fosters

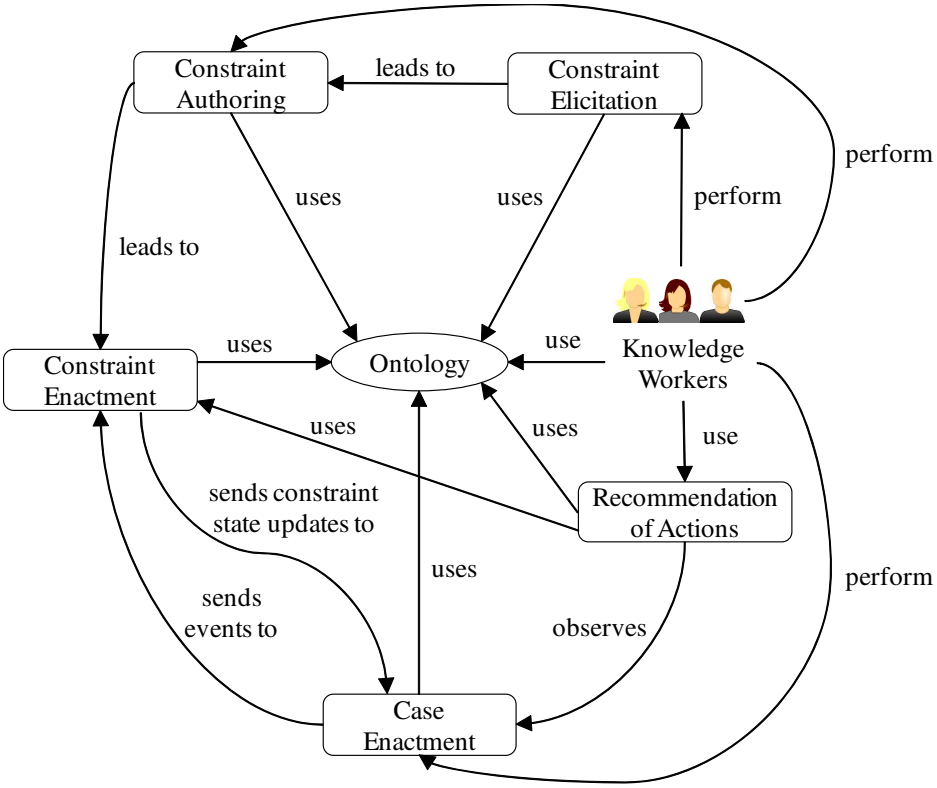


Figure 4.1.: Overview of Compliance Support Framework

a greater scope of possibilities instead (e.g., instead of proposing the action 'payment by credit card', the more general action 'payment' can be proposed).

(Targeted Stakeholder) *Knowledge workers* are the main drivers of the ACM system. They handle cases to the best of their knowledge while having to cope with unforeseeable situations commonly. Their knowledge is the foundation of this constraint framework. On the one hand, their knowledge is directly used to define the domain ontology. On the other hand, their knowledge is automatically processed to propagate it to other knowledge workers which enables an automatic knowledge transfer among knowledge workers.

(Functionality 1) *Constraint Elicitation* is concerned with transforming regulatory laws, standards, or best practices into internal policies, and in a next step to constraints that can be automatically enacted.

(Functionality 2) *Constraint Authoring* is concerned with defining, modifying and managing constraints. The language used to create constraint expression is partly derived from the ontology and can make use of the domain concepts and their relations. Other parts of the constraint language are related to specific constraint patterns (e.g., [23], [73]).

(Functionality 3) *Constraint Enactment* monitors constraint instances. Reacting to constraint modifications and carrying out the necessary steps (e.g., removal of an obsolete constraint instance) are important responsibilities of this component. Constraint enactment reevaluates the states of constraint instances with every new relevant received event. State changes, in particular those that violate constraints, are reported to the knowledge worker.

(Functionality 4) *Case Enactment* comprises the daily business of knowledge workers, namely the handling of cases to progress towards specific goals while coping with unforeseeable circumstances on a regular basis. Documents and data objects are continuously created, modified and reviewed. Tasks can be delegated to, and carried out by, knowledge workers belonging to different professions. During case enactment, knowledge workers are continuously informed about constraint violations.

(Functionality 5) *Recommendation of Actions* enables automated knowledge transfers between knowledge workers. Whenever the knowledge worker seeks advice from this component, the current circumstances and the history of the case are used to find one or more appropriate candidate actions that are in line with the decisions made by knowledge workers that have been in a similar situation. This component propagates information, such as evolved best practices and compensation actions between knowledge workers.

Knowledge workers are characterized as being knowledgeable in their business domain. Most often, this knowledge exists merely as implicit knowledge, so it is not explicitly available in an organization or company. For the proposed framework, this has two important consequences:

- The application of their knowledge must not be hindered by the IT system (i.e., a prescrip-

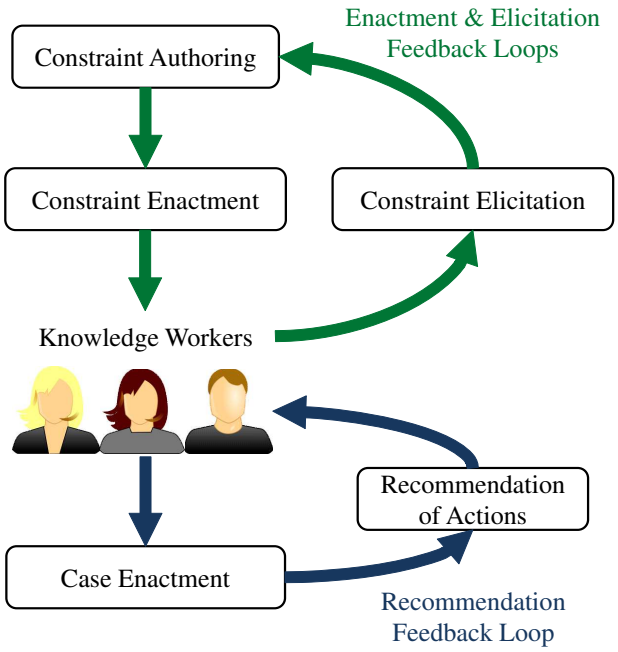


Figure 4.2.: Knowledge Transfer Cycle

tive approach must be avoided).

- This implicit knowledge is an opportunity to improve the enactment of cases in general (e.g., less experienced business users might benefit from the knowledge of more experienced colleagues).

The proposed constraint framework aims at providing adequate support for these two implications. On the one hand, the framework does not intend to prohibit any action of knowledge workers. On the other hand, it is designed to support knowledge transfers through the ACM system to other knowledge workers. Figure 4.2 schematically shows the knowledge transfer cycle in the design of the framework. There exist three feedback loops that potentially cause the actions of knowledge workers to become influenced by other knowledge workers.

4.2.1. Recommendation Feedback Loop

In Figure 4.2, the loop starts with *Knowledge Workers* who handle cases (*Case Enactment*). The *Recommendation of Actions* component observes their actions continuously and learns from them. Eventually, *Knowledge Workers* can query *Recommendation of Actions* whenever they require advice. If a similar situation has occurred in the past, then a proposal for potential next actions is made. This loop directly feeds back the decisions of knowledge workers to other

knowledge workers. Consequently, the function of the *Recommendation Feedback Loop* is the fast propagation of formerly implicit knowledge. This may include compensation actions to recover from a constraint violation and evolving practices how to handle specific situations.

4.2.2. Enactment Feedback Loop

This loop starts with *Knowledge Workers* who observe that a large quantity of *Constraint Enactments* of a specific constraint are in a state of violation. This might be an indication for changing this constraint, so *Constraint Elicitation* is performed to evaluate this particular constraint. If the set of constraints is adapted (*Constraint Authoring*), *Constraint Enactment* will propagate this change to *Knowledge Workers* during *Case Enactment*. Consequently, the main objective of the *Enactment Feedback Loop* is to react to constraint violations. If a large number of constraint instances with a particular constraint in a state of violation is observed, then there might be a need to adapt the set of constraints. Maybe the rules of the business have changed or an error was introduced into the set of constraints unintentionally.

4.2.3. Elicitation Feedback Loop

This loop starts with *Knowledge Workers* who see the need for *Constraint Elicitation* (e.g., new constraints are needed to implement a compliance document which describes upcoming compliance requirements) and introduce new constraints by *Constraint Authoring* that become effective as constraint instances in *Constraint Enactment*. The *Case Enactment* by *Knowledge Workers* might be influenced by the changed set of constraints. In contrast to the *Enactment Feedback Loop*, this loop is not triggered by violations occurring during *Constraint Enactment*. Consequently, the main objective of the *Elicitation Feedback Loop* is to integrate knowledge workers directly in the constraint elicitation and creation process. Knowledge workers participate in constraint elicitation and actively contribute their domain knowledge while working on new internal policies that they formally specify as constraints. Once new policies are enacted as constraints of the ACM system, the knowledge workers' future decisions are potentially influenced by them.

4.3. Framework Components

4.3.1. Ontology

Ontologies are an efficient way to organize information. The set of components of ontologies include concepts, attributes, relations and instances. A domain ontology represents the knowledge of a specific business domain. Depending on the business domain, the ontology can look

fairly different. For the ACM system, it is important that domain concepts are related to technical concepts of the ACM domain. Multiple business ontologies can be combined with this single ACM ontology. Through this, the ACM system could, for example, be made aware that the domain concept 'Payment' is actually in technical terms a 'Task' (Figure 4.3). The example given in Figure 4.3 illustrates conceptually how the ontology of the domain can be connected to the ontology of the ACM system and how both are integrated in a constraint language (for illustration purpose also described by concepts and relations). Through this architecture, it becomes possible to define constraints on the basis of higher-level concepts. For example, there could be a policy "Shipped orders must eventually be paid" which can be represented in a textual constraint language as "Shipping is finished leads to Payment is finished". There are concepts derived from Payment, like Payment by Credit Card that are also covered by this constraint. Even a tighter integration of the business ontology is possible: consider that not only concepts can be mapped to ACM elements but also larger structures of the domain ontology. For example, consider two concepts, 'Order' and 'Customer', and their relation 'can be placed by'. This would allow to derive a task 'Place Order' with performer 'Customer' directly from the domain ontology and further allows to use this domain knowledge not only during case enactment but also for the definition of constraints.

4.3.2. Constraint Elicitation & Constraint Authoring

The main purpose of *Constraint Elicitation* is the extraction of internal policies from compliance documents which will in further consequence be introduced to the ACM system as constraints, and the evaluation of existing constraints with regards to internal policies. In a next step, these policies are fed into the ACM system as constraints (*Constraint Authoring*).

Figure 4.4 illustrates *Constraint Authoring*. Knowledge workers can use the *Constraint Editor* to author constraints. Moreover, they can define the ontology of their business domain. A *Constraint Editor* provides the functionality to create *Natural Language Constraints*. These constraints consist of parts stemming from a *Constraint Grammar* that abstracts underlying formal verification techniques, and of other expressions that reflect the concepts and relations defined in the *Ontology* (cf. Chapter 3).

Since business users do not necessarily have a technical background, constraints should be easily understood. Consequently, technical constraint authoring must be avoided and a natural language approach is suggested. In the current state-of-the-art, the grammar or feature set of the language is rarely defined as part of the ontology (cf. [74]), but rather separate from it in a dedicated grammar that makes use of the ontology elements (cf. [75]). Either way, the core function of the domain ontology is the support of domain-specific concepts and relations, which are important not only for the definition of constraints but for the case enactment in general since

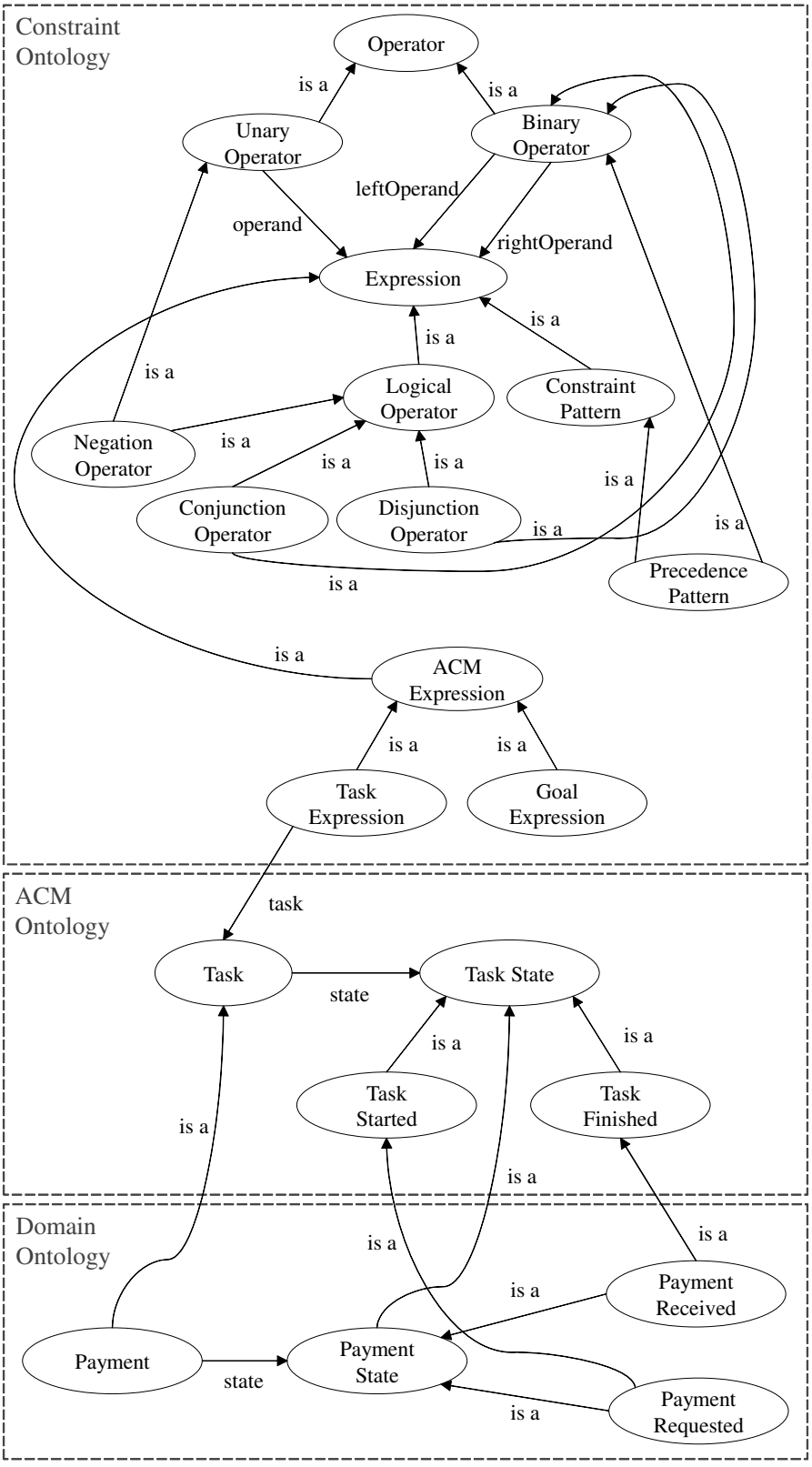


Figure 4.3.: Ontology Example

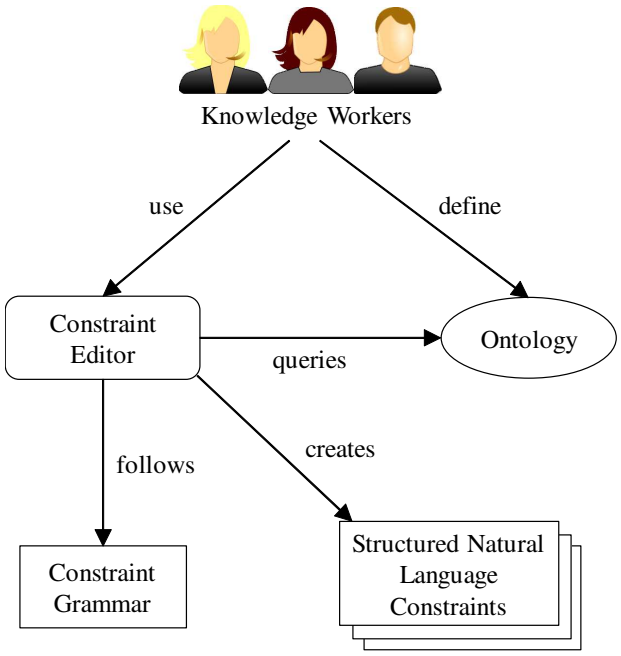


Figure 4.4.: Constraint Authoring

knowledge workers operate based on their known domain concepts instead of technical terms.

Each created constraint has *five properties*. The first three properties are proposed by Leitner et al. [76] for flow-driven business processes. We adapt them to fit into the scope of ACM, discuss them in relation to our framework, and add an important fourth and fifth property named *Coverage* and *Regionality*:

(1) *Localization*: In which case instances must the constraint hold?

- Inter-Organizational Localization: The constraint is enacted for instances of cases that exist in more than a single organization. If the IT system is incoherent, the constraint must be either propagated to other organizations for enactment on their side, or the constraint is enacted centrally with the propagation of results to other organizations. Our framework assumes a single, coherent ACM system which can be used by several organizations (e.g., companies).

- Inter-Case-Concept Localization: The constraint is enacted for all instances of several cases. Hence the constraint is imposed on multiple case concepts. This kind of localization requires a classification of a case instance to relate it to a case concept. This can be done either automatically (e.g., a specific case template is instantiated) or manually (e.g., the knowledge worker opens a case instance and decides for specific case concepts), depending on the circumstances. It is supported by the framework.

- Intra-Case-Concept Localization: The constraint is enacted for all instances of a specific case concept. It is supported by the framework.
- Intra-Case-Instance Localization: The constraint is enacted for a specific case instance only. This will be rarely the case since specialized compliance treatment of a case instance is rather unlikely. Consequently, Intra-Case-Instance Localization is not supported by the framework.

(2) *Span*: What happenings are observed for the enactment of the constraint?

- Intra-Case-Instance Span: The happenings of a single case instance are considered. That is, the constraint enactment does not observe happenings in other case instances for this constraint instance. This is the most common kind of span (cf. e.g., [23], [56], [77], [78]). Since we have not yet encountered the requirement for other kinds of span in case studies (e.g., [79], [75]) as well, our solution supports this kind of span exclusively. Nevertheless, if a need for other kinds of span evolves, the proposed constraint support framework can consider the other kinds of span as well.
- Inter-Organizational Span: The happenings of all case instances of multiple case concepts that exist in multiple organizations are considered.
- Trans-Organizational Span: The happenings of all case instances of a single case concept that exists in multiple organizations are considered.
- Inter-Case-Concept Span: The happenings of all case instances of multiple case concepts are considered.
- Intra-Case-Concept Span: The happenings of all case instances of a case concept are considered.

(3) *Dependency*: A constraint is either independent of its previous enactments or dependent. If it is dependent, then it becomes, for example, possible to enact it only every second time. Since we have not yet discovered a practical use case for the *Dependency* property, it is not supported by the proposed framework.

(4) *Coverage*: A constraint covers specific periods of time. There exist two periods:

- Enactment Period: The constraint is enacted in this period of time. If the constraint is not yet active (current time $<$ enactment start time) or already expired (current time $>$ enactment end time), the constraint enactment component does not consider it.

- Retrospect Period: Past events, which occurred before the constraint entered the enactment period, can have an influence on the current state of a constraint, and must therefore be taken into account by constraint enactment for a defined period of time. Obviously, the retrospect period ends with the start of the enactment period. The retrospect period requires the ACM system to provide events falling into this period of time.

(5) *Regionality*: If the organization is (or the organizations are) spread over different regions (e.g., different countries), each region might have different regulations that must be met. Additionally, there might be cultural differences in the way in which cases are handled by knowledge workers. Our approach considers this by allowing the assignment of constraints to specific case concepts for different regions that are derived from more general case concepts.

4.3.3. Case Enactment

Case Enactment is the basic functionality of every ACM system. Knowledge workers collaboratively work towards goals by performing tasks, and data and content (e.g., documents) are continuously modified. This component must capture the state and actions of a case extensively. Only then is the constraint framework working effectively. If knowledge workers perform actions that are not disclosed to the IT system, then Constraint Enactment possibly cannot provide notifications regarding violations, and knowledge transfers through the Recommendation of Actions would not work. It is important to raise the awareness of knowledge workers to properly document all of their (manual) actions.

4.3.4. Constraint Enactment

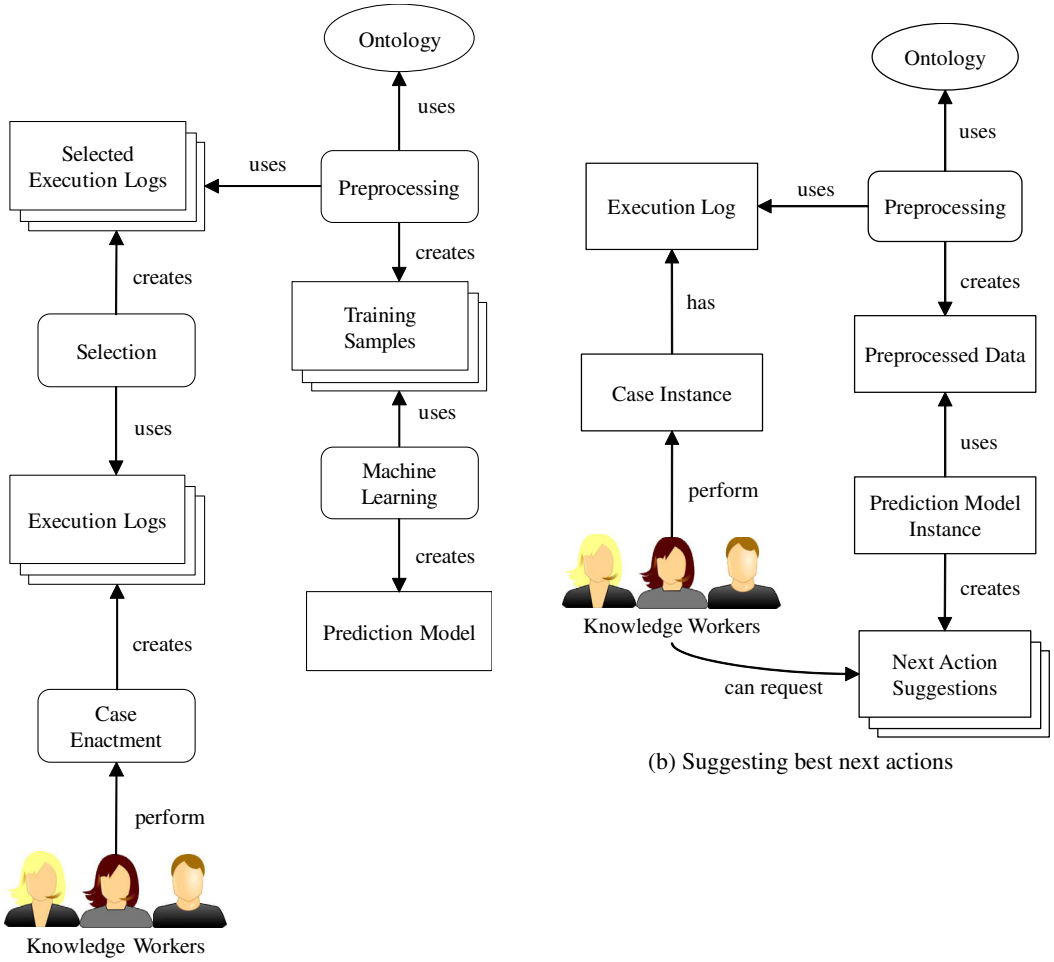
There exist various possibilities for enabling *Constraint Enactment*. Linear Temporal Logic (LTL) is an established formal way to describe specifications for both design time (cf. [15], [80]) and runtime verification (cf. [42], [56]). Dwyer et al. propose a set of property specification patterns that abstract temporal logic formulas to high-level order and occurrence patterns. Elgammal et al. build upon this pattern set and create a compliance language [23]. Temporal patterns can be represented by different underlying formalisms and checking techniques. For example, Complex Event Processing (CEP) can process a large quantity of events in close to real-time and is applied for temporal pattern-based runtime verification of business processes in recent studies ([77], [79]). For a detailed survey on approaches and categorization based on functionalities, we refer the interested reader to the Compliance Monitoring Framework (CMF) as proposed by Ly et al. [48]. The CMF framework does not yet contain criteria to categorize the support for constraint authoring and maintainability, or the integration of ontologies. Many existing compliance monitoring approaches still require well-trained, rather technical personnel

for creating and maintaining constraints, which is still an obstacle for the practical adoption in real-world applications. Shifting the language of constraint specification towards natural and domain specific terminology might enable non-technical business users to actively take part in managing constraints.

4.3.5. Recommendation of Actions

As Ly et al. point out in [48], so far there exists no compliance monitoring approach which supports more than seven of their identified ten CMFs. To illustrate the challenges involved with supporting specific existing CMF combinations, let us consider the combination of CMF 2 (data) and CMF 8 (pro-active management). To the best of our knowledge, none of the existing approaches provides support for this combination. Existing approaches realize pro-active support by using Constraint Programming (CP) to plan a case execution by finding solutions that satisfy all constraints eventually (cf. [81], [82]). It is hardly surprising that these approaches focus on the order of tasks, because this level of abstraction allows the modelling of an optimization problem of a reasonable size (assuming that the number of tasks is not very large). Existing approaches work as follows: The start and completion events of tasks are represented by an integer value and, given an objective function and a set of constraints, the optimization tries to find an optimal order for these events. For example, objective functions may minimize time or costs, and a constraint may demand that a variable is smaller than another variable to represent common compliance patterns, such as Precedence or Response. If data were included in this optimization process, there would be a time and a specific value for each data adaption. Consequently, the optimization problem would become infeasible to solve. Moreover, planning specific future data changes seems like a pointless exercise because it simply cannot be planned beforehand. In ACM, data is an essential part of every case, so constraints that involve data are likely to occur, but existing pro-active compliance support approaches are not able to support data-based constraints.

To overcome this limitation, we propose leveraging the decisions made by knowledge workers for the automated recommendation of subsequent actions. The *Recommendation of Actions* component seeks to leverage past actions of knowledge workers to learn from their decisions and to subsequently provide support for other knowledge workers who are in similar situations. Tran et al. propose a *User Trained Agent (UTA)* for the recommendation of actions [83]. The current functionality of the UTA is as follows: every time a knowledge worker adds an ad hoc task to a case, a training sample is generated. Once several training samples under a specific goal have been collected, features are selected [84]. Based on these features, a clustering decision tree is generated [85]. The UTA preprocesses data to some extent but does not yet consider temporal relationships that might have an influence on the decisions of knowledge workers. Moreover,



(a) Learning from the taken actions of knowledge workers

(b) Suggesting best next actions

Figure 4.5.: Recommendation

it does not yet sufficiently integrate knowledge stemming from the ontology and constraints. A prototypical extension of the UTA aims at behavioral constraint integration and improving the preprocessing capabilities while preserving a reasonable computational complexity.

The *Recommendation of Actions* component seeks to learn from actions of knowledge workers and to propagate the acquired knowledge to other knowledge workers. This can be achieved by machine learning, after proper preparation of the inputs for the learning process. Figure 4.5 (a) illustrates the learning process. The enactment of cases by knowledge workers is recorded as *Execution Logs*. *Selection* is a filtering process to consider only those execution logs that are needed to create a prediction model with a specific purpose (e.g., to help compensating specific

compliance violation). By *Preprocessing*, the *Selected Execution Logs* are prepared for *Machine Learning* as *Training Samples*. Finally, a machine learning approach creates a *Prediction Model* on basis of the provided training samples.

Figure 4.5 (b) shows how suggestions for next actions are made. A knowledge worker requests a *Next Action Suggestion* for a *Case Instance* that she or he is working on. Each enacted case instance pertains information of all happenings in this instance in an *Execution Log*. *Preprocessing* prepares the data of the log as inputs for the instantiation of a *Prediction Model*. The *Prediction Model Instance* contains probabilities for performing specific next actions.

Selection

An important aspect of preprocessing is the selection of execution logs. For example, if a knowledge worker seeks help to compensate for a compliance violation, only those execution logs might be included for the learning decisions where this compliance violation was successfully resolved. If the current case execution is compliant, it could be harmful to include execution logs into the learning process that could lead to non-compliance.

Preprocessing

A log must contain all the information that might have an influence on the decisions of knowledge workers. That includes events related to activities, or data adaption with meta-data, such as the performer and timestamp of the action, as well as information about constraint violations.

The preprocessing component prepares the raw data from logs for the automated learning process. To improve the learning process, we propose including temporal aspects of a decision (cf. [86], [87]) into the learning process. Moreover, we propose the integration of domain knowledge given by the ontology and the states of constraints. This involves diverse ways of extracting useful data, such as:

- Relative Intra-Instant Data Consideration: A relative or aggregated value is created from two or more data values for the same instant in time. For example, the account balance is computed from incoming and outgoing payments.
- Relative Inter-Instant State Preprocessing: The relative time between states of task, goals, events and constraints, such as the start or end time is calculated. For example, the time between the end of a medical examination and the start of a surgery is computed.
- Relative Inter-Instant Data Preprocessing: The relative change of data values from one instant in time to another instant in time is computed. For example, the difference of temperature data is computed from its values at instant t-1 and instant t.

- Absolute Inter-Instant Data Preprocessing: Not only relative but also a series of absolute values of the same data in different instants in time can be useful. For example: The temperature of a patient is above 39 degrees Celsius for a longer time which causes a special decision of a knowledge worker.

The more information is prepared by this preprocessing, the bigger is the resulting learning problem. Consequently, it must be carefully decided on how to enrich the provided raw data for the learning process:

- By having domain knowledge available from the ontology, preprocessing may focus on concepts which are related to each other.
- Temporal windows may be used to focus on happenings that date back a specific amount of time and to abstract from events having happened earlier.

Machine Learning

Automated learning from the decisions of knowledge workers is realized by the supervised learning approach *classification*. Based on the classifiers learned from the training data, new observations can be assigned to the category that they belong to. Categories are in the context of this framework equivalent to tasks, so that it can provide recommendations for next actions. An observation is equivalent to the state of the case execution for which the knowledge worker requests a proposal and must in the same way be preprocessed as the training samples to create a matching set of attributes. Please note that while the framework does not prescribe any specific machine learning approach for its implementation, the prototypical implementation is based on decision tree learning.

4.4. Implementation

Parts of the prototypical extension of the ISIS Papyrus ACM software that are related to behavioral constraint authoring are discussed in Section 3.7. Compliance rule evaluation at runtime updates the user interface of the related case instance and informs the business user about the current compliance state (cf. Figure 4.6). Figure 4.7 shows a screenshot of the user interface of a prototypical implementation of the recommendation component for a healthcare case. In this specific implementation of the recommendation component, the confidence of the best next action proposal is indicated by a 5-star rating[1], runtime states of behavioral constraints are included in the decision learning process, and proposals that are non-compliant with existing behavioral constraints are dropped and never shown to the knowledge worker.

[1]The 5 stars rating is an available feature of ISIS Papyrus' User-Trained Agent that has been reused.

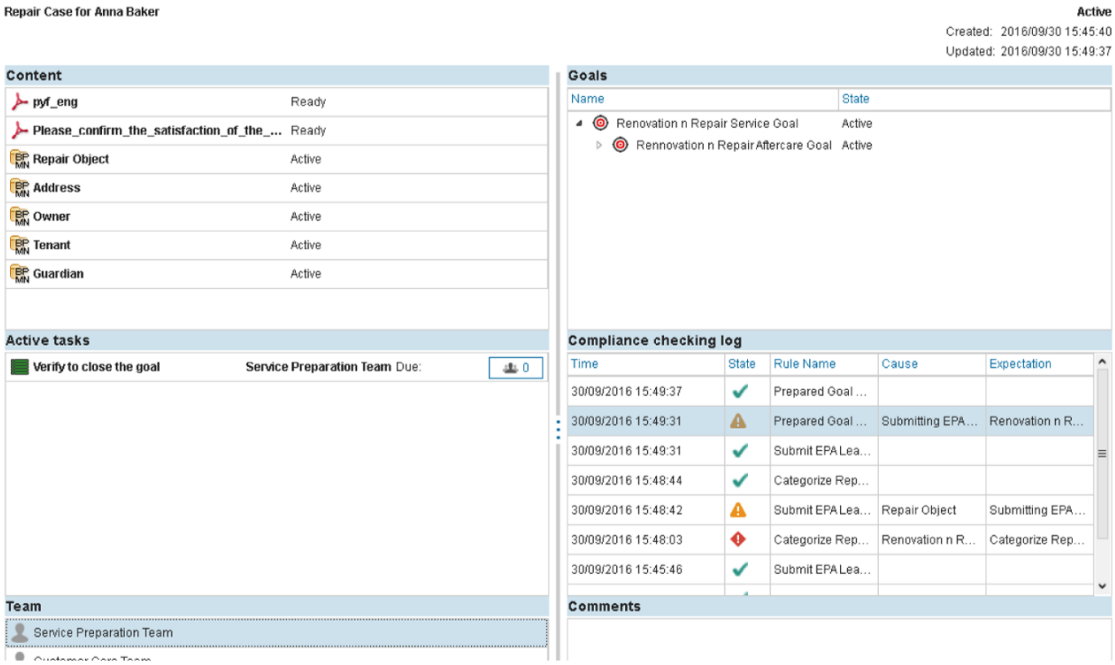


Figure 4.6.: Compliance notifications in ISIS Papyrus

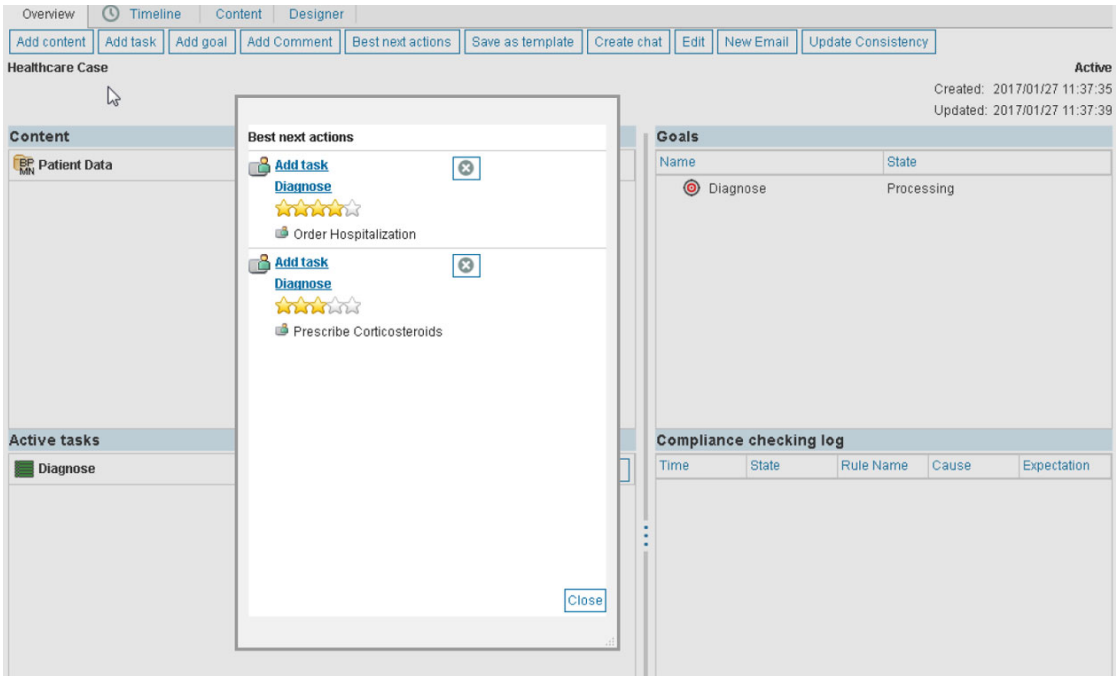


Figure 4.7.: Recommendations in ISIS Papyrus

4.5. Discussion

We present the framework in the context of ACM, but flexible business process management approaches can benefit from the framework in general. In such systems, a fast way to react to changing compliance requirements is needed, which can be provided by the proposed ontology-based constraint editor. Additionally, an automated support for deciding on next actions can be beneficial when business users are not confined to enacting predefined, flow-driven business processes.

To make knowledge explicit in form of constraints, it would be possible to extend the proposed framework by constraint discovery. By supporting the automated discovery of constraints, the framework would gain an additional feedback loop. The main objective of this loop would be the capturing of recurring behavior that is existent in a large quantity of case instances but not yet available as explicit constraint. Current constraint discovery research is predominantly focused on discovering LTL-based temporal patterns (cf. Ciccio & Mecella [78]). Another approach mines pattern-based organizational constraints for the DPIL approach (cf. Schönig et al. [88]). The integration of domain knowledge stemming from an ontology is not yet considered in existing approaches. Taxonomies and relations might be of great use in improving constraint mining results and making mined constraints more understandable for business users.

Lakshmanan et al. propose a Markov prediction model for data-driven semi-structured business processes [89]. By exploiting process mining techniques, the approach discovers a classical process model to learn decisions for determined decision points. Whether the recommendation approach of the proposed framework can benefit from this additional preprocessing step is uncertain since classical process mining tends to create spaghetti models when analyzing enactments of unstructured business processes, such as those often present in ACM.

4.6. Conclusion & Future Work

This chapter proposes a framework that supports knowledge workers to avoid non-compliance. An ontology-based constraint editor allows knowledge workers to rapidly react to new circumstances that require an adaption of the realization of compliance requirements by enabling the creation of constraints in business terminology. Consequently, long maintenance cycles—usually involved with realizing compliance requirements in an IT system—can be avoided. Decisions of knowledge workers, such as those related to compensating a compliance violation, are being automatically learned to provide support to knowledge workers that encounter a similar situation.

There exist several opportunities for future research. Knowledge workers might benefit from

extending the framework with automated constraint discovery. How to integrate the process of discovery with the ontology and how to make the discovered results usable by knowledge workers would be an interesting direction. While the prototypical implementation of the framework as extension of the ISIS Papyrus ACM software implies general feasibility, user studies on specific aspects of the framework, such as the usability of the constraint and ontology editor, could be used to further investigate the practical applicability of the framework.

5. Plausibility Checking of Behavioral Constraints Formalized in Linear Temporal Logic

Business users are shielded from the complexity of temporal logic formulas by making use of a pattern-based approach, as discussed in the previous two chapters. Patterns describe a specific intent in structured natural language (e.g., A *leads to* B) instead of exposing the underlying technical representation. Nonetheless, it must be guaranteed that the underlying formula of a pattern correctly implements the desired intent. As this chapter will show, alignment of intent and formula is a problem even in established pattern repositories, and furthermore the creation of new patterns is far from being trivial. This chapter proposes a plausibility-checking approach for LTL-based specifications. The proposed approach can provide confidence in an LTL formula if plausibility checking is successfully passed. If the formula does not pass the plausibility checks, a counterexample trace and the truth values of both the LTL formula and the plausibility specification are generated and can be used as a starting point for correction.

5.1. Introduction

Keeping business processes in line with requirements stemming from various sources (e.g., laws, regulations, standards, internal policies, best practices) has become an important research field due to increasing flexibility demands in business process management, especially in knowledge-intensive environments. In recent years, both academia and industry have been working towards solutions for enabling the flexible handling of business processes while providing support for meeting necessary requirements. Two closely related categories of such supporting approaches have been extensively investigated. The first are compliance-enabling approaches developed for checking behavioral constraints (also called compliance rules) at runtime (e.g., Ly et al. [48]) and design time (e.g., Awad et al. [47]) of business processes. The other are constraint-based business processes models (also called declarative workflows), in which a set of constraints is used to describe the business process; these constraints become the basis for the enactment of the process. A prominent example for the declarative workflow approach is Declare (cf. Pešić

& van der Aalst [56]) which provides a graphical front end with mappings to Linear Temporal Logic (LTL; cf. Pnueli [26]) as underlying formalism. Temporal logics, such as LTL and CTL (Computation Tree Logic), are established ways of describing desired system properties for verification. Especially LTL has become a *de facto* standard for defining system specifications due to its extensive use in model checking (cf. Rozier [90]) and the possibility to automatically translate LTL formulas to nondeterministic finite automata (NFA) for runtime verification on finite traces [63].

The creation of LTL formulas is, however, a challenging and error-prone task that requires considerable knowledge of and experience with LTL. It is hardly surprising that higher levels of abstraction, such as the property specification patterns proposed by Dwyer et al. [27], are often preferred over authoring new LTL formulas. There are two major issues when trying to rely exclusively on a pattern-based approach. Firstly, formal patterns that precisely match the intention of the user might not be available. As a result, manually defining the constraint (for instance, by modifying or combining existing patterns) is still required. Secondly, if an existing candidate pattern has been identified, how can the user be sure that her intention is really met by the pattern? Such problems in the specifications could result in severe consequences, for example, legal issues due to the violation of compliance requirements. Thus, it is highly important to provide better support for creating correct specifications.

Our plausibility checking approach aims at supporting the creation process of LTL formulas and at increasing the confidence in existing LTL formulas by ensuring that the user's intention matches a corresponding LTL specification. Whenever an LTL formula is (re)written, the user also creates a plausibility specification which is used to find out whether an LTL formula is contradictory to this specification. The approach performs reasoning on finite traces through Complex Event Processing (CEP)—with plausibility specifications encoded in Event Processing Language (EPL)—and Nondeterministic Finite Automata (NFA)—representing LTL formulas. We conducted experiments (cf. Chapter 6 and Chapter 7) which suggest that EPL-based plausibility specifications are easier to understand than LTL formulas which is assumed to be important for the practical applicability and usability of the approach. Furthermore, we discuss the practical use of our approach by the following scenarios:

- The detection and correction of an incorrect LTL formula found in the constraint pattern collection by Dwyer et al. [27].
- The implementation of a new constraint pattern stemming from an EPA (Environmental Protection Agency) compliance document [13].

5.2. Related Work

To the best of our knowledge, only very few studies exist on keeping LTL formulas in line with the users' understanding of the formula. Salamah et al. [91] propose to use a set of manually created test cases to check the plausibility of pattern-generated LTL formulas. However, this involves the user in the generation process of all the sample traces and the expected truth values at the end of this traces. As a result, the count of test cases remains marginal because the specification of tests is time-consuming. Yan et al. [67] claim to keep natural-language requirements and their corresponding formulas consistent by translating specifications in structured English grammar to LTL, mainly by mapping English words to LTL operators. While the approach provides relief for specifying discrete time properties, the direct mapping of LTL operators to words does not really simplify the creation process of LTL specifications. Thus, there is still the risk of creating formulas that contradict the actual intention of the author.

Other plausibility checking approaches do not focus on the consistency between the users' intention and its actual formal representation in LTL, but they check the internal consistency of LTL formulas (e.g., Barnat et al. [92]). Vacuity detection is concerned with avoiding tautologies and subformulas that are not relevant for the satisfaction of the formula (e.g., Simmonds et al. [93]). Consistency checking of LTL formulas means finding contradicting parts of a formula or contradictions in sets of formulas (e.g., compliance rule collections) that are generally unsatisfiable (e.g., Awad et al. [94]). While these plausibility checking approaches are also very important, their focus is entirely different from our work.

The execution of automata is an integral part of our approach. To the best of our knowledge, the Declare approach by Pešić & van der Aalst [56] is the first approach that proposed the execution of automata for runtime verification of business processes. Declare is a declarative workflow approach which provides graphical notation for the specification of constraints with underlying LTL formalism. Our approach is, in contrast to Declare, not based on the execution of Büchi automata but relies on the `LTL2NFA` translation approach by De Giacomo et al. [63] which creates NFA that are specialized for reasoning on finite traces.

Our work presented in this chapter is also related to many existing studies (e.g., Holmes et al. [95]) that have successfully applied CEP in various research contexts. Closest to our approach is the work by Awad et al. [77] that is concerned with the runtime detection of compliance violations based on anti-patterns. Anti-patterns cause a firing of the CEP engine for every violation, but recovery from violations is not covered. In contrast, our CEP approach is based on LTL runtime states, namely temporarily satisfied, permanently satisfied, temporarily violated or permanently violated (cf. Pešić et al. [96], De Giacomo et al. [97], Maggi et al. [98]), to ensure identical behavior when comparing against an NFA. Only to consider violations would not be

sufficient for plausibility checking.

5.3. Preliminaries

This section provides background information on the LTL formalism and EPL as a language for plausibility specifications.

5.3.1. Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) was originally proposed as a specification formalism for non-terminating systems which are characterized by infinite traces [26]. In recent years, LTL has been more and more used for reasoning on finite traces [56], [99]. Regardless whether reasoning on finite or infinite traces, the syntax of LTL remains the same. In the following we describe LTL and its semantics interpreted over finite traces [99] because the proposed plausibility checking approach will operate on finite execution traces of the underlying system.

Let AP be a finite set of propositional variables and the syntax of LTL be defined as $\varphi ::= v \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (\mathcal{X}\ \varphi) \mid (\varphi_1\ \mathcal{U}\ \varphi_2)$ where $v \in AP$, $\mathcal{X}$ is the *next* operator and $\mathcal{U}$ is the *until* operator. Every boolean formula is an LTL formula. The length of a trace is a function $length(t)$ which counts the number of elements of the traces t. The function $t(i)$ accesses the element at position i of trace t where $\{i \in \mathbb{N} \mid 0 \leq i < length(t)\}$. The satisfaction relationship $t, i \models \varphi$ between a trace t at an instant i and a formula φ is inductively defined as

- $t, i \models v$ iff $v \in t(i)$,
- $t, i \models \neg\varphi$ iff $t, i \not\models \varphi$,
- $t, i \models \varphi_1 \wedge \varphi_2$ iff $t, i \models \varphi_1$ and $t, i \models \varphi_2$,
- $t, i \models \mathcal{X}\ \varphi$ iff $i < length(t) - 1$ and $t, i + 1 \models \varphi$,
- $t, i \models \varphi_1\ \mathcal{U}\ \varphi_2$ iff $\exists j \in \mathbb{N} \mid i \leq j \leq length(t) - 1$ it holds that $t, j \models \varphi_2$ and $\forall k \in \mathbb{N} \mid i \leq k \leq j$ it holds that $t, k \models \varphi_1$.

Intuitively, $\mathcal{X}\ \varphi$ means that φ must be met in the next state, and $\varphi_1\ \mathcal{U}\ \varphi_2$ means that φ_1 must hold in every state until φ_2 eventually holds. There are additional operators that can be derived from the aforementioned operators.

- $\mathcal{F}\ \varphi \equiv true\ \mathcal{U}\ \varphi$,
- $\mathcal{G}\ \varphi \equiv \neg F\ \neg\varphi$,

- $\varphi_1 \mathcal{W} \varphi_2 \equiv (\mathcal{G} \varphi_1) \vee (\varphi_1 \mathcal{U} \varphi_2)$.

Intuitively, $\mathcal{F} \varphi$ implies that φ must eventually hold, $\mathcal{G} \varphi$ demands that φ always holds, and $\varphi_1 \mathcal{W} \varphi_2$ is a weak variant of until which is also satisfied when φ_1 always holds.

The semantics of LTL over infinite traces is defined as follows: LTL formulas are interpreted as infinite words over the alphabet 2^{AP} (i.e., the alphabet are all possible propositional interpretations of the propositional symbols in AP). $\pi(i)$ denotes that state of the trace π at time instant i. We define $\pi, i \vDash \psi$ (i.e., a trace π at time instant i satisfies the LTL formula ψ) as follows:

- $\pi, i \vDash a$, for $a \in AP$ iff $a \in \pi(i)$.
- $\pi, i \vDash \neg\psi$ iff $\pi, i \nvDash \psi$.
- $\pi, i \vDash \psi \wedge \phi$ iff $\pi, i \vDash \psi$ and $\pi, i \vDash \phi$.
- $\pi, i \vDash \psi \vee \phi$ iff $\pi, i \vDash \psi$ or $\pi, i \vDash \phi$.
- $\pi, i \vDash \mathcal{X}\psi$ iff $\pi, i+1 \vDash \psi$.
- $\pi, i \vDash \mathcal{F}\psi$ iff $\exists j \geq i$, such that $\pi, j \vDash \psi$.
- $\pi, i \vDash \mathcal{G}\psi$ iff $\forall j \geq i$, such that $\pi, j \vDash \psi$.
- $\pi, i \vDash \psi \,\mathcal{U}\, \phi$ iff $\exists j \geq i$, such that $\pi, j \vDash \phi$, and $\forall k, i \leq k < j$, we have $\pi, k \vDash \psi$.
- $\pi, i \vDash \psi \,\mathcal{R}\, \phi$ iff $\forall j \geq i$, iff $\pi, j \nvDash \phi$, then $\exists k, i \leq k < j$, such that $\pi, k \vDash \psi$.

In model checking, LTL formulas commonly have two possible truth value states, namely `true` (`satisfied`) and `false` (`violated`). In case of monitoring an LTL specification in a running system, it might be the case, that it is not only of interest if a specification is satisfied or violated but also whether further state changes are possible that could resolve or cause a violation of a specification. That is, the state of a specification is either temporary (i.e., the state may change) or permanent (i.e., the state may not longer change). Consequently, to enable a more fine-grained analysis of the plausibility of LTL formulas, we employ the semantics of Runtime Verification Linear Temporal Logic (RV-LTL; cf. Bauer et al. [100]) that supports four truth value states. In particular, an LTL behavioral constraint specification at runtime is either `temporarily satisfied`, `temporarily violated`, `permanently satisfied`, or `permanently violated`.

The semantics of RV-LTL is defined as follows:

- $[u \vDash \psi]_{RV} = \top$ (ψ permanently satisfied by u) if for each possible finite continuation v of u : $uv \vDash \psi$.

- $[u \vDash \psi]_{RV} = \bot$ (ψ permanently violated by u) if for each possible finite continuation v of $u : uv \nvDash \psi$.
- $[u \vDash \psi]_{RV} = \top^p$ (ψ possibly/temporarily satisfied by u) if $u \vDash \psi$ and there exists a possible finite continuation v of $u : uv \nvDash \psi$.
- $[u \vDash \psi]_{RV} = \bot^p$ (ψ possibly/temporarily violated by u) if $u \nvDash \psi$ and there exists a possible finite continuation v of $u : uv \vDash \psi$.

Several existing studies make use of the concept of four LTL truth value states (cf. Pešić et al. [96], De Giacomo et al. [97], Maggi et al. [98], Falcone et al.[101], Joshi et al. [102], and Morse et al. [103]).

5.3.2. Event Processing Language (EPL)

In this section, we discuss plausibility specifications for LTL-based behavioral constraints based on the Event Processing Language (EPL; cf. EsperTech Inc. [104]). An EPL-based plausibility specification consists of an initial truth value (`temporarily satisfied` or `temporarily violated`) and one or more query-listener pairs that we will name *Temporal Queries*. A *Temporal Query* causes a truth value change of the constraint as soon as a matching event pattern is observed in the event stream. Consequently, an EPL-based specification always consists of EPL queries that are composed of EPL operators and listeners that causes truth value changes (to `temporarily satisfied`, `temporarily violated`, `permanently satisfied`, `permanently violated`) to which the state of the specification is set to by a positive match of an expression in the event stream. Obviously, further truth value changes are not possible once a permanent state (i.e., `permanently violated` or `permanently satisfied`) has been reached. The semantics of those EPL operators is given as follows (cf. [104]):

- The *and* operator e_1 `and` e_2 is a logical conjunction that is matched once both e_1 and e_2 (in any order) have occurred.
- The *or* operator e_1 `or` e_2 is a logical disjunction that is matched once either e_1 or e_2 has occurred.
- The *not* operator `not` e is a logical negation that is matched if the expression e is not matched.
- The *every* operator `every` e not just observes the first occurrence of the expression e in the event stream but also each subsequent one.

- The *leads-to* operator e_1 `->` e_2 specifies that first e_1 must be observed and only then is e_2 matched. Intuitively, the whole expression is matched once e_1 is followed by e_2 at the occurrence of e_2.
- The *until* operator e_1 `until` e_2 matches the expression e_1 until e_2 occurs. In practice, this operator is commonly used in the expression `not` e_1 `until` e_2 that demands the absence of e_1 before the occurrence of e_2.

5.4. Plausibility Checking Approach

Authoring LTL formulas usually starts with thinking about some kind of constraint or requirement in natural language. In this case, the user intends to create an LTL formula that matches the description in natural language. Alternatively, a natural-language description and the corresponding LTL formula could already be extant, and the user be interested in finding out whether the LTL formula is a plausible representation of the natural-language description. For the creation of plausibility specifications, we propose using Temporal Queries (TQs), which are a way for specifying truth value changes while observing finite traces, like the execution trace of a business process instance. In addition to the creation of the TQs, it is necessary to define the initial truth value of the specification (i.e., temporarily satisfied or temporarily violated) otherwise the truth value of the plausibility specification would be undefined until a TQ causes a truth value change.

An overview of *Plausibility Checking* is shown in Figure 5.1. Plausibility checking requires two inputs, namely an LTL formula and its corresponding plausibility specifications. The LTL formula is then transformed into a nondeterministic finite automaton (NFA). The plausibility specifications consist of an initial truth value and TQs. The TQs are transformed to *event query statements* and *listeners* that can be used by a Complex Event Processing (CEP) engine.

In particular, there can be up to four listeners, namely one for each runtime verification state. Once a *permanent* state is reached, the truth value becomes immutable. Both the NFA and CEP receive inputs which are the elements of finite traces. These inputs will lead to changes of both the *Truth Value* and the *Reference Truth Value*. A change of the *Reference Truth Value* of the plausibility specification occurs once a listener is triggered because the temporal query matches the current trace. The *Truth Value* reflects the current acceptance state of the automaton.

In order to achieve a positive plausibility checking result, there must not be any deviation between the *Truth Value* and the *Reference Truth Value* for all inputs. A large set of test cases can be created and checked automatically. There are two options: *Option 1:* All words over the alphabet of the NFA having sufficient length can be used as inputs. A moderate maximum trace length between 7 and 10 is, in most cases we encountered so far, sufficient. *Option 2:* Only a

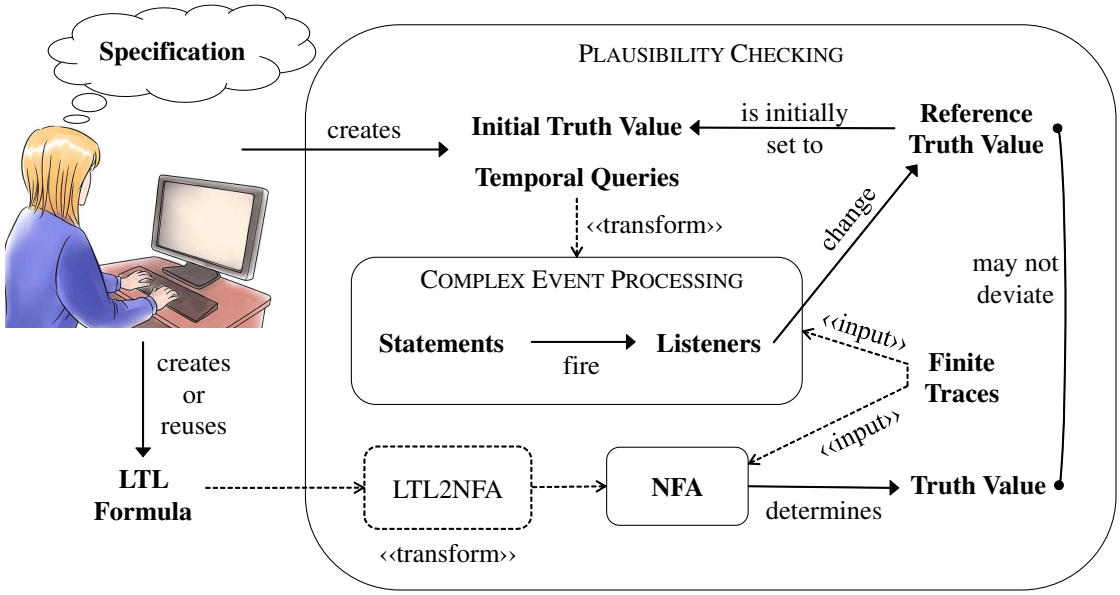


Figure 5.1.: Approach Overview [105]

subset of traces with greater maximum size is created randomly, and checked automatically as well. The alphabet of the automaton always consists of the variables of the LTL formula and a single additional variable that functions as a surrogate for all other variables that are not part of the LTL formula. If the formula does not meet the plausibility specifications, a counterexample trace and the truth values of both the LTL formula and the plausibility specification are being made available as a starting point for the correction of the LTL formula. The approach has been fully implemented in a prototype which makes use of the open source CEP engine *Esper* (cf. EsperTech Inc. [106]) and the LTL2NFA algorithm (cf. De Giacomo [63]).

In this way, the plausibility checking can be leveraged for aiding users during the creation of a new constraint patterns as well as for analyzing existing patterns to gain confidence in the proposed LTL representation of the pattern.

5.5. Reviewing Existing LTL Patterns

In 1999, Dwyer et al. published a paper entitled "Patterns in property specifications for finite-state verification" [27] alongside a constraint pattern collection called "Property Specification Patterns", which is available online[1]. In the FAQs the following is stated: *"Mappings were validated primarily by peer review amongst the project members, with assistance from several other*

[1] `http://patterns.projects.cis.ksu.edu`

people on selected pattern mappings. Some of the mappings also underwent testing by running existing FSV[2] tools to analyze small finite-state transition systems which encode (un)satisfying sequences of states/events.". Consequently, we cannot assume the correctness of a pattern representation. As an example for an LTL formula that does not match our understanding of the corresponding pattern, we are now going to discuss the *Precedence After* pattern (*after a: b precedes c*) and its LTL representation $\mathcal{G}(\neg a) \vee \mathcal{F}(a \wedge (\neg c \; \mathcal{W} \; b))$. We formulate a single TQ with a query a `leads-to not` b `until` c and a listener to switch to `permanently violated` once there has been an a but thereafter no b until c occurs. Plausibility checking notifies us of the counterexample $[a, c, a]$ where the LTL formula is satisfied. According to the pattern scopes defined by Dwyer et al. [27], the after scope *after a* starts at the first occurrence of a. Thus, with the occurrence of the trace $[a, c]$ the pattern becomes permanently violated because b should have happened in between the first occurrence of a and the occurrence of c. Consequently, every suffix of $[a, c]$ must not cause any further change of the truth value of the pattern, so there must be something wrong with the LTL formula. The reason why the LTL formula is incorrect becomes obvious when we substitute the *weak until* by one of its equivalences. Then the modified formula is given as $\mathcal{G}(\neg a) \vee \mathcal{F}(a \wedge ((\neg c \, \mathcal{U} \; b) \vee \mathcal{G}(\neg c)))$. The trace $[a, c, a]$ meets the LTL formula by satisfying the subformula $\mathcal{F}(a \wedge \mathcal{G}(\neg c))$ through the second occurrence of a because the trace ends there and c is not present after this a. From our point of view, a correct LTL formula for this pattern is $(\mathcal{G}\neg a) \vee (\neg a \, \mathcal{U} \; (a \wedge (\neg c \; \mathcal{W} \; b)))$ because here it is ensured that only the first a starts the scope.

5.6. Creation of LTL Formulas

Pattern collections, most notably the "Property Specification Patterns" by Dwyer et al. [27], contain a large number of patterns suitable for many cases. However, such collections are far from being complete. If no suitable pattern is available to realize a certain constraint, a new formula must be created. We are now going to illustrate this scenario by a practical example extracted from our prior research work related to capturing and formalizing real-world compliance requirements (cf. Tran et al. [75]).

Let us consider a compliance guidance document [13] published by the United States Environmental Protection Agency (EPA) regarding buildings built before or after 1978—when lead-based paints were prohibited for use in residential and public buildings in USA—that states "*In housing built before 1978, you must: Distribute EPA's lead pamphlet [...] to the owner and occupants before renovation starts.*". This rule involves three tasks, namely, checking whether the house is built before 1978, distributing EPA's lead pamphlets, and starting the renovation

[2] Finite State Verification

process. It makes sense to distribute the lead pamphlet in case the house was built before 1978. Hence, a confirmation that the building was built before 1978 must *coexist* with the distribution of lead pamphlets in the same process instance *before* the renovation is started. Now we are facing the problem that the pattern catalog by Dwyer et al. [27] does not introduce *Coexistence* patterns. The declarative workflow approach Declare (cf. Pešić & van der Aalst [56]), another important source of patterns, does offer a Coexistence pattern but only in the *global*-scoped—instead of the needed *before*-scoped—variant. To the best of our knowledge, the *Coexistence Before* pattern does not exist, thus it must be created. During this process, we will leverage the proposed plausibility checking approach.

A brief description of the pattern in natural language, such as *before c: a coexists with b*, outlines what we aim for. At first, we create appropriate TQs for the pattern. On the one hand, we need a query `(`a `leads-to` b `leads-to c) or (`b `leads-to` a `leads-to` c`)` and a listener that turns the truth value to `permanently satisfied` and another query `(not` c `and not` b `until` a `leads-to not` b `until` c`) or (not` c `and not` a `until` b `leads-to not` a `until` c`)` that turns the truth value to `permanently violated`. Creating these formulas is relatively straightforward. The pattern becomes permanently satisfied when a and b, in any order, occur before c. A permanent violation of the pattern occurs if one of the following conditions is satisfied.

- there is no c and no b until a occurs and thereafter is again no b until c occurs.
- there is no c and no a until b occurs and thereafter is again no a until c occurs.

On the other hand, we need to create the LTL formula. As soon as we believe that the LTL formula corresponds to the meaning of the pattern, we can run the plausibility check. For example, when we perform plausibility checking on $\mathcal{F}\ c \rightarrow ((\neg c\,\mathcal{U}\ a) \wedge (\neg c\,\mathcal{U}\ b))$, our approach reports a plausibility issue in relation with the trace $[c]$. The plausibility specification is still in its initial state, namely *temporarily satisfied* while the LTL formula indicates already a violation. We intended that the formula becomes only violated if either *a* exists before *c* but *b* does not, or *b* exists before *c* but *a* does not. Since this is not the case for the trace $[c]$, the plausibility checking approach correctly identifies an issue. Hence, we must revise the LTL formula. Eventually, we come up with the formula $(\mathcal{F}\ c \rightarrow ((\neg c\,\mathcal{U}\ a) \rightarrow (\neg c\,\mathcal{U}\ b))) \wedge (\mathcal{F}\ c \rightarrow ((\neg c\,\mathcal{U}\ b) \rightarrow (\neg c\,\mathcal{U}\ a)))$ that passes the plausibility checks. Therefore, we have found a plausible representation of the

pattern in LTL. Now, we can encode the compliance requirement in LTL as

$$
\begin{aligned}
(\mathcal{F}\ &\text{“Renovation started”} \rightarrow ((\neg\text{“Renovation started”}\ \mathcal{U}\ \text{“Housing build before 1978} \\
&\quad \text{confirmed”}) \rightarrow (\neg\text{“Renovation started”}\ \mathcal{U}\ \text{“Distribute pamphlet finished”}))) \\
&\wedge\ (\mathcal{F}\ \text{“Renovation started”} \rightarrow ((\neg\text{“Renovation started”}\ \mathcal{U}\ \text{“Distribute pamphlet} \\
&\quad \text{finished”}) \rightarrow (\neg\text{“Renovation started”}\ \mathcal{U}\ \text{“Housing build before 1978 confirmed”}))).
\end{aligned}
$$

To complete the formalization of the compliance rule, it is additionally required that an execution of the *Housing build before 1978* task happens before the start of the renovation process. This can be realized by using the already existing *Precedence Global* pattern [44].

5.7. Discussion

Although the origin and reason for proposing plausibility checking is clearly related to business process management, the approach is applicable to other domains, such as the verification of software in general. The current plausibility checking approach assumes a single state per instant of time, which has been sufficient for plausibility-checking problems that we have encountered until now. Multiple states per instant are currently not considered due to the resulting exponential blowup of traces that need to be considered during plausibility checking. Nevertheless, it is feasible to extend the current plausibility-checking approach to also support multiple states per instant if the application domain demands it, but the computational efforts would be enormous if we want to check all words over the alphabet of the automaton exhaustively. Even if working with a single state per instant, the main drawback of the approach is the exponential growth of traces that should be considered for the plausibility checks. It is, however, in the scenarios we encountered so far sufficient to work with moderate maximum trace lengths because the count of propositional variables that are present in most semantic constraints is often low (cf. Elgammal et al. [23]) and most issues are already discoverable in short traces (cf. Section 5.7). Alternatively, if LTL formulas involve a high count of propositional variables, our approach still can perform a large quantity of plausibility checks with randomly generated longer traces automatically, which obviously is a better approach than generating only a few test cases manually.

Our empirical studies (cf. Chapter 6 and Chapter 7) suggest that EPL specifications are more understandable than LTL specifications, which we consider crucial for the practical applicability of the approach.

5.8. Conclusion and Future Work

This chapter proposes an approach for plausibility checking of LTL specifications based on Nondeterministic Finite Automata (NFA) and Complex Event Processing (CEP). The approach has been discussed in the context of a practical scenario, namely the creation of a new constraint pattern stemming from a compliance document. Existing pattern-based approaches, such as Declare, can benefit from our approach. Whenever it becomes necessary to extend the set of supported constraints, our approach can provide assistance. Not only runtime verification or declarative workflow techniques, but also design-time approaches can benefit, since specifications for model checking are often encoded in LTL. In our future work, we plan to further evaluate the approach through performance, scalability and user experiments and to apply our approach to other formalisms that have, for example, a notion of quantitative time (cf. Autili et al. [107]).

Part III.

Empirical Studies on the Understandability of Behavioral Constraint Representations

6. On the Understandability of Behavioral Constraints Formalized in Linear Temporal Logic, Event Processing Language and Property Specification Patterns

Business-driven behavioral constraint authoring and plausibility checking involve different behavioral constraint representations, namely Property Specification Patterns (PSP) that abstract underlying formal Linear Temporal Logic (LTL) representations, and Event Processing Language (EPL) for plausibility specifications. In this chapter, we study the understandability of these three major behavioral constraint representations. We conducted two controlled experiments with 216 participants in total, using a completely randomized design with one alternative per experimental unit. We hypothesized that PSP, as a highly abstracting pattern language, is easier to understand than LTL and EPL, and that EPL, due to separation of concerns (as one or more queries can be used to explicitly define the truth value change that an observed event pattern causes), is easier to understand than LTL. We found evidence supporting our hypotheses which was statistically significant and reproducible.

6.1. Introduction

Behavioral constraints focus on the execution of a system, which usually involves changing states at different points in time during system execution. They play a major role in many domains, such as satellite systems (cf. Esteve et al. [108]), health care (cf. Rovani et al. [14]), banking (cf. Bianculli et al. [109]), automotive (cf. Post et al. [110]), to name a few. They are used in the context of verification and validation activities, both at design time (cf. Kherbouche et al. [111], Czepa et al. [112], Morimoto [113], and Bucchiarone et al. [114]) and at run time (cf. Mulo et al. [115], Knuplesch et al. [116], Ly et al. [48], and de Silva & Balasubramaniam [117]).

In this study, we consider a representative set of established approaches for the specification

of behavioral constraints, namely:

- Linear Temporal Logic (LTL; cf. Pnueli [26]),
- Property Specification Patterns (PSP; cf. Dwyer et al. [27]), and
- Event Processing Language (EPL; cf. EsperTech Inc. [104]).

Linear Temporal Logic (LTL) is a widely used and established language for the specification of behavioral constraints. It is a logic-based approach that supports not only logical but also temporal operators. Many existing model checkers leverage LTL as a specification language (cf. Cimatti et al. [35] for NuSMV[1], Blom et al. [118] for LTSmin[2], Holzmann [119] for SPIN[3]). Originally developed for reasoning on infinite traces, LTL can also be applied for reasoning on finite traces (cf. De Giacomo & Vardi [99]). The `LTL2NFA` algorithm (cf. De Giacomo et al. [120]) describes the transformation of an arbitrary LTL formula to a non-deterministic finite automaton (NFA), which can be executed for runtime checking of LTL-based behavioral constraints.

The Property Specification Patterns (PSP) are a collection of recurring temporal patterns. The relevance of the patterns discovered by Dwyer et al. [27] was confirmed even 13 years after the original study took place, in a survey by Bianculli et al. [109] based on 104 scientific case studies. Each pattern represents a specific intent with a mapping to underlying formal representations, most notably LTL and CTL (Computation Tree Logic; cf. Clarke et al. [41]). Many existing approaches reuse PSP or extend the original pattern catalog with more specific context-dependent patterns. Among them are the DecSerFlow language for declarative service descriptions (cf. van der Aalst & Pešić [42]), the declarative workflow approach Declare (cf. Pešić et al. [121]), the Compliance Request Language (abbrev. CRL; cf. Elgammal et al. [23]), and the PROPOLS approach for the verification of BPEL service composition schemes (cf. Yu et al. [66]).

The Event Processing Language (EPL) can be used to encode specific event patterns in queries that cause the firing of event listeners once the pattern is observed in the event stream of a Complex Event Processing (CEP) environment (cf. Wu et al. [122]). EPL is part of the open source CEP engine *Esper*[4]. Numerous studies make use of EPL (cf. Awad et al. [77], Holmes et al. [95], Boubeta-Puig et al. [123], Kunz et al. [124], Adam et al. [125], Aniello et al. [126], to name but a few). EPL is well-suited as a representative for CEP query languages as it supports common CEP query language concepts, such as *leads-to* (*sequence*, *followed-by*) and *every*

[1] `http://nusmv.fbk.eu/`
[2] `http://fmt.cs.utwente.nl/tools/ltsmin/`
[3] `http://spinroot.com/`
[4] `http://www.espertech.com/esper`

(*each*) operators, that are present in many CEP query languages and engines (e.g., Siddhi[5] and TESLA [127]).

6.1.1. Problem Statement

Despite the long presence of many major behavioral constraint specification approaches (e.g., Linear Temporal Logic was first proposed in 1977, the Property Specification Patterns exist since 1999), the core focus of most researchers has been on the formal/technical perspective of these approaches, whereas studying the usage point of view from an empirical perspective has not drawn much attention. Indeed, we are not aware of any existing work that provides an empirical study on the understandability of different representative behavioral constraint specification approaches. Gaining more insights into the understandability of behavioral constraint representations is crucial for evaluating their suitability for practical use and finding potential ways for their improvement with regards to understandability.

LTL, PSP, and EPL are all powerful approaches for automated behavioral constraint verification and validation, but very little is known about their understandability. Intuitively, we might hypothesize that the temporal pattern-based approach PSP is more understandable than the temporal logic-based approach LTL because the former is abstracting the latter, but scientific evidence is required to back up such claims. In this chapter, we investigate this and similar hypotheses by applying suitable statistical methods on the gathered empirical data.

Studying the currently existing empirical research gaps in this field is not only interesting from a purely scientific point of view, but it is also important for industrial applications. For example, from the cooperation with our industry partners (see e.g., [79]), their customers, and other company representatives at conferences and workshops, we realized that industry has a huge demand for, and shows a strong interest in, behavioral constraint specification approaches that are applicable in practice by supporting a comprehensible, fast and accurate adoption of compliance requirements as well as their automated enactment and verification. All representative behavioral constraint specification approaches that we study in this chapter are well-suited for automated computer-aided checking, but BPM vendors are still often reluctant to expose their customers to such approaches, and our discussions with industry partners (see e.g. [75], [128]), that indicate uncertainty regarding how understandable behavioral constraint representations are, is among the reason for this.

The application of behavioral constraint specifications for supporting software architecture compliance in the software engineering & SWA domain faces a similar issue: architecture descriptions and design decisions (cf. Medvidovic et al. [129] and Zdun et al. [130]) must be

[5] `https://github.com/wso2/siddhi`

documented in a comprehensible manner for different stakeholders in the software development process. Nowadays this is still often done in natural language, which cannot be directly used (i.e., without semi-automatic natural language processing; cf. Czepa et al. [131]) for automated software architecture compliance checking. By using a behavioral constraint language for capturing architectural descriptions and decisions, we can directly leverage those architectural descriptions for automated architecture compliance checking.

Empirical research on behavioral constraint understandability has the potential to influence practitioners in making the decision for adopting a specific existing behavioral constraint language and in designing future industrial behavioral constraint specification approaches. Consequently, one of the goals of this empirical study is to pave the way for industrial or practical exploitation of behavioral constraint specification approaches.

6.1.2. Research Objectives

This empirical study has the objective to investigate the understandability of representative behavioral constraint representations. The understandability construct focuses on how well (in terms of correct understanding) and fast (in terms of the response time) a participant understands a given behavioral constraint representation.

We state the experimental goal using the GQM (Goal Question Metric) goal template (cf. Basili et al. [132]) as follows:

Analyze the LTL, PSP, and EPL behavioral constraint approaches
for the purpose of their evaluation
with respect to their understandability
from the viewpoint of the novice and moderately advanced software architect, designer or developer
in the context (environment) of the Distributed System Engineering Lab and the Advanced Software Engineering Lab courses at the Faculty of Computer Science of the University of Vienna.

6.1.3. Context

The study consists of two controlled experiments with 216 participants in total:

- The first run was carried out with 70 computer science students who enrolled in the course "Advanced Software Engineering Lab (ASE)" (mandatory part of the master in computer science curricula) at the University of Vienna in the winter term 2015/2016.

- The second run was carried out with 92 computer science students who enrolled in the course "Distributed System Engineering Lab (DSE)" (optional part of the bachelor and master in computer science curricula) at the University of Vienna and 54 computer science students who enrolled in the course "Advanced Software Engineering Lab (ASE)" (mandatory part of the master in computer science curricula) at the University of Vienna in the winter term 2016/2017.

Consequently, we can differentiate between DSE and ASE participants. While the former are used as proxies for novice to moderately advanced software architects, designers or developers, the latter are used as proxies for moderately advanced software architects, designers or developers. According to Kitchenham et al. [133], using students *"is not a major issue as long as you are interested in evaluating the use of a technique by novice or nonexpert software engineers. Students are the next generation of software professionals and, so, are relatively close to the population of interest"*. Besides, a number of our students work while studying; some even have a few years of industry experience (cf. Section 6.4.2). Several existing studies take it even a step further by suggesting that students can be representatives for professionals under certain circumstances (cf. Höst et al. [134], Runeson [135], Svahnberg et al. [136], and Salman et al. [137]).

6.1.4. Guidelines

This work follows and respects existing guidelines for conducting and reporting empirical research in software engineering: Jedlitschka et al. [138] propose guidelines and a structured approach for reporting experiments in software engineering, which had a strong influence on the general structure and contents of this chapter. Those guidelines integrate (among others) the "Preliminary guidelines for empirical research in software engineering" by Kitchenham et al. [133] and standard books on empirical software engineering (cf. Wohlin et al. [34] and Juristo & Moreno [139]). Moreover, we considered and applied the "Robust Statistical Methods for Empirical Software Engineering" by Kitchenham et al. [140] for the statistical evaluation of the acquired data.

6.2. Background on Behavioral Constraint Representations

In this section, we discuss the general properties of the behavioral constraint representations that are the focus of this study. Readers already familiar with one (or more) of the discussed behavioral constraint representations may consider skipping (parts of) this section.

6.2.1. Linear Temporal Logic (LTL)

Propositional logic is not expressive enough to describe the behavior of systems (i.e., the ordering of events in time), so the notion of temporal logic has been introduced in 1977 (cf. Pnueli [26]). In particular, a logic called Linear Temporal Logic (LTL) for reasoning over linear traces with the temporal operators $\mathcal{G}$ (or $\Box$) for "globally" and $\mathcal{F}$ (or $\Diamond$) for "finally" is proposed. Additional temporal operators are $\mathcal{U}$ for "until", $\mathcal{W}$ for "weak until", $\mathcal{R}$ for "release", and $\mathcal{X}$ (or $\circ$) for "next". $\mathcal{G}\psi$ (or $\Box\psi$) states that ψ must be true in every point in time. $\mathcal{F}\psi$ (or $\Diamond\psi$) states that ψ must be true at some future point in time. $\psi\,\mathcal{U}\,\phi$ states that ψ remains true at least until the point in time when ϕ becomes true. $\psi\,\mathcal{R}\,\phi$ states that ψ remains true at least until and including the point in time when ϕ becomes true. $\mathcal{X}\psi$ (or $\circ\psi$) states that ψ must be true at the next point in time. LTL formulas are composed of the aforementioned temporal operators, atomic propositions (the set AP), and the boolean operators $\wedge$ (for "and"), $\vee$ for "or", $\neg$ for "not", $\rightarrow$ for "implies" (cf. Baier & Katoen [141]). The weak-until operator $\psi\,\mathcal{W}\,\phi$ is defined as $(\mathcal{G}\,\psi) \vee (\psi\,\mathcal{U}\,\phi)$.

An LTL formula is inductively defined as follows: For every $a \in AP$, a is an LTL formula. If ψ and ϕ are LTL formulas, then so are $\mathcal{G}\psi$ (or $\Box\psi$), $\mathcal{F}\psi$ (or $\Diamond\psi$), $\psi\,\mathcal{U}\,\phi$, $\psi\,\mathcal{R}\,\phi$, $\mathcal{X}\psi$ (or $\circ\psi$), $\psi \wedge \phi$, $\psi \vee \phi$, and $\neg\psi$.

The semantics of LTL over infinite traces is defined as follows: LTL formulas are interpreted as infinite words over the alphabet 2^{AP} (i.e., the alphabet consists of all possible propositional interpretations of the propositional symbols in AP). $\pi(i)$ denotes that state of the trace π at time instant i. We define $\pi, i \vDash \psi$ (i.e., a trace π at time instant i satisfies the LTL formula ψ) as follows:

- $\pi, i \vDash a$, for $a \in AP$ iff $a \in \pi(i)$.
- $\pi, i \vDash \neg\psi$ iff $\pi, i \nvDash \psi$.
- $\pi, i \vDash \psi \wedge \phi$ iff $\pi, i \vDash \psi$ and $\pi, i \vDash \phi$.
- $\pi, i \vDash \psi \vee \phi$ iff $\pi, i \vDash \psi$ or $\pi, i \vDash \phi$.
- $\pi, i \vDash \mathcal{X}\psi$ iff $\pi, i+1 \vDash \psi$.
- $\pi, i \vDash \mathcal{F}\psi$ iff $\exists j \geq i$, such that $\pi, j \vDash \psi$.
- $\pi, i \vDash \mathcal{G}\psi$ iff $\forall j \geq i$, such that $\pi, j \vDash \psi$.
- $\pi, i \vDash \psi\,\mathcal{U}\,\phi$ iff $\exists j \geq i$, such that $\pi, j \vDash \phi$, and $\forall k, i \leq k < j$, we have $\pi, k \vDash \psi$.
- $\pi, i \vDash \psi\,\mathcal{R}\,\phi$ iff $\forall j \geq i$, iff $\pi, j \nvDash \phi$, then $\exists k, i \leq k < j$, such that $\pi, k \vDash \psi$.

In model checking, LTL formulas commonly have two possible truth value states, namely `true` (`satisfied`) and `false` (`violated`). In case of monitoring an LTL specification in a running system, it might be the case, that it is not only of interest if a specification is satisfied or violated but also whether further state changes are possible that could resolve or cause a violation of a specification. That is, the state of a specification is either temporary (i.e., the state may change) or permanent (i.e., the state may not longer change). Consequently, to enable a more fine-grained analysis of the participants' understanding of LTL in the experiment, we employ the semantics of Runtime Verification Linear Temporal Logic (RV-LTL; cf. Bauer et al. [100]) that supports four truth value states. In particular, an LTL behavioral constraint specification at runtime is either `temporarily satisfied`, `temporarily violated`, `permanently satisfied`, or `permanently violated`.

The semantics of RV-LTL is defined as follows:

- $[u \vDash \psi]_{RV} = \top$ (ψ permanently satisfied by u) if for each possible finite continuation v of $u : uv \vDash \psi$.
- $[u \vDash \psi]_{RV} = \bot$ (ψ permanently violated by u) if for each possible finite continuation v of $u : uv \nvDash \psi$.
- $[u \vDash \psi]_{RV} = \top^p$ (ψ possibly/temporarily satisfied by u) if $u \vDash \psi$ and there exists a possible finite continuation v of $u : uv \nvDash \psi$.
- $[u \vDash \psi]_{RV} = \bot^p$ (ψ possibly/temporarily violated by u) if $u \nvDash \psi$ and there exists a possible finite continuation v of $u : uv \vDash \psi$.

Several existing studies make use of the concept of four LTL truth value states (cf. Pešić et al. [96], De Giacomo et al. [97], Maggi et al. [98], Falcone et al.[101], Joshi et al. [102], and Morse et al. [103]).

6.2.2. Property Specification Patterns (PSP)

Having been inspired by software design patterns, Dwyer et al. have proposed the Property Specification Patterns (PSP) [27], a collection of recurring behavioral constraints in software engineering. For each pattern, there exist transformation rules to underlying formal representations (including LTL and CTL)[6]. The patterns are categorized into *Occurrence Patterns* and *Order Patterns* as follows:

- Occurrence Patterns:

[6] `http://patterns.projects.cs.ksu.edu/documentation/patterns.shtml`

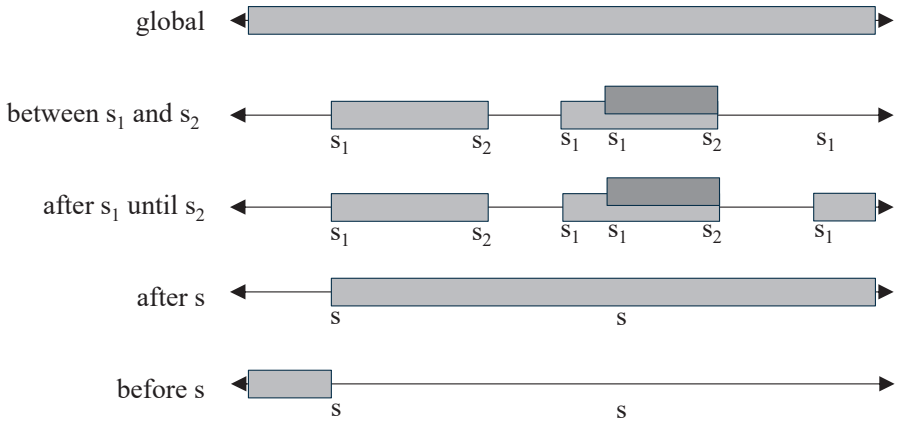


Figure 6.1.: Available scopes for Property Specification Patterns (shaded areas indicate the extent over which the pattern must hold)

  - Absence: a `never occurs`
  - Universality: a `always occurs`
  - Existence: a `occurs`
  - Bounded Existence:
    a `occurs at most` n `times`

- Order Patterns:
  - Precedence: a `precedes` b
  - Response: a `leads to` b
  - 2 Cause-1 Effect Precedence Chain:
    `(`a`, `b`)` `precedes` c
  - 1 Cause-2 Effect Precedence Chain:
    a `precedes (`b`, `c`)`
  - 2 Stimulus-1 Response Chain:
    `(`a`, `b`)` `leads to` c
  - 1 Stimulus-2 Response Chain:
    a `leads to (`b`, `c`)`

Moreover, each pattern has a scope. Figure 6.1 shows the available scopes and their area of effect:

- The *global* scope defines that a pattern must hold during the entire execution of a system. This scope is implicitly assumed when no other scope is defined.

- The *before* scope `before` s `[` p `]` defines that a pattern p must hold before the first occurrence of s.
- The *after* scope `after` s `[` p `]` defines that a pattern p must hold after the first occurrence of s.
- The *between* scope `between` s_1 `and` s_2 `[` p `]` defines that a pattern p must hold between every s_1 (i.e., starting the scope) that is followed by s_2 (i.e., closing the scope).
- The *after-until* scope `after` s_1 `until` s_2 `[` p `]` defines that a pattern p must hold after every s_1 (i.e., starting the scope) by no later than s_2 (i.e., closing the scope).

6.2.3. Event Processing Language (EPL)

In this section, we discuss the Event Processing Language (EPL; cf. EsperTech Inc. [104]) and how it can be applied for runtime monitoring of behavioral constraints. An EPL-based specification consists of an initial truth value (either `temporarily satisfied` or `temporarily violated`) and one or more query-listener pairs. A query-listener pair causes a truth value change in the behavioral constraint as soon as a matching event pattern is observed in the event stream. Consequently, an EPL-based behavioral constraint specification always consists of EPL queries that are composed of EPL operators and listeners that causes truth value changes (to `temporarily satisfied`, `temporarily violated`, `permanently satisfied`, `permanently violated`) to which the state of the behavioral constraint specification is set to by a positive match of an expression in the event stream. The semantics of those EPL operators is given as follows (cf. [104]):

- The *and* operator e_1 `and` e_2 is a logical conjunction that is matched once both e_1 and e_2 (in any order) have occurred.
- The *or* operator e_1 `or` e_2 is a logical disjunction that is matched once either e_1 or e_2 has occurred.
- The *not* operator `not` e is a logical negation that is matched if the expression e is not matched.
- The *every* operator `every` e not just observes the first occurrence of the expression e in the event stream but also each subsequent one.
- The *leads-to* operator e_1 `->` e_2 specifies that first e_1 must be observed and only then is e_2 matched. Intuitively, the whole expression is matched once e_1 is followed by e_2 at the occurrence of e_2.

- The *until* operator e_1 `until` e_2 matches the expression e_1 until e_2 occurs. In practice, this operator is commonly used in the expression `not` e_1 `until` e_2 that demands the absence of e_1 before the occurrence of e_2.

Obviously, further truth value changes are not possible once a permanent state (i.e., `permanently violated` or `permanently satisfied`) has been reached.

6.3. Experiment Planning

6.3.1. Goals

This experiment has the goal of measuring the construct *understandability* of behavioral constraint specifications expressed in different representations, namely Linear Temporal Logic (LTL), Property Specification Patterns (PSP), and Event Processing Language (EPL). The focus is on the *correctness* and *response time* of the answers given by the participants.

6.3.2. Experimental Units

All participants are students who enrolled in the course "Advanced Software Engineering Lab (ASE)" (which is a mandatory course in the master curriculum) or in the course "Distributed System Engineering Lab (DSE)" (which is optional in both the bachelor and master curricula) at the University of Vienna in the winter term 2015/2016 or the winter term 2016/2017.

We differentiate between two kinds of participants:

- Participants of DSE are used as proxies for novice to moderately advanced software architects, designers or developers.
- Participants of ASE are used as proxies for moderately advanced software architects, designers or developers.

The first experiment run aims to evaluate the languages with moderately advanced software architects, designers or developers, whereas the second experiment run considers both novice to moderately advanced and moderately advanced software architects, designers or developers. Another difference between the two experiment runs concerns the incentive for participation, the sampling strategy, and the setting. In the first experiment run, the experiment was carried out as a normal course assignment. Consequently, attendance was mandatory, and the submitted solutions were graded as an integral part of the course with up to 10 points (10% of the total course points). In the second experiment run, we changed to optional attendance that was rewarded by up to 10 bonus points. In both cases, the participants' performance in the experiment

determined the achieved points, and the participants were randomly allocated to the treatments (i.e., the three behavioral constraint representations).

6.3.3. Experimental Material & Tasks

The behavioral constraint specifications used in the tasks of this empirical study are based on recurring behavioral constraint specification patterns (cf. Dwyer et al. [27] and Bianculli et al. [109]). Each task of the experiment consists of a behavioral constraint definition and six combinations of an execution trace and a truth value. To optimize the execution of the experiment and to be independent from a specific application domain, the traces only consist of capital letters that represent surrogates of events (e.g. capital letter *A* could represent a task event *"Apply for Loan started"* in the BPM domain or a function/method invocation event in the SWA & SWE domain). For each combination the participant must evaluate whether it is correct or incorrect (i.e., whether the truth value is correct for the given trace). For example, Figure 6.2 (a) shows a task of the PSP group that is concerned with the *Precedence* pattern in the *Between* scope. In this task, only the choices *b)* and *f)* are correct. The same task is shown for the LTL group in Figure 6.2 (b) and for the EPL group in Figure 6.2 (c). Obviously, the expression of the behavioral constraint in each case is changed to the appropriate formalism. Furthermore, a different set of letters is used as a preventive measure against cheating (in addition to the seating arrangements).

The experiment document consisted of 10 tasks in the first experiment run. We reduced the number of tasks in the second experiment run to 9 tasks because a relatively large number of participants could not complete the first experiment run in time. Another difference between the two experiment runs is the order of tasks and answer choices. In the first run, the order was randomized between the groups whereas in the second run there was no difference in order between the groups. Randomization has the advantage that cheating is hampered, but it might introduce an unwanted variable to the experiment. For example, one group might have an easy first task, while another group has a hard one that hinders further progression and/or frustrates the participant. To avoid such unwanted effects, we kept the order unchanged in the second experiment run.

For the creation of the tasks of the experiment, we used an algorithm that generates traces and computes the correct truth value of a behavioral constraint specification that corresponds to each trace automatically. This algorithm leverages both the LTL and EPL specifications used in this experiment. For checking a trace against an LTL specification, the LTL formula is transformed to a non-deterministic finite automaton (cf. De Giacomo & Vardi [99]). By executing the automaton and analyzing its accepting states, the truth value of the LTL formula can be determined. Moreover, EPL behavioral constraint specifications are enacted in a CEP engine to

Please select the correct answer(s) for the following constraint description:
between Z and M [T precedes S]

a) At the end of trace ***[S, T, M, M, T, T, M, M, T, L]*** the truth value of the constraint is ***permanently violated***.
b) At the end of trace ***[T, M, S, T, Z, L, L, Z, Z, L]*** the truth value of the constraint is ***temporarily satisfied***.
c) At the end of trace ***[S, L, S, M, Z, L, T, L, L, Z]*** the truth value of the constraint is ***permanently violated***.
d) At the end of trace ***[L, L, Z, Z, L, T, T, Z, S, M]*** the truth value of the constraint is ***temporarily satisfied***.
e) At the end of trace ***[Z, S, L, Z, S, M, M, M, T, L]*** the truth value of the constraint is ***temporarily satisfied***.
f) At the end of trace ***[S, Z, S, M, S, M, T, Z, S, T]*** the truth value of the constraint is ***permanently violated***.

(a) PSP group

Please select the correct answer(s) for the following constraint description:
globally((V and not K and finally K) implies (not J until (H or K)))

a) At the end of trace ***[J, H, K, K, H, H, K, K, H, L]*** the truth value of the constraint is ***permanently violated***.
b) At the end of trace ***[H, K, J, H, V, L, L, V, V, L]*** the truth value of the constraint is ***temporarily satisfied***.
c) At the end of trace ***[J, L, J, K, V, L, H, L, L, V]*** the truth value of the constraint is ***permanently violated***.
d) At the end of trace ***[L, L, V, V, L, H, H, V, J, K]*** the truth value of the constraint is ***temporarily satisfied***.
e) At the end of trace ***[V, J, L, V, J, K, K, K, H, L]*** the truth value of the constraint is ***temporarily satisfied***.
f) At the end of trace ***[J, V, J, K, J, K, H, V, J, H]*** the truth value of the constraint is ***permanently violated***.

(b) LTL group

Please select the correct answer(s) for the following constraint description:
initial truth value: temporarily satisfied
permanently violated query: every(L leads-to not S and not V until M leads-to S)

a) At the end of trace ***[M, V, S, S, V, V, S, S, V, T]*** the truth value of the constraint is ***permanently violated***.
b) At the end of trace ***[V, S, M, V, L, T, T, L, L, T]*** the truth value of the constraint is ***temporarily satisfied***.
c) At the end of trace ***[M, T, M, S, L, T, V, T, T, L]*** the truth value of the constraint is ***permanently violated***.
d) At the end of trace ***[T, T, L, L, T, V, V, L, M, S]*** the truth value of the constraint is ***temporarily satisfied***.
e) At the end of trace ***[L, M, T, L, M, S, S, S, V, T]*** the truth value of the constraint is ***temporarily satisfied***.
f) At the end of trace ***[M, L, M, S, M, S, V, L, M, V]*** the truth value of the constraint is ***permanently violated***.

(c) EPL group

Figure 6.2.: *Precedence Between* task in the three different treatment/group variants in the second experiment run

evaluate their truth value. Using either LTL or EPL would suffice to create the tasks for the experiment. Nevertheless, we used both to double check the correctness of the behavioral constraint representations. Please note that it is not possible to use PSP specifications directly for execution (i.e., they are an abstraction of formal languages such as LTL and EPL), so they cannot be used for automated task generation. After the automated generation, we manually checked each task for correctness.

A slightly adapted version of the algorithm was used for the second experiment run. For the first run, the truth value of an answer choice was randomly altered to another truth value to create both wrong and correct answer choices. That kind of alteration might affect the results of the EPL group because the EPL approach explicitly contains truth values in its specifications. That is, some answer choices can be ruled out by matching the truth value of an answer choice against the set of possible truth values in the EPL specification. As we will discuss later (in the evaluation of the experiments in Section 6.7.1), apparently, these answer choices did not introduce bias that affected the EPL results positively, but they even had a negative impact on the response times in the EPL group in the first experiment run. We eliminated that threat to validity in the second experiment run by limiting random alterations of truth values in the answer choices of all groups to the set of possible truth values of a specification.

The tasks of both controlled experiment runs are available online (cf. Czepa & Zdun [142]) to support a replication of the study. In addition, code was released as open source that supports the automated generation of experiment tasks.[7]

6.3.4. Hypotheses, Parameters, and Variables

We hypothesized that PSP, as a highly abstract pattern language, is easier to understand than LTL and EPL, and that EPL, due to separation of concerns (as one or more queries can be used to explicitly define the truth value change that an observed event pattern causes), is easier to understand than LTL. Consequently, we formulated the following hypotheses for the two controlled experiment runs:

- $H_{0,1}$: There is no difference in terms of understandability between PSP and LTL.
- $H_{1,1}$: PSP has a higher level of understandability than LTL.

- $H_{0,2}$: There is no difference in terms of understandability between PSP and EPL.
- $H_{1,2}$: PSP has a higher level of understandability than EPL.

[7] `https://gitlab.swa.univie.ac.at/christoph.czepa/experimentgenerator/`

- $H_{0,3}$: There is no difference in terms of understandability between EPL and LTL.
- $H_{1,3}$: EPL has a higher level of understandability than LTL.

In both runs of this controlled experiment, there are two dependent variables, namely:

- the *correctness* achieved in trying to mark the correct answers, and
- the *response time*, which is the time it took to complete the 10 tasks in the first experiment run / the 9 tasks in the second experiment run.

These two dependent variables are commonly used to measure the construct understandability (cf. Feigenspan et al. [143] and Hoisl et al. [144]). The independent variable (also called factor) has three treatments, namely the three behavioral constraint representations (LTL, EPL, and PSP).

6.3.5. Experiment Design & Execution

We used a completely randomized design with one alternative per experimental unit, which is appropriate for the stated goal. Through this, we tried to avoid learning effects among the participants. Moreover, chances of selection bias are limited by using a computer-aided randomization for the assignment of participants to groups. The experiment is designed as a multiple-choice test for automated processing by the e-learning platform *Moodle*[8] to avoid experimenter bias in the analysis of the answers submitted. For that reason, the participants marked the answers in an answer sheet that was scanned and evaluated automatically. In some cases, it was necessary to correct some issues (e.g., imprecise markings) manually. To further limit the chances of experimenter bias, we used the four eyes principle while performing any such manual actions.

Two weeks before each experiment run, we handed out preparation material to the participants. This material consists of two documents: a document that provided a general introduction to the behavioral constraint language and slides that represent a kind of quick reference guide with the important aspects of the behavioral constraint representations and further examples. The participants were allowed to use the preparation material also during the experiment session.

The preparation material is based on (informal) natural language descriptions of the approaches and practical examples of application. There are two main reasons for this design of the preparation material. Firstly, we needed to ensure that all three languages were presented by the same educational methods at a comparable level of detail to not introduce unnecessary bias

[8] `https://moodle.org`

into our experiment. Secondly, we tried to present the approaches in an approachable manner to the participants as suggested by numerous existing research on teaching undergraduate students in theoretical computer science, formal methods, and logic (cf. Habiballa & Kmet' [145], Knobelsdorf & Frede [146], Carew et al. [147], Spichkova [148], and Richardson & Suinn [149]).

Please note that the tasks used in the experiment were randomly generated and not taken from the learning material. There were similarities between the behavioral constraints used in some of the experiment tasks and those used in the examples discussed in the learning material, but we could not find any indication of bias introduced by these similarities in the gathered data. In particular, the number of possibly affected experiment tasks was almost balanced between the groups, and the measured correctness of possibly affected tasks was overall similar to those of the remaining tasks (cf. Section 6.8.1).

Since the first experiment run also involved two qualitative questions regarding all behavioral constraint representations, we made the decision to provide the preparation materials of all three behavioral constraint representations to every participant. That is, the participants studied all behavioral constraint languages, and were unaware to which group they had been assigned until the start of the experiment session. However, having knowledge of all the representations could have introduced bias. For example, learning a representation could lead to a better understanding of another one, or the languages were mixed up unintentionally. As a result, we handed out preparation material for each group individually in the second experiment run.

The preparation material is available online (cf. Czepa & Zdun [142]) to support a replication of the study.

6.3.6. Procedure

The first experiment run had a duration of 90 minutes for working on the 10 tasks plus an additional 10 minutes for answering the two qualitative questions. The second experiment run had a total duration of 90 minutes for working on the 9 given tasks. No qualitative questions were asked in the second run. Seating arrangements were made to limit opportunity for misbehavior (i.e., cheating). At the beginning of each experiment run, the experiment material was handed out in the form of printed documents. Furthermore, we provided copies of the preparation materials for those participants who did not bring their own. Next, the participants were informed about the procedure of the experiment. This involved time tracking and how to mark answers correctly in the answer sheet for automatic processing. Following this, the participant had to fill out a general question sheet from which we gathered information about the participants' previous knowledge and experience. Next, the main part of the experiment started, in which the participants tried to solve the tasks of the experiment. The experiment runs were carried out following this plan without known deviations.

Table 6.1.: Summary of dropped participants

Group	Correctness	Response Time	Course	Reason
PSP	43.9 %	-	DSE	Time records missing completely
PSP	16.3 %	22.4 minutes	DSE	Suspicious time record for one task (10 second duration)
PSP	43.0 %	42.0 minutes	ASE	Already participated in first experiment run
PSP	77.8 %	40.1 minutes	ASE	Already participated in first experiment run
LTL	18.1 %	-	DSE	Time records missing for three tasks
LTL	14.8 %	-	ASE	Time records missing for one task; Already participated in first experiment run
LTL	31.7 %	-	DSE	Time records missing for one task
LTL	22.2 %	47.7 minutes	ASE	Already participated in first experiment run
LTL	-	50.0 minutes	DSE	Wrong answer sheet used
EPL	50.6 %	-	DSE	Time records missing for three tasks

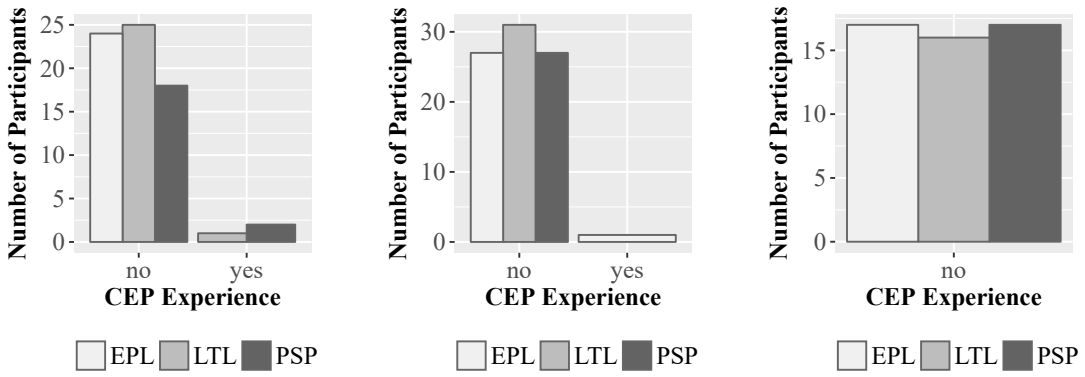
6.4. Analysis

6.4.1. Data Set Preparation

The first experiment run considered the overall response time per participant only. In the second run, we introduced a more fine-grained approach for time tracking on a per task basis. Unfortunately, a small number of the participants of the second experiment run failed to perform the time tracking per task correctly. Moreover, one participant used an answer sheet of a different group, and a few students had already participated in the first experiment run in the course of their previous studies. Due to the large number of remaining observations, we decided to drop the incomplete and potentially unreliable data of those participants. All dropped participants are summarized in Table 6.1.

6.4.2. Descriptive Statistics

The purpose of this section is to present the collected data (cf. Czepa & Zdun [142]) with the help of descriptive statistics. First, we analyze the previous knowledge and experience of the participants. By comparing the previous knowledge and other features (e.g., age of the participants) of the different groups, we try to find out whether the random allocation of participants



(a) 1st run (b) 2nd run: DSE (c) 2nd run: ASE

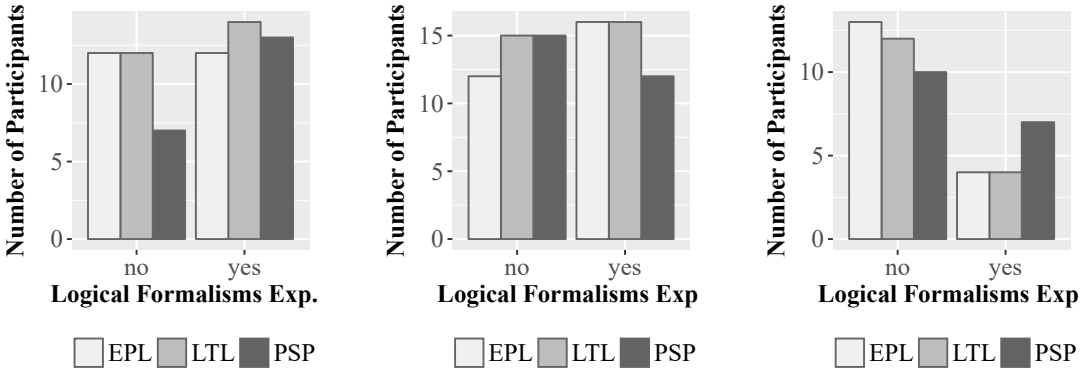
Figure 6.3.: Bar charts of the participants' experience with Complex Event Processing per group and experiment run

to groups has led to balanced groups or not. Following this, we will use descriptive statistics to analyze the dependent variables.

Descriptive Statistics of Previous Knowledge, Experience and Other Features of Participants

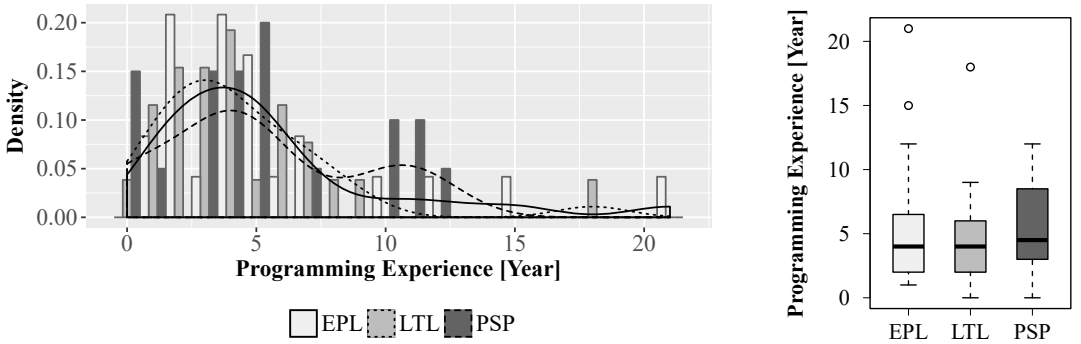
Figure 6.3 shows a bar chart of the participants' previous knowledge of Complex Event Processing (CEP). The distribution between the groups is relatively well-balanced. Overall, only a very few participants are experienced with CEP. Figure 6.4 shows a bar chart of the participants' previous knowledge of logical formalisms (e.g., first-order logic) in general. Again, the distribution between the groups is relatively well-balanced. Interestingly, the students in ASE seem to be less experienced with logical formalisms than the DSE students in the second experiment run. A possible reason for this might be that more time has passed between attending the respective lectures introducing the formalisms for master students in ASE than for bachelor students in DSE and the diverse background of our master students (i.e., coming from various faculties and countries with different curricula).

Next, we investigate the participants' programming experience and work experience in the software industry. Figure 6.5 shows a kernel density plot and box plot of the programming experience per group in the first experiment run. The peak density of all groups is at about 3 to 4 years of programming experience. Another peak is at about 11 years in the PSP group. Both the EPL and LTL group have a small amount of participants that have 15 and more years of programming experience (shown as outliers in the box plot) while the PSP group has a slightly larger number of participants that have between 10 and 13 years of experience in programming. According to



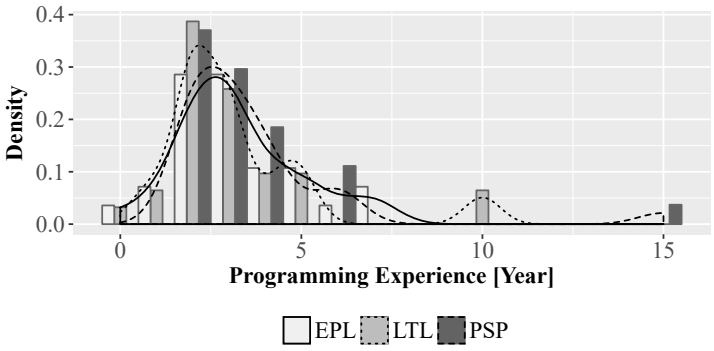
(a) 1st run (b) 2nd run: DSE (c) 2nd run: ASE

Figure 6.4.: Bar charts of the participants' experience with logical formalisms per group and experiment run

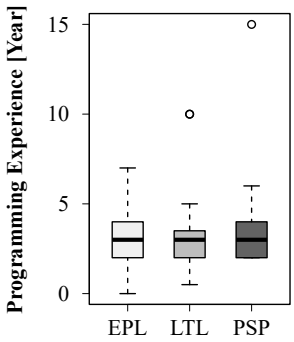


(a) Kernel density plot (b) Box plot

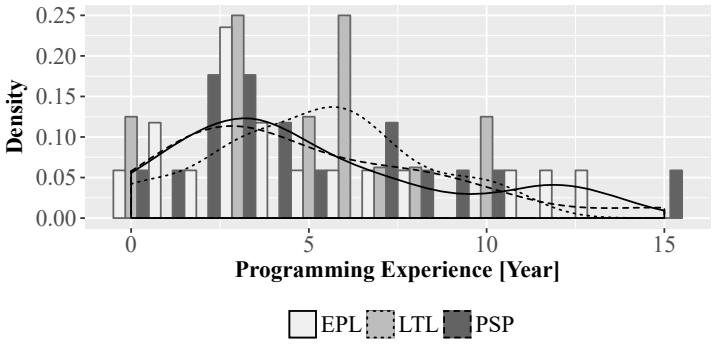
Figure 6.5.: Kernel density plot and box plot of the participants' programming experience per group in the first experiment run



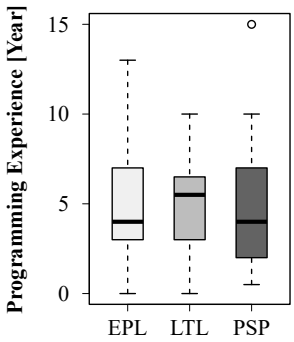
(a) Kernel density plot: DSE



(b) Box plot: DSE

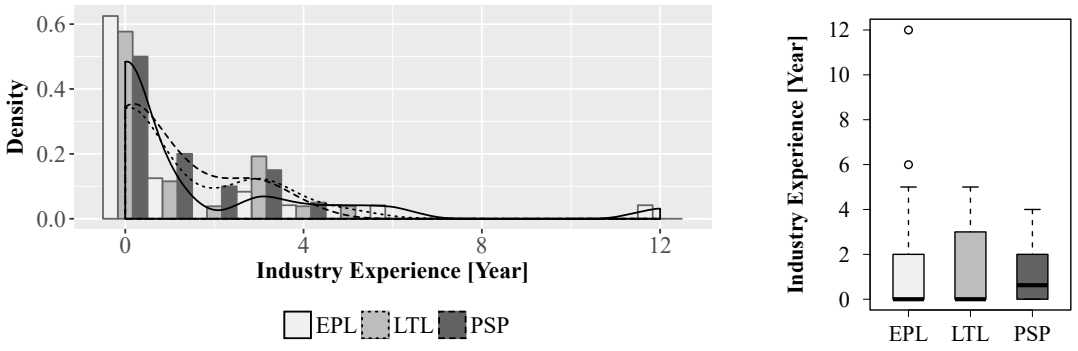


(c) Kernel density plot: ASE



(d) Box plot: ASE

Figure 6.6.: Kernel density plots and box plots of the participants' programming experience per group in the second experiment run



(a) Kernel density plot

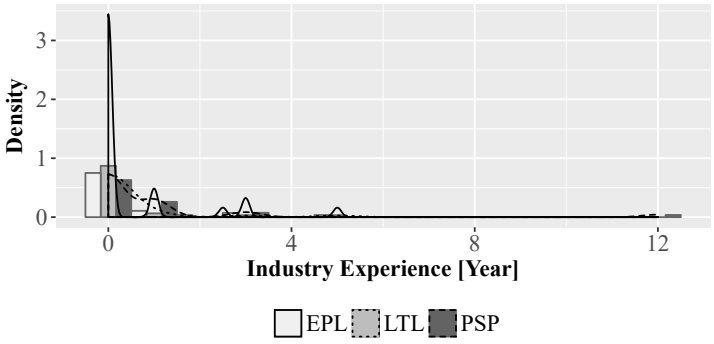
(b) Box plot

Figure 6.7.: Kernel density plot and box plot of the participants' software industry experience per group in the first experiment run

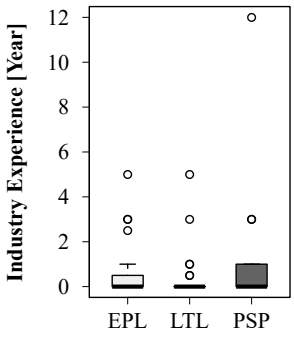
the plots, the participants of the PSP group seem to be slightly more experienced in programming. Figure 6.6 contains a kernel density plot and box plot of the participants' programming experience in the second experiment run. DSE participants have the peak density in all groups at about 3 years. Only a handful of participants have more than 7 years of programming experience in all groups. According to these plots, the distribution is similar in all three experiment groups. In ASE, we can observe a difference in the central tendency in the LTL group which has its peak density at about 6 years, whereas the peak density of the two other groups is at about 4 years. Above 10 years of experience occurs only in the EPL and PSP groups. Thus, those groups contain a few highly experienced programmers, and the LTL group appears to be slightly more experienced on the average.

Figure 6.7 shows the participants' experience with regard to working in the software industry in the first experiment run. The majority of participants do not have any such work experience at all. Overall, the shapes and peaks of the distributions are rather similar. Some EPL participants have a higher number of years of experience in comparison to the participants in the other groups (shown as outliers in the box plot). The PSP group has slightly more work experience on the average. In Figure 6.8, the participants' industrial experience in the second experiment run is shown. The peak density of all groups in DSE is at zero years. In ASE, the LTL group has slightly less working experience than the two other groups apparently.

In the second experiment run, we additionally gathered information regarding the age and gender of the participants. In Figure 6.9, the participants' age per group is shown. On the average, DSE students of the LTL group are slightly older than their colleagues in the PSP group, and DSE students of the EPL group are younger than their colleagues in the two other groups. Overall, the majority of the DSE participants shares the same age group (20–25). In ASE, the

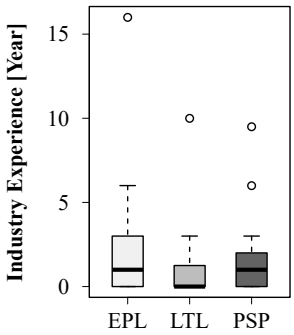


(a) Kernel density plot: DSE



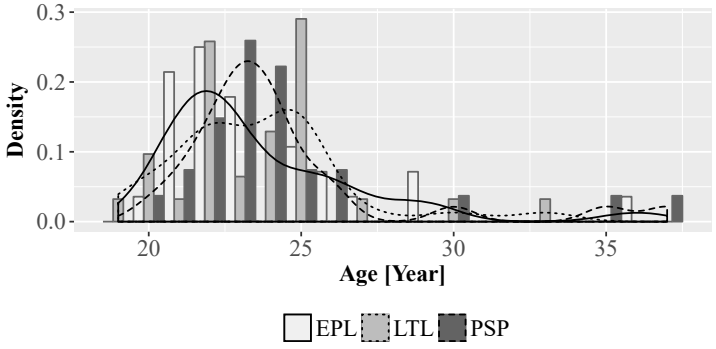
(b) Box plot: DSE

(c) Kernel density plot: ASE

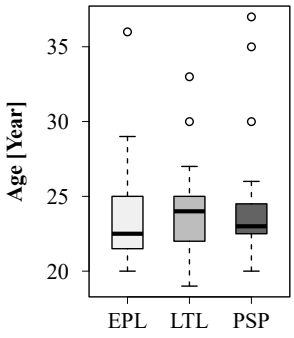


(d) Box plot: ASE

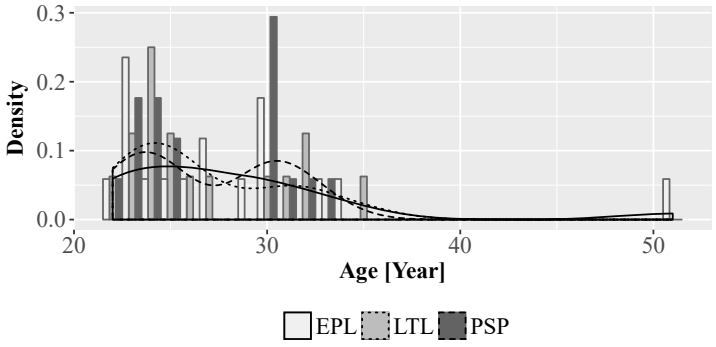
Figure 6.8.: Kernel density plots and box plots of the participants' software industry experience per group in the second experiment run



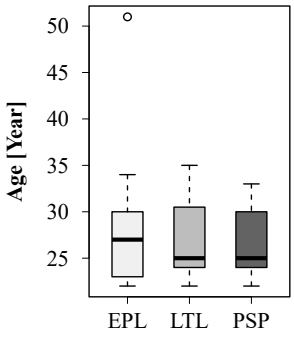
(a) Kernel density plot: DSE



(b) Box plot: DSE

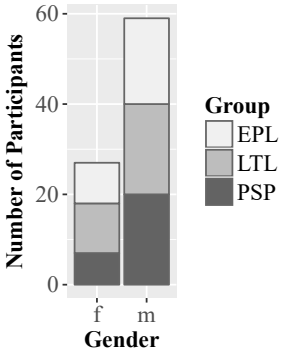


(c) Kernel density plot: ASE

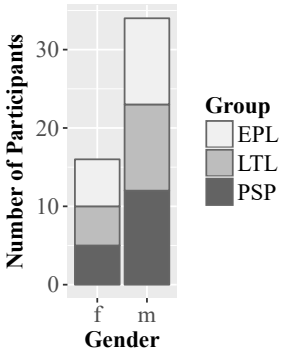


(d) Box plot: ASE

Figure 6.9.: Kernel density plots and box plots of the participants' age per group in the second experiment run



(a) DSE



(b) ASE

Figure 6.10.: Bar charts of the participants' gender per group in the second experiment run

Table 6.2.: Number of observations, central tendency and dispersion per group of the first experiment run

	LTL	**PSP**	**EPL**
Number of observations	26	20	24
Mean correctness [%]	33.04	69.55	50.70
Standard deviation [%]	15.39	25.46	28.52
Median correctness [%]	31.3	78	48.7
Median absolute deviation [%]	12.79	23.87	42.48
Min. correctness [%]	5	12.7	10.5
Max. correctness [%]	63	100	94.7
Skew (correctness)	0.02	−0.56	0.01
Kurtosis (correctness)	−0.83	−1.01	−1.61
Mean response time [min]	69.85	58.25	72.12
Standard deviation [min]	15.25	20.86	21.47
Median response time [min]	73	57.50	78.5
Median absolute deviation [min]	17.05	25.95	17.05
Min. response time [min]	35	28	11
Max. response time [min]	90	90	90
Skew (response time)	−0.44	0.13	−1.44
Kurtosis (response time)	−0.74	−1.53	1.25

participants of the EPL group are on average slightly older. Moreover, the kernel density plot suggests that there are two age groups, namely younger students (aged 22–27) and older students (aged 30–35). The fraction of female participants is slightly lower in the PSP group in DSE (cf. Figure 6.10). Overall, the distribution of male and female participants is balanced.

According to the descriptive statistics, the groups in both experiment runs are similar with regards to previous knowledge, experience, and also with regards to age and gender in the second run. No major differences between the groups are noticeable.

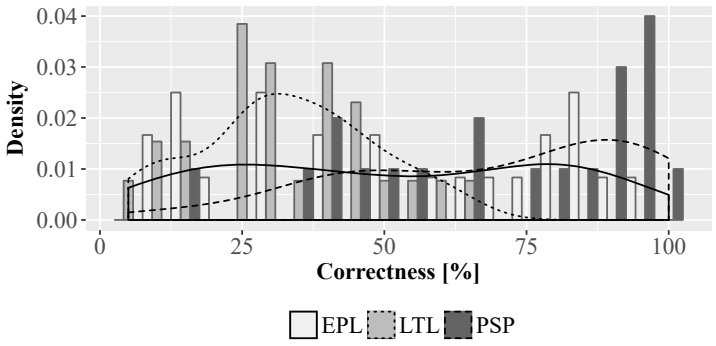
Descriptive Statistics of Dependent Variables

Table 6.2 & Table 6.3 contain the number of observations, central tendency measures and dispersion measures of the dependent variables (correctness and response time) per behavioral constraint representation of the first and second experiment run. The second experiment run consists of measurements in two courses, namely DSE and ASE. That is, we tested our hypotheses three times, namely in the first experiment run in ASE, and in the second experiment run in DSE and ASE. In all three cases, the PSP group reached the highest mean and median correctness (about 70–75%), followed by the EPL group (about 50–55% correctness) and the LTL group (about 30–35% correctness). The maximum measured response time in the first run is the 90 minutes limit in all groups. In response to this, we reduced the number of tasks in the second run by one

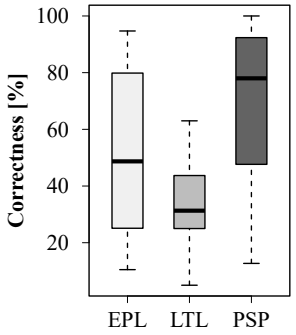
Table 6.3.: Number of observations, central tendency and dispersion per group of the second experiment run

		LTL	PSP	EPL
DSE	Number of observations	31	27	28
	Mean correctness [%]	32.45	70.55	53.83
	Standard deviation [%]	17.23	20.89	23.04
	Median correctness [%]	31.7	73.70	54.10
	Median absolute deviation [%]	18.09	18.09	23.5
	Min. correctness [%]	6.5	16.30	5.6
	Max. correctness [%]	70.6	97.20	86.70
	Skew (correctness)	0.36	−0.87	−0.37
	Kurtosis (correctness)	−0.62	−0.11	−0.86
	Mean response time [min]	51.03	36.65	43.80
	Standard deviation [min]	14.95	14.18	14.71
	Median response time [min]	51	33.05	42.76
	Median absolute deviation [min]	13.42	15.25	13.2
	Min. response time [min]	19	17.35	23
	Max. response time [min]	88	63.08	84.63
	Skew (response time)	0.25	0.56	0.81
	Kurtosis (response time)	−0.10	−1.10	0.38
ASE	Number of observations	16	17	17
	Mean correctness [%]	36.42	72.41	54.4
	Standard deviation [%]	17.32	18.17	21.06
	Median correctness [%]	38.60	71.9	53.70
	Median absolute deviation [%]	9.71	18.09	17.35
	Min. correctness [%]	3.7	33.50	8.9
	Max. correctness [%]	67.6	100	87.6
	Skew (correctness)	−0.08	−0.23	−0.32
	Kurtosis (correctness)	−0.63	−0.78	−0.70
	Mean response time [min]	55.32	39.12	44
	Standard deviation [min]	11.51	8.95	15.33
	Median response time [min]	53.15	39.5	44.83
	Median absolute deviation [min]	11.48	9.64	19.74
	Min. response time [min]	35.5	23.47	23
	Max. response time [min]	78	52.93	70.5
	Skew (response time)	0.27	−0.23	0.30
	Kurtosis (response time)	−0.90	−1.19	−1.35

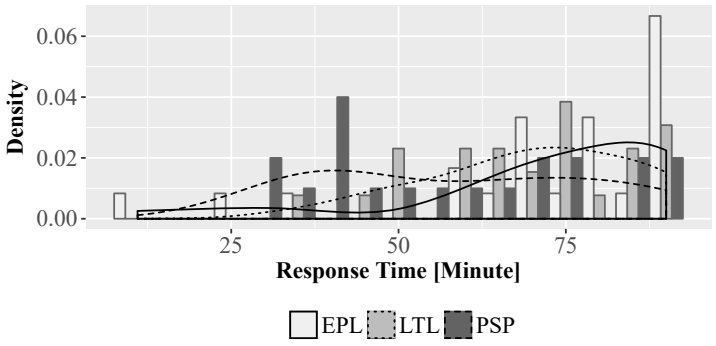
(from 10 to 9). In the second run, the maximum response time is 88 minutes. Interestingly, students in the second run in ASE managed to finish on the average about 20–40% faster than their colleagues in the first run, which cannot have been caused by the removal of a single task alone, as the expected response time reduction would be only about 10%. We suspect that this difference is caused by the change from total experiment time recordings in the first experiment run to per task time recordings in the second experiment run, and the late assignment of participants to groups at the beginning of the experiment session in the first run. Obviously, the time recordings of the participants in the first experiment run included times such as pauses, task switching times, and times spent on consulting the accompanying documents that are not directly related to solving a specific task. In the first experiment run the participants had to be prepared for all three representations, and the experiment group was assigned at the beginning at the experiment session. Up to this point in time, the participants did not know to which experiment group they were assigned to. As a result, once it became clear which of the three approaches must be applied, the participants revisited the learning material related to the assigned representation intensely. In the second experiment run, group assignment was clear beforehand, so this initial consulting of the info material did not take place in a comparable intensity. Furthermore, the mean (72.12 minutes) and median response times (78.5 minutes) of the EPL group are longer than those of the LTL group (69.85 minutes mean and 73 minutes median) in the first run. With regard to the hypotheses of this experiment, the response time measurements in the first experiment run are an unexpected result since we expected that the response times in the EPL group would be faster than in the LTL group. In contrast, the EPL group has a faster response time than the LTL group in the second run. We suspect that this effect could have been caused by the task design which contained truth value states in the answer choices that are not part of the EPL behavioral constraint definition. Originally (i.e., at the time the first run was completed, and before the second run was carried out), we thought that there might have been a bias present in the first experiment run in favor of the EPL group, because wrong answer choices could have been potentially easier to identify by the EPL participants. However, these answer choices seemingly confused the participants rather than helping them. During the first experiment run, EPL participants repeatedly asked whether there was an error in the exercise or whether it was really that easy to solve it. Due to their confusion, EPL participants spent considerably more time on solving the tasks in the first experiment run. The skew values of the correctness variable are balanced (i.e., close to zero) for the LTL and EPL groups in the first run. That is, the distribution is rather symmetric. The negative PSP correctness skew value (-0.56) suggests that the distribution is left-tailed. A positive value such as the skew of the response time variable in the second run in DSE indicates a right-tailed distribution. Kurtosis, another measure for the shape of a distribution, focuses on the general tailedness of a distribution. A negative kurtosis indicates fat tails, and vice versa,



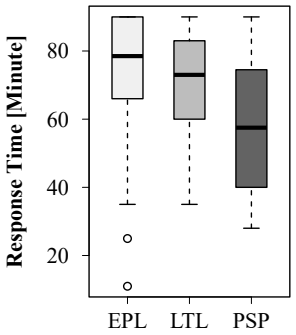
(a) Kernel density plot: Correctness



(b) Box plot: Correctness



(c) Kernel density plot: Response time



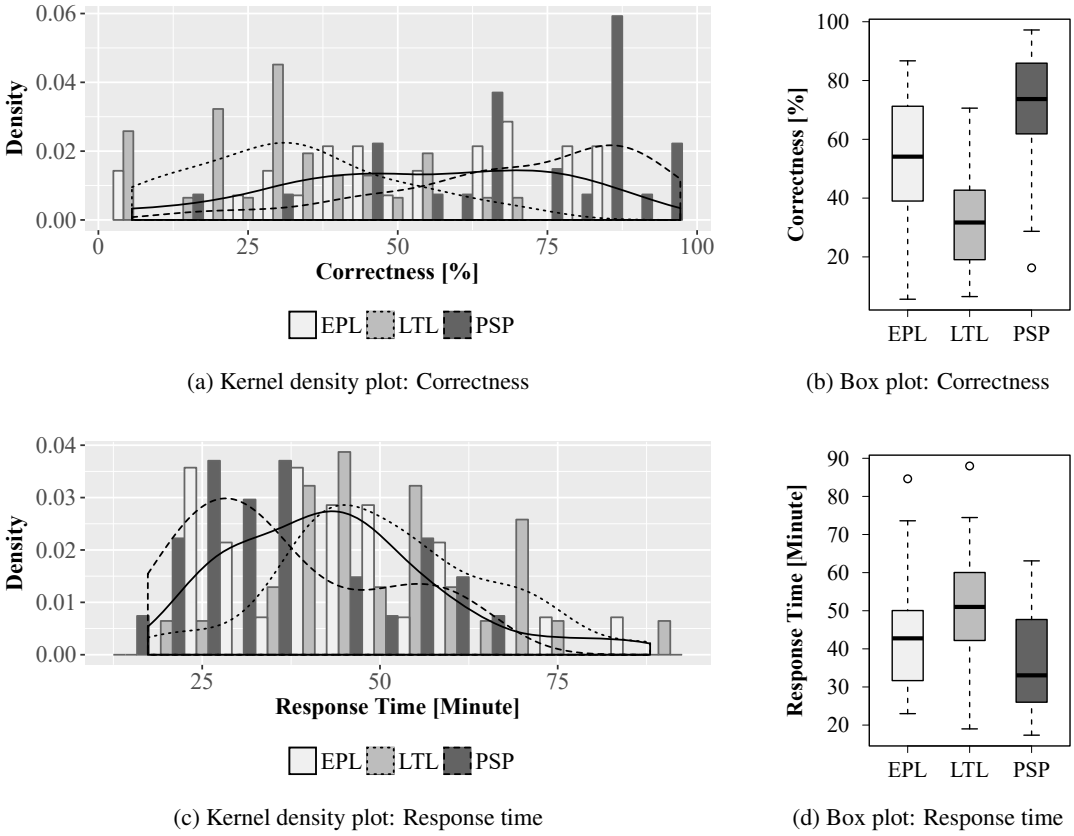
(d) Box plot: Response time

Figure 6.11.: Kernel density plots and box plots of the participants' overall correctness of the given answers and the overall response time per group in the first experiment run

a positive kurtosis indicates skinny tails with a distribution toward the mean. In general, the differences in skew and kurtosis between the groups indicate differences in the shape of their distributions. Consequently, the skew and kurtosis values in Table 6.2 suggest that there exist changes in distribution of the dependent variables between the groups.

In addition to the descriptive statistics in Table 6.2, we will now perform a graphical analysis that is based on kernel density plots and box plots to further study the dependent variables. Kernel density plots are well-suited to visualize the distribution of the data whereas box plots are used to visualize the quartiles and outliers.

In the first experiment run (as shown in Figure 6.11), the EPL correctness distribution is extremely long-tailed and flat. The distribution of the PSP correctness has its peak close to the maximum and a long left tail. LTL has the steepest correctness distribution, with its peak at about 30% and a right tail that already ends at about 75% correctness. The EPL response time distribution has its peak close to the maximum of 90 minutes and a slope until about 50



(a) Kernel density plot: Correctness

(b) Box plot: Correctness

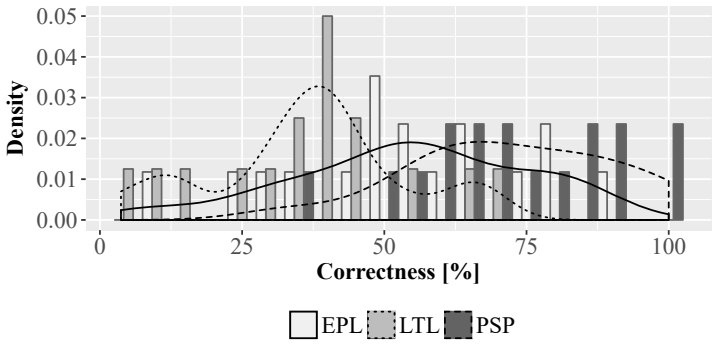
(c) Kernel density plot: Response time

(d) Box plot: Response time

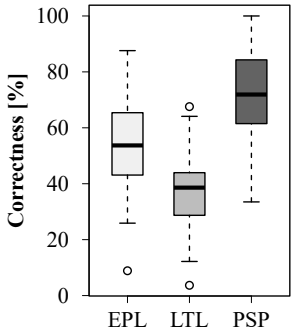
Figure 6.12.: Kernel density plots and box plots of the DSE participants' overall correctness of the given answers and the overall response time per group in the second experiment run

minutes, where the density is already low, but remaining nearly constant from that point on. LTL response time has its peak density at about 75 minutes and a long left tail that ends at about 25 minutes. There are two response time outliers in the EPL group which do not result from measuring errors.

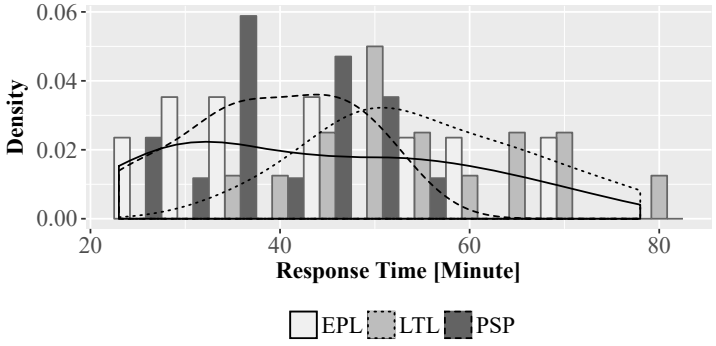
In the second experiment run with DSE participants (as shown in Figure 6.12), the EPL correctness distribution is still rather flat, but less extreme than in the first run. PSP has its peak correctness at about 90% and a long left tail. The kernel density plot shows the peak correctness in the LTL group at about 30%, and the right tail of the distribution ending at about 75%. In contrast to the first run, we observe faster response times overall and especially in the EPL group. The peaks of the LTL and EPL response time distributions share nearly the same location at about 45 minutes. Apart from that, the distributions are fairly different because the EPL group has a higher density on the left tail whereas the LTL group has a higher density on the right tail.



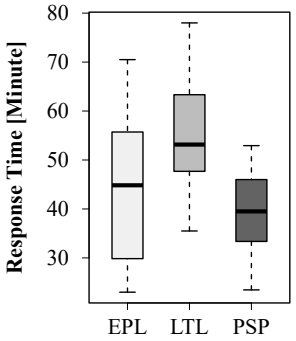
(a) Kernel density plot: Correctness



(b) Box plot: Correctness



(c) Kernel density plot: Response time

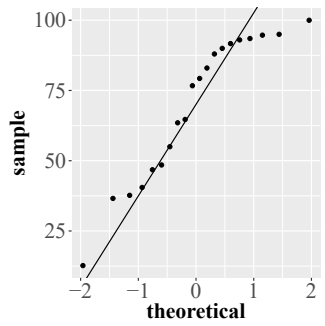


(d) Box plot: Response time

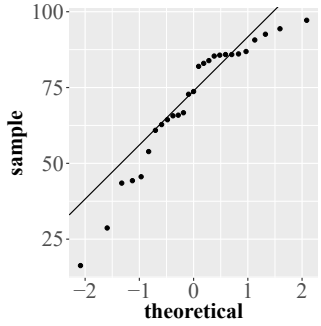
Figure 6.13.: Kernel density plots and box plots of the ASE participants' overall correctness of the given answers and the overall response time per group in the second experiment run

In the PSP group, the highest density is located at about 25–30 minutes with another smaller peak at about 55–60 minutes. There is a single correctness outlier in the PSP group, and there are two response time outliers, one in the LTL group and another in the EPL group. Since those outliers are not caused by measuring errors, we see no reason for exclusion.

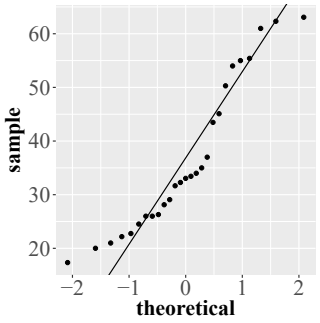
In the second experiment run with ASE participants (as shown in Figure 6.13), the peak density of the LTL group correctness is located at about 35–40%. Two tiny peaks can be found at about 10–15% and 60–65% correctness. The peak density of the PSP group is located at 60–65%, and the density drops merely slowly on the right tail which indicates a high level of correctness in this group. The EPL group has its peak correctness at about 55%, and the shape indicates that the right tail has a slightly higher density than the left tail. Like in Figure 6.11 and Figure 6.12, the response time distribution is relatively flat in the EPL group. The highest density can be found at about 30–35 minutes. From that point on, the density is slowly decreasing. PSP



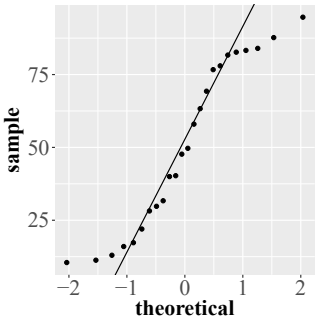
(a) Correctness data of PSP group in the 1st experiment run



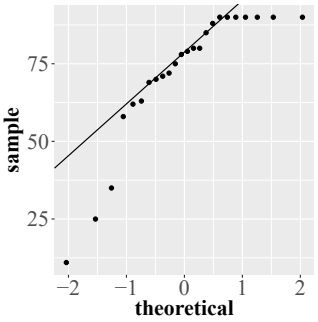
(b) Correctness data of PSP group in the 2nd experiment run (DSE)



(c) Response time data of PSP group in the 2nd experiment run (DSE)



(d) Correctness data of EPL group in the 1st experiment run



(e) Response time data of EPL group in the 1st experiment run

Figure 6.14.: Normal QQ plots

has the steepest response time distribution with its peak at about 45 minutes and higher density on the left tail. In contrast, the LTL response time distribution has a higher density on the right tail, and the peak response time is located at about 50 minutes. There are two correctness outliers in the LTL group, and a single correctness outlier in the EPL group. Again, those are valid measurements, and we see no reason for excluding them.

Generally, the distributions look fairly different, which implies unequal variances in the different groups, and there are obvious differences in central tendency. All present outliers appear to be valid measurements, so there is not enough evidence to drop them.

A graphical analysis by normal Q-Q plots and Shapiro-Wilk tests of normality (cf. Table 6.4) suggest that the univariate normality assumption does not hold in multiple cases. In the following, we discuss the most severe cases. Specifically, the univariate normality assumption does not hold for

Table 6.4.: Shapiro-Wilk test of normality (* for $\alpha = 0.05$, ** for $\alpha = 0.01$, * for $\alpha = 0.001$)

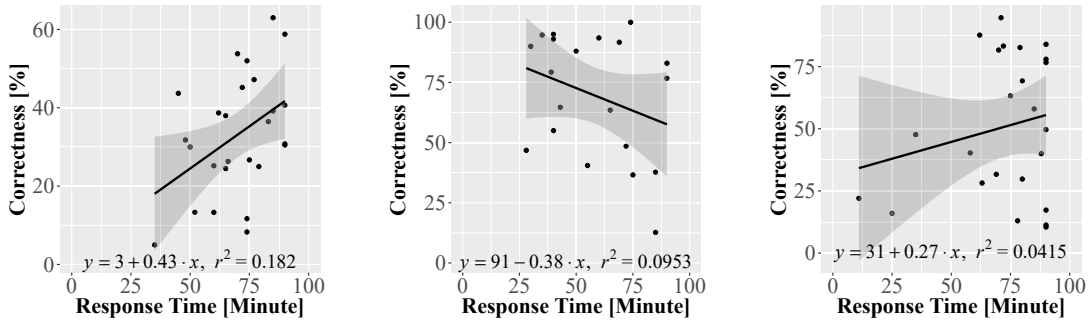
Group	Dependent Variable	1st Run	2nd Run: DSE	2nd Run: ASE
LTL	Correctness	$W = 0.9782$ $p = 0.8328$	$W = 0.96$ $p = 0.3096$	$W = 0.9598$ $p = 0.6581$
	Response Time	$W = 0.9501$ $p = 0.2326$	$W = 0.976$ $p = 0.696$	$W = 0.9838$ $p = 0.9867$
PSP	Correctness	$W = 0.902$ $p = 0.045$ *	$W = 0.9062$ $p = 0.0186$ *	$W = 0.9725$ $p = 0.8606$
	Response Time	$W = 0.9216$ $p = 0.1063$	$W = 0.9047$ $p = 0.0172$ *	$W = 0.9598$ $p = 0.6277$
EPL	Correctness	$W = 0.9109$ $p = 0.0369$ *	$W = 0.9539$ $p = 0.2473$	$W = 0.9753$ $p = 0.9023$
	Response Time	$W = 0.7947$ $p = 0.0002$ ***	$W = 0.9402$ $p = 0.112$	$W = 0.9314$ $p = 0.2298$

- the correctness variable of the PSP group in the first and second (DSE) experiment run (cf. Figure 6.14 (a) & (b)),
- the response time variable of the PSP group in the second (DSE) experiment run (cf. Figure 6.14 (c)),
- the correctness and response time variable of the EPL group in the first experiment run (cf. Figure 6.14 (d) & (e)).

Scatter plots (as shown in Figure 6.15) and Kendall's rank correlation tau tests (summarized in Table 6.5) do not indicate any significant correlation of the two dependent variables (i.e., correctness and response time). Please note that the scatter plots in the second experiment run reveal a similar picture, so they are omitted intentionally in Figure 6.15.

6.5. Statistical Inference

The multivariate analysis of variance (MANOVA) is a suitable statistical inference procedure in the presence of two dependent variables. However, necessary assumptions must be met. Please note that we will not discuss each and every assumption or report its violation if a specific other (more elementary) assumption already indicates a violation that hinders any meaningful application of the method on the given data set. Both the graphical analysis (by kernel density plots



(a) LTL (b) PSP (c) EPL

Figure 6.15.: Scatter plots of response time vs. correctness in first experiment run with linear trend lines, 95% confidence regions, and coefficients of determination (r^2)

Table 6.5.: Kendall's rank correlation tau (* for $\alpha = 0.05$, ** for $\alpha = 0.01$, * for $\alpha = 0.001$)

Group	**1st Run**	**2nd Run: DSE**	**2nd Run: ASE**
LTL	$\tau = 0.2638$ $z = 1.8589$ $p = 0.0631$	$\tau = -0.0693$ $z = -0.5445$ $p = 0.5861$	$\tau = 0.1833$ $T = 71$ $p = 0.3502$
PSP	$\tau = -1.2691$ $z = -1.2691$ $p = 0.2044$	$\tau = -0.0029$ $z = -0.0209$ $p = 0.9834$	$\tau = -0.155$ $z = -0.8658$ $p = 0.3866$
EPL	$\tau = -0.0151$ $z = -0.1006$ $p = 0.9198$	$\tau = 0.15$ $z = 1.1263$ $p = 0.26$	$\tau = 0.0441$ $T = 71$ $p = 0.8393$

Table 6.6.: Cliff's d of first experiment run, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Correctness	$p_1 = P(X > Y)$	0.8769	0.7021	0.6715
	$p_2 = P(X = Y)$	0	0.0042	0
	$p_3 = P(X < Y)$	0.1231	0.2938	0.3285
	d	−0.7539	−0.4083	−0.343
	s_d	0.1097	0.1575	0.162
	z	−6.8699	−2.5933	−2.1157
	CI low	−0.8847	−0.633	−0.5789
	CI high	−0.513	−0.1203	−0.0539
	p	8.8×10^{-9}	0.0065	0.0198
	FDR adjusted p	5.3×10^{-8}	0.0195	0.0297
	level of significance	***	*	*
	effect size magnitude	large	medium	medium
Response Time	$p_1 = P(X > Y)$	0.3115	0.2833	0.564
	$p_2 = P(X = Y)$	0.0442	0.0417	0.0577
	$p_3 = P(X < Y)$	0.6442	0.675	0.3782
	d	0.3327	0.3917	−0.1859
	s_d	0.1693	0.1641	0.164
	z	1.9649	2.3874	−1.1336
	CI low	0.0312	0.0931	−0.4376
	CI high	0.5787	0.6256	0.0928
	p	0.0279	0.0108	0.1313
	FDR adjusted p	0.0335	0.0216	0.1313
	level of significance	*	*	-
	effect size magnitude	medium	medium	-

and normal Q-Q plots) and Shapiro-Wilk tests of the data indicate that the univariate normality assumption does not hold in multiple cases. The linearity assumption demands that all of the dependent variables are linearly related to each other, but scatter plots and Residuals vs. Fitted plots suggest that the linearity assumption is not met by the data sufficiently. As a result, the power of the multivariate and parametric MANOVA test might be affected, and its results would be unreliable. Multivariate and parametric testing might lead to unreliable results due to unsatisfied model assumptions, so we fall back to univariate non-parametric testing. The univariate non-parametric Kruskal-Wallis test is strongly affected by unequal variances (cf. Kitchenham et al. [140]), so its result might be not reliable because the kernel density plots of the data show distributions that look different in many cases which implies unequal variances in the different groups.

As a consequence, we use *Cliff's delta* (cf. Cliff [150] and Rogmann [151]), a robust non-

Table 6.7.: Cliff's d of second experiment run (DSE participants), one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Correctness	$p_1 = P(X > Y)$	0.902	0.709	0.7661
	$p_2 = P(X = Y)$	0	0.0053	0.0012
	$p_3 = P(X < Y)$	0.098	0.2857	0.2327
	d	-0.8041	-0.4233	-0.5334
	s_d	0.0847	0.1388	0.1275
	z	-9.4883	-3.0494	-4.1823
	CI low	-0.9053	-0.6238	-0.7107
	CI high	-0.6163	-0.1706	-0.2924
	p	1.5×10^{-13}	0.0018	5.0×10^{-5}
	FDR adjusted p	8.9×10^{-13}	0.0027	0.0002
	level of significance	***	**	***
	effect size magnitude	large	medium	large
Response Time	$p_1 = P(X > Y)$	0.2473	0.3585	0.3502
	$p_2 = P(X = Y)$	0.0024	0.0027	0.0023
	$p_3 = P(X < Y)$	0.7503	0.6389	0.6474
	d	0.503	0.2804	0.2972
	s_d	0.1345	0.153	0.1452
	z	3.7393	1.8324	2.0474
	CI low	0.251	0.0135	0.0432
	CI high	0.6911	0.51	0.5151
	p	0.0002	0.0363	0.0226
	FDR adjusted p	0.0004	0.0363	0.0271
	level of significance	***	*	*
	effect size magnitude	large	medium	medium

Table 6.8.: Cliff's d of second experiment run (ASE participants), one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Correctness	$p_1 = P(X > Y)$	0.9154	0.7405	0.7427
	$p_2 = P(X = Y)$	0	0	0.0037
	$p_3 = P(X < Y)$	0.0846	0.2595	0.2537
	d	-0.8309	-0.481	-0.489
	s_d	0.0968	0.1691	0.1761
	z	-8.5879	-2.8448	-2.777
	CI low	-0.9352	-0.7108	-0.7248
	CI high	-0.5937	-0.1585	-0.1506
	p	5.3×10^{-10}	0.0038	0.0046
	FDR adjusted p	3.2×10^{-9}	0.0069	0.0069
	level of significance	***	**	**
	effect size magnitude	large	large	large
Response Time	$p_1 = P(X > Y)$	0.125	0.4187	0.2794
	$p_2 = P(X = Y)$	0.0037	0	0.0037
	$p_3 = P(X < Y)$	0.8713	0.5813	0.7169
	d	0.7463	0.1626	0.4375
	s_d	0.1172	0.2063	0.1814
	z	6.3672	0.7883	2.4124
	CI low	0.4852	-0.1854	0.0972
	CI high	0.8852	0.4744	0.6862
	p	2.2×10^{-7}	0.2	0.011
	FDR adjusted p	6.5×10^{-7}	0.2182	0.0132
	level of significance	***	-	*
	effect size magnitude	large	-	large

parametric test that is unaffected by change in distribution, non-normal-data and possible non-stable variance. The results of the test are shown in Table 6.6 for the first experiment run and Table 6.7 & Table 6.8 for the second experiment run where

- p_1 represents the probability that a subject chosen from group X has a higher value than a randomly chosen subject from group Y,
- p_2 reflects the probability that a subject chosen from group X has an equal value to a randomly chosen subject from group Y,
- p_3 is the probability of superiority of Y over X,
- d denotes Cliff's delta for independent groups (i.e., the difference between the probability that a randomly chosen Y measurement has a higher value than a randomly chosen X measurement and the probability for the opposite),
- s_d is the unbiased sample estimate of the delta standard deviation,
- z is the z-score of Cliff's delta, and
- $(CI\ low,\ CI\ high)$ denotes the confidence interval.

Multiple testing ($n = 6$ because of the two dependent variables and three treatments) requires us to lower the significance level in order to avoid Type I errors (i.e., detection of an effect that is not present). As a classical and widely used method, the Bonferroni correction suggests lowering the alpha value to $\alpha = \frac{0.05}{6} = 0.008\dot{3}$, but the method is also known to skyrocket Type II errors (i.e., failing to detect an effect that is present). As an alternative that is more robust against Type II errors, we consider FDR (False Discovery Rate) adjusted p-values (cf. Benjamini & Hochberg [152]). According to these FDR adjusted p-values, there is evidence for the rejection of the null hypotheses of this study.

In the first experiment run (cf. Table 6.6), almost all test results are significant, which suggests a rejection of $H_{0,1}$ and $H_{0,2}$. $H_{0,3}$ can only be rejected on the basis of the correctness variable since the test result does not indicate any significant difference in the response times of the EPL and LTL group. Moreover, the results suggest that the difference in terms of correctness between the PSP and LTL group are highly significant, with a large effect size magnitude. All remaining significant test results of the first experiment run show a medium-sized effect.

In the second experiment run (cf. Table 6.7), the majority of the test results is significant. Only one test, namely the PSP/EPL response time with ASE participants, has no significant result, which means that $H_{0,2}$ (in ASE) can only be rejected on basis of the correctness result. All other test results range from significant ($\alpha = 0.05$) to highly significant ($\alpha = 0.001$), which

suggests a rejection of the null hypotheses. Moreover, all significant results show a large or medium effect size magnitude. It is striking that all PSP/LTL test results are highly significant with a large-sized effect.

The statistics software R[9] was used for all statistical analyses. In particular, we used the following libraries in the course of our statistical evaluation: *biotools* [153], *car* [154], *ggplot2* [155], *mvnormtest* [156], *mvoutlier* [157], *orddom* [151], *psych* [158], *usdm* [159].

6.6. Analysis of Qualitative Data

In addition to the controlled experiment, we invited the participants of the first experiment run to share their thoughts with regards to the following two tasks:

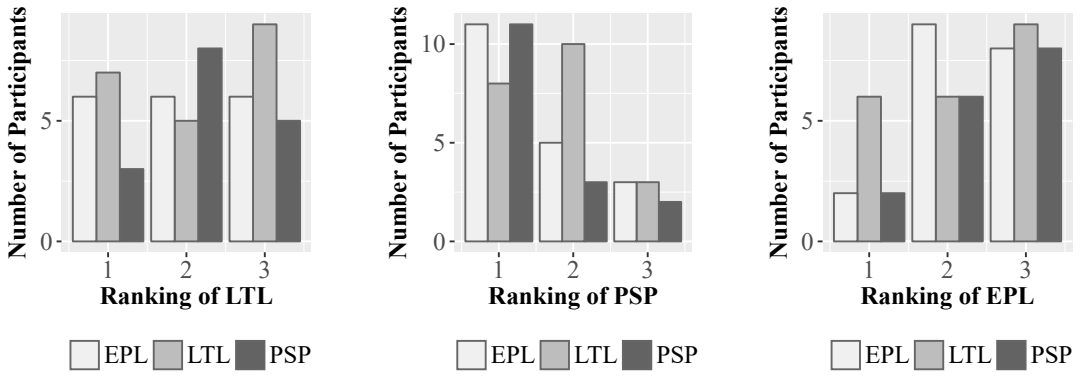
1. *"Please rank the languages according to your preference and state reasons for this ranking."*, and
2. *"Please discuss for which sort of users each language is (not) appropriate and why."*.

The purpose of this survey was to assess the participants' (subjective) preference towards a specific behavioral constraint representation. By that, we tried to gain insights into the users' acceptance of the tested behavioral constraint representations. Please note that we intentionally did not replicate this survey in the second experiment run, because we wanted to avoid the (necessary for this survey) cross-contamination of treatments to improve the validity of the controlled experiment in the second run. Our analysis of the textual answers of the participants has been inspired by the summative content analysis approach [160]. Since the majority of answers given by the participants is very short and in note form, running a full-blown summative content analysis, which usually focuses on journal manuscripts or specific content in textbooks, is impossible. Nevertheless, it is possible to use the core idea of the technique, namely the counting of occurrences of identified keywords and the interpretation of the context associated with the use of the word or phrase. In the following, we present the results of this analysis.

Figure 6.16 shows the personal preference ranking of the tested behavioral constraint representations per group. While the LTL ranking does not show any clear trends, the ranking of the PSP representations indicates a trend towards the first place and the EPL representation towards the third place.

Figure 6.17 (a) shows a bar chart that contains the number of users that are positive or negative towards a specific behavioral constraint representation. In Figure 6.17 (b), the number of mentions of user groups for which a specific behavioral constraint representation is considered

[9] `https://www.r-project.org/`

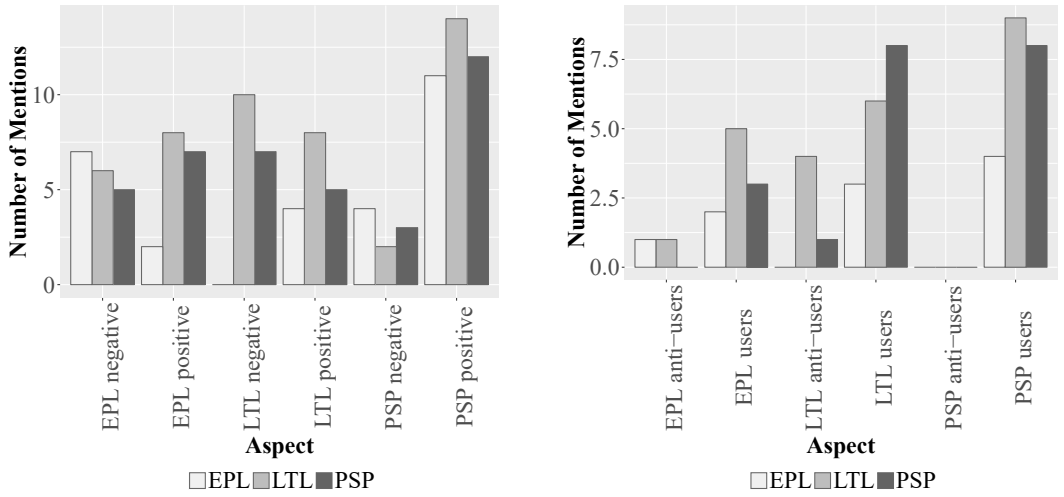


(a) Personal preference ranking of LTL

(b) Personal preference ranking of PSP

(c) Personal preference ranking of EPL

Figure 6.16.: Bar charts of the participants' personal preference ranking of the three approaches in the first experiment run



(a) Positive and negative aspects

(b) Potential users and anti-users

Figure 6.17.: Bar chars of the number of participants mentioning specific aspects of the behavioral constraint representations per group in the first experiment run

to be well-suited (i.e., users) or rather problematic (i.e., anti-users) is shown. In all three groups, positive mentions of PSP are dominant. Moreover, the count of mentioned PSP users is overall higher than for the other representations. There has not been a single mention of a user group that should or could not use PSP.

A detailed summary of the positive and negative aspects mentioned, and of users and anti-users, is shown in Table 6.9 and Table 6.10, respectively. All groups mentioned in the same extent and relatively often (8 times) that PSP is easy to understand. Also the temporal scopes (e.g., After ... until ...) that are present in PSP were mentioned positively. Interestingly, participants other than those of the EPL group relatively often considered EPL as clear and easy to use, while EPL participants apparently did not. Also the separation of concerns in EPL (i.e., through several temporal queries that contain the truth value state change as well) was considered to be a positive aspect of EPL by the LTL group (7 mentions) and the PSP group (2 mentions). Some user comments contradict the comments of other users. For example, one EPL participant stated that EPL is for advanced users while another stated that it is suitable for novice users. Neither participant mentioned any potential anti-user of the PSP representation.

6.7. Discussion

6.7.1. Evaluation of Results and Implications

Most results of this study are in accordance with the initial expectations of this study, but there are some deviations that must be further discussed. In the first experiment run, $H_{0,3}$ cannot be rejected for the response time variable. We suspect that this effect could be related to the experimental tasks of the first experiment run that offered answer choices with truth value states that are not part of the EPL behavioral constraint specification. Apparently, these answer choices caused confusion that resulted in longer response times. To avoid potential bias, the answer choices in the second experiment run included only truth value states that are mentioned in the EPL behavioral constraint specification. In the second experiment run, $H_{0,2}$ cannot be rejected for the response time variable. In this case, we could not find any plausible interpretation other than the sample size of ASE students in the second experiment run. With 50 participants, the sample size is borderline, and we cannot rule out disturbing effects. Nevertheless, aside from that, the statistical inference shows significant results with medium to large effect size magnitudes. Consequently, the controlled experiment runs of this study clearly indicate that

- PSP specifications provide a higher level of understandability than LTL specifications,
- PSP specifications provide a higher level of understandability than EPL specifications, and

Table 6.9.: Summary of mentioned positive and negative aspects per group with number of occurrences (if aggregated)

Aspect	LTL Group	PSP Group	EPL Group
LTL positive	can handle all cases, can express everything, powerful logic, most expressive, fleshed-out, many operators, most robust, clear, formal, easy	very/most powerful (2), easy (2), better syntax than PSP, operators for detailed formulas, clear	readability (2), less complicated, easy, clear
LTL negative	hard to read and/or understand (3), nesting (3), complex (2), confusing (2), long formulas are hard to understand	operators are hard to understand, nesting, long formulas are hard to understand, unintuitive, complicated, difficult, complex, hard to comprehend	-
PSP positive	easy (8), scopes (6), clear (2), intuitive (2), self-explaining, very powerful, very logical, high-level	easy (8), scopes (5), intuitive (2), mapping to natural language, common sense, sound set of operators, compact	easy (8), scopes (4), least keywords, most understandable, precise, most readable, close to natural language
PSP negative	complex, operators are hard to understand	insufficient preparation material, scopes hard to understand, confusing, no clear understanding to which state it changes	hard to understand (2), complex keywords, complicated
EPL positive	queries/states (7), clear (6), easy (3)	easy (5), queries/states (2), clear, operators	queries/states (2)
EPL negative	complicated due to multiple queries (2), too many operators, operator precedence, too few operators, event-based character, less powerful	difficult (2), too simple, too complicated tasks, sometimes confusing, complex	complicated (3), demanding (2), poor readability, poor logic, complex

Table 6.10.: Summary of mentioned user and anti-user aspects per group with number of occurrences (if aggregated)

Aspect	LTL Group	PSP Group	EPL Group
LTL users	users with basic knowledge, software engineers, expert programmers, developers, programming background, programmers / mathematicians, physicists with prior logic background	experts with many years of experience / experienced users (3), software developers (2), modeling user, users performing model checking, all users including programmers and admins, enduser	mainstream users, experienced users, bank employees after training
LTL anti-users	users with minimal programming experience, all, endusers, economists / simple users without without logical background	project managers	-
PSP users	users with minimal to no experience, diverse users, software architects, in executive presentations, people working with large systems, endusers, small or no programming background, business users, high-level programmer	novice users / beginners (2), programmers (2), modeling user, all kind of users (application users, developers, testing experts), workflow designers, enduser, high-level language for architects	specialized users, unexperienced users, IT-affine people, untrained
PSP anti-users	-	-	-
EPL users	not highly trained staff / novice users (2), software engineers, endusers, database users	interface language between modeling users and programmers, advanced users/modeling users, users experienced with CEP	advanced users, novice users
EPL anti-users	users with minimal programming experience	-	general users

- EPL specifications provide a higher level of understandability than LTL specifications.

When it comes to the personal preference of the participants (cf. Section 6.6), PSP seems to be the most preferred behavioral constraint representation. This result is in accordance with the outcome of the controlled experiment runs as well. In contrast, the personal preference ranking of the EPL and LTL representations does not seem to match the results of the controlled experiment runs, since the EPL representation seems to be less popular among the participants than LTL. However, the survey on which the ranking is based must be interpreted with caution, because the sample size might not be large enough to draw valid conclusions on the basis of the data. Please note that we did not replicate the survey in the second experiment run intentionally, so as to improve the validity of the controlled experiment (cf. Section 6.8). Moreover, the constructs "personal preference" and "understandability" might be inherently different and not comparable. In either case, this peculiarity is important to report, and it might be a possible cornerstone for further investigations in future empirical studies.

Both in terms of understandability and the personal preference of the participants, the PSP representation outperformed the other two approaches examined. The pattern-based, high-level nature of the approach seems to make it highly appealing as a behavioral constraint representation. However, a major limitation of the approach is its inflexibility in the case where the set of available patterns does not fit the purpose. In such a case, the pattern set must be extended, i.e., the creation of underlying low-level behavioral constraint representations is required. Both EPL and LTL are more low-level behavioral constraint representations that can be used either as underlying behavioral constraint representations for PSP, or to directly create behavioral constraint specifications for automated verification. EPL supports runtime monitoring, whereas LTL can be used for both runtime monitoring by non-deterministic finite automata (cf. De Giacomo et al. [120]), and design time verification by model checking (cf. Cimatti et al. [35], Blom et al. [118], Holzmann [119]). If a behavioral constraint representation is solely used for runtime monitoring, the study would—based on the measured understandability—imply a preference for EPL over LTL. Another scenario is conceivable as well: During the creation of new PSP patterns, easier to understand EPL behavioral constraint specifications can be used as plausibility specifications for harder to understand LTL formulas to countercheck whether a created LTL formula contains errors (cf. Chapter 5). However, an obstacle could be the possibly low user acceptance of EPL (cf. Section 6.6), which must be further investigated.

6.8. Threats to Validity

6.8.1. Threats to Internal Validity

The internal validity is concerned with the causal relationship of independent variables and dependent variables. Threats to internal validity are unknown or unobserved variables that might have an influence on the outcome of the experiment. Diverse threats to internal validity must be addressed:

- *History effects* refer to events that occur in the environment and change the conditions of a study. The short duration of the study limits the possibility of changes in environmental conditions. Actually, we are not aware of any history effects during the study, but we cannot entirely rule out any such effect, prior to the study taking place. However, in such a case, it would be extremely unlikely that the scores of one group are more affected than another because of the random allocation of participants to groups.

- *Maturation effects* refer to the impact that time has on an individual. Since the duration of the experiment was very short (max. 90 minutes), maturation effects are considered to be of minor importance.

- *Testing effects* comprise learning effects and experimental fatigue. *Learning effects* were avoided by dropping results of the second run in case of a prior participation in the first run. That is, each person was only tested once. *Experimental fatigue* is concerned with occurrences during the experiment that exhaust the participant either physically or mentally. Neither did we observe any signs of fatigue nor did any participant report such.

- *Instrumental bias* occurs if the measuring instrument (i.e., a physical measuring device or the actions/assessment of the researcher) changes over time during the experiment. We avoided such effects by using an experimental design that enables an automated and standardized evaluation of the test results.

- *Selection bias* is present if the experimental groups are unequal before the start of the experiment (e.g., severe differences in relevant experience, age, or gender). Usually, selection bias is likely to be more threatening in quasi-experimental research. By using an experimental research design with the fundamental requirement to randomly assignment participants to the different groups of the experiment, we can avoid selection bias to a large extent. In addition, our investigation of the composition of the groups did not indicate any major differences between them.

- *Experimental mortality* is only likely to occur if the experiment lasts for a long time because the chances for dropouts increase (e.g., location change). Consequently, it has not been a problem in our study at all.

- *Diffusion of treatments* occurs if a group of the experiment is contaminated in some way. By design, in the first run of the experiment, making information about all three behavioral constraint languages available to every participant was necessary for the survey. That is, we accepted the risk of cross-contamination intentionally in the first experiment run. In the second experiment run, the survey was not replicated to avoid the diffusion of treatments, and the preparation material was distributed on a per treatment basis. Since the participants participate in the same social group, and are interacting outside the research process as well, we cannot entirely rule out a cross-contamination between the groups.

- *Compensatory rivalry* is present if participants of a group put in extra effort when they have the impression that the treatment of another group might lead to better results than their own treatment. For example, participants of the LTL group might be aware that their assigned behavioral constraint language is more difficult than PSP. We tried to mitigate the risk of compensatory rivalry by communicating that while there might be differences in difficulty, that would be considered in the grading process.

- *Demoralization* could occur if a participant is assigned to a specific group that she/he does not want to be part of. We did not observe any signs of demoralization such as increased dropout rates or complaints regarding group allocation.

- *Experimenter bias* refers to undesired effects on the dependent variables that are unintentionally introduced by the researcher. We tried to avoid such effects by designing the experiment in a way that limits any such chances. In particular, all participants worked on the same set of tasks (only the behavioral constraint representation differs), and the results of the controlled experiment runs were processed automatically. The tasks used in the experiment were randomly generated, but there were similarities between the behavioral constraints used in some of the experiment tasks and those used in the examples discussed in the learning material. Such similarities might facilitate solving of related experiment tasks. To investigate this threat, we identified tasks that have similarities with the provided learning examples (cf. Table 6.11, Table 6.12, and Table 6.13). If there was a bias, such tasks should show a central tendency towards a relative high level of correctness while the remaining tasks should show a central tendency towards a relative low level of correctness. According to the acquired data, behavioral constraints with more predicates appear to be more difficult than those with less predicates, so we decided to normalize the measured

Table 6.11.: Evaluation of the impact of similarities between experiment tasks and training examples on correctness in the first experiment run (* indicates similarity)

Pattern	# Predicates	PSP	EPL	LTL
Absence_AfterUntil	3	63.89	38.89 *	31.94
Absence_Between	3	78.33	71.67	52.50 *
Existence_AfterUntil	3	52.08	61.46 *	34.38 *
Existence_Between	3	81.25	54.17 *	39.58
Precedence_After	3	63.33	65.00 *	25.83
Response_After	3	52.78 *	50.00	54.17 *
Precedence_AfterUntil	4	75.00	40.83	36.67
Precedence_Between	4	77.08 *	45.83	16.67 *
Response_AfterUntil	4	40.00 *	36.67	25.00
Response_Between	4	56.67 *	42.50	20.83 *
Tasks similar to examples mean (not normalized)		56.63	54.88	35.71
Remaining tasks mean (not normalized)		68.98	47.92	31.80
Tasks similar to examples mean (normalized)		53.33	41.16	28.66
Remaining tasks mean (normalized)		54.86	42.85	26.94

correctness by the formula $correctness \times number_of_predicates \ / \ max_predicates$ to enable a fair comparison between all tasks. We could not find any indication of bias introduced by those similarities in the gathered data. In particular, the number of possibly affected experiment tasks was almost balanced between the groups, and the measured correctness of possibly affected tasks was overall similar to those of the remaining tasks (cf. Table 6.11, Table 6.12, and Table 6.13). All approaches are presented by the same educational methods at a comparable level of detail to not introduce unnecessary bias into the experiment. A different choice of training material (e.g., formal semantics of LTL or the use of Structured English Grammar [107]) could have impacted the results. Also the design decision of using four instead of two truth value states (for a more fine-grained analysis of the understandability of a specification) might have had an impact on the results. Since however all groups had to cope with four runtime states, neither group was disadvantaged.

6.8.2. Threats to External Validity

The external validity is concerned with the generalizability of the results of our study. In the following, we discuss potential threats that hinder a generalization. There exist different types of generalizations that must be considered:

Table 6.12.: Evaluation of the impact of similarities between experiment tasks and training examples on correctness in the second experiment run - DSE (* indicates similarity)

Pattern	**# Predicates**	**PSP**	**EPL**	**LTL**
Absence_AfterUntil	3	69.44	64.29 *	40.32
Absence_Between	3	87.65	54.76	25.81 *
Existence_Between	3	77.78	63.10 *	46.24
Precedence_After	3	78.70	61.61 *	24.19
Response_After	3	74.81 *	60.71	54.84 *
Precedence_AfterUntil	4	67.59	50.89	32.26
Precedence_Between	4	45.68 *	54.76	19.35 *
Response_AfterUntil	4	56.30 *	45.00	18.71
Response_Between	4	77.04 *	29.29	30.32 *
Tasks similar to examples mean (not normalized)		63.46	63.00	32.58
Remaining tasks mean (not normalized)		76.23	49.24	32.34
Tasks similar to examples mean (normalized)		58.78	47.25	27.54
Remaining tasks mean (normalized)		60.55	44.42	26.81

Table 6.13.: Evaluation of the impact of similarities between experiment tasks and training examples on correctness in the second experiment run - ASE (* indicates similarity)

Pattern	**# Predicates**	**PSP**	**EPL**	**LTL**
Absence_AfterUntil	3	61.76	63.24 *	40.63
Absence_Between	3	88.24	47.06	25.00 *
Existence_Between	3	88.24	58.82 *	54.17
Precedence_After	3	80.88	61.76 *	43.75
Response_After	3	81.18 *	60.00	47.50 *
Precedence_AfterUntil	4	72.06	47.06	46.88
Precedence_Between	4	50.98 *	52.94	18.75 *
Response_AfterUntil	4	51.76 *	49.41	16.25
Response_Between	4	76.47 *	49.41	35.00 *
Tasks similar to examples mean (not normalized)		65.10	61.27	31.56
Remaining tasks mean (not normalized)		78.24	50.98	40.34
Tasks similar to examples mean (normalized)		60.02	45.96	27.03
Remaining tasks mean (normalized)		62.28	46.52	33.41

- *Generalizations across populations*: Through statistical inference, we try to make generalizations from the sample to the immediate population. The study considers two populations, namely computer science students that enrolled in the DSE course as proxies for novice to moderately advanced software architects, designers or developers, as well as computer science students that enrolled in the ASE course as proxies for moderately advanced software architects, designers or developers. The results of our study show similar results for both populations, but it is unclear to what extent these results are generalizable to different or broader populations. Therefore, we do not intend to claim generalizability without further empirical evidence. For example, it might be plausible that people working in the software industry with many years of experience or business administrators perform similarly, but the present study can neither support nor reject such claims.

- *Generalizations across treatments*: Since the treatments are equivalent to specific behavioral constraint representations, treatment variations are inherently impossible.

- *Generalizations across settings/contexts*: The participants of this study are students who enrolled in computer science courses at the University of Vienna, Austria. Apparently, a majority of the students are Austrian citizens, but there is a large presence of foreign students as well. Surely, it would be interesting to repeat the experiment in different settings/context to evaluate the generalizability in that regard. For example, the majority of the participants are non-native English speakers, which could be an obstacle for understanding the preparation material or task descriptions, so repeating the experiment with native speakers might lead to different (presumably better) results.

- *Generalizations across time*: We performed the experiment at two points in time (one year apart) with similar results. Master students in particular are a rather heterogeneous group as they often come from other countries and faculties. This heterogeneity might explain the differences in previous experience with formal logic of master students in ASE between the first and second experiment run. Students in DSE appear to be more homogeneous, maybe because receiving training in formal logic is part of the bachelor program in computer science at the University of Vienna. In general, it is hard to predict whether the results of this study hold over time. For example, if teaching of LTL or EPL is intensified, then the students would bring in more LTL-related or EPL-related expertise, which likely has an impact on the results of the controlled experiment.

6.8.3. Threats to Construct Validity

There are potential threats to the validity of the construct that must be discussed:

- *Inexact definition & Construct confounding*: This study considers the construct *understandability* that is measured by the variables *correctness* and *response time*. To our best knowledge, this construct is exact and adequate. Several existing studies that evaluate different representations (e.g., domain specific languages) use this construct and its variables (cf. Feigenspan et al. [143] and Hoisl et al. [144]).

- *Mono-operation bias*: In this study, the independent variable is the behavioral constraint language. Currently, we do not differentiate this construct any further. For example, the tasks of the experiment are based on a representative set of behavioral constraint patterns with different numbers of propositional variables, but we do not perform further investigations on the basis of the number of propositional variables. Such finer-grained analyses are tempting, but a much larger number of tasks and/or answer choices would be necessary in order to be able to perform meaningful statistical analyses, and increasing the number of tasks and/or answer choices would likely result in experimental fatigue due to prolonged experiment sessions.

- *Mono-method bias*: To measure the correctness of answers, the evaluation by an automated method appears to be the most accurate measure as it does not suffer from experimenter bias or instrumental bias. For organizational reasons, keeping time records was the personal responsibility of each participant. Certainly, this leaves room for measuring errors, and an alternative measuring method (e.g., video records with timestamps or performing the experiment with an online tool that handles record keeping) would reduce the threat to construct validity. Participants who made obvious errors in their time records were not considered in this study (cf. Section 6.4.1).

- *Reducing levels of measurements*: Both the correctness and response time are continuous variables. That is, the levels of measurements are not reduced.

- *Treatment-sensitive factorial structure*: In some empirical studies it might be the case that a treatment sensitizes the participant into developing a different view on the construct (e.g., differentiation between different types of stress). Since we did not ask questions regarding the subjective level of understandability of behavioral constraint specifications in the controlled experiment runs, but tried to measure the actual level of understandability objectively, this threat is considered to be irrelevant.

  The survey questions asked in addition to the controlled experiment in the first experiment run are concerned with subjective preference rankings and subjective thoughts on practical applicability of the behavioral constraint languages (or the lack of the same), so they are neither meant nor used to measure the understandability construct in this study.

6.8.4. Threats to Content Validity

Content validity is concerned with the relevance and representativeness of the elements of a study for the construct that is measured:

- *Relevance*: All tasks are based on the Property Specification Patterns (cf. Dwyer et al. [27]), which is a set of commonly occurring behavioral constraint patterns. Thus, we claim that the contents of the experiment are highly relevant for measuring the understandability of behavioral constraint representations. However, using the patterns as basis for our tasks might be a threat to validity for measuring the understandability of LTL and EPL, because the expressiveness of these approaches goes far beyond the pattern-based approach, which is limited to a set of patterns. In that regard, for future work, it would be interesting to design an experiment that focuses on LTL and EPL with tasks that are not based on patterns. In the context of the presented study, it was necessary to base the tasks on patterns, otherwise it would not have been possible to include PSP in the study.

- *Representativeness*: A representative subset of existing Property Specification Patterns was used for the tasks of the experiment. To reduce chances for experimental fatigue, we did not include all of the available patterns, but we selected the most commonly used patterns according to Dwyer et al. [27]. A survey by Bianculli et al. [109] based on 104 scientific case studies reproduced the results of the survey in [27] even 13 years after the original study took place. In particular, the *Response Chain*, *Precedence Chain*, *Constrained Chain* and *Bounded Existence* patterns are omitted, because they are rarely used. The study by Bianculli et al. [109] investigated the PSP used in a set of industrial service-based applications. Interestingly, the patterns found in the requirement specifications of 100 randomly selected service interfaces were more concerned with non-functional requirements like the maximum number of events in a certain time interval within a certain time window, rather than the qualitative order or existence/absence of events. That is, patterns used in practice might be different from those in scientific studies. Please note, however, that the generalizability of these results is rather limited as the results might only apply to service-oriented computing in that specific company.

6.8.5. Threats to Conclusion Validity

Retaining outliers might be a threat to conclusion validity. However, all outliers appear to be valid measurements, so deleting them would pose a threat to conclusion validity as well. We performed a thorough evaluation of the model assumptions of all relevant statistical tests and

selected the test with the greatest statistical power. That course of action is considered to be extremely beneficial to the conclusion validity of this study.

6.9. Related Work

To the best of our knowledge, we are not aware of any existing empirical studies that investigate the differences in understandability of representative behavioral constraint languages in a similar way and depth as the presented study does. However, there exist related empirical studies that evaluate representations of languages/models in software engineering. This section will focus on those studies.

The first study we would like to present in the field of software architecture and engineering is indirectly related to behavioral constraint specifications as it focuses on architecture descriptions in general. Heijstek et al. [29] try to find out whether there are differences in understanding of textual and graphical software architecture descriptions in a controlled experiment with 47 participants. Interestingly, participants who used textual architecture descriptions performed significantly better, which suggests that textual architectural descriptions could be superior to their graphical counterparts.

An eye-tracking experiment carried out by Sharafi et al. [30] with 28 participants investigates the understandability of graphical and textual software requirement models. They observed no statistically significant difference in terms of correctness of the two approaches, but the response times of participants working with the graphical representations were slower.

Czepa et al. [131] compared the understandability of three languages for behavioral software architecture compliance checking, namely the Natural Language Constraint language (NLC), the Cause-Effect Constraint language (CEC), and the Temporal Logic Pattern-based Constraint language (TLC), in a controlled experiment with 190 participants. The NLC language is simply using the English language for software architecture descriptions. CEC is a high-level structured architectural description language that abstracts EPL and enables nesting of cause parts, that observe an event stream for a specific event pattern, and effect parts, that can contain further cause-effect structures and truth value change commands. TLC is a high-level structured architectural description language that abstracts temporal patterns (such as the Property Specification Patterns by Dwyer et al. [27]). Interestingly, the statistical inference of this study suggests that there is no difference in understandability between the tested languages. This could indicate that the high-level abstractions employed bring those structured languages closer to the understandability of unstructured, natural-language architecture descriptions. Moreover, it might suggest that natural language leaves more room for ambiguity, which is detrimental for its understanding. Overall, the understandability of all three approaches is at a high level. However, the results

must be interpreted with caution. Potential limitations of that study are that its tasks are based on common architectural patterns/styles (i.e., a participant possibly recognizes the meaning of a constraint more easily by having knowledge of the related architectural pattern) and the rather small set of involved behavioral constraint patterns (i.e., only very few behavioral constraint patterns were necessary to represent the architecture descriptions). In contrast, the controlled experiment runs presented in this chapter do not focus on software architecture compliance. Instead, we try to be independent from specific areas of application to evaluate the behavioral constraint representations in a more general context. While the software architecture compliance constraints in that study wrap only a very few patterns in high-level structured languages, the empirical study presented in this chapter is based on a larger, representative set of behavioral constraint patterns, and is focuses on the formalisms' core features instead of high-level, domain-specific abstractions of them.

Hoisl et al. [144] conducted a controlled experiment on three notations for defining scenario based model tests with 20 participants. In particular, they tested a semi-structured natural-language scenario notation, a diagrammatic scenario notation, and a fully-structured textual scenario notation. The authors conclude that the semi-structured natural-language scenario notation is recommended for scenario-based model tests, because the participants of this group were able to solve the given tasks faster and more correctly. However, the validity of the experiment is strongly limited by the small sample size and the lack of statistical hypothesis testing.

6.10. Conclusion and Future Work

6.10.1. Summary

This chapter reports two controlled experiments on the understandability of behavioral constraint representations with 216 participants in total (70 in the first run and 146 in the second run). The results of the statistical evaluation suggest that PSP-based behavioral constraint specifications are significantly easier to understand than EPL behavioral constraint specifications, that are based on Complex Event Processing (CEP), and LTL (Linear Temporal Logic) behavioral constraint specifications. Moreover, the results imply that EPL behavioral constraint specifications are significantly easier to understand than LTL behavioral constraint specifications. Despite the threats to validity listed in Section 6.8, we consider the validity of our results as high because of the repetition and replication by a second experiment run with two different populations, the overall large sample size, the automated generation of the tasks, the automated evaluation of the given answers, and the thorough statistical evaluation.

6.10.2. Impact

This study seems to support the original assumption that the pattern-based PSP approach is the most user-friendly behavioral constraint representation for novice and moderately advanced users. Therefore, if possible (i.e., if the approach is applicable to the domain), the results suggest that the pattern-based behavioral constraint approach should be preferred. Since many existing approaches (e.g., the Compliance Request Language CRL by Elgammal et al. [23] and the PROPOLS approach for the verification of BPEL service composition schemes by Yu et al. [66]) reuse PSP or extend the original pattern catalog (cf. Dywer et al. [27]) with more specific context-dependent patterns, there is strong evidence that the results of the study hold for these approaches as well. However, in contrast to the two other behavioral constraint approaches tested in this study, the pattern-based approach is the most limited one in terms of its expressiveness. That is, if the set of supported patterns is incompatible with a specific requirement (e.g., a company internal policy that must be covered by the IT system), it is necessary to extend the pattern catalog. Since the pattern-based approach merely abstracts other behavioral constraint representations (most often LTL formulas), creating new patterns always requires the creation of the underlying behavioral constraint specifications as well. Creating those underlying behavioral constraint specifications is considered to be difficult and error-prone. Plausibility checking (cf. Chapter 5) tries to alleviate the risk to create incorrect LTL specifications by leveraging EPL specifications to countercheck if the LTL formula contains errors. Since EPL behavioral constraint specifications are more understandable than LTL formulas, the results of the presented study can be seen as an empirical evaluation of the plausibility checking approach as well.

6.10.3. Future Work

The present study focuses on the understandability of already given behavioral constraints. That is, the authoring of behavioral constraint specifications is not yet sufficiently covered. It is possible to further investigate the understandability of behavioral constraint languages by running different kinds of experiments. In particular, we plan to study the understandability of behavioral constraint representations during the authoring process as well. We suspect that creating correct behavioral constraint specifications from scratch is more difficult than interpreting already given behavioral constraint specifications correctly. Moreover, we are curious whether the measured significant differences in understandability between the three behavioral constraint representations are also present during the creation process of behavioral constraint specifications. Another interesting opportunity for future work is studying the understandability of behavioral constraint specifications with professionals working in the industry (e.g., senior system administrators and senior software architects). Studying whether there exist differences in understandability be-

tween textual and graphical behavioral constraint representation is another interesting opportunity for future work. In particular, it would be interesting to find out whether the results of the studies by Heijstek et al. [29] and Sharafi et al. [30], that investigated the differences in understandability of textual and graphical models in the software architecture and engineering domain with results in favor of the textual approaches, are transferable to behavioral constraint specifications. In this context, it might be interesting as well to compare textual LTL representations against the graphical NFA representations since NFAs are often the transformation product of LTL formulas.

7. Modeling Compliance Specifications in Linear Temporal Logic, Event Processing Language and Property Specification Patterns

In contrast to the controlled experiments reported in the previous chapter, which focused on the understandability with regards to already given behavioral constraints, this chapter investigates the understandability construct with regards to behavioral constraint authoring. We conducted a controlled experiment with 215 participants on the understandability of modeling behavioral constraints in the three representative modeling languages that we used in our technical contributions, namely Linear Temporal Logic (LTL), the Complex Event Processing based Event Processing Language (EPL), and Property Specification Patterns (PSP). The results of our study show that the formalizations in PSP were overall more correct, which indicates that the pattern-based approach provides a higher level of understandability than EPL and LTL. More advanced users, however, seemingly are able to cope equally well with PSP and EPL in modeling behavioral constraints.

7.1. Introduction

Many domains are subject to a vast and ever-growing number of rules and constraints stemming from sources including legislation, regulations, standards, guidelines, contracts, and best practices. One example is compliance in the corporate and financial sector, regulated for example by the Sarbanes-Oxley Act of 2002 (SOX) [11], a federal law composed in reaction to major corporate accounting scandals in the U.S. (e.g., Enron and WorldCom), or the Basel III [12] framework, established in response to weaknesses in financial regulation responsible for the financial crisis in 2007/2008. Another example of heavily regulated domains is the construction industry. Compliance rules in this domain are often related to occupational safety and health hazards. For example, certain precautions and safe practices are required if lead contamination is present (or to be presumed present) in buildings built before 1978 that undergo renovation

(cf. United States Environmental Protection Agency's Lead-Based Paint Renovation, Repair and Painting Rule [13]). A third example is the healthcare sector, where processes in hospitals must comply with state-of-the-art medical knowledge and treatment procedures (e.g., Rovani et al. [14]).

From cooperations with industry partners (e.g., [79]), their customers, and other company representatives at conferences and workshops, we were able to gain valuable insights into the current handling on compliance rules in practice. Most often, compliance documents are transformed into internal policies first. They are often written in natural language, but there is also a shift towards structured approaches, such as the Semantics of Business Vocabulary and Business Rules (SBVR) standard [24]. Later these internal policies are considered in business process models (e.g., BPMN [20]) or other behavioral models (e.g., UML activity diagrams), and/or they are hard-coded in a programming language. That often leads to consistency problems and to a poor maintainability and traceability between compliance specifications, internal policies, models, and the final source code. This is especially the case when compliance specifications change frequently. Additionally, practitioners report that it often takes a long time until new compliance specifications are actually supported by their software. Often the compliance rule has long become obsolete before the implementation is ready (cf. [49], [161]). Consequently, the industry shows a strong interest in approaches that are applicable in practice. Such approaches should support a comprehensible, fast and accurate adoption of compliance specifications as well as their automated enactment and verification. All modeling languages that we study are well suited for automated computer-aided compliance checking or monitoring. Nonetheless, companies are still often reluctant to expose their customers or employees to such approaches. In discussions with industry partners (cf. [75], [128]), uncertainty regarding how understandable these approaches are, became evident. This uncertainty was stated as one of the major reasons for the reluctance in practical adoption.

7.1.1. Problem Statement

Most existing work on design time verification and runtime monitoring focuses on technical contributions rather than empirical contributions. From the perspective of a potential end user who has to implement compliance specifications, the understandability of an offered formal specification language appears to be of major interest. To the best of our knowledge, there are no empirical studies that investigate and compare the understandability of representative languages with respect to the formal modeling of compliance specifications. In particular, the following representative specification languages are considered in this empirical study:

- *Linear Temporal Logic (LTL)* was proposed in 1977 by Pnueli [26]. LTL is a popular way

for defining compliance rules according to Reichert & Weber [25]. In general, LTL is a widely used specification language commonly applied in model checking (cf. Cimatti et al. [35] for NuSMV[1], Blom et al. [118] for LTSmin[2], Holzmann [119] for SPIN[3]) and runtime monitoring by nondeterministic finite automata (cf. De Giacomo & Vardi [99] and De Giacomo et al. [120]).

- *Event Processing Language (EPL)* is the query language of the open source Complex Event Processing engine *Esper*[4]. EPL is well-suited as a representative for CEP query languages as it supports common CEP query language concepts, such as *leads-to* (*sequence*, *followed-by*) and *every* (*each*) operators, that are present in many CEP query languages and engines (e.g., Siddhi[5] and TESLA [127]). Several existing studies on compliance monitoring make use of EPL (cf. Awad et al. [77], Holmes et al. [95], and Tran et al. [162]).

- *Property Specification Patterns (PSP)* are a collection of recurring temporal patterns proposed by Dwyer et al. [27], [73]. This pattern-based approach abstracts underlying technical and formal languages, most notably LTL and CTL (Computation Tree Logic; cf. Clarke et al. [41]). Numerous existing approaches are based on PSP. Among them are the *Compliance Request Language* proposed by Elgammal et al. [23] and the declarative business process approach *Declare* proposed by Pešić et al. [121].

In previous studies (cf. Chapter 6) the understandability of already existing formal specifications was studied through experiments in these three languages. To further study the understandability of these languages, it is crucial to consider the modeling itself (i.e., authoring or creation of behavioral constraints) as well.

7.1.2. Research Objectives

This empirical study has the research objective of investigating the understandability construct of representative languages with regards to the modeling of compliance specifications. The understandability construct focuses on the degree of correctness achieved and on the time spent on modeling compliance specifications.

The experimental goal, using the relevant template from the Goal Question Metric proposed by Basili et al. [132], is stated as follows:

[1] http://nusmv.fbk.eu/
[2] http://fmt.cs.utwente.nl/tools/ltsmin/
[3] http://spinroot.com/
[4] http://www.espertech.com/esper
[5] https://github.com/wso2/siddhi

Table 7.1.: Questions based on the goal

Identifier	Question
Q1	How understandable are the tested approaches for participants at the bachelor level (attending the Software Engineering 2 Lab course)?
Q2	Are there differences in understandability between the tested approaches for participants at the bachelor level (attending the Software Engineering 2 Lab course)?
Q3	How understandable are the tested approaches for participants at the master level (attending the Advanced Software Engineering Lab course)?
Q4	Are there differences in understandability between the tested approaches for participants at the master level (attending the Software Engineering 2 Lab course)?
Q5	How understandable are the tested approaches for participants with industrial working experience?
Q6	Are there differences in understandability between the tested approaches for participants with industrial working experience?

Analyze LTL, PSP, and EPL
for the purpose of their evaluation
with respect to their understandability related to modeling compliance specifications
from the viewpoint of the novice and moderately advanced software engineer, designer or developer
in the context/environment of the Software Engineering 2 Lab and the Advanced Software Engineering Lab courses at the Faculty of Computer Science of the University of Vienna in the summer term 2017.

Based upon the stated goal, questions concerning *understandability* were generated as shown in Table 7.1.

The *understandability* is measured by three dependent variables, namely the *syntactic correctness* and *semantic correctness* achieved in trying to formally model compliance specifications, as well as the *response time*. Correctness and response time are commonly used to measure the understandability construct, for example in the empirical studies by Feigenspan et al. [143] and Hoisl et al. [144]. The study design enables a more fine-grained analysis of the correctness by differentiating between syntactic and semantic correctness, as suggested by numerous existing studies, such as Ferri et al. [163], Hindawi et al. [164], and Harel & Rumpe [165].

Besides the main research goal, which focuses on understandability, this work also addresses subjective aspects, namely the *perceived ease of application* and the *perceived correctness*, which are measures of self-assessment and not directly related to the understandability construct.

7.1.3. Guidelines

This work follows existing guidelines for conducting, evaluating, and reporting empirical studies (cf. Section 6.1.4).

7.2. Background

For an introduction to the specification languages used in this study, we refer the reader to Section 6.2.

7.3. Experiment Planning

This section describes the outcome of the experiment planning phase, and it provides all information that is required for a replication of the study.

7.3.1. Goals

The primary goal of the experiment is measuring the *understandability* construct of representative languages that are suitable for modeling compliance specifications. This construct is defined by the *syntactic correctness*, *semantic correctness*, and *response time* of the answers given by the participants.

This study differentiates between syntactic and semantic correctness as it enables a more fine-grained analysis. This is in line with Chomsky [166], who stressed that the study of syntax must be independent from the study of semantics. Numerous existing studies differentiate between syntactic and semantic correctness (cf. Ferri et al. [163], Hindawi et al. [164], and Harel & Rumpe [165]).

An LTL formula can be syntactically totally correct without catching the desired meaning. For example, the specification "activity 2 must not happen unless activity 1 has already happened" is not covered at all in a semantic way by the formula "$\mathcal{F}\ activity_1\ \wedge\ \mathcal{F}\ activity_2$" (meaning activity 1 must finally happen and activity 2 must finally happen), which is syntactically correct. In contrast, the formula "$\neg\ activity_2\,\mathcal{U}\ activity_1$" is both syntactically and semantically correct for the chosen example.

In addition to the understandability construct, the experiment aims at studying the *perceived ease of application* of the languages and the *perceived correctness* of the formalized compliance specifications.

7.3.2. Experimental Units

All 215 participants of the experiment are students who enrolled in the courses "Software Engineering 2 Lab (SE2)" and "Advanced Software Engineering Lab (ASE)" in the summer semester 2017 at the Faculty of Computer Science, University of Vienna, Austria. Two kinds of participants can be differentiated:

- 149 participants of the bachelor-level course SE2 are used as proxies for *novice* software engineers, designers or developers.
- 66 participants of the master-level course ASE are used as proxies for *moderately advanced* software engineers, designers or developers.

Using students as proxies for nonexpert users is not an issue according to Kitchenham et al. [133]. Other studies even suggest that students can be used as proxies for experts under certain circumstances (cf. Höst et al. [134], Runeson [135], Svahnberg et al. [136], and Salman et al. [137]). As an incentive for participation and proper preparation, up to 10 bonus points (10% of total course points) were awarded based on the participant's performance in the experiment. All participants were randomly allocated to experiment groups.

7.3.3. Experimental Material & Tasks

In total, the experiment comprised five distinct tasks stemming from three different domains, as shown in Table 7.2. Tasks 1 and 2 are related to compliance in the context of lending, Task 3 focuses on compliance in hospital processes, and Tasks 4 and 5 are based on compliance specifications in the construction industry. Each task was presented to the participants by stating first the context, then the specification, and finally the available elements that are to be used during formal modeling of the specification. For an example of how experimental tasks were presented to the participants see Figure 7.1. The full experimental material is available online (cf. Czepa et al. [167]). For sample solutions of all experimental tasks see Appendix A.1. It is important to note that these sample solutions show just one possible way to model the compliance specifications. In the grading process, each proposed solution was carefully assessed under constant consideration that the sample solution might not be the only way to correctly formalize the specification.

7.3.4. Hypotheses, Parameters, and Variables

PSP abstracts underlying formal representations, such as LTL formulas, by high-level patterns with the intention to facilitate reuse and to improve ease of use. That is, the pattern representations are assumed to provide a better understandability than their underlying LTL formulas.

Table 7.2.: Experimental tasks

Task No.	Context/Source	Compliance Specification in Natural Language	Available Elements for Modeling
1	Request for a loan (cf. Elgammal et al. [23])	The branch office manager has to evaluate the loan risk before signing the contract officially. No one else is allowed to evaluate the loan risk and to sign the contract.	Tasks: • Evaluate Loan Risk • Officially Sign Contract Roles: • Branch Office Manager
2	Request for a loan (cf. Elgammal et al. [23])	The checking of the customer bank privilege is followed by checking of the credit worthiness. Both activities must take place before determining the risk level of the loan application.	Tasks: • Check Customer Privilege • Check Credit Worthiness • Evaluate Loan Risk
3	Medical treatment and surgery of malignant gastric diseases (cf. Rovani et al. [14])	The preoperative screening is performed before any surgical treatment in order to assess whether the patient's conditions are good enough for the surgery to be performed and to estimate potential risks. As far as the surgical technique is concerned, the gastric resection for malignant diseases can be performed by using either a laparoscopic surgery or a traditional open approach, but not both. Furthermore, in both cases a nursing period is needed to monitor the patient after the operation.	Tasks: • Preoperative Screening • Laparoscopic Gastrectomy • Open Gastrectomy • Nursing
4	Renovation work and lead-based paint (cf. United States Environmental Protection Agency [13])	Once a lead contamination has been identified, a certified renovator must be present all time while any cleaning activity is performed until the end of the renovation work.	Tasks: • Renovation • Cleaning • Presence of Certified Renovator Events: • Lead Contamination identified
5	Renovation work and lead-based paint (cf. United States Environmental Protection Agency [13])	Contractors, property managers, and others who perform renovations for compensation in residential houses, apartments, and child-occupied facilities built before 1978 are required to distribute a lead pamphlet before starting renovation work.	Tasks: • Renovation • Distribute Lead Pamphlet • Classify Building • Enter Building Date Data: • Year of construction • Type of building

Task 2

Use your constraint language to describe the requirement below. It might be necessary to use multiple constraints to represent the requirement. Just write "C1:" to start you first constraint, "C2:" for the second, and so on. Use the given letters (e.g., p for Check Customer Privilege) to refer to a task in your constraint(s). Please always keep records of the time when working on this task, and don't forget to answer the two questions below at the completion of this task.

Start Time					
End Time					
Duration					

Total Duration	

Context:

Request for a loan (Kreditantrag)

Requirement:

"The checking of the customer bank privilege is followed by checking of the credit worthiness. Both activities must take place before determining the risk level of the loan application."

Tasks:

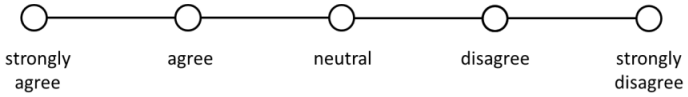
p = Check Customer Privilege

w = Check Credit Worthiness

e = Evaluate Loan Risk

Please fill out the survey at the completion of this task:

1. I think that my transformation of the requirement to the constraint language is correct.



2. It has been easy for me to create the constraint(s) for the requirement.

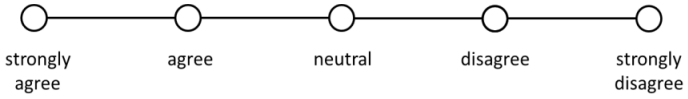


Figure 7.1.: Sample task as presented to the participants

EPL-based constraints are composed of an initial truth value and one or more query-listener pairs that change the truth value state. In contrast to LTL, where meaning is encoded in a formula, in EPL-based constraints different concerns, namely defining the initial truth value and change criteria for the truth value, are separated from each other. This separation of concerns is assumed to facilitate the understandability of EPL-based constraints as opposed to LTL formulas where this separation is not present.

Consequently, we hypothesized that PSP, as a highly abstract pattern language, is easier to understand than LTL and EPL, and that EPL, due to separation of concerns, is easier to understand than LTL. Consequently, the following hypotheses for the controlled experiment were formulated:

- $H_{0,1}$: There is no difference in terms of *understandability* between PSP and LTL.
- $H_{A,1}$: PSP has a higher level of *understandability* than LTL.

- $H_{0,2}$: There is no difference in terms of *understandability* between PSP and EPL.
- $H_{A,2}$: PSP has a higher level of *understandability* than EPL.

- $H_{0,3}$: There is no difference in terms of *understandability* between EPL and LTL.
- $H_{A,3}$: EPL has a higher level of *understandability* than LTL.

The *understandability* construct is measured by three interval-scaled dependent variables, namely:

- the *syntactic correctness* achieved in trying to formally model the compliance specifications,
- the *semantic correctness* achieved in trying to formally model the compliance specifications,
- the *response time*, which is the time it took to complete the experimental tasks.

In addition, there are hypotheses that are concerned with the participants' opinion on the languages under investigation, namely:

- $H_{0,4}$: There is no difference in terms of *perceived correctness* between PSP and LTL.

- $H_{A,4}$: PSP has a higher level of *perceived correctness* than LTL.

- $H_{0,5}$: There is no difference in terms of *perceived correctness* between PSP and EPL.
- $H_{A,5}$: PSP has a higher level of *perceived correctness* than EPL.

- $H_{0,6}$: There is no difference in terms of *perceived correctness* between EPL and LTL.
- $H_{A,6}$: EPL has a higher level of *perceived correctness* than LTL.

- $H_{0,7}$: There is no difference in terms of *perceived ease of application* between PSP and LTL.
- $H_{A,7}$: PSP has a higher level of *perceived ease of application* than LTL.

- $H_{0,8}$: There is no difference in terms of *perceived ease of application* between PSP and EPL.
- $H_{A,8}$: PSP has a higher level of *perceived ease of application* than EPL.

- $H_{0,9}$: There is no difference in terms of *perceived ease of application* between EPL and LTL.
- $H_{A,9}$: EPL has a higher level of *perceived ease of application* than LTL.

The dependent variables associated with these hypotheses are ordinal and scaled since the data were gathered by agree-disagree scales. In accordance with the results of a study by Revilla et al. [168], each scale had five categories.

7.3.5. Experiment Design & Execution

According to Wohlin et al. [34], "it is important to try to use a simple design and try to make the best possible use of the available subjects". For that reason, a completely randomized experiment design with one alternative per experimental unit was used. That is, each participant is randomly assigned to exactly one experiment group. The assignment procedure was fully automated in an unbiased manner.

Preparation documents were distributed to the participants one week before the experiment run. In these documents, the basics of the approaches are discussed, and the participants were encouraged to prepare for the experiment by applying the assigned behavioral constraint representation before the experiment session. To avoid bias, all three preparation documents were similar in length and depth of detail. The approaches were presented in an easy to understand manner to the participants, as suggested by extant research on teaching undergraduate students in theoretical computer science, formal methods, and logic (cf. Habiballa & Kmeť [145], Knobelsdorf & Frede [146], Carew et al. [147], and Spichkova [148]). The training material provided is available online (cf. Czepa et al. [167]).

7.3.6. Procedure

To ensure a smooth procedure and to avoid unnecessary stress, the preparation document informed the participants about the experiment procedure in as much detail as possible. Seating arrangements were such as to reduce chances for misbehavior, and the participants were instructed on finding a suitable seat. The participants were allowed to use printouts of the preparation material and notes at their own discretion. After a brief discussion of the contents and structure of the experiment document by the experimenters, the participants started trying to solve the experiment tasks. The duration of the experiment was limited to 90 minutes. For organizational reasons, the experiment was done on paper, and time record keeping was the responsibility of each participant (please see Section 7.5.2 for a discussion of this potential threat to validity). After the experiment execution, the given answers were evaluated. For that purpose a method proposed by Lytra et al. [169] was applied, which comprises the independent evaluation of the answers by three experts. The attempted formalization in each experiment task was graded independently by the first, second, and third author, who are experts in the investigated languages. To mitigate the risk of grading bias, the participant's given answers were graded in random order by each of the experts. In case of large differences in grading, a discussion took place until a consensus was achieved. Figure 7.2 and Figure 7.3 depict the grading process schematically, respectively from the individual and overall perspective. This evaluation of more than a thousand distinct answers comprising approximately $17,000$ constraints took about half a year, alongside the authors' normal teaching and research responsibilities. All other given answers, which are related to previous knowledge, time records, and agree-disagree scale responses, were digitized and double-checked subsequently.

7.4. Analysis

This section is concerned with the treatment and statistical analysis of the data.

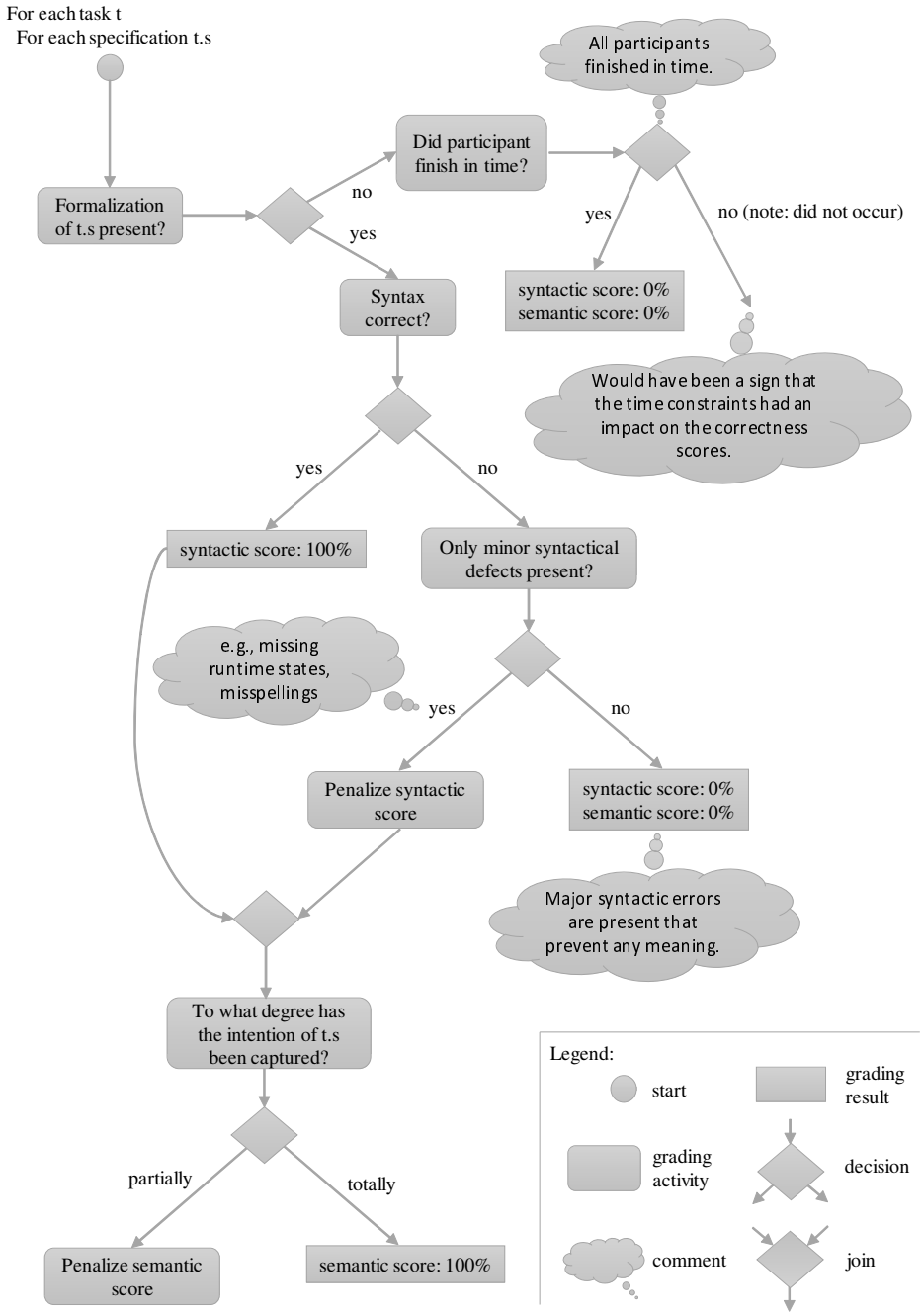


Figure 7.2.: Individual grading procedure

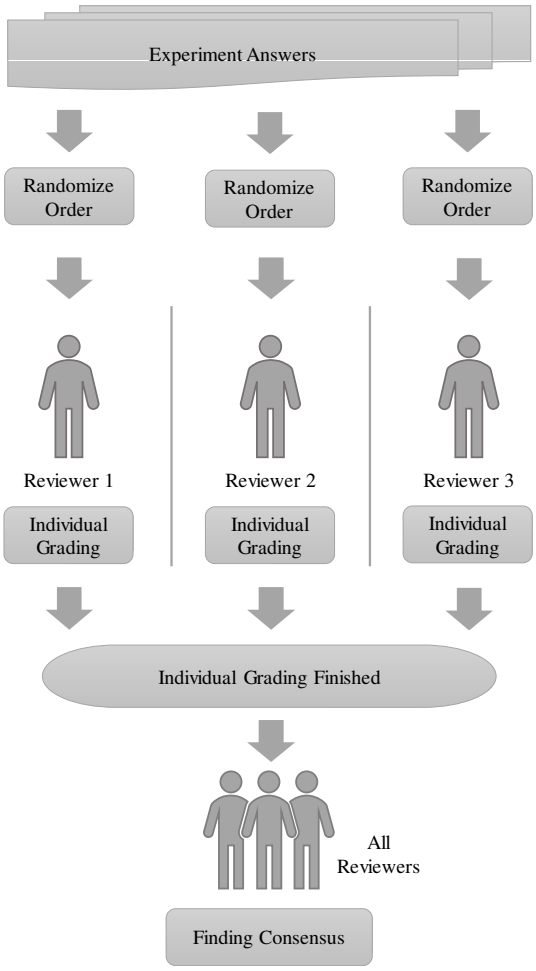


Figure 7.3.: Overall grading procedure

Table 7.3.: Summary of dropped participants

Group	Course	Reason
PSP	SE2	The participant gave up after the first task.
PSP	SE2	The participant did not apply PSP, but used a language/formalism that was not part of the study.
LTL	SE2	The participant was assigned to LTL, but gave answers in PSP.
LTL	SE2	The participant gave positive perceived difficulty and correctness ratings for unsolved tasks.
PSP	ASE	The participant did not apply PSP, but wrote basic Boolean formulas.
PSP	ASE	The participant came unprepared.
PSP	ASE	The participant did not apply PSP, but drew UML activity diagrams.
LTL	ASE	The participant gave up after the first task.

7.4.1. Data Set Preparation

To preserve the integrity of the acquired data, it was necessary to drop potentially unreliable items. In total, the data of eight participants were not considered in the statistical evaluations. Table 7.3 summarizes all dropped participants including the reasons for non-consideration.

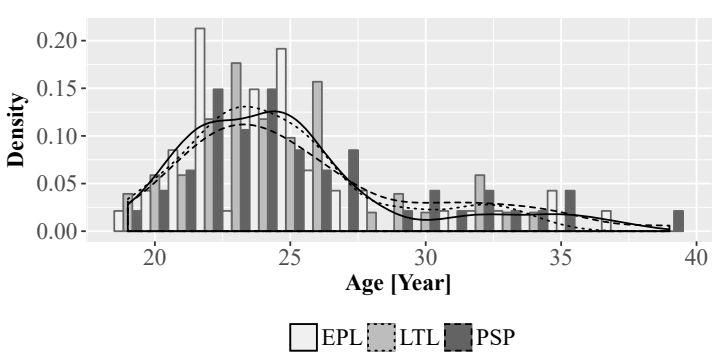
7.4.2. Descriptive Statistics

In this section, we analyze the acquired data (cf. Czepa et al. [167]) with the help of descriptive statistics.

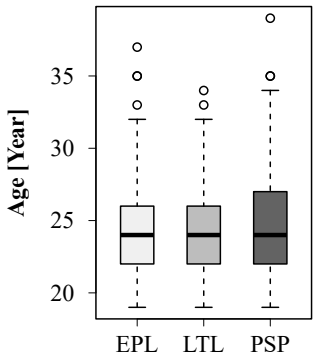
Descriptive Statistics of Previous Knowledge, Experience and Other Features of Participants

For the validity of the study, it is crucial to find out whether the randomized distribution to experiment groups resulted in well-balanced groups. This section provides descriptive statistics of the age, gender, programming experience, complex event processing experience, logical formalisms experience, and industry experience of the participants per experiment group for that purpose.

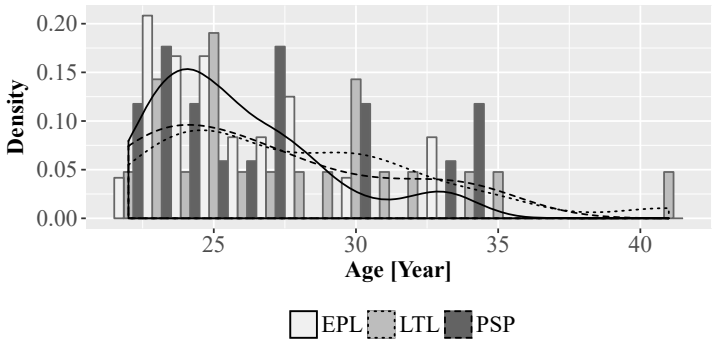
Both the kernel density plot in Figure 7.4 (a) and the box plot in Figure 7.4 (b) show a nearly identical age distribution in all experiment groups of the SE2 course with a central tendency at 24 years. There are few (i.e., two each in LTL and PSP, and three in EPL) outliers, which represent students that are older than the majority of their colleagues. In contrast to the nearly identical age distribution in SE2, there seem to be minor differences in age distribution between the experiment groups of the ASE course. The kernel density plot in Figure 7.4 (c) and the



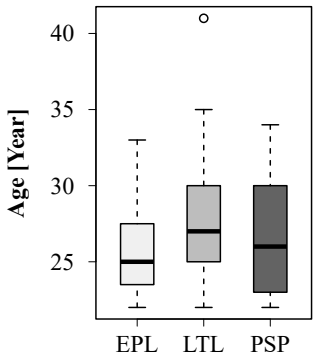
(a) Kernel density plot: SE2



(b) Box plot: SE2

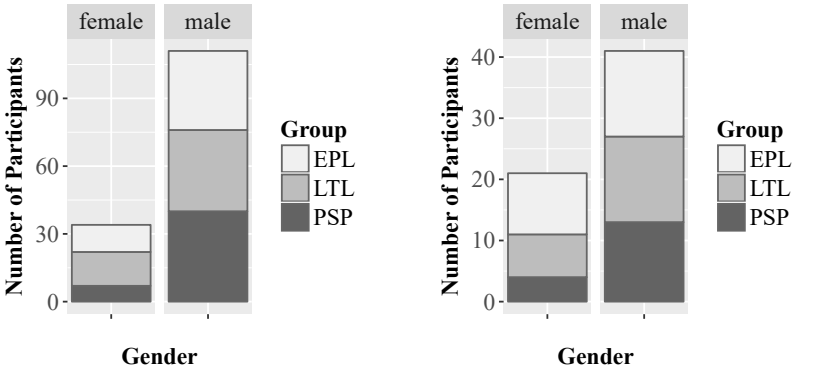


(c) Kernel density plot: ASE



(d) Box plot: ASE

Figure 7.4.: Kernel density plots and box plots of the participants' age per group and course



(a) Software Engineering 2 (bachelor-level course)

(b) Advanced Software Engineering (master-level course)

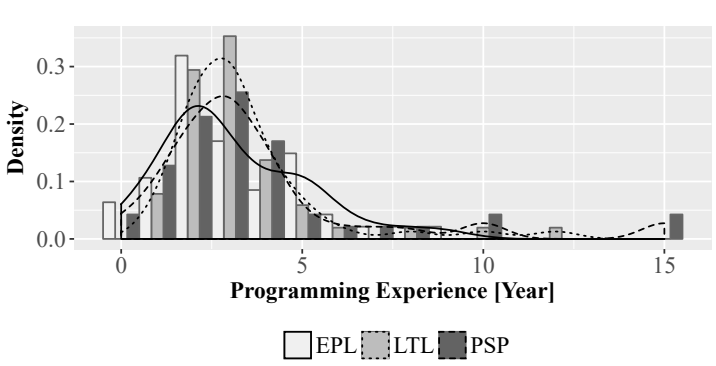
Figure 7.5.: Bar charts of the participants' gender per group and course

box plot in Figure 7.4 (d) indicate that the share of younger participants is larger in the EPL group than in the two remaining experiment groups. Overall, LTL participants are slightly older than participants of the other groups. Moreover, there is a single age outlier in the LTL group representing a student of greater age.

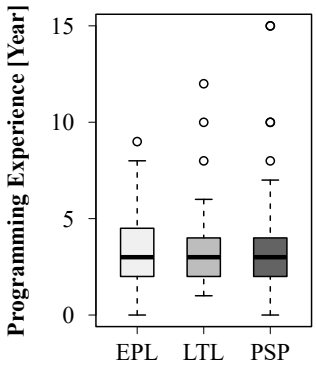
Figure 7.5 shows the gender distribution. With 111 men and 34 women, there are about three times as many male than female participants in SE2. The share of women is larger in ASE with a ratio of about $1:2$ (21 female to 41 male participants). In detail, the gender distribution is as follows:

- Software Engineering 2 (SE2):
  - EPL: 12 female (25.5%) and 35 male participants (74.5%)
  - LTL: 15 female (29.4%) and 36 male participants (70.6%)
  - PSP: 7 female (14.9%) and 40 male participants (85.1%)
- Advanced Software Engineering (ASE):
  - EPL: 10 female (41.7%) and 14 male participants (58.3%)
  - LTL: 7 female (33.3%) and 14 male participants (66.7%)
  - PSP: 4 female (23.5%) and 13 male participants (76.5%)

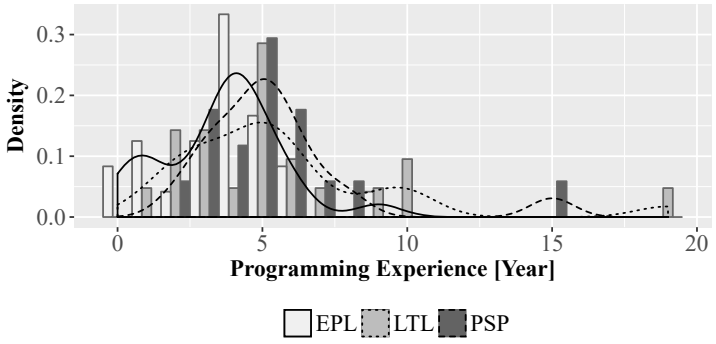
In both courses, the share of female participants is smallest in PSP. There are about twice as many women in the LTL group in SE2 and in the EPL group in ASE as in the corresponding PSP groups, which indicates an imbalance in the distribution of female participants. Since the share



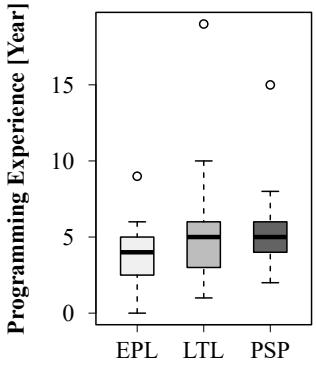
(a) Kernel density plot: SE2



(b) Box plot: SE2



(c) Kernel density plot: ASE

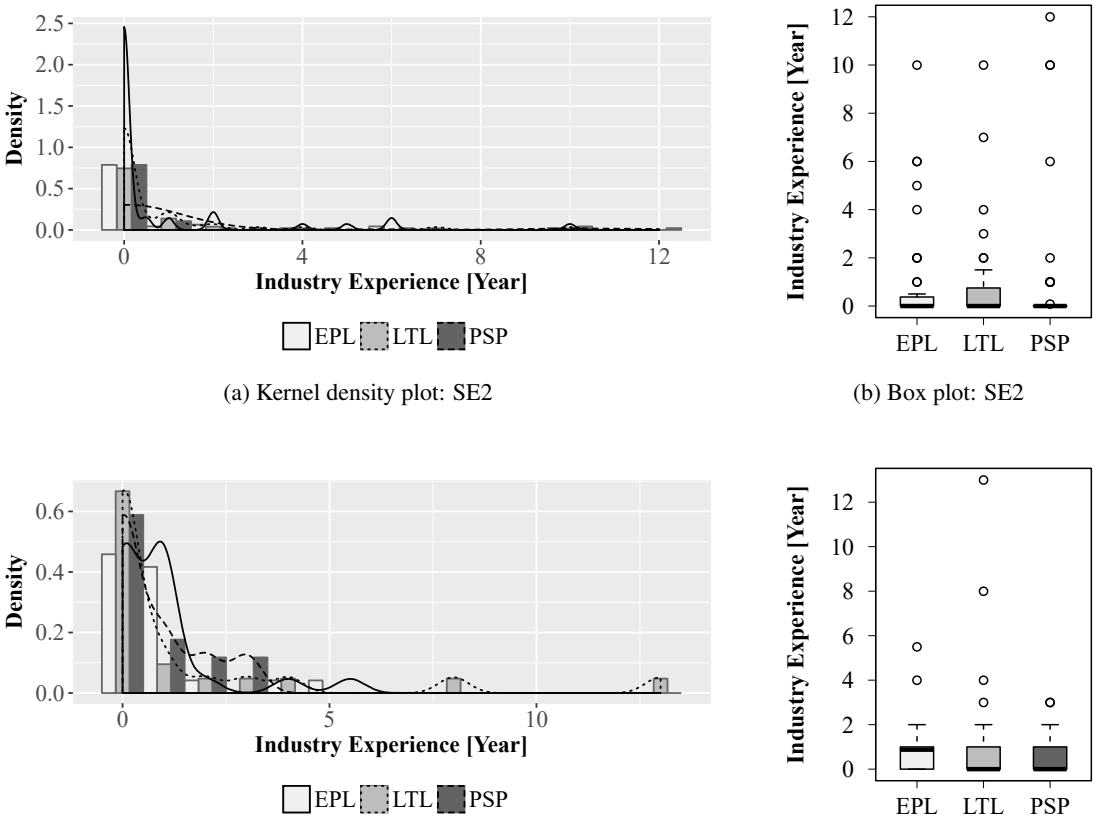


(d) Box plot: ASE

Figure 7.6.: Kernel density plots and box plots of the participants' programming experience per group and course

of women is overall low, the magnitude of potential disturbing effects is assumed to be low as well.

Figure 7.6 shows the participants' programming experience. According to the kernel density plot in Figure 7.6 (a) and the box plot in Figure 7.6 (b), the central tendency is balanced at 2-3 years in SE2. There are three outliers each in the LTL and PSP groups, and a single one in EPL, which indicate participants with long-term programming experience relatively to their colleagues in the same experiment group. In line with expectations, the participants of the master-level course ASE have more programming experience than their colleagues in the bachelor course SE2 (cf. Figure 7.6 (c) & (d)). The peak density is at 5 years programming experience in LTL and PSP. The participants of the EPL group appear to be slightly less experienced with a peak density at 4 years and a higher density in the range of 0 to 1 years. Each group has a single



(a) Kernel density plot: SE2

(b) Box plot: SE2

(c) Kernel density plot: ASE

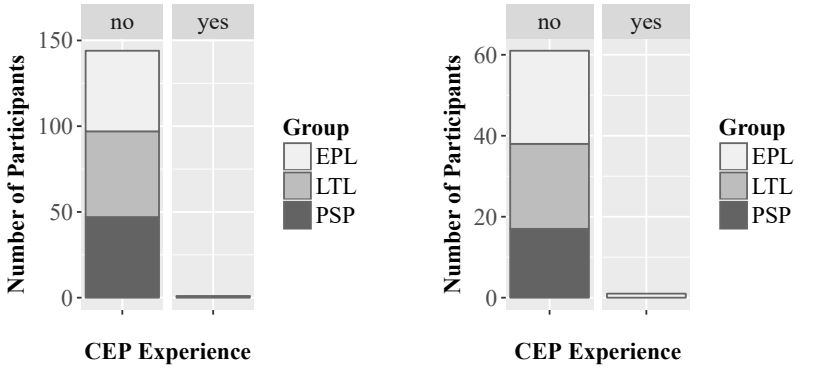
(d) Box plot: ASE

Figure 7.7.: Kernel density plots and box plots of the participants' software industry work experience per group and course

outlier with more years of programming experience than the other participants of the same experiment group. Some master students are rather inexperienced in programming, which seems to be caused by their diverse background (i.e., coming from various faculties and countries with different curricula).

Overall, the degree of industry experience is low in SE2, as to be expected in a bachelor-level course (cf. Figure 7.7 (a) & (b)). In ASE, a considerable number of the students have already begun to work in the industry (cf. Figure 7.7 (c) & (d)). Interestingly, EPL participants in ASE seem to have slightly more industry experience (cf. Figure 7.7 (c) & (d)) than their colleagues in the same course despite reporting less programming experience (cf. Figure 7.6 (c) & (d)).

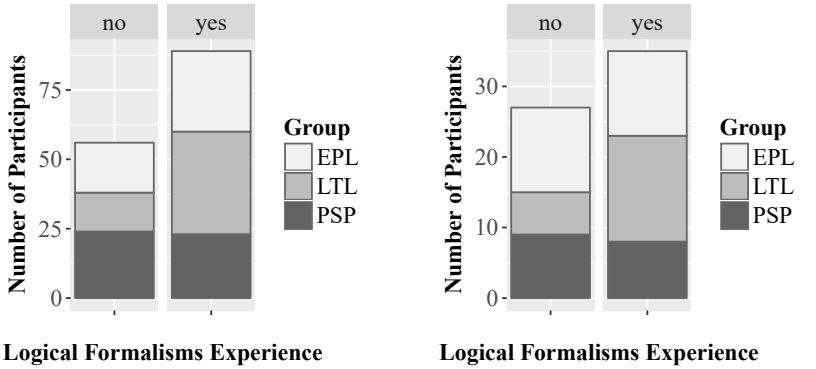
Figure 7.8 shows the participants' prior experience with Complex Event Processing (CEP). Almost all participants do not have any experience with CEP. Just two participants, one in the



(a) Software Engineering 2 (bachelor-level course)

(b) Advanced Software Engineering (master-level course)

Figure 7.8.: Bar charts of the participants' prior experience with Complex Event Processing per group and course



(a) Software Engineering 2 (bachelor-level course)

(b) Advanced Software Engineering (master-level course)

Figure 7.9.: Bar charts of the participants' prior experience with logical formalisms per group and course

PSP group in SE2 and one in the EPL group in ASE, have prior experience with CEP. In contrast, the level of experience with logical formalisms is high (cf. Figure 7.9). Overall, the share of experienced participants is larger in the master-level course ASE than in the bachelor-level course SE2. Interestingly, the share of prior experience with logical formalisms is higher in the LTL group than in the other groups. A potential reason could be that some of the LTL participants misunderstood this question by falsely assuming studying LTL (which has "logic" in its name) for this experiment qualifies as "prior knowledge".

Apart from minor differences between the experiment groups, which are to be expected in a completely randomized experiment design, the groups appear to be overall well-balanced. That is, there are no clear indications of disturbing effects on the measurement of the dependent variables resulting from unbalanced groups.

Descriptive Statistics of Dependent Variables

Table 7.4 & Table 7.5 show the number of observations, central tendency and dispersion of the dependent variables *syntactic correctness*, *semantic correctness* and *response time* per group. In the bachelor-level course *Software Engineering 2*, the sample size is relatively large and evenly distributed ($49 : 47 : 49$). In the master-level course *Advanced Software Engineering*, there are less than half as many observations. Unfortunately, the number of participants of the group with the smallest number of observations, namely PSP, was further diminished by the exclusion of three participants (cf. Section 7.4.1). In consequence, the distribution in the ASE course is $21 : 17 : 24$. The median and mean correctness values of the LTL groups both in SE2 and ASE are smaller than those of the other two groups. In SE2, the mean syntactic correctness of the LTL group is 56.52, thus about 5% less than in the EPL group (61.82%) and about 12% less than in the PSP group (68.64%), and the mean semantic correctness of the LTL group is at 28.49%, so about 10% below the EPL group (38.20%) and 22% below the PSP group (50.19%). In ASE, the mean syntactic correctness of the LTL group is 57.01%, thus about 8% less than in the PSP group (65.13%) and about 15% less than in the EPL group (71.91%). While the PSP group overall achieved a higher syntactic and semantic correctness than the LTL group in SE2, this ranking is reversed in the ASE course, where EPL participants overall achieved a higher syntactic and semantic correctness than their colleagues of the PSP group. The mean syntactic correctness achieved by the PSP group (65.13%) is about 7% higher than in the EPL group (71.91%) in SE2 whereas the EPL group achieved an about 7% higher mean syntactic correctness (71.91%) than the PSP group (65.13%) in ASE. In SE2, the mean semantic correctness of the PSP group (50.19%) is about 12% higher than in the EPL group (38.20%). In ASE, the mean semantic correctness is about 3% higher in the EPL group (49.71%) than in the PSP group (46.93%). The mean and median response times are overall faster in the SE2 course than in the ASE course.

Table 7.4.: Number of observations, central tendency and dispersion of the dependent variables semantic/syntactic correctness and response time per group in SE2

		LTL	PSP	EPL
	Total number of observations	51	49	49
	Number of considered observations	49	47	49
Syntactic Correctness	Arithmetic mean [%]	56.52	68.64	61.82
	Standard deviation (SD) [%]	16.40	16.99	16.85
	Median [%]	57.84	72.55	61.76
	Median absolute deviation (MAD) [%]	19.19	13.37	18.61
	Minimum [%]	9.02	24.51	21.18
	Maximum [%]	96.27	98.82	89.22
	Skew	−0.3	−0.55	−0.53
	Kurtosis	0.01	−0.09	−0.4
Semantic Correctness	Arithmetic mean [%]	28.49	50.19	38.20
	Standard deviation (SD) [%]	14.48	15.74	14.73
	Median [%]	27.06	49.61	36.08
	Median absolute deviation (MAD) [%]	13.66	15.12	13.66
	Minimum [%]	2.75	18.04	10
	Maximum [%]	68.43	80.59	72.55
	Skew	0.75	−0.08	0.27
	Kurtosis	0.24	−0.68	−0.56
Response Time	Arithmetic mean [Minute]	43.49	48.68	44.87
	Standard deviation (SD) [Minute]	13.10	14.39	14.07
	Median [Minute]	40.50	45.67	47.22
	Median absolute deviation (MAD) [Minute]	11.98	17.49	13.66
	Minimum [Minute]	15.07	27.00	14.58
	Maximum [Minute]	75.40	79.93	75.00
	Skew	0.33	0.38	0.14
	Kurtosis	−0.35	−0.93	−0.41

Table 7.5.: Number of observations, central tendency and dispersion of the dependent variables semantic/syntactic correctness and response time per group in ASE

		LTL	PSP	EPL
	Total number of observations	22	20	24
	Number of considered observations	21	17	24
Syntactic Correctness	Arithmetic mean [%]	57.01	65.13	71.91
	Standard deviation (SD) [%]	15.62	21.02	13.78
	Median [%]	56.67	67.84	72.06
	Median absolute deviation (MAD) [%]	18.90	26.74	10.47
	Minimum [%]	29.61	21.76	31.76
	Maximum [%]	81.96	89.41	94.71
	Skew	−0.15	−0.5	−0.9
	Kurtosis	1.22	−1.02	1.05
Semantic Correctness	Arithmetic mean [%]	30.85	46.93	49.71
	Standard deviation (SD) [%]	12.96	17.14	13.46
	Median [%]	29.61	47.84	51.57
	Median absolute deviation (MAD) [%]	14.54	19.19	12.06
	Minimum [%]	12.75	17.65	19.41
	Maximum [%]	63.14	75.69	76.86
	Skew	0.6	0.06	−0.37
	Kurtosis	−0.41	−1.12	−0.45
Response Time	Arithmetic mean [Minute]	52.32	55.99	58.82
	Standard deviation (SD) [Minute]	15.36	13.64	14.15
	Median [Minute]	49.00	62.00	58.00
	Median absolute deviation (MAD) [Minute]	16.88	11.64	15.64
	Minimum [Minute]	28.00	29.50	37.17
	Maximum [Minute]	84.00	73.08	81.78
	Skew	0.42	−0.61	0.15
	Kurtosis	−0.94	−1.09	−1.19

In SE2, the mean response time of the LTL group (43.49 minutes) is slightly faster than in EPL (44.87 minutes), and a few minutes faster than in the PSP group (48.68 minutes). In ASE, the mean response time of the LTL group (52.32 minutes) is $3 - 4$ minutes faster than in the PSP group (55.99 minutes), and $6 - 7$ minutes faster than in the EPL group (58.82 minutes).

Skew is a measure of the shape of a distribution. A positive skew value indicates a right-tailed distribution (i.e., more cases of low correctness than high correctness), a negative skew value indicates a left-tailed distribution (i.e., more cases of high correctness than low correctness), and a skew value close to zero indicates a symmetric distribution. Differences in skew are, for example, present

- between the semantic correctness distributions of LTL (0.75 indicating that the mass of the distribution is concentrated at lower levels of correctness) and PSP (-0.08 indicating a rather symmetric distribution) in SE2,
- between the syntactic correctness distributions of LTL (-0.15 indicating a curve that is slightly leaned to the right) and EPL (-0.9 indicating a distribution with only few measurements in lower correctness ranges) in ASE,
- between the semantic correctness distributions of LTL (0.6 indicating higher densities in lower correctness ranges) and EPL (-0.37 indicating higher densities in higher correctness ranges) in ASE, and
- between the response time distributions of LTL (0.42 indicating a left-leaning curve) and PSP (-0.61 indicating a right-leaning curve) in ASE.

Kurtosis is another measure for the shape of a distribution which focuses on the general tailedness. Positive kurtosis values indicate skinny tails with a steep distribution, whereas negative kurtosis values indicate fat tails. The most severe difference in kurtosis is present between the syntactic correctness distributions of the LTL group (1.22) and PSP group (-1.02) in ASE.

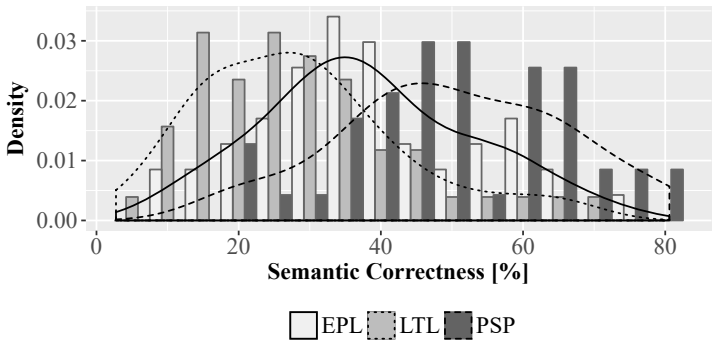
So far, the dependent variables were analyzed on the basis of separating between course groups, which reflects the participants' academic level of progression. Next, the dependent variables are investigated focusing on participants with industrial working experience. Table 7.6 summarizes the descriptive statistics of the dependent variables when focusing on participants with industrial working experience of one year and above. Based on the demographic data collected (cf. Section 7.4.2), we consider this subset of participants to be close to the population of industrial practitioners with basic to modest experience. The mean syntactic correctness in the LTL group (58.65%) is about 8% lower than in the PSP (66.79%) and EPL (66.01%) groups. The PSP group achieved the highest degree of semantic correctness (48.58%), closely followed by the EPL group (44.46%). The LTL group achieved 30.51% semantic correctness, which

Table 7.6.: Number of observations, central tendency and dispersion of the dependent variables semantic/syntactic correctness and response time per group of participants with working experience ≥ 1 year

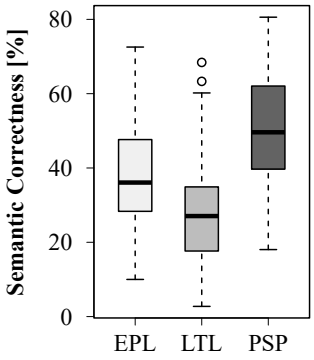
		LTL	**PSP**	**EPL**
	Number of observations	20	17	22
Syntactic Correctness	Arithmetic mean [%]	58.65	66.79	66.01
	Standard deviation (SD) [%]	14.68	17.76	14.82
	Median [%]	58.82	67.84	70.20
	Median absolute deviation (MAD) [%]	16.42	13.08	12.50
	Minimum [%]	31.18	21.76	26.67
	Maximum [%]	81.96	89.41	89.22
	Skew	-0.33	-0.89	-0.87
	Kurtosis	-1.03	0.24	0.32
Semantic Correctness	Arithmetic mean [%]	30.51	48.58	44.46
	Standard deviation (SD) [%]	16.04	16.93	15.20
	Median [%]	28.73	49.22	45.78
	Median absolute deviation (MAD) [%]	16.86	20.93	18.46
	Minimum [%]	8.24	17.65	15.69
	Maximum [%]	63.33	75.69	72.55
	Skew	0.55	0.2	-0.1
	Kurtosis	-0.72	-1.27	-1.07
Response Time	Arithmetic mean [Minute]	49.31	49.19	48.64
	Standard deviation (SD) [Minute]	16.81	13.34	14.03
	Median [Minute]	47.94	48.85	48.13
	Median absolute deviation (MAD) [Minute]	15.80	20.36	15.52
	Minimum [Minute]	15.07	29.50	24.07
	Maximum [Minute]	84.00	66.00	76.08
	Skew	0.29	0.21	0.22
	Kurtosis	-0.42	-1.56	-0.87

is noticeable lower than in the two other groups. Present differences in skew and kurtosis are indications of differences in central location and distribution shape.

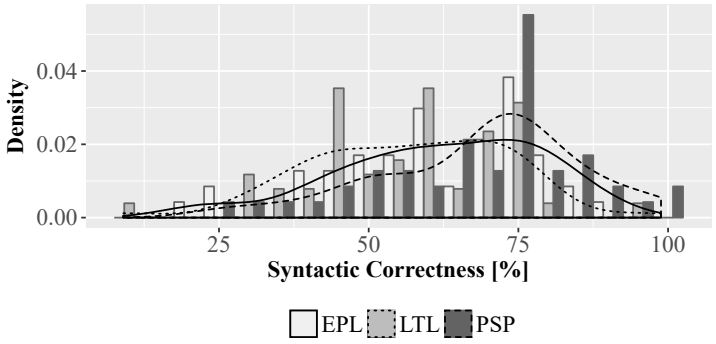
Figure 7.10 shows kernel density plots and box plots of the dependent variables *syntactic correctness*, *semantic correctness* and *response time* in the SE2 course. As the kernel density plot in Figure 7.10 (a) clearly indicates, there are differences in central location and shape between the semantic correctness distributions of the groups. While the LTL group has a very low density in the range of $50 - 100\%$ semantic correctness, the PSP has a high density in the range of $40 - 70\%$ semantic correctness. The central location of the EPL group (about 35% semantic correctness) is located between the peaks of the two other distributions. Figure 7.10 (b) shows two outliers in the LTL group, which represent participants who were able to achieve a higher level of correctness than most of their colleagues in the same experiment group. In Figure 7.10 (c), a



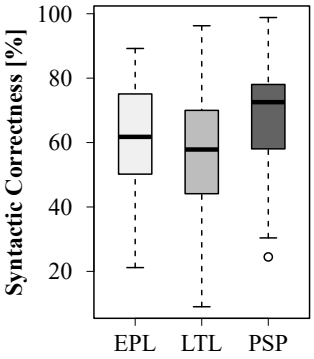
(a) Kernel density plot: Semantic Correctness



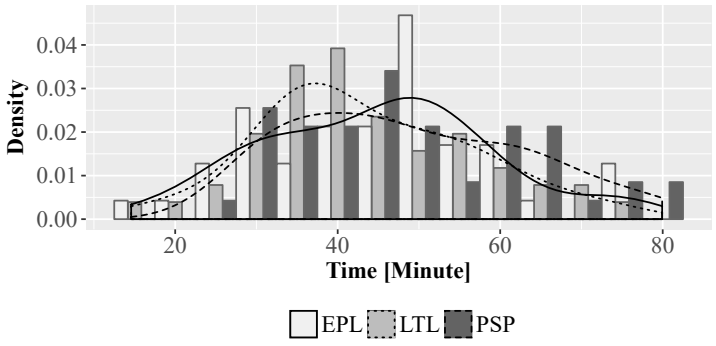
(b) Box plot: Semantic Correctness



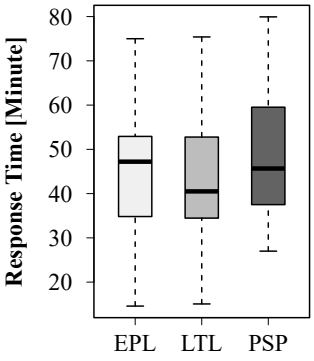
(c) Kernel density plot: Syntactic Correctness



(d) Box plot: Syntactic Correctness



(e) Kernel density plot: Response time



(f) Box plot: Response time

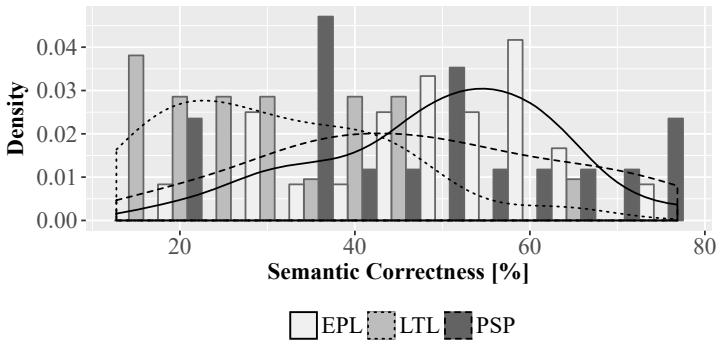
Figure 7.10.: Kernel density plots and box plots of the participants' semantic/syntactic correctness and response time of the given answers per group in the Software Engineering 2 (bachelor-level) course

kernel density plot of the syntactic correctness is shown. All distributions have their central location at $70 - 75$ minutes, but their shapes are different. The PSP distribution has a particularly high density directly at the central location, whereas the remaining distributions show higher densities in the lower correctness ranges. There is a single outlier in the PSP group, indicating a participant who has achieved a slightly lower level of syntactic correctness (cf. Figure 7.10 (d)). Both the kernel density plot in Figure 7.10 (c) and the box plot in Figure 7.10 (d) indicate a clear difference in distribution. The assumption of equal variance seems violated. The same applies to the response time distributions shown in Figure 7.10 (e) & (f).

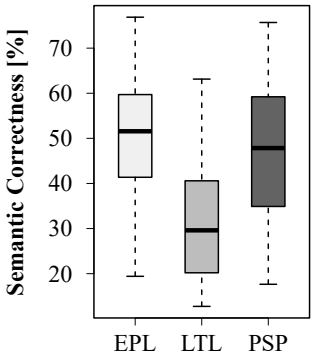
Figure 7.11 visualizes the data of the dependent variables *syntactic correctness*, *semantic correctness* and *response time* of ASE participants by kernel density plots and box plots. In Figure 7.11 (a), the PSP semantic correctness distribution is rather flat with its peak at about 45%. While the LTL semantic correctness distribution has a high density in the lower correctness range ($10 - 45\%$) with its peak density at $20 - 25\%$, the EPL distribution has a high density in the range of $45 - 65\%$. Thus, all semantic correctness distributions appear to be different in shape and central location. Regarding syntactic correctness (cf. Figure 7.11 (c)), the LTL distribution appears to be bi-modal with peaks at 50% and 70%. The EPL syntactic correctness distribution is steeper than the others with its peak at $70 - 75\%$. In contrast, the PSP syntactic correctness distribution is strikingly flat with a slightly higher density in the higher syntactic correctness ranges. There is a single outlier in the EPL group showing a low level of syntactic correctness. The PSP group has its peak response time density at 65 minutes, and there are indications of bi-modality with a second small peak at about 35 minutes. Both remaining response time distributions are rather similar of shape, but their central locations differ. LTL has its central location at 45 minutes whereas PSP has it at 55 minutes.

Figure 7.12 shows kernel density plots and box plots of the dependent variables *syntactic correctness*, *semantic correctness* and *response time* for the subset of participants with industry experience. The peak density in semantic correctness in the LTL group can be found at about 20% while the other groups have their peaks at $50 - 60\%$. The syntactic correctness distribution of the LTL group is less steep than the ones of the other two groups with higher densities in the lower syntactic correctness ranges. While there are only minor differences in distribution shape of the response time variable between the LTL and EPL groups with their peak density in the range of $45 - 50$ minutes, the PSP group has its peak density in the range of $60 - 65$. Overall, the distributions display marked differences in central location and shape.

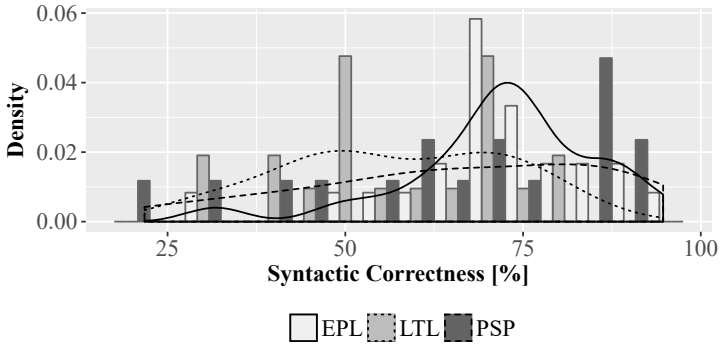
According to the scatter plots in Figure 7.13, there is a positive linear correlation between the dependent variables *syntactic correctness* and *semantic correctness*. That is, syntactic and semantic correctness are not isolated metrics, which is not surprising, because correct application of syntax is a prerequisite for enabling meaning. There is no correlation between the correct-



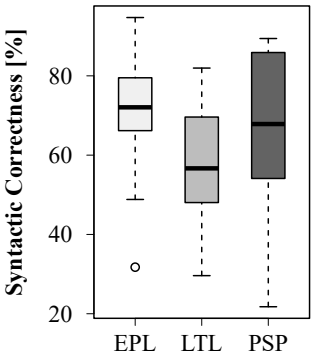
(a) Kernel density plot: Semantic Correctness



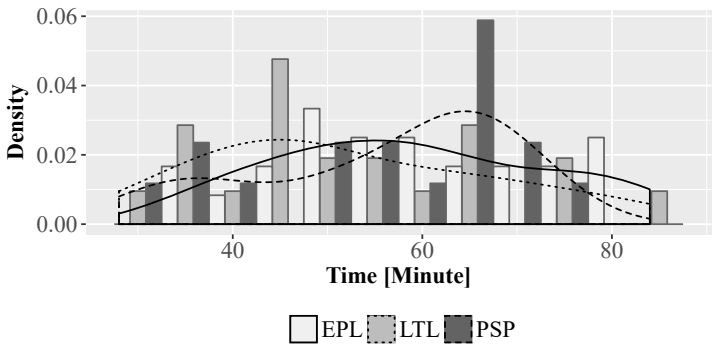
(b) Box plot: Semantic Correctness



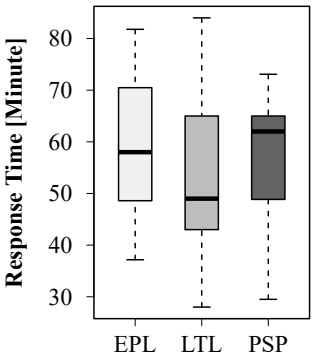
(c) Kernel density plot: Syntactic Correctness



(d) Box plot: Syntactic Correctness

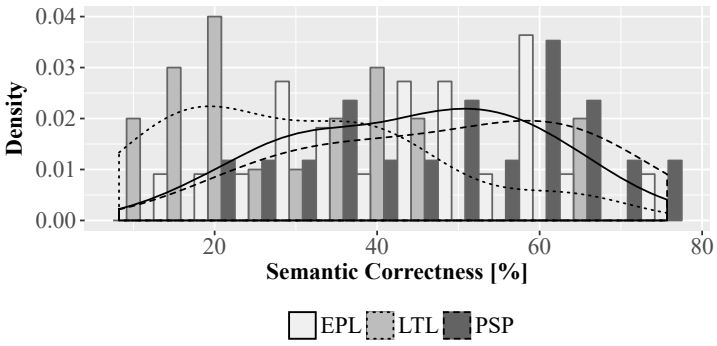


(e) Kernel density plot: Response time

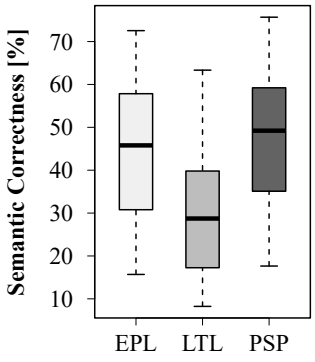


(f) Box plot: Response time

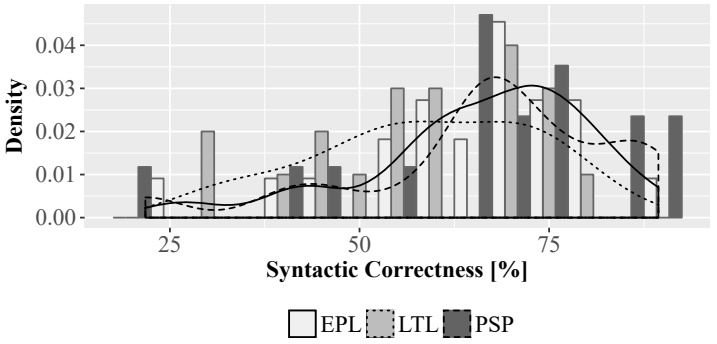
Figure 7.11.: Kernel density plots and box plots of the participants' semantic/syntactic correctness and response time of the given answers per group in the Advanced Software Engineering (master-level) course



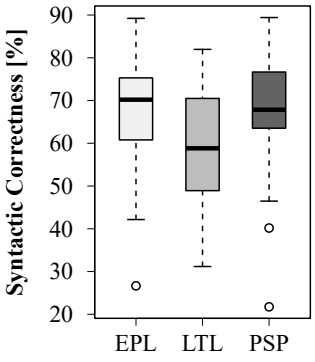
(a) Kernel density plot: Semantic Correctness



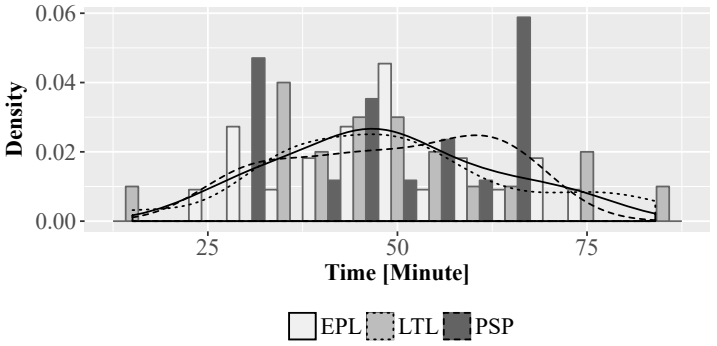
(b) Box plot: Semantic Correctness



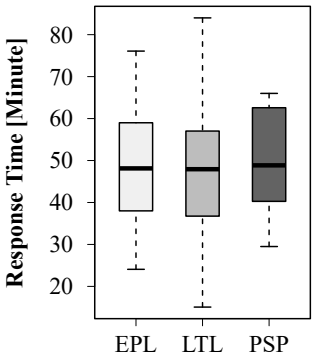
(c) Kernel density plot: Syntactic Correctness



(d) Box plot: Syntactic Correctness

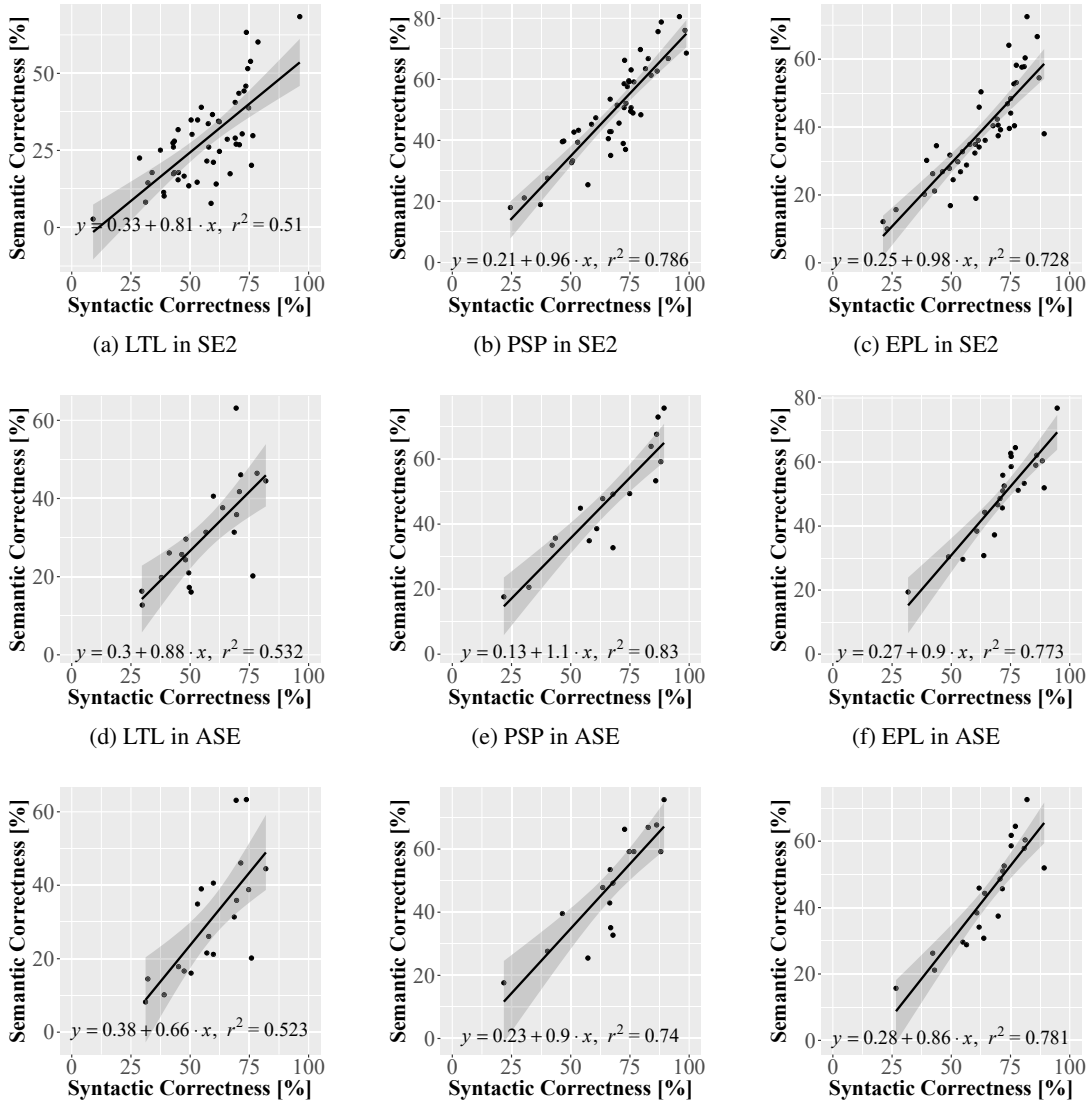


(e) Kernel density plot: Response time



(f) Box plot: Response time

Figure 7.12.: Kernel density plots and box plots of the participants' semantic/syntactic correctness and response time of the given answers per experiment group of participants with ≥ 1 year industry experience



(a) LTL in SE2 (b) PSP in SE2 (c) EPL in SE2

(d) LTL in ASE (e) PSP in ASE (f) EPL in ASE

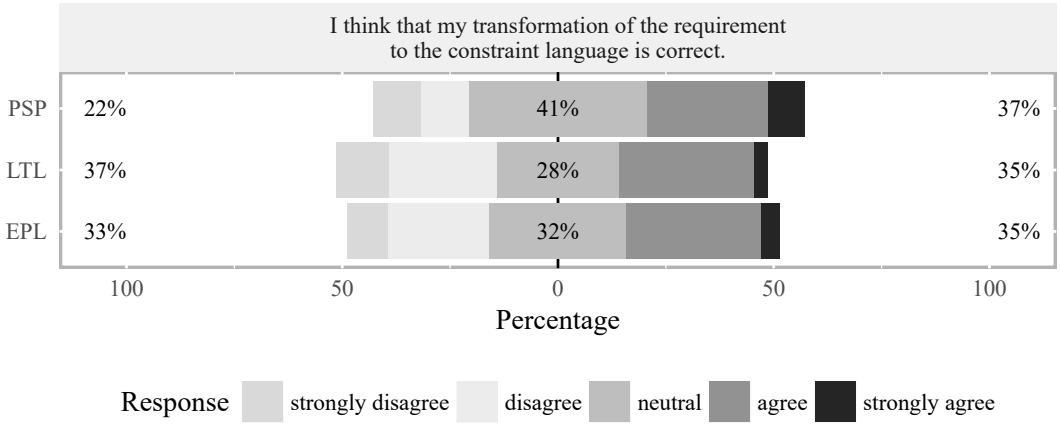
(g) LTL $\geq$ 1 year industry experience (h) PSP $\geq$ 1 year industry experience (i) EPL $\geq$ 1 year industry experience

Figure 7.13.: Scatter plots of syntactic vs. semantic correctness with linear trend lines, 95% confidence regions, and coefficients of determination (r^2)

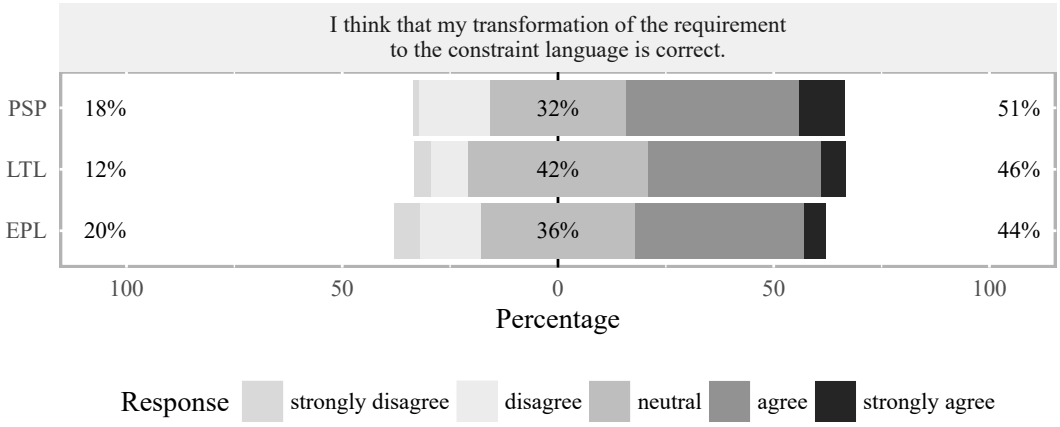
ness variables and the dependent variable *response time*. Consequently, the amount of time spent working on the experiment tasks by the participants did not necessarily result in higher correctness values.

With regards to the stacked bar chart (cf. Bryer & Speerschneider [170]) in Figure 7.14 (a) showing the *perceived correctness* in SE2, the share of *strongly agree* responses to the statement "I think that my transformation of the requirement to the constraint language is correct" is 2% higher in PSP (37%) than in the other two groups, and the share of *(strongly) disagree* answers is 22% in PSP while it is higher in LTL (37%) and EPL (33%). With 41% the share of *neutral* answers is largest in PSP. In ASE (cf. Figure 7.14 (b)), the participants appear to be overall slightly more confident regarding the correctness of their formalizations. The largest share of *(strongly) agree* responses is again present in the PSP group (51%), followed by LTL (46%) and EPL (44%). According to the stacked bar charts in Figure 7.14, the perceived correctness of PSP appears to be slightly higher than in the other experiment groups in SE2, while EPL has a slightly lower perceived correctness than the other languages in ASE. According to Figure 7.14 (c), a large share (44%) of participants with industry experience in the PSP group is undecided whether the given answer is correct. The percentage of *neutral* answers of participants with industry experience is lowest in the EPL group (30%) and only slightly higher in the LTL group. The largest share of *(strongly) agree* responses of participants with industry experience is present in the EPL group (42%), followed by LTL (38%) and PSP (34%).

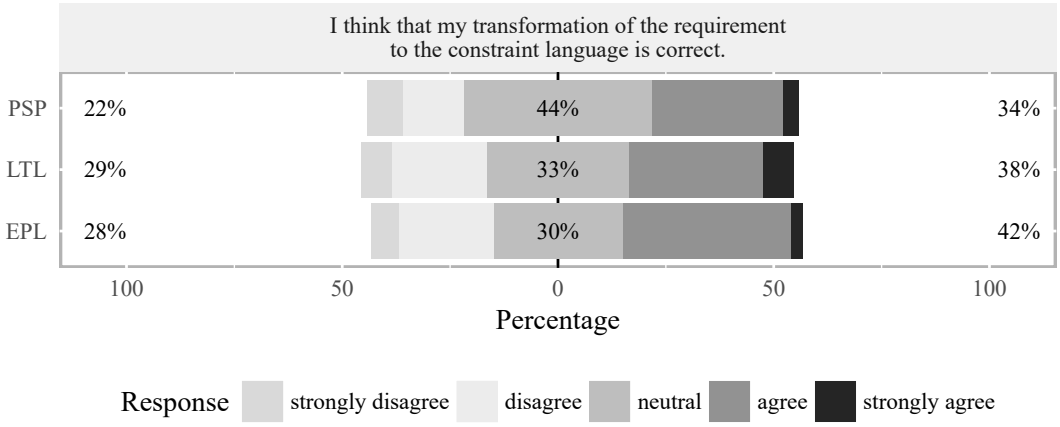
Figure 7.15 contains stacked bar charts of the participants' *perceived ease of application* of the tested languages. Interestingly, there appears to be a strong similarity between the *perceived correctness* and *perceived ease of application* responses in SE2 regarding the ranking of the approaches (cf. Figure 7.15 (a) and Figure 7.14 (a)). PSP with 25% *(strongly) agree*ing and 42% *(strongly) disagree*ing appears to be slightly easier to apply than EPL with 21% *(strongly) agree*ing and 48% *(strongly) disagree*ing, and LTL with 17% *(strongly) agree*ing and 48% *(strongly) disagree*ing is perceived as slightly more difficult to apply than EPL. In ASE (cf. Figure 7.15 (b)), the application of PSP is perceived to be even easier than in SE2. Interestingly, EPL is perceived to be similarly easy as PSP with regards to application. Like in SE2, LTL is ranked last in *perceived ease of application*. Figure 7.15 (c) focuses on participants with industry experience and reveals striking differences between the groups. The perceived ease of application is highest rated in the EPL group with 33% *(strongly) agree*ing and 38% *(strongly) disagree*ing, which means that there is still a shift towards a negative rating. The strongest shift towards low ease of application is present in the LTL group with only 7% *(strongly) agree*ing and 52% *(strongly) disagree*ing. In between are the results of the PSP group with 22% *(strongly) agree*ing and 49% *(strongly) disagree*ing.



(a) Software Engineering 2 (SE2)

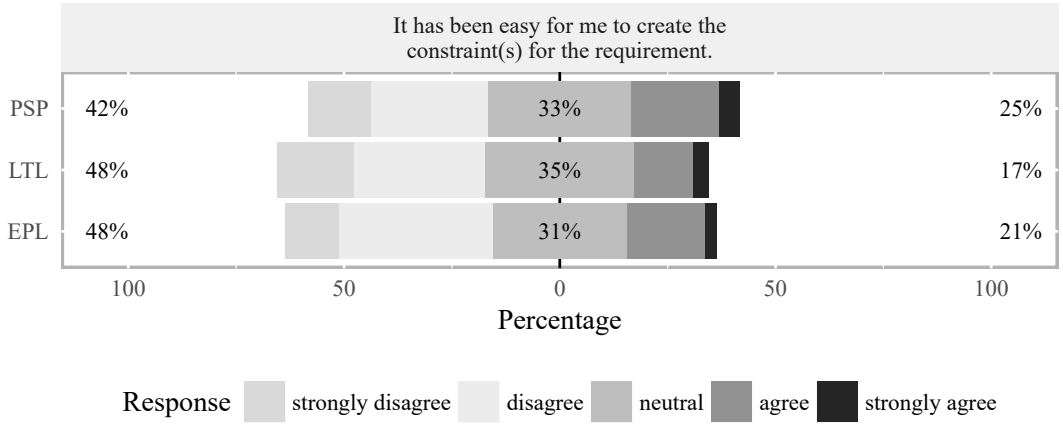


(b) Advanced Software Engineering (ASE)

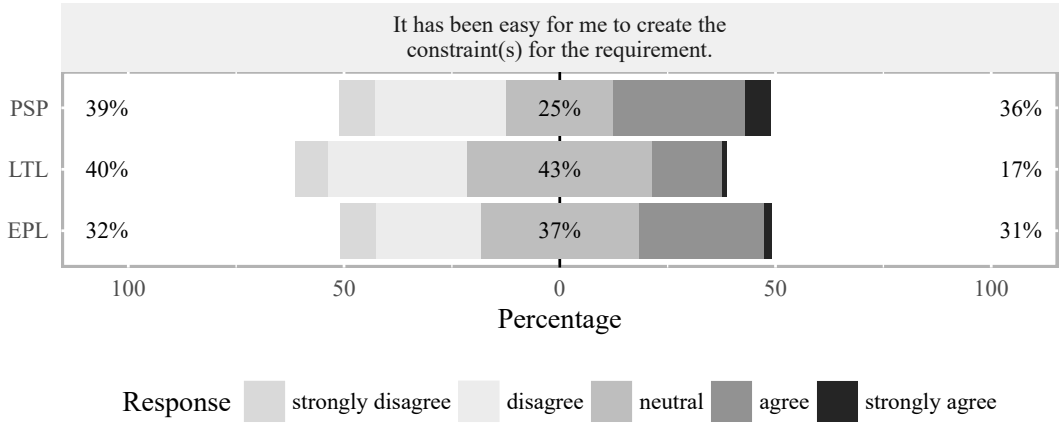


(c) Participants with industry experience $\geq$ 1 year

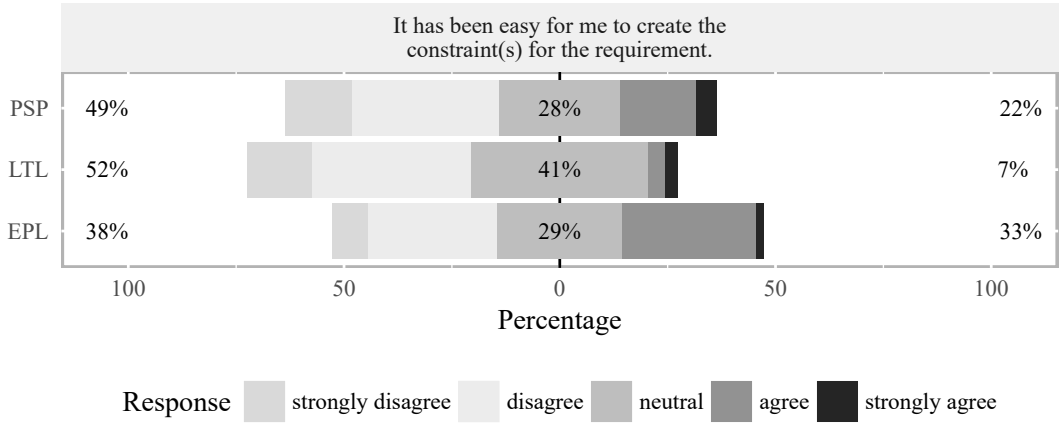
Figure 7.14.: Participants' perceived correctness



(a) Software Engineering 2 (SE2)



(b) Advanced Software Engineering (ASE)



(c) Participants with industry experience $\geq$ 1 year

Figure 7.15.: Participants' perceived ease of application

7.4.3. Statistical Inference

Before applying any statistical test, its model assumption must be tested and met. For a discussion whether or not the normality assumption is violated by the acquired data, see Appendix A.2. Since there is uncertainty regarding normality, a core assumption of parametric testing, non-parametric testing is the preferable approach.

Standard non-parametric tests like Kruskal-Wallis cannot be applied if distribution shapes differ apart from their central location (cf. descriptive statistics in Section 7.4.2), so *Cliff's delta* (cf. Cliff [150] and Rogmann [151]), a robust non-parametric test, is applied. Table 7.7 summarizes the test results for the bachelor-level course SE2. To take multiple testing into account, the p-values are adjusted based on the method proposed by Benjamini & Hochberg [152]. There is a highly significant result with a medium effect size magnitude indicating that PSP provides a higher syntactic correctness than LTL. After p-value adjustments, no such result is present in the remaining syntactic correctness tests. All semantic correctness test results are highly significant with medium- to large-sized effects. There is no significant difference between the response times. Consequently, $H_{0,1}$ is rejected on the basis of syntactic and semantic correctness whereas $H_{0,2}$ and $H_{0,3}$ can only be rejected based on semantic correctness.

In the master-level course ASE (cf. Table 7.8), there is a large-sized difference in syntactic correctness between EPL and LTL. Regarding semantic correctness, there are large-sized effects between PSP/LTL and EPL/LTL, indicating that the former outperforms the latter. As in SE2, there are no significant differences regarding the response times. Consequently, $H_{0,1}$ can only be rejected on the basis of semantic correctness whereas $H_{0,3}$ is rejected based on both types of correctness.

Tables 7.9, 7.10 & 7.11 summarize the test results regarding perceived correctness and perceived ease of application. Almost all test results are not significant with two exceptions: (1) A significant test result ($p = 0.0316$) with a medium-sized effect is present in SE2 between PSP and LTL with regards to perceived correctness. Consequently, $H_{0,4}$ can be rejected in SE2. That is, PSP participants are significantly more confident that their formalizations are correct than LTL participants at the bachelor level, while such an effect is not measurable at the master level or within the sample of participants with industry experience. (2) Participants with industry experience rate the ease of application of EPL significantly higher than that of LTL ($p = 0.0023$). Consequently, $H_{0,9}$ can be rejected for participants with industry experience.

Table 7.12 contains the test results for participants with industry experience. There is no significant difference in terms of syntactic correctness and response time. Similarly to ASE, there is no significant difference in semantic correctness between PSP and EPL while there are significant differences with large-sized effects when comparing PSP against LTL and EPL against LTL.

Table 7.7.: Cliff's d of syntactic/semantic correctness and response time in SE2, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Syntactic Correctness	$p_1 = P(X > Y)$	0.7059	0.6071	0.6028
	$p_2 = P(X = Y)$	0.0038	0.0014	0.0046
	$p_3 = P(X < Y)$	0.2904	0.3916	0.3926
	d	-0.4155	-0.2155	-0.2103
	s_d	0.1054	0.1166	0.1148
	z	-3.9412	-1.8477	-1.8308
	$CI\ low$	-0.5733	-0.3976	-0.3899
	$CI\ high$	-0.2281	-0.0171	-0.0152
	p	7.7×10^{-5}	0.0339	0.0351
	$FDR\ adjusted\ p$	0.0004	0.0658	0.0658
	$level\ of\ significance$	***	-	-
	$effect\ size\ magnitude$	medium	-	-
Semantic Correctness	$p_1 = P(X > Y)$	0.8448	0.7153	0.6913
	$p_2 = P(X = Y)$	0.1356	0.0032	0.0058
	$p_3 = P(X < Y)$	0.1535	0.2816	0.3029
	d	-0.6913	-0.4337	-0.3884
	s_d	0.0794	0.1057	0.1066
	z	-8.7104	-4.1028	-3.6445
	$CI\ low$	-0.8006	-0.5909	-0.549
	$CI\ high$	-0.5374	-0.2447	-0.2002
	p	4.4×10^{-14}	4.4×10^{-5}	0.0002
	$FDR\ adjusted\ p$	6.5×10^{-13}	0.0003	0.0008
	$level\ of\ significance$	***	***	***
	$effect\ size\ magnitude$	large	large	medium
Response Time	$p_1 = P(X > Y)$	0.5928	0.5632	0.5298
	$p_2 = P(X = Y)$	0.0017	0.0023	0.0029
	$p_3 = P(X < Y)$	0.4055	0.4346	0.4673
	d	0.1873	-0.1286	-0.0626
	s_d	0.1153	0.119	0.1185
	z	1.625	-1.0808	-0.5281
	$CI\ low$	-0.3685	-0.3176	-0.2533
	$CI\ high$	0.0076	0.0702	0.1329
	p	0.0537	0.1413	0.2993
	$FDR\ adjusted\ p$	0.0895	0.1766	0.2993
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-

Table 7.8.: Cliff's d of syntactic/semantic correctness and response time in ASE, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Syntactic Correctness	$p_1 = P(X > Y)$	0.6303	0.4069	0.7718
	$p_2 = P(X = Y)$	0.0058	0	0.006
	$p_3 = P(X < Y)$	0.3697	0.5931	0.2222
	d	-0.2605	0.1863	-0.5496
	s_d	0.1923	0.1991	0.1404
	z	-1.3547	0.9354	-3.9153
	$CI\ low$	-0.5429	-0.1513	-0.7397
	$CI\ high$	0.0748	0.4849	-0.2783
	p	0.092	0.1777	0.0002
	$FDR\ adjusted\ p$	0.1971	0.2961	0.0012
	$level\ of\ significance$	-	-	**
	$effect\ size\ magnitude$	-	-	large
Semantic Correctness	$p_1 = P(X > Y)$	0.7815	0.4461	0.8373
	$p_2 = P(X = Y)$	0	0.0025	0.002
	$p_3 = P(X < Y)$	0.2185	0.5515	0.1607
	d	-0.563	0.1054	-0.6766
	s_d	0.1517	0.1938	0.1229
	z	-3.7109	0.5438	-5.5023
	$CI\ low$	-0.7633	-0.2153	-0.8322
	$CI\ high$	-0.2641	0.4055	-0.4221
	p	0.0003	0.2948	9.6×10^{-7}
	$FDR\ adjusted\ p$	0.0017	0.3641	1.4×10^{-5}
	$level\ of\ significance$	**	-	***
	$effect\ size\ magnitude$	large	-	large
Response Time	$p_1 = P(X > Y)$	0.5686	0.4755	0.6349
	$p_2 = P(X = Y)$	0.0112	0	0.002
	$p_3 = P(X < Y)$	0.4202	0.5245	0.3631
	d	-0.1485	0.049	-0.2718
	s_d	0.194	0.1894	0.1697
	z	-0.7652	0.2588	-1.6016
	$CI\ low$	-0.4451	0.2595	-0.5243
	$CI\ high$	0.1775	0.3485	0.0246
	p	0.2246	0.3985	0.0583
	$FDR\ adjusted\ p$	0.3062	0.4703	0.1507
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-

Table 7.9.: Cliff's d of perceived correctness and ease of application in SE2, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Perceived Correctness	$p_1 = P(X > Y)$	0.4336	0.4087	0.392
	$p_2 = P(X = Y)$	0.2485	0.2589	0.259
	$p_3 = P(X < Y)$	0.3179	0.3324	0.349
	d	-0.1157	-0.0763	-0.043
	s_d	0.05	0.0511	0.0502
	z	-2.3139	-1.494	-0.8566
	$CI\ low$	-0.197	-0.1597	-0.1253
	$CI\ high$	-0.0328	0.0081	0.0398
	p	0.0105	0.0679	0.196
	$FDR\ adjusted\ p$	0.0316	0.1019	0.2262
	$level\ of\ significance$	*	-	-
	$effect\ size\ magnitude$	medium	-	-
Perceived Ease of Application	$p_1 = P(X > Y)$	0.4213	0.4005	0.3881
	$p_2 = P(X = Y)$	0.2518	0.2569	$0,2631$
	$p_3 = P(X < Y)$	0.3269	0.3426	0.3488
	d	-0.0945	-0.0579	-0.0394
	s_d	0.0502	0.0513	0.0501
	z	-1.8827	-1.1272	-0.7857
	$CI\ low$	-0.1762	-0.1417	-0.1214
	$CI\ high$	-0.0114	0.0268	0.0432
	p	0.0302	0.1301	0.2162
	$FDR\ adjusted\ p$	0.0658	0.1766	0.2317
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-

Table 7.10.: Cliff's d of perceived correctness and ease of application in ASE, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Perceived Correctness	$p_1 = P(X > Y)$	0.3675	0.4013	0.3095
	$p_2 = P(X = Y)$	0.3039	0.2914	0.324
	$p_3 = P(X < Y)$	0.3286	0.3074	0.3664
	d	-0.0389	-0.0939	0.0569
	s_d	0.0808	0.0778	0.0722
	z	-0.481	-1.2065	0.7882
	$CI\ low$	-0.1706	-0.22	-0.0623
	$CI\ high$	0.0942	0.0352	0.1745
	p	0.3155	0.1145	0.2157
	$FDR\ adjusted\ p$	0.3641	0.2147	0.3062
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-
Perceived Ease of Application	$p_1 = P(X > Y)$	0.4338	0.3752	0.4233
	$p_2 = P(X = Y)$	0.2613	0.2616	0.2891
	$p_3 = P(X < Y)$	0.3049	0.3632	0.2876
	d	-0.129	-0.012	-0.1356
	s_d	0.0827	0.0807	0.0725
	z	-1.5594	-0.1481	-1.8697
	$CI\ low$	-0.262	-0.144	-0.2526
	$CI\ high$	0.009	0.1205	-0.0147
	p	0.0603	0.4412	0.0314
	$FDR\ adjusted\ p$	0.1507	0.4412	0.1178
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-

Table 7.11.: Cliff's d of perceived correctness and ease of application for participants with industry experience, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Perceived Correctness	$p_1 = P(X > Y)$	0.3586	0.344	0.37
	$p_2 = P(X = Y)$	0.2778	0.2872	0.2745
	$p_3 = P(X < Y)$	0.3636	0.3689	0.3555
	d	0.0051	0.0249	-0.0145
	s_d	0.0813	0.0793	0.0768
	z	0.0622	0.3142	-0.1894
	$CI\ low$	-0.1283	-0.1055	-0.1402
	$CI\ high$	0.1383	0.1545	0.1116
	p	0.4752	0.3769	0.425
	$FDR\ adjusted\ p$	0.4752	0.4752	0.4752
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-
Perceived Ease of Application	$p_1 = P(X > Y)$	0.4078	0.3006	0.5014
	$p_2 = P(X = Y)$	0.2734	0.2524	0.2555
	$p_3 = P(X < Y)$	0.3188	0.447	0.2432
	d	-0.0889	0.1463	-0.2581
	s_d	0.0826	0.0802	0.0733
	z	-1.0765	1.8252	-3.5228
	$CI\ low$	-0.2226	0.0124	-0.3744
	$CI\ high$	0.048	0.2751	-0.1339
	p	0.1416	0.0348	0.0003
	$FDR\ adjusted\ p$	0.3033	0.1042	0.0023
	$level\ of\ significance$	-	-	**
	$effect\ size\ magnitude$	-	-	medium

Table 7.12.: Cliff's d of syntactic/semantic correctness and response time for participants with industry experience $\geq$ 1 year, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151]), adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$], and effect size magnitudes (cf. Kitchenham et al. [140])

		PSP/LTL	**PSP/EPL**	**EPL/LTL**
Syntactic Correctness	$p_1 = P(X > Y)$	0.6471	0.5321	0.6636
	$p_2 = P(X = Y)$	0.0029	0	0.0023
	$p_3 = P(X < Y)$	0.35	0.4679	0.3341
	d	-0.2971	-0.0642	-0.3295
	s_d	0.1875	0.1934	0.1702
	z	1.5845	-0.3318	-1.9358
	$CI\ low$	-0.5697	-0.3682	-0.5774
	$CI\ high$	0.0345	0.2523	-0.0260
	p	0.061	0.371	0.0299
	$FDR\ adjusted\ p$	0.1526	0.4752	0.1043
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-
Semantic Correctness	$p_1 = P(X > Y)$	0.7824	0.5802	0.7295
	$p_2 = P(X = Y)$	0	0	0.0023
	$p_3 = P(X < Y)$	0.2176	0.4198	0.2682
	d	-0.5647	-0.1604	-0.4614
	s_d	0.1501	0.1907	0.1582
	z	-3.7622	-0.8412	-2.9159
	$CI\ low$	-0.7632	-0.4513	-0.6812
	$CI\ high$	-0.2687	0.1613	-0.1652
	p	0.0003	0.2028	0.0029
	$FDR\ adjusted\ p$	0.0023	0.3803	0.0145
	$level\ of\ significance$	**	-	*
	$effect\ size\ magnitude$	large	-	large
Response Time	$p_1 = P(X > Y)$	0.5059	0.5134	0.4909
	$p_2 = P(X = Y)$	0.0029	0	0.0045
	$p_3 = P(X < Y)$	0.4912	0.4866	0.5045
	d	-0.0147	-0.0267	0.0136
	s_d	0.1986	0.191	0.1822
	z	-0.074	-0.14	0.0749
	$CI\ low$	-0.3314	-0.331	-0.2808
	$CI\ high$	0.305	0.2825	0.3057
	p	0.4707	0.4447	0.4704
	$FDR\ adjusted\ p$	0.4752	0.4752	0.4752
	$level\ of\ significance$	-	-	-
	$effect\ size\ magnitude$	-	-	-

The statistics software R[6] was used for all statistical analyses. In particular, the following libraries were used in the course of the performed statistical evaluations: *biotools* [153], *car* [154], *ggplot2* [155], *mvnormtest* [156], *mvoutlier* [157], *orddom* [151], *psych* [158], *usdm* [159], and *likert* [170].

7.5. Discussion

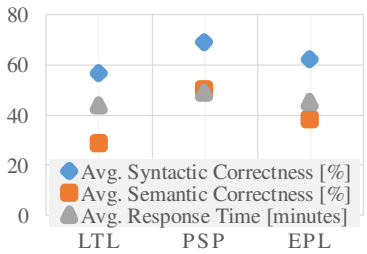
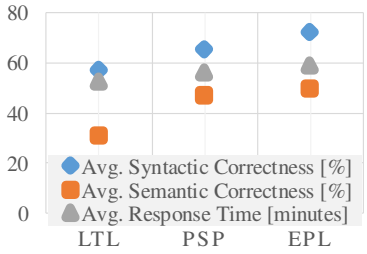
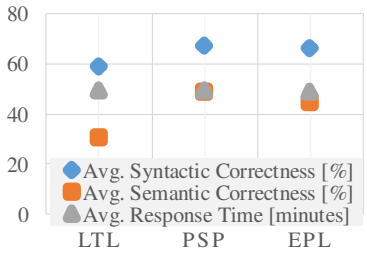
This sections discusses the results and threats to validity of the study.

7.5.1. Evaluation of Results and Implications

The experimental goal was stated as ***Analyze*** *LTL, PSP, and EPL* ***for the purpose of*** *their evaluation* ***with respect to*** *their understandability related to modeling compliance specifications* ***from the viewpoint of*** *the novice and moderately advanced software engineer, designer or developer* ***in the context/environment of*** *the Software Engineering 2 Lab and the Advanced Software Engineering Lab courses at the Faculty of Computer Science of the University of Vienna.* Due to the large number of participants with industry experience, it became possible to consider a third population, namely participants with industry experience, who function as proxies for industrial practitioners with basic to modest industry experience. Based upon the stated goal, questions concerning *understandability* were generated. The understandability construct focuses on the degree of syntactic and semantic correctness achieved and on the time spent on modeling compliance specifications. The results per question are summarized in Table 7.13. By differentiating between syntactic and semantic correctness, it became possible to reveal that differences in understandability in formal modeling of compliance specifications predominately lie in semantic correctness. Almost all test results regarding semantic correctness are highly significant with large-sized effects. Interestingly, no significant difference in semantic correctness is present between the pattern-based PSP approach and the CEP-based EPL language in the master-level course ASE and in the subset of participants with industry experience. That might imply that more experienced users are able to cope equally well with both approaches. Aside from that, the results suggest that the pattern-based PSP approach is more understandable than EPL and LTL, and that EPL provides a higher level of understandability than LTL. In terms of syntactic correctness, PSP seems to be more understandable than LTL for less experience users while EPL seems to be more understandable than LTL for more experienced users. This study did not reveal any significant differences in response time. Regarding perceived correctness and perceived ease of application, there are two significant test results, which imply that transformations to PSP

[6] `https://www.r-project.org/`

Table 7.13.: GQM summary

ID	Question	Summary of results
Q1	How understandable are the tested approaches for participants at the bachelor level (attending the Software Engineering 2 Lab course)?	
Q2	Are there differences in understandability between the tested approaches for participants at the bachelor level (attending the Software Engineering 2 Lab course)?	There are significant differences between all tested approaches in terms of semantic correctness, and between PSP and LTL in terms of syntactic correctness.
Q3	How understandable are the tested approaches for participants at the master level (attending the Advanced Software Engineering Lab course)?	
Q4	Are there differences in understandability between the tested approaches for participants at the master level (attending the Advanced Software Engineering Lab course)?	There are significant differences in terms of semantic and syntactic correctness between EPL and LTL, and between PSP and LTL in terms of semantic correctness.
Q5	How understandable are the tested approaches for participants with industrial working experience?	
Q6	Are there differences in understandability between the tested approaches for participants with industrial working experience?	There are significant differences in terms of semantic correctness between PSP and LTL as well as between EPL and LTL.

are perceived to be more correct than LTL transformations by less experienced users, and more experienced users with industry experience find that EPL is easier to apply than LTL.

Overall, the results imply that the pattern-based PSP approach has advantages with regards to understandability. Therefore, the pattern-based approach seems to be particularly well-suited for modeling compliance specifications. Moreover, the results indicate that EPL is more understandable than LTL. This could be important in cases where the set of available PSP patterns is not sufficient to model a compliance specification. In such cases, the compliance specification could be encoded in EPL for runtime verification, or an extension of the pattern catalog could be undertaken. In this regard, EPL specifications could be used to aid the creation of new patterns with underlying LTL formalizations by checking the plausibility of the LTL formula (cf. Chapter 5).

Moreover, the results are overall in line with two studies on the understandability of already existing formal specifications in LTL, EPL, and PSP (cf. Chapter 6). The results of these controlled experiments with 216 participants in total suggest that existing specifications in PSP are significantly easier to understand than existing specifications in EPL and LTL. Moreover, the results imply that existing specifications in EPL are significantly easier to understand than existing specifications in LTL. The correctness of understanding was evaluated by letting the participant decide whether a truth value is correct for a specification, given a specific trace. In contrast to the current study, which focuses on the formal modeling of compliance specifications, no major differences between novice and moderately advanced users were found in the understandability of existing specifications. Interestingly, the response times between the experimental groups were significantly different in most cases, an effect which appears to be absent during modeling (cf. Section 7.4.3).

7.5.2. Threats to Validity

In the following, all known threats that might have an impact on the validity of the results of this study are discussed.

Threats to Internal Validity

Threats to internal validity are unobserved variables that might have an undesired impact on the result of the experiment by disturbing the causal relationship of independent and dependent variables. There exist several threats to internal validity, which must be discussed:

- *History effects* refer to events that happen in the environment resulting in changes in the conditions of a study. The short duration of the study limits the possibility of changes in environmental conditions, and none were observed. The occurrence of such effects prior

to the study cannot be entirely ruled out. However, in such a case, it would be extremely unlikely that the scores of one experiment group are more affected than another, because of the random allocation of participants to groups.

- *Maturation effects* refer to the impact the passage of time has on an individual. Like history effects, maturation effects are rather problematic in long-term studies. Since the duration of the experiment was short, maturation effects are considered to be of minor importance, and none were observed.

- *Testing effects* comprise learning effects and experimental fatigue. *Learning effects* were avoided by testing each person only once. *Experimental fatigue* is concerned with events during the experiment that exhaust the participant either physically or mentally. The short time frame of the experiment session limits chances of fatigue. Neither were any signs of fatigue observed nor were there any reports by participants indicating fatigue.

- *Instrumental bias* occurs if the measuring instrument (i.e., a physical measuring device or the actions/assessment of the researcher) changes over time during the experiment. Since the answers given in the experiment tasks were evaluated manually, this is a serious threat to validity. It might be the case that the experience gained in scoring some answers had an influence on subsequent evaluations. This threat was mitigated by evaluating the results in no specific prescribed order by multiple authors, and in case of substantial differences in grading, a discussion took place until consensus was achieved.

- *Selection bias* is present if the experimental groups are unequal before the start of the experiment (e.g., severe differences in previous experience). Selection bias is likely to be more threatening in quasi-experimental research. By using an experimental design with the fundamental requirement to randomly assignment participants to the different groups of the experiment, it became possible to avoid selection bias to a large extent. In addition, the investigation of the composition of the groups did not reveal any major differences between them. (cf. Section 7.4.2).

- *Experimental mortality* more likely occurs in long-lasting experiments, since the chances for dropouts increase (e.g., participants leaving the town). Due to the short time frame of this study, experimental mortality did not occur.

- *Diffusion of treatments* is present if at least one group is contaminated by the treatments of at least one other group. Since the participants share the same social group, and they are interacting outside the research process as well, a cross-contamination between the groups cannot be entirely ruled out.

- *Compensatory rivalry* is present if participants of a group put in extra effort when the impression arises that the treatment of another group might lead to better results than their own treatment. This threat was mitigated by clarifying that different degrees of difficulty will be considered and compensated in the calculation of bonus points.

- *Demoralization* could occur if a participant is assigned to a specific group that she/he does not want to be part of. No indications of demoralization such as increased dropout rates or complaints regarding group allocation were observed.

- *Experimenter bias* refers to undesired effects on the dependent variables that are unintentionally introduced by the researcher. All participants received a similar preparation for the experiment and worked on the same set of tasks. A manual evaluation of the given answers regarding their correctness was performed. To mitigate the threat of experimenter bias in that regard, the first, second, and third author performed the evaluation of all tasks individually. Differentiating between semantic and syntactic correctness overall simplified the evaluation process by enabling a separation of concerns. A potential threat in that regard could be falsely classifying defects. Therefore, after the completion of all individual evaluations, in case of substantial differences in grading, a discussion took place until consensus was achieved.

Threats to External Validity

The external validity of a study focuses on its generalizability. In the following, potential threats that hinder a generalization are discussed. Different types of generalizations must be considered:

- *Generalizations across populations*: By statistical inference, generalizations from the sample to the immediate population are made. The initial study design considered two populations, namely computer science students that enrolled in the SE2 course as proxies for novice software engineers, designers or developers, as well as computer science students that enrolled in the ASE course as proxies for moderately advanced software engineers, designers or developers. Due to a sufficient number of participants with industry experience, it became possible to consider a third population, namely participants with industry experience, who function as proxies for industrial practitioners with basic to modest industry experience. The results of this study show interesting discrepancies between these populations. In particular, there are no significant differences in understandability between PSP and EPL for more advanced users, while a significant difference is measurable among less experienced users. In general, this study does not intend to claim generalizability to other populations without further empirical evidence. For example, it

might be plausible that leading experts, working in the software industry, or as business administrators, perform similarly to ASE participants or the subset of participants with industry experience, but this study can neither support nor reject such claims.

- *Generalizations across treatments*: The treatments are equivalent to specific tested languages. Treatment variations would likely be related to changing the content, amount, or difficulty of experiment tasks, or the amount of preparation provided. The experiment design attempts to be as general as possible by using compliance specifications stemming from different domains and applying a moderate amount of training.

- *Generalizations across settings/contexts*: The participants of this study are students who enrolled computer science courses at the University of Vienna, Austria. The majority of the students are Austrian citizens, but there is a large presence of foreign students as well. Surely, it would be interesting to repeat the experiment in different settings/context to evaluate the generalizability in that regard. For example, repeating the experiment with native English-speakers might lead to different and presumably better results.

- *Generalizations across time*: It is hard to foresee whether the results of this study will hold over time. For example, if teaching of a specific tested language is intensified in the computer science curricula at the University of Vienna, then the students would bring in more expertise, which likely would have an impact on the results.

Threats to Construct Validity

There are potential threats to the validity of the construct that must be discussed:

- *Inexact definition & Construct confounding*: This study has a primary focus on the *understandability* construct, which is measured by the dependent variables *syntactic correctness*, *semantic correctness* and *response time*. This construct is exact and adequate, and the dependent variables *syntactic correctness* and *semantic correctness* make even a more fine-grained analysis possible than in existing studies that measure correctness by a single variable (cf. Feigenspan et al. [143] and Hoisl et al. [144]).

- *Mono-method bias*: Due to organizational reasons, keeping time records was the personal responsibility of each participant. The participants were carefully instructed how to record start and end times, and we did not detect any irregularities (e.g., overlapping time frames or long pauses) in those records. Nonetheless, this measuring method leaves room for measuring errors, and an additional or alternative measuring method (e.g., direct observation by experimenters or performing the experiment with an online tool that

handles record keeping) would reduce this threat. However, these methods would have influenced the overall study design and potentially could have introduced other threats to validity (e.g., prolonged experiment execution potentially leading to an exposure of the experiment task contents or technical problems during experiment execution). To avoid mono-method bias in evaluating the syntactic and semantic correctness, the grading was not performed by a single but by three experimenters individually.

- *Reducing levels of measurements*: Both correctness variables and the response time are continuous variables. That is, the levels of measurements are not reduced. The Likert scales used in this study offer 5 answer categories rather than 7 or 11, because the latter mentioned would produce data of lower quality according to Revilla et al. [168].
- *Treatment-sensitive factorial structure*: In some empirical studies a treatment might sensitize participants to develop a different view on a construct. The actual level of understandability based on the task solutions provided was measured, so the participants' view on this construct appears to be irrelevant.

Threats to Content Validity

Content validity is concerned with the relevance and representativeness of the elements of a study for the measured construct:

- *Relevance*: The tasks of this study are based on realistic scenarios stemming from three different domains in which compliance is highly relevant (cf. Elgammal et al. [23], Rovani et al. [14], and United States Environmental Protection Agency [13]).
- *Representativeness*: In the formal modeling of the compliance specifications, the use of all core temporal LTL operators and EPL operators was required, which means that the construct understandability was measured comprehensively. The use of each PSP pattern was required two or more times (cf. sample solutions of experimental tasks in Appendix A.1). Unfortunately, it was not possible to test all available pattern-scope combinations. However, the majority of specifications are based on the *global* scope (cf. Dwyer et al. [27], [73]), which is as well reflected in the realistic specifications used in the tasks of this experiment (cf. experimental tasks in Table 7.2 and sample solutions in Appendix A.1). That is, a representative subset of PSP was tested.

Threats to Conclusion Validity

Thorough statistical investigations of model assumptions were performed before applying the most suitable statistical test with the greatest statistical power, given the properties of the ac-

quired data. That course of action is considered to be highly beneficial to the conclusion validity of this study. The decision to retain outliers might be a threat to conclusion validity, but all outliers appear to be valid measurements, so deleting them would pose a threat to conclusion validity as well.

7.6. Related Work

We are not aware of any empirical studies evaluating the understandability related to the formal modeling of compliance specifications in particular. There exists, however, related work focusing on similar issues.

Related studies in the field of business process management are concerned with declarative workflows (cf. van der Aalst [171]), which use graphical patterns with underlying formal representations in LTL (cf. Montali [58]) or event calculus (cf. Montali et al. [57]). Haisjackl & Zugal [28] investigated differences between textual and graphical declarative workflows in an empirical study with 9 participants. The descriptive statistics of this study indicate that the graphical representation is advantageous in terms of perceived understandability, error rate, duration, and mental effort. The lack of hypothesis testing and the small number of participants are severe threats to the validity of this study. Zugal et al. [172] investigated the understandability of hierarchies on basis of the same data set. The results of their research indicate that hierarchies must be handled with care. While information hiding and improved pattern recognition are considered to be positive aspects of hierarchies since the mental effort for understanding a process model is lowered, the fragmentation of processes by hierarchies might lower overall understandability of the process model. Another important finding of their study is that users appear to approach declarative process models in a sequential manner even if the user is definitely not biased through previous experiences with sequential/imperative business process models. They conclude that the abstract nature of declarative process models does not seem to fit the human way of thinking. Moreover, they observe that the participants of their study tried to reduce the number of constraints to consider by putting away sheets that describe irrelevant sub-process or by using their hand to hide parts of the process model that were irrelevant. As with the previously discussed study, it must be assumed that the validity of this study is strongly limited by the extremely small sample size. Haisjackl et al. [173] investigated the understanding of declarative business process models, again on the same data set. As in the previously mentioned study, they point out that users tend to read such models sequentially, despite the declarative nature of the approach. The larger a model, the more often hidden dependencies were overlooked, which indicates that increasing numbers of constraints lower overall understanding. Moreover, they report that while individual constraints are overall well understood, there seem to be problems with

understanding the *precedence* constraint. As the authors point out, this kind of confusion could be related to the graphical arrow-based representation of the constraints, where subtle differences determine the actual meaning. That is, the arrow could be confused with a sequence flow as present in flow-driven, sequential business processes. As previously stated for the other two studies that are based on the same data set, the validity of this study is possibly strongly affected by the small sample size. De Smedt et al. [174] tried to improve the understandability of declarative business process models by explicitly revealing hidden dependencies. They conduced an experiment with 95 students. The result suggests that explicitly showing hidden dependencies enables a better understandability of declarative business process models. Pichler et al. [175] compared the understandability of imperative and declarative business process modeling notations. The results of this study are in line with Zugal et al. [172] and suggest that imperative process models are significantly more understandable than declarative models, but the authors also state that the participants had more prior experience with imperative process modeling than with declarative process modeling. The small sample size (28 participants) is a threat to validity of this study. Rodrigues et al. [176] compared the understandability of textual and graphical BPMN [20] business process descriptions with 32 students and 41 practitioners. They conclude that experienced users understand a process better if it is presented by a graphical BPMN process model, whereas for inexperienced users there is no difference in understandability between the textual and graphical process descriptions. Jost et al. [177] compared the intuitive understanding of process diagrams with 103 students. They conclude that UML activity diagrams provide a higher level of understandability than BPMN diagrams and EPCs.

Software architecture compliance, which focuses on the alignment of software architecture and implementation, and requirements engineering are also related to this study. Czepa et al. [131] compared the understandability of three languages for behavioral software architecture compliance checking, namely the Natural Language Constraint language (NLC), the Cause-Effect Constraint language (CEC), and the Temporal Logic Pattern-based Constraint language (TLC), in a controlled experiment with 190 participants. NLC simply refers to using the English language for documenting software architectures. CEC is a high-level structured architectural description language that abstracts EPL. It supports the nesting of cause parts, which observe an event stream for a specific event pattern, and effect parts, which can contain further cause-effect structures and truth value change commands. TLC is a high-level structured architectural description language based on PSP. Interestingly, the statistical inference of this study suggests that there is no difference in understandability of the tested languages. This could indicate that the high-level abstractions employed bring those structured languages closer to the understandability of unstructured natural language architecture descriptions. Moreover, it might also suggest that natural language leaves more room for ambiguity, which is detrimental to its under-

standing. Potential limitations of that study are that its tasks are based on common architectural patterns/styles (i.e., a participant possibly recognizes the meaning of a constraint more easily by having knowledge of the related architectural pattern) and the rather small set of involved patterns (i.e., only very few patterns of PSP were necessary to represent the architecture descriptions). A controlled experiment carried out by Heijstek et al. [29] with 47 participants focused on finding differences in understanding of textual and graphical software architecture descriptions. Interestingly, participants who predominantly used textual architecture descriptions performed significantly better, which suggests that textual architectural descriptions could be superior to their graphical counterparts. An eye-tracking experiment with 28 participants by Sharafi et al. [30] on the understandability of graphical and textual software requirement models did not reveal any statistically significant difference in terms of correctness of the approaches. The study also reports that the response times of participants working with the graphical representations were slower. Interestingly though, the participants preferred the graphical notation. Hoisl et al. [144] conducted a controlled experiment on three notations for scenario-based model tests with 20 participants. In particular, they evaluated the understandability of a semi-structured natural language scenario notation, a diagrammatic scenario notation, and a fully-structured textual scenario notation. According to the authors, the purely textual semi-structured natural language scenario notation is recommended for scenario-based model tests, because the participants of this group were able to solve the given tasks faster and more correctly. That is, the study might indicate that a textual approach outperforms a graphical one for scenario based model test, but the validity of the experiment is limited by the small sample size and the absence of statistical hypothesis testing.

7.7. Conclusion and Future Work

The main goal of this empirical study was testing and comparing the understandability of representative approaches for the formal modeling of compliance specifications. The experiment was conducted with 215 participants in total. Major differences were found especially in semantic correctness of the approaches. Since formalizations in the Property Specification Patterns (PSP) were overall more correct than in Linear Temporal Logic (LTL) and Event Processing Language (EPL), there is evidence that the pattern-based PSP approach provides a higher level of understandability. More advanced users, however, seemingly are able to cope equally well with PSP and EPL. That is, for more advanced users, these approaches can be used interchangeably as fitting best to a concrete domain or task. Moreover, EPL provides a higher level of understandability than LTL. Therefore, EPL is well-suitable in situations that demand runtime verification in which the set of available patterns in PSP is not sufficient to model a compliance specification

or to aid the creation of new patterns with underlying LTL formalizations (cf. Chapter 5).

Moreover, the results are overall in line with two controlled experiments with 216 participants in total on the understandability of already existing formal specifications in LTL, EPL, and PSP (cf. Chapter 6). In contrast to the current study, which focuses on the formal modeling of compliance specifications, no major differences between novice and moderately advanced users were found in understandability of existing specifications. Interestingly, the response times between the experimental groups were significantly different in most cases, an effect which appears to be absent during modeling.

Opportunities for further empirical research are the consideration of an extended set of representations including, for example, event calculus (cf. Kowalski & Sergot [178]) or Declare (cf. Pešić & van der Aalst [121]) and studying the understandability construct in different settings with other user groups (e.g., business administrators or professional software engineers). Moreover, besides the understandability construct, additional metrics such as changeability (i.e., "Is one representation easier to change when taking new/amended compliance specifications into account?") and verifiability (i.e., "Are there differences between the representations when it comes to assessing whether a given compliance specification is fully covered?") could be investigated.

8. On the Understandability of Graphical and Textual Pattern-Based Behavioral Constraint Representations

Unlike the Property Specification Patterns, which were examined in the controlled experiments presented in the two previous chapters and display a high level of understandability, pattern-based behavioral constraints are not necessarily textual, but can be represented graphically. The controlled experiments presented in two previous chapters show a high level of understandability for textual pattern-based behavioral constraints. However, pattern-based behavioral constraints can also be represented graphically.

In this chapter, we investigate the understandability of graphical and textual pattern-based behavioral constraint representations on the basis of already given constraints. This chapter reports a controlled experiment with 116 participants on the understandability of representative graphical and textual pattern-based behavioral constraint representations from the viewpoint of novice software designers. In particular, the subjects of this study are graphical and textual behavioral constraint patterns present in the declarative business process language *Declare* and textual behavioral constraints based on *Property Specification Patterns*. In addition to measuring the understandability construct, this study assesses subjective aspects such as perceived difficulties regarding learning and application of the tested approaches.

An interesting finding of this study is the overall low correctness achieved in the experimental tasks, which seems to indicate that pattern-based behavioral constraint representations are hard to understand for novice software designers in the absence of additional support means. The results of the descriptive statistics regarding achieved correctness are slightly in favor of the textual representations, but the inference statistics do not indicate any significant differences in terms of understandability between graphical and textual behavioral constraint representations.

8.1. Introduction

Since the early days of computer science, supporting the correctness of computer programs has been a recurring research interest. In 1977, Pnueli introduced an approach for the verification of

sequential and parallel programs that is based on temporal reasoning [26]. His approach became widely popular under the term *Linear Temporal Logic (LTL)*. A plethora of different temporal logics have been proposed since then. For example, in 1988, Clarke and Emerson [41] applied the *Computation Tree Logic (CTL)*, a branching time logic, for model checking of computer programs. Both LTL and CTL are still very popular and supported as a specification language by many of today's model checkers (e.g., NuSMV by Cimatti et al. [35] and SPIN by Holzmann [119]).[1]

In 1998, Dwyer et al. [27], [73] proposed the *Property Specification Patterns (PSP)*, a pattern-based approach that abstracts underlying LTL, CTL, or other formal logic formulas.[2] As the main reason for collecting the patterns, the authors state that practitioners had been reluctant to apply formal methods due to unfamiliarity with specification processes, notations, and strategies. Consequently, the pattern catalog was primarily meant to enable ease of reuse of existing patterns. The proposed set of patterns was evaluated against 555 specifications from more than 35 different sources, and 92.1% (511) of the considered specifications are covered by the proposed set of patterns. A survey by Bianculli et al. [109] based on 104 scientific case studies reproduced these results.

Numerous studies make use of the PSP approach either directly or by extending the original idea of pattern-based behavioral constraints. In the following, we will discuss a selection of them to emphasize the importance and applicability of PSP for various purposes and to introduce the *Declare* approach (cf. Pešić et al. [121]), a popular graphical behavioral constraint approach, which has been greatly inspired by PSP.

Corbett et al. [179] apply PSP for the verification of Java programs. PSP can be used for modeling requirements in requirements engineering (see e.g., Cheng & Atlee [180]). Hatcliff et al. [181] apply PSP for the verification of component-based systems. Krismayer et al. [182] propose an approach that mines constraints from event logs on the basis of PSP. Li et al. [183] use a structured textual specification language based on PSP for the behavioral verification of webservices at runtime. Wong and Gibbons [184] apply a superset of PSP to construct behavioral properties of BPMN (Business Process Model and Notation) [20] models. Namiri and Stojanovic [185] propose a PSP-based approach for modeling internal controls that are required by regulations (e.g., Sarbanes-Oxley Act of 2002) for business process compliance. Elgammal et al. [23] adopted some of the PSP in the *Compliance Request Language (CRL)*. Dou et al. [186] extended the Object Constraint Language (OCL) with support for temporal constraints based on PSP. The *PROPEL (PROPerty Elicitation)* approach by Smith et al. [187] provides support for the specification of PSP-based constraints using two different notations, namely an

[1] `http://nusmv.fbk.eu`, `http://spinroot.com`
[2] `http://patterns.projects.cs.ksu.edu`

extended finite-state automaton representation and a structured natural language representation. The *PROPOLS* approach for the verification of BPEL service compositions schemas (cf. Yu et al. [66]) is based on PSP as well. Awad et al. [188] propose a PSP-based visual language, called *BPMN-Q*, to express compliance requirements with visual shapes that are similar to those used in imperative business process modeling.

Both the graphical declarative workflow approach *Declare* (formerly known as *ConDec*) by Pešić et al. [121] and the graphical *DecSerFlow (Declarative Service Flow Language)* approach by van der Aalst and Pešić [42] were strongly inspired by the PSP approach.[3] *Declare* appears to be the most wide-spread graphical behavioral constraint approach in business process management (cf. Goedertier et al. [189], Schonenberg et al. [4], and van der Aalst et al. [190]), and its abstractions are generic constraint language abstractions. That is, it can be seen as representative for, and generalized to, a broad set of possible other graphical behavioral constraint languages.

8.1.1. Problem Statement

Behavioral constraint approaches are highly relevant in various domains, such as healthcare (cf. Rovani et al. [14]), banking (cf. Bianculli et al. [109]), the automotive industry (cf. Post et al. [110]), software architecture (cf. Czepa et al. [131]), and business process management (cf. Elgammal et al. [23]), to name but a few. Pattern-based behavioral constraints can be used to shield the user from the complexity of formal temporal logics used in the context of formal verification methods such as model checking (cf. Rozier [90] and Baier & Katoen [141]) and runtime monitoring by nondeterministic finite automata (cf. De Giacomo et al. [99], [120]). Many textual and graphical pattern-based behavioral constraint approaches exist (e.g., [23], [42], [66], [188]) which originated from PSP. However, current studies predominantly focus on technical contributions in specific application areas. Only a few studies focus on empirical evaluations of behavioral constraint representations [131], [172]–[175], [191], and even fewer are concerned with comparing graphical and textual behavioral constraint representations specifically [28], [192]. Interestingly, the body of existing studies (cf. Section 8.8, which discusses those related works in depth) on this topic yields contradictory results, which indicates that the understandability of pattern-based graphical and textual behavioral constraints is not yet well understood. Two prior empirical studies (cf. Chapter 6) indicated that the pattern-based PSP representation provides a high level of understandability (about 70% on average in the specific setup of those studies), but these studies did not consider graphical pattern-based behavioral constraints. Since our experience from multiple industry projects shows that industry experts in areas such as business process management tend to prefer graphical over textual constraint

[3] `http://www.win.tue.nl/declare/2011/11/declare-renaming/`

representations when given the choice, it would be important to test if their "gut feeling" can be empirically confirmed. Also, non-expert users seem to prefer graphical models over structured text and textual descriptions when the goal is to understand a process (cf. Figl & Recker [193]). It is yet unknown whether there are differences in understandability between graphical and textual pattern-based behavioral constraint approaches. In addition, it is unknown whether there exist problematic language elements that pose an obstacle for correct understanding of textual and graphical pattern-based behavioral constraint representations. The discovery of such problematic elements could provide a starting point for improving the comprehensibility of the representations and making them more applicable in practice.

Studying the understandability of graphical and textual behavioral constraints is not only interesting from a purely scientific point of view, but is also important for industrial applications. For example, from the collaboration with our industry partners (see e.g., [79]), their customers, and other company representatives at conferences and workshops, we realized that the industry has a huge demand for, and shows a strong interest in, behavioral constraint approaches that are applicable in practice for supporting the comprehensible, fast and accurate adoption of compliance requirements, as well as their automated enactment and verification. The pattern-based behavioral constraint representations that we study in this chapter are well suited for automated computer-aided verification at runtime and design time, but vendors are still often reluctant to expose their customers to such approaches. Our discussions with industry partners (see e.g. [75], [128]) indicate uncertainty regarding how understandable the constraints are, and this could be among the reasons for this reluctance.

In addition to triggering further empirical evaluations and thus new insights, empirical research on behavioral constraints has the potential to influence practitioners in decision-making for adopting a specific behavioral constraint language and in designing future industrial solutions. Consequently, a farther-reaching goal for our research on behavioral constraint representations is to pave the way for their future industrial or practical exploitation.

8.1.2. Research Objectives

This empirical study has the objective to investigate the *understandability* of representative graphical and textual behavioral constraint representations. The understandability construct focuses on how well (in terms of *correct understanding*) and fast (in terms of the *response time*) a participant understands a given behavioral constraint representation. Particularly, this empirical study considers the *Property Specification Patterns*, which are the origin of numerous existing behavioral constraint approaches (cf. Section 8.1), and the *Declare* approach, which seems to be the most popular graphical behavioral constraint approach in the field of business process management (cf. Goedertier et al. [189], Schonenberg et al. [4], and van der Aalst et al. [190]).

We are not aware of any other graphical behavioral constraint language of comparable significance. Originally, *Declare* was proposed in the domain of business process management (cf. Pešić & van der Aalst [56]) and also applied in service-oriented computing (cf. van der Aalst & Pešić [42]), but there seem to be no limiting factors for the application of *Declare* in different domains. Its graphical pattern-based representation is versatile and transformable to underlying formal representations (e.g., LTL [58] and event calculus [57]) for verification at design time (i.e., model checking) and runtime verification in general. *Declare* is considered in three variants, namely as a purely graphical, a purely textual, and a hybrid (mixed graphical/textual) behavioral constraint approach.

We state the experimental goal using the GQM (Goal Question Metric) goal template (cf. Basili et al. [132]) as follows:

Analyze the textual Property Specifications Patterns (PSP) based representation approach, the purely graphical *Declare* representation approach (DG), the purely textual *Declare* representation approach (DT), and the hybrid (i.e., showing a textual label in addition to the graphical relation) *Declare* representation approach (DGT)
for the purpose of their evaluation
with respect to their understandability
from the viewpoint of the novice software designer
in the context (i.e., environment) of the Distributed System Engineering and the Software Engineering 2 courses at the Faculty of Computer Science of the University of Vienna in the winter term 2017.

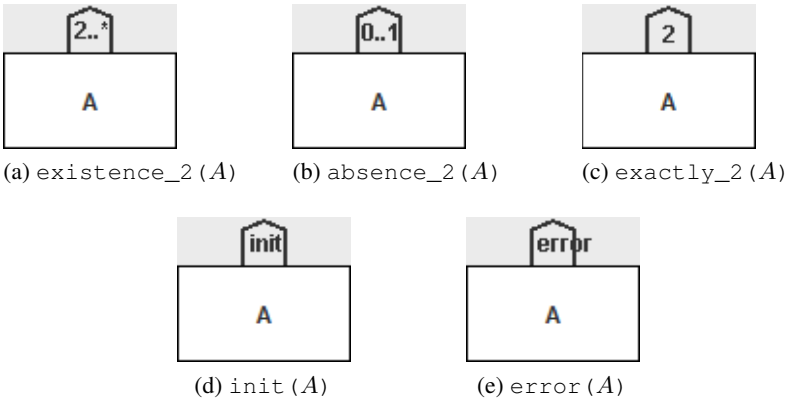
8.1.3. Guidelines

This work follows existing guidelines for conducting, evaluating, and reporting empirical studies (cf. Section 6.1.4).

8.2. Background on Pattern-Based Behavioral Constraint Representations

8.2.1. Property Specification Patterns

For an introduction to the *Property Specification Patterns (PSP)*, we refer to Section 6.2.1.



(a) `existence_2(`A`)` (b) `absence_2(`A`)` (c) `exactly_2(`A`)`

(d) `init(`A`)` (e) `error(`A`)`

Figure 8.1.: Graphical representations of *Existence* patterns in Declare

8.2.2. Declare

Declare (cf. Pešić & van der Aalst [121]), also known by the names *DecSerFlow* (cf. van der Aalst & Pešić [42]) and *ConDec* (cf. Pešić & van der Aalst [56]), is a graphical declarative business process modeling language and approach. There exist transformations of its high-level graphical representations to *Linear Temporal Logic (LTL)* (cf. Pnueli [26] and Montali [58]) and *Event Calculus (EC)* (cf. Kowalski & Sergot [178] and Montali et al. [57]). As of *Declare Version 2.1.0*, the available constraint templates are organized as follows:[4]

- Existence Patterns (cf. Figure 8.1 for graphical representations):
  - "at least"
    - `existence_`n`(`A`)`: State A must occur at least n times.
  - "at most"
    - `absence_`n`(`A`)`: State A must occur at most $n - 1$ times.
  - "exactly"
    - `exactly_`n`(`A`)`: State A must occur exactly n times (i.e., not more, not less).
  - "position"
    - `strong_init(`A`)`: A must start and complete first.
    - `init(`A`)`: A must start first, and it must complete first or remain active indefinitely.

[4] `http://www.win.tue.nl/declare/download/`

(a) `responded_existence(`A`, `B`)`

(b) `co-existence(`A`, `B`)`

(c) `response(`A`, `B`)`

(d) `precedence(`A`, `B`)`

(e) `succession(`A`, `B`)`

Figure 8.2.: Graphical representations of *Relation* patterns in Declare

(a) `alternate_response(`A`, `B`)`

(b) `alternate_precedence(`A`, `B`)`

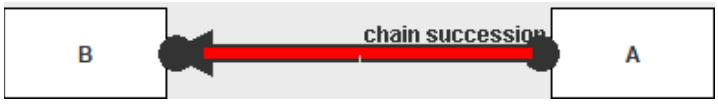
(c) `alternate_succession(`A`, `B`)`

(d) `alternate(`A`, `B`)`

Figure 8.3.: Graphical representations of *Alternate Relation* patterns in Declare

(a) `chain_response`$(A,\ B)$

(b) `chain_precedence`$(A,\ B)$

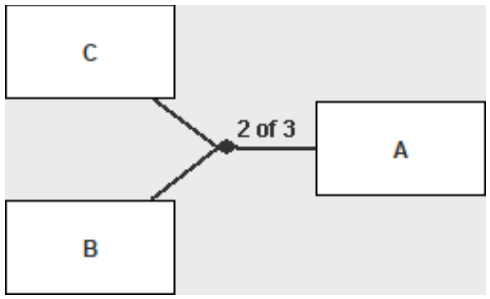


(c) `chain_succession`$(A,\ B)$

Figure 8.4.: Graphical representations of *Alternate Relation* patterns in Declare

(a) `choice`$(A,\ B)$

(b) `exclusive_choice`$(A,\ B)$



(c) `exclusive_choice_2_of_3`$(A,\ B,\ C)$

Figure 8.5.: Graphical representations of *Choice* patterns in Declare

(a) `not_co-existence(`A`, `B`)`

(b) `not_succession(`A`, `B`)`

(c) `not_chain_succession(`A`, `B`)`

Figure 8.6.: Graphical representations of *Negative Relation* patterns in Declare

    - `last(`A`)`: A must be the last occurring element. There must not occur any other element than A after A.
  - `error(`A`)`: This appears to be an auxiliary pattern to detect a completion of A that should not occur if A has never been started.

- Relation Patterns:
  - "no order" (cf. Figure 8.2 (a&b) for graphical representations)
    - `responded_existence(`A`, `B`)`: If state A happens (at least once), then state B must have happened (at least once) before state A or must happen after state A.
    - `co-existence(`A`, `B`)`: If state A happens (at least once), then state B must have happened (at least once) before state A or must happen after state A, and vice versa.
  - "order" (cf. Figure 8.2 (c-e) for graphical representations)
    - "simple"
      - `response(`A`, `B`)`: Whenever state A happens, state B must occur afterwards eventually.
      - `precedence(`A`, `B`)`: The occurrence of state A is a precondition for state B. State B is only allowed to happen if state A has happened already.
      - `succession(`A`, `B`)`: Whenever state A happens, state B must occur afterwards eventually. The occurrence of state A is a precondition for state

B. State B is only allowed to happen if state A has happened already. That is, this pattern is a combination of `response` and `precedence`.

* "alternate" (cf. Figure 8.3 for graphical representations)
  - `alternate_response(`A`, `B`)`: Whenever state A happens, state B must occur afterwards eventually, but A is not allowed to occur a second time until then.
  - `alternate_precedence(`A`, `B`)`: A must occur before the first B, then the occurrence of another A is the precondition for the next B, and so forth.
  - `alternate_succession(`A`, `B`)`: This pattern is a combination of `alternate_response` and `alternate_precedence`.
  - `alternate(`A`, `B`)`: After A there must not be another A indefinitely or until B occurs
* "chain" (cf. Figure 8.4 for graphical representations)
  - `chain_response(`A`, `B`)`: Whenever state A happens, state B must occur next.
  - `chain_precedence(`A`, `B`)`: B can only be executed directly after A.
  - `chain_succession(`A`, `B`)`: This pattern is a combination of `chain_response` and `chain_precedence`.

- Choice Patterns (cf. Figure 8.5 for graphical representations):
  - "simple"
    * `choice(`A`, `B`)`: State A or state B must occur. That is, either of them occurring alone would satisfy this constraint, but both may occur anyway. For example, the traces [A, B], [A], [B] would satisfy this constraint while an empty trace [] would cause a violation.
    * `choice_`n`_of_`N`(list[`N`])`: This pattern is the generalization of `choice` where $n, N \in \mathbb{N}$, $N \geq 2$, and $1 \leq n < N$. For example, the traces [A, B, C], [A, B], [A, C], [B, C] would satisfy the constraint `choice_2_of_3`(A, B, C) while the traces [], [A], [B], [C] would cause violations.
  - "exclusive"
    * `exclusive_choice(`A`, `B`)`: State A or state B must occur, but not both. That is, either of them occurring alone would satisfy this constraint, and the

constraint would be violated if both of them occur. For example, the traces [A], [B] would satisfy this constraint while the traces [], [A, B] would cause violations.

* `exclusive_choice_`n`_of_`N`(list[`N`])`: Generalization of the pattern `exclusive_choice` where $n, N \in \mathbb{N}$, $N \geq 2$, and $1 \leq n < N$. For example, the traces [A, B], [A, C], [B, C] would satisfy `exclusive_choice_2_of_3`(A, B, C) while the traces [], [A], [B], [C], [A, B, C] would cause violations.

- Negative Relation Patterns (cf. Figure 8.6 for graphical representations):
  - "no order"
    * `not_co-existence(`A`, `B`)`: Either state A or state B can occur, but not both.
  - "order"
    * `not_succession(`A`, `B`)`: Before state B there cannot be state A and after state A there cannot be state B.
  - "chain"
    * `not_chain_succession(`A`, `B`)`: A and B must not occur next to each other in this order.

8.3. Experiment Planning

8.3.1. Goals

The primary goal of the experiment is measuring the *understandability* construct of graphical and textual pattern-based behavioral constraint representations by the *correctness* and *response time* of the answers given by the participants.

Additionally, the experiment aims at studying the perceived learning difficulty, the perceived difficulty of applying the learned behavioral constraint representation approach (i.e., the perceived application difficulty), the personal interest in using the representation, the perceived practical applicability, and the perceived potential for further improvement of the behavioral constraint representations.

8.3.2. Experimental Units

All 116 participants were students at the University of Vienna, Austria, who enrolled in the courses "Distributed System Engineering Lab (DSE)" and "Software Engineering 2 (SE2)" in the winter term 2017.

This study aims to evaluate the understandability of pattern-based behavioral constraints from the perspective of novice software designers, which makes undergraduate students suitable test subjects. The attendance was optional and rewarded by extra credits (i.e., bonus points) for the course based on performance in the experiment (i.e., the achieved correctness and the completeness of time records). Alternatively, the students were given the chance to gain extra credits in other lab activities by going beyond the normal course requirements (e.g., by implementing more functionalities than required, or by paying attention to excellent code quality). As required for a valid controlled experiment setup, all participants were randomly allocated to the four experiment groups (i.e., one for each of the four notations being studied).

8.3.3. Experimental Material & Tasks

In total, three documents were used per representation:

- An *info sheet* about the assigned behavioral constraint representation was made available to the participants one week before the experiment execution for preparation purposes. The descriptions used in these documents are based on the pattern descriptions provided by *Declare* and the *Property Specification Patterns*.[5] To keep the number of language elements to remember low, the experiment design considers limitations in human capacity for processing information (cf. Miller [194]). That is, the info sheets of all groups were limited to introducing at most nine language elements. The experiment itself was similar to a closed book exam, so no additional means of assistance were allowed. This step was taken to ensure unbiased testing of the participants' understanding of the textual terms and graphical shapes of a notation under the exclusion of potential effects resulting from looking up graphical shapes or textual terms.

- A *question sheet* consisting of general questions on the background of the participant (age, gender, level of education, years of work experience, etc.), the experimental tasks, and a Likert scale-based questionnaire to gain insights on how the different representations are subjectively perceived (e.g., perceived learning difficulty) was handed out at the beginning of the experiment session.

[5] `http://www.win.tue.nl/declare/`, `http://patterns.projects.cs.ksu.edu/`

- An *answer sheet* accompanied the question sheet for marking the answers to the questions of the experimental tasks. This document made an automated evaluation by the e-learning platform *Moodle* possible.[6]

For the creation of the tasks of the experiment, we used an algorithm that randomly generates traces and computes the correct truth value of a constraint (i.e., fitting to the trace) automatically. The implementation makes use of the Event Processing Language (EPL) [104] to encode the behavioral constraint patterns in the Complex Event Processing (CEP) engine *Esper*.[7] Truth values were automatically randomly altered to another truth value to create both wrong and correct answer choices. After that automated generation of the task, we manually checked each answer choice to make sure that correct and incorrect answer choices will be treated in the right way (i.e., wrong answer choices are treated as incorrect and correct answer choices are indeed treated as correct) during the automated processing by Moodle.

In total, there were 18 experimental tasks, each consisting of a behavioral constraint, and the instruction to select the correct answers in the answer sheet and to keep time records. Per task six multiple choice answer options were available, each of them consisting of an execution trace and a (correct or incorrect) truth value. For each option the participant had to decide whether it is correct or incorrect (i.e., whether the truth value is correct for the given trace). Figure 8.7 shows the first task for each of the four groups, which is based on the Succession pattern. Please note that the instruction text and the table for time tracking is only shown in Figure 8.7 (a) and omitted in (b), (c), and (d). In case a participant works on a task several times, the time tracking table offers not just a single column, but four columns with four separate start and end times. Instead of letters that may be suggestive of a sequence of events based on the alphabetical order (after "A" comes "B") of the used letters, we use the abstract concepts "space" and "time" (cf. behavioral constraints in Figure 8.7), which do not indicate any kind of chronological order. In Figure 8.7 (a), the answer choices c) and d) are correct.

When monitoring a behavioral constraint in a system at runtime, it might be the case that it is not only of interest if a specification is satisfied or violated, but also whether further state changes that could resolve or cause a violation of a specification are possible. That is, the state of a specification can be either temporary (i.e., the state may change) or permanent (i.e., the state may not longer change). Consequently, to enable a more fine-grained analysis of the participants' understanding of behavioral constraints in the experiment, we employ the concept of runtime states (cf. Bauer et al. [100], [195]) which support four truth value states. In particular, a behavioral constraint at runtime is either `temporarily satisfied`, `temporarily violated`, `permanently satisfied`, or `permanently violated`. Several existing

[6] `http://moodle.org`
[7] `http://www.espertech.com/esper/`

Table 8.1.: Realization of *Declare* patterns by combining available PSP patterns

Declare Constraint	**PSP Constraint**
choice(A, B)	A occurs or B occurs
exclusive_choice(A, B)	(A occurs and B never occurs) or (B occurs and A never occurs)
responded_existence(A, B)	before A [B occurs] or after A [B occurs]
co-existence(A, B)	(A occurs and B occurs) or (A never occurs and B never occurs)
not_succession(A, B)	before B [A never occurs] and after A [B never occurs]
not_co-existence(A, B)	after A [B never occurs] and after B [A never occurs]

studies make use of the concept of four LTL truth value states (cf. Pešić et al. [96], De Giacomo et al. [97], Maggi et al. [98], Falcone et al.[101], Joshi et al. [102], Morse et al. [103], to name but a few).

To reduce chances of misbehavior, the order of the answer choices was randomized between the experimental groups (cf. Figure 8.7 (a)-(d)). That is, the answer choices remained the same in each group, only their order of presentation was different. Moreover, in the design of the experiment, orientation variations (i.e., the connector shapes were also presented rotated 180 degrees) in the pattern presentation (cf. Figure 8.8 and Figure 8.7 (b)) were considered since the orientation possibly has an impact on understandability. However, with regards to orientation variations, the results did not reveal any conclusive impact on understandability.

Since the *Succession* pattern is not explicitly covered in PSP, it was realized by a combination of the *Response* and *Precedence* patterns (cf. Figure 8.7 (a)). Table 8.1 summarizes other *Declare* patterns that are represented in PSP by combining available PSP patterns.

To support a replication of the study, we made the experimental material available online (cf. Czepa & Zdun [196]).

8.3.4. Hypotheses, Parameters, and Variables

Primarily, this controlled experiment focuses on the following hypotheses:

- $H_{0,1}$: There is no difference in terms of *understandability* between the representations.
- $H_{A,1}$: The approaches differ in terms of their *understandability*.

The *understandability* construct consists of two dependent variables, namely:

1) Please keep time records and select the correct answer(s) for the following constraint description:

Start Times (hh:mm:ss)					
End Times (hh:mm:ss)					Task Duration:
Durations (mm:ss)					

(space leads to time) and (space precedes time)

a) At the end of trace ***[space, time, space, space, other]*** the truth value is ***permanently violated***.
b) At the end of trace ***[space, time, time, other, other]*** the truth value is ***permanently satisfied***.
c) At the end of trace ***[space, other, space, other, time]*** the truth value is ***temporarily satisfied***.
d) At the end of trace ***[time, space, time, space, space]*** the truth value is ***permanently violated***.
e) At the end of trace ***[time, space, other, other, other]*** the truth value is ***temporarily satisfied***.
f) At the end of trace ***[other, space, time, space, other]*** the truth value is ***temporarily satisfied***.

(a) PSP group (with instructions and time record table)

a) At the end of trace ***[space, other, space, other, time]*** the truth value is ***temporarily satisfied***.
b) At the end of trace ***[time, space, other, other, other]*** the truth value is ***temporarily satisfied***.
c) At the end of trace ***[space, time, time, other, other]*** the truth value is ***permanently satisfied***.
d) At the end of trace ***[time, space, time, space, space]*** the truth value is ***permanently violated***.
e) At the end of trace ***[other, space, time, space, other]*** the truth value is ***temporarily satisfied***.
f) At the end of trace ***[space, time, space, space, other]*** the truth value is ***permanently violated***.

(b) DG group (instructions and time record table omitted)

succession(space, time)

a) At the end of trace ***[space, time, space, space, other]*** the truth value is ***permanently violated***.
b) At the end of trace ***[time, space, time, space, space]*** the truth value is ***permanently violated***.
c) At the end of trace ***[space, other, space, other, time]*** the truth value is ***temporarily satisfied***.
d) At the end of trace ***[space, time, time, other, other]*** the truth value is ***permanently satisfied***.
e) At the end of trace ***[time, space, other, other, other]*** the truth value is ***temporarily satisfied***.
f) At the end of trace ***[other, space, time, space, other]*** the truth value is ***temporarily satisfied***.

(c) DT group (instructions and time record table omitted)

a) At the end of trace ***[space, time, time, other, other]*** the truth value is ***permanently satisfied***.
b) At the end of trace ***[time, space, other, other, other]*** the truth value is ***temporarily satisfied***.
c) At the end of trace ***[space, time, space, space, other]*** the truth value is ***permanently violated***.
d) At the end of trace ***[other, space, time, space, other]*** the truth value is ***temporarily satisfied***.
e) At the end of trace ***[space, other, space, other, time]*** the truth value is ***temporarily satisfied***.
f) At the end of trace ***[time, space, time, space, space]*** the truth value is ***permanently violated***.

(d) DGT group (instructions and time record table omitted)

Figure 8.7.: Example of a task in all four group variants (more specifically: *Task 1* - based on the *Succession* pattern)

Figure 8.8.: Orientation variation in the presentation of the *Succession* pattern (i.e., the connector shape is rotated 180 degrees)

- the *correctness* achieved in trying to mark the correct answers, and
- the *response time*, which is the time it took to complete the 18 tasks.

These dependent variables are commonly used to measure the understandability construct (cf. Feigenspan et al. [143] and Hoisl et al. [144]). The independent variable (also called factor) focuses on the four behavioral constraint representations.

Secondarily, there are hypotheses that are concerned with the participants' opinion on the tested behavioral constraint representations:

- $H_{0,2}$: There is no difference in terms of *perceived learning difficulty* between the representations.
- $H_{A,2}$: The representations differ in terms of *perceived learning difficulty*.
- $H_{0,3}$: There is no difference in terms of *perceived application difficulty* between the representations.
- $H_{A,3}$: The representations differ in terms of *perceived application difficulty*.
- $H_{0,4}$: There is no difference in terms of *personal interest in using the approach* between the representations.
- $H_{A,4}$: The representations differ in terms of *personal interest in using the approach*.
- $H_{0,5}$: There is no difference in terms of *perceived practical application potential* between the representations.
- $H_{A,5}$: The representations differ in terms of *perceived practical application potential*.
- $H_{0,6}$: There is no difference in terms of *perceived improvement potential* between the representations.

- $H_{A,6}$: The representations differ in terms of *perceived improvement potential*.

8.3.5. Experiment Design

Wohlin et al. [34] and Kitchenham et al. [133] recommend using a simple experiment design that is appropriate for the goal of a study. In consequence, we applied a completely randomized design with one alternative per participant, which is both a simple design and appropriate for the stated goals (cf. Section 8.3.1). The participants were assigned to representations in an unbiased manner by using a computerized random allocation to groups.

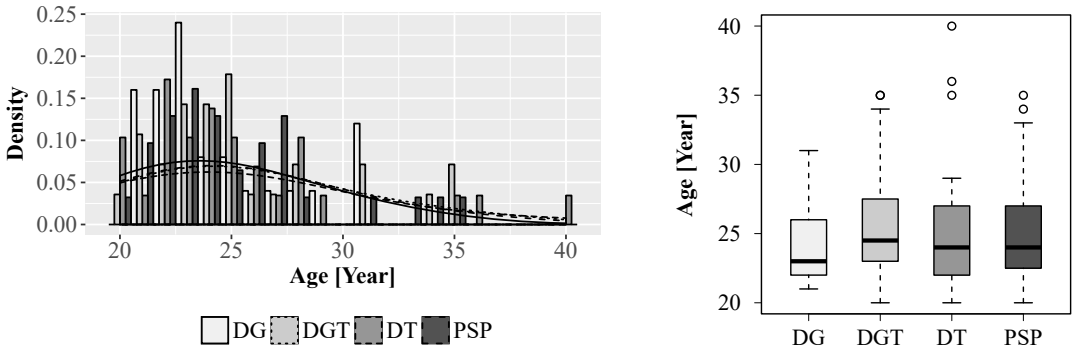
8.3.6. Procedure

In total, the experiment had a duration of 90 minutes. The experimental material, namely the question and answer sheet, was provided in the form of a printed document. The participants were informed about the procedure of the experiment with instructions on how to track time, how to mark answers correctly on the answer sheet, and a pointer to the questionnaire on the last page of the question sheet. During the whole experiment session, a clock was displayed by a projector, and the participants were instructed to write down the displayed time when starting and completing work on a task. Seating arrangements were made to limit chances for misbehavior. To limit chances for experimenter bias, the experiment was designed as a multiple-choice test that supports automated processing of the given answers by the e-learning platform *Moodle*. In case of necessary manual interventions (e.g., imprecise markings that we had to clarify), we always made use of the "four eyes principle". Moreover, the time recordings and questionnaire answers were processed manually and double-checked subsequently.

8.4. Analysis

8.4.1. Data Set Preparation

The data set was prepared as follows: We had to remove the data of two participants from the data set. One participant used an answer sheet of a different group, which may have been wrongly distributed by the experimenters. To be on the safe side, we decided not to consider this answer sheet as it might have led to confusion (e.g., the results might have accidentally been assigned to the wrong group). The experiment procedure was rigorously implemented in accordance to the planning described in Section 8.3. Unfortunately, one participant used unauthorized means of aid during the experiment, which led to the exclusion of this participant from the study. Missing values (5.6% of the dependent variables excluding correctness) were substituted by the



(a) Kernel density plot

(b) Box plot

Figure 8.9.: Participants' age per group

arithmetic mean (in case of interval scale data) and median (in case of ordinal scale data) of the data attribute per group.

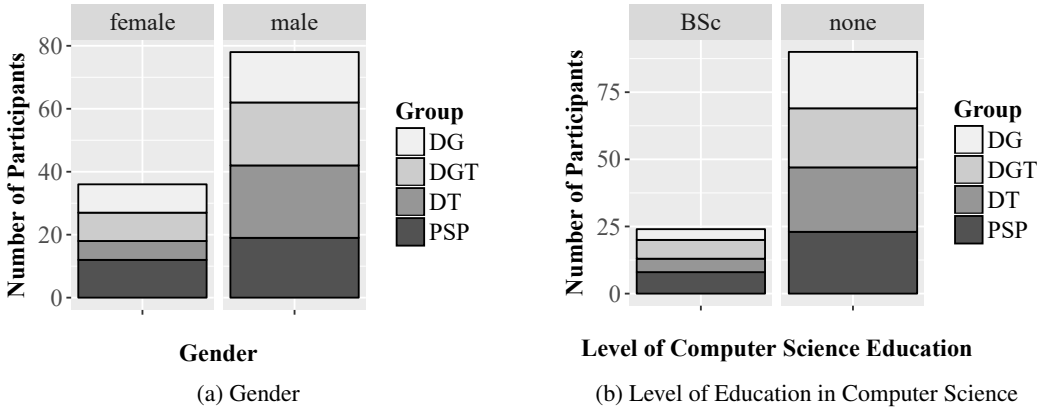
8.4.2. Analysis of Previous Knowledge, Experience and Other Features of Participants

In Figure 8.9, a kernel density plot and box plot of the participants' age per group is shown. The peak density of the participants' age is 23 years, and a high density can be found in the range between 21 and 25 years (cf. Figure 8.9 (a)). Only very few participants are older than 28 years (cf. Figure 8.9 (a)), some of them are shown as outliers in the box plot (cf. Figure 8.9 (b)). The graphical inspection of the data indicates no major differences in the age distribution between the groups. Neither do statistical significance tests indicate any significant differences between the experiment groups (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).

Figure 8.10 (a) shows a bar chart of the participants' gender distribution. In total, there were 36 female (31.6%) and 78 male participants (68.4%). Inside of the groups, the gender distribution was as follows:

- DG: 9 female (36%) and 16 male participants (64%)
- DGT: 9 female (31%) and 20 male participants (69%)
- DT: 6 female (20.7%) and 23 male participants (79.3%)
- PSP: 12 female (38.7%) and 19 male participants (61.3%)

No significant differences were found in gender distribution (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).



(a) Gender

(b) Level of Education in Computer Science

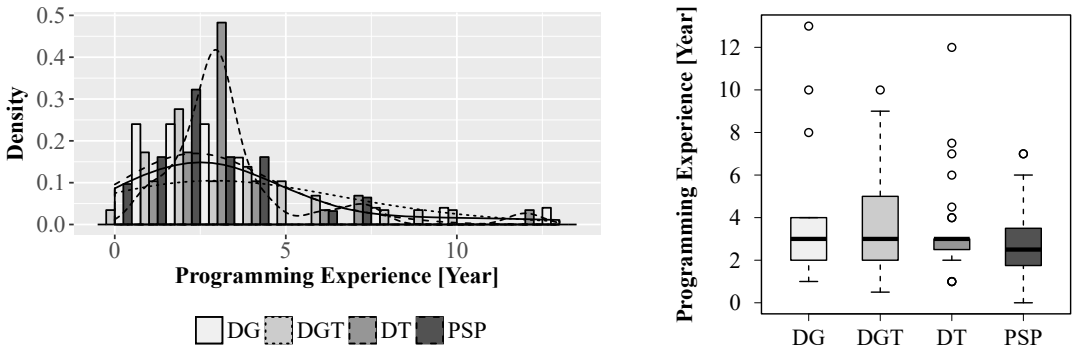
Figure 8.10.: Participants' gender and level of computer science education per group

The participants' level of education in computer science is shown in Figure 8.10 (b). Since the courses we recruited the participants from are primarily targeting bachelor students, only 21.1% of the participants hold a Bachelor of Science (BSc) degree in computer science. The distribution between the groups was as follows:

- DG: 4 participants with BSc degree (16%) and 21 participants without any computer science degree (84%)
- DGT: 7 participants with BSc degree (24.1%) and 22 participants without any computer science degree (75.9%)
- DT: 5 participants with BSc degree (17.2%) and 24 participants without any computer science degree (82.8%)
- PSP: 8 participants with BSc degree (25.8%) and 23 participants without any computer science degree (74.2%)

Both the DGT group and PSP group have a slightly larger share of participants with a BSc degree in computer science than the DG and DT groups, but no significant differences were found in level of education between the groups (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).

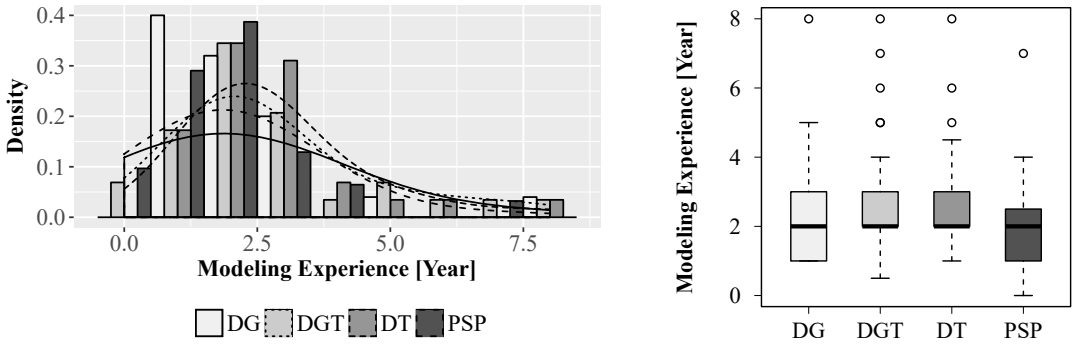
With regards to programming experience (cf. Figure 8.11), all groups have a high density in the range of 1 to 4 years of experience. Only very few participants have less than 1 year or more than 4 years of programming experience. Overall, the groups are similar with regards to programming experience. The steeper distribution shape of the DT group appears to be no



(a) Kernel density plot

(b) Box plot

Figure 8.11.: Participants' programming experience per group



(a) Kernel density plot

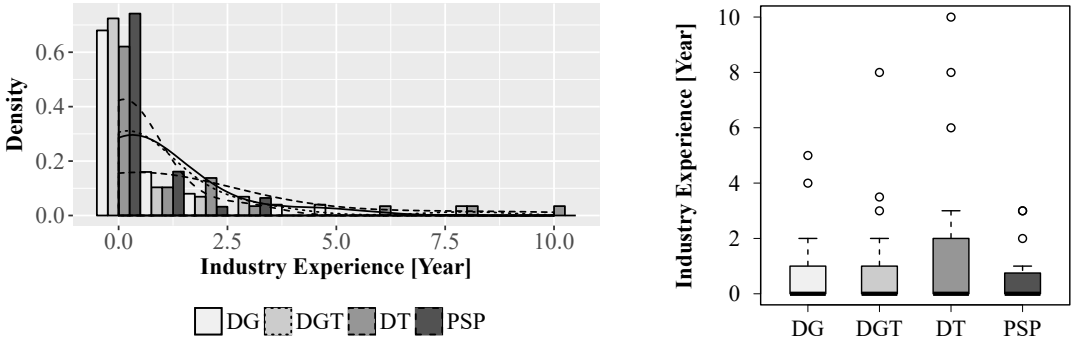
(b) Box plot

Figure 8.12.: Participants' modeling experience per group

major difference since we could not find any significant difference in programming experience between the groups (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).

In the majority of cases, the participants' modeling experience is between 1 and 3 years (cf. Figure 8.12). There are no significant differences in modeling experience between the groups (all $p > 0.05$ when p-values are adjusted to take multiple testing into account [152]); test applied: two-sided Cliff's delta [150], [151]).

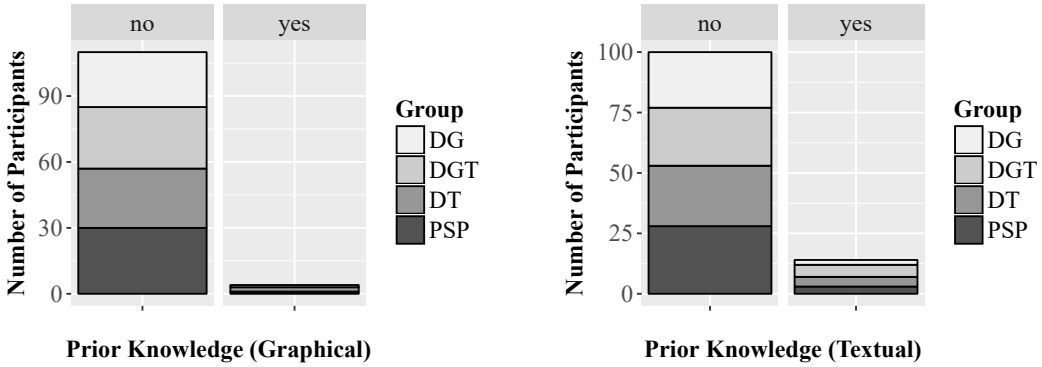
Since the computer science curricula at the University of Vienna are designed for full-time studying, the majority of the participants have little to no work experience in the software industry. Some of the students work beside studying or had been working for years in the software industry prior to becoming computer science students. These circumstances are very well reflected in Figure 8.13. The industry experience of the different groups appears to be simi-



(a) Kernel density plot

(b) Box plot

Figure 8.13.: Participants' industry experience per group



(a) Graphical approaches

(b) Textual approaches

Figure 8.14.: Participants' prior knowledge of graphical and textual pattern-based behavioral constraint approaches per group

larly low. There are no significant differences in industry experience between the groups (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).

Almost all participants did not have any prior knowledge of graphical pattern-based behavioral constraint approaches (e.g., Declare [121], Compliance Rule Graphs [197], or Dynamic Condition Response Graphs [23]), and a great majority of the participants did not have any prior knowledge of textual pattern-based behavioral constraint approaches (e.g., Property Specification Patterns [27], or the Compliance Request Language [23]), as can be seen in Figure 8.14. The share of participants with prior knowledge on pattern-based behavioral constraint approaches is as follows:

- DG: textual: 2 participants (8%), graphical: 0 participants (0%)

Table 8.2.: Number of observations, central tendency and dispersion of the dependent variables *correctness* and *response time* per group

		DT	DG	DGT	PSP
	Total number of observations	29	26	29	32
	Number of considered observations	29	25	29	31
Correctness	Arithmetic mean [%]	43.64	37.79	34.13	41.46
	Standard deviation (SD) [%]	27.93	29.08	24.86	25.6
	Median [%]	33.61	21.67	26.76	34.44
	Median absolute deviation (MAD) [%]	28.55	17.57	16.89	28.28
	Minimum [%]	8.33	6.67	1.85	10
	Maximum [%]	98.61	98.15	97.22	91.67
	Skew	0.48	0.8	1.17	0.51
	Kurtosis	−1.23	−0.89	0.53	−1.16
Response Time	Arithmetic mean [Minute]	36.28	35.56	33.87	33.09
	Standard deviation (SD) [Minute]	12.88	13.08	9.87	8.85
	Median [Minute]	36.14	35.85	32.58	33.43
	Median absolute deviation (MAD) [Minute]	11.46	11.74	8.75	9.2
	Minimum [Minute]	15.68	14.87	19.87	13.63
	Maximum [Minute]	69.75	72.65	58.18	51.8
	Skew	0.67	0.71	0.79	−0.01
	Kurtosis	−0.03	0.49	−0.08	−0.62

- DGT: textual: 5 participants (17.2%), graphical: 1 participant (3.4%)
- DT: textual: 4 participants (13.8%), graphical: 2 participants (6.9%)
- PSP: textual: 3 participants (9.7%), graphical: 1 participant (3.2%)

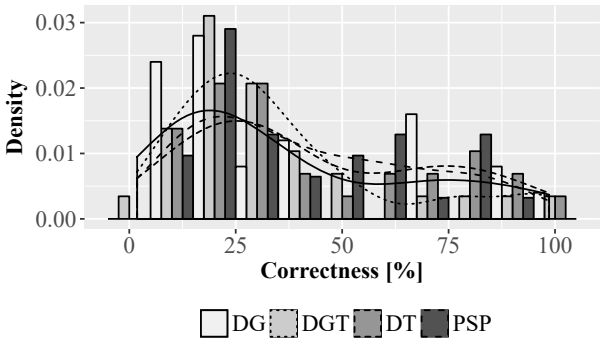
There are no significant differences with regards to prior knowledge of graphical and textual pattern-based behavioral constraint approaches between the groups (all $p > 0.05$; test applied: two-sided Cliff's delta [150], [151]).

Overall, with the exception of minor differences, which are to be expected in a completely randomized group allocation, the groups are well-balanced. We could not find any significant differences. That is, there are no indications of disturbing effects on the dependent variables that might have resulted from unbalanced groups.

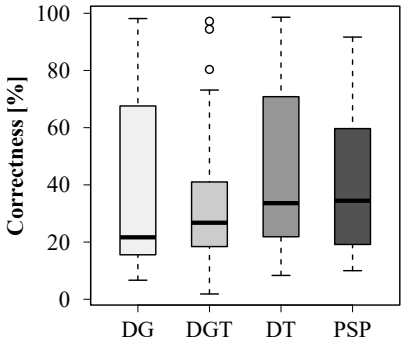
8.4.3. Descriptive Statistics of Dependent Variables

This section presents the descriptive statistics of the dependent variables. All gathered data have been made publicly available (cf. Czepa & Zdun [196]).

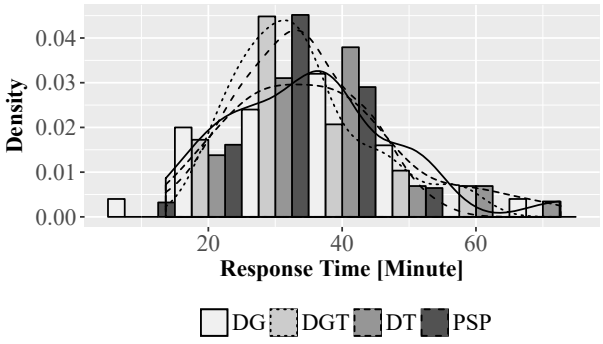
Table 8.2 contains the number of observations, central tendency and dispersion of the dependent variables *correctness* and *response time* per group. In consequence of the completely



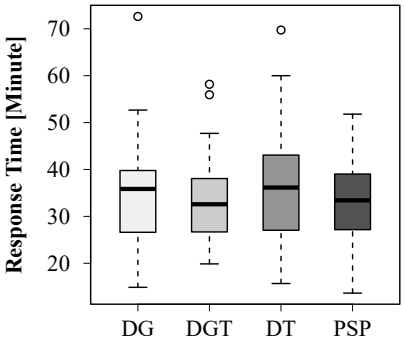
(a) Kernel density plot: Correctness



(b) Box plot: Correctness



(c) Kernel density plot: Response time

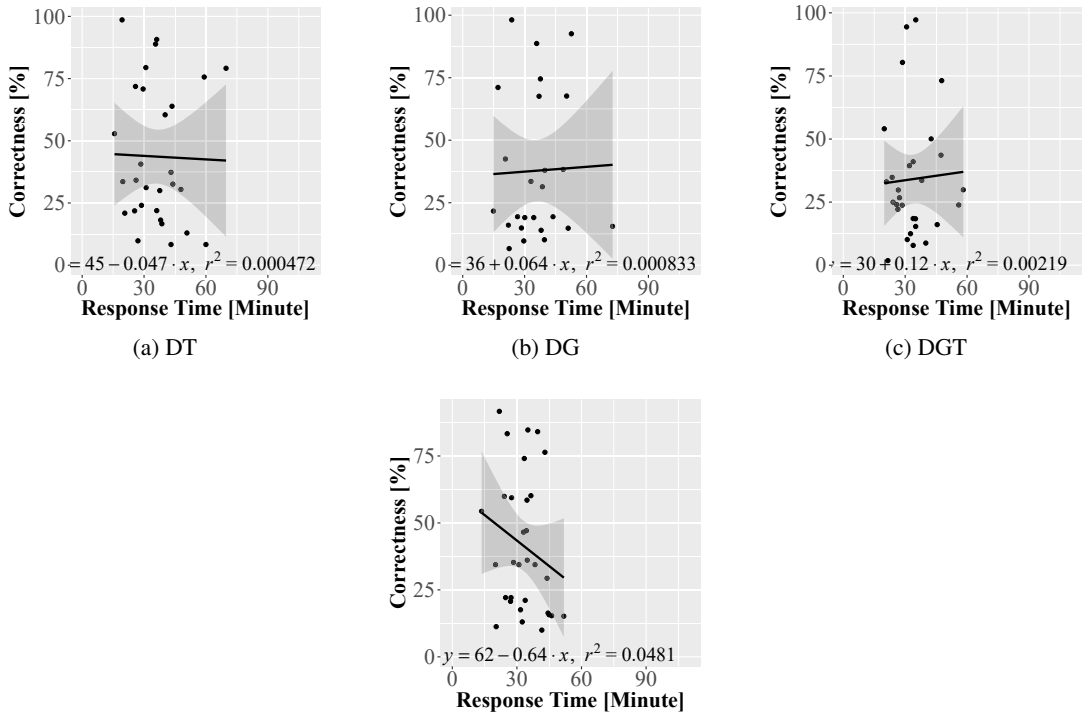


(d) Box plot: Response time

Figure 8.15.: Kernel density plots and box plots of the dependent variables *correctness* and *response time* per group

random allocation to groups, there were 29 participants in the DT group, 26 participants in DG, 29 participants in DGT, and 32 participants in PSP. Due to irregularities (cf. Section 8.4.1), we had to exclude the data of one DG participant and one PSP participant. With 43.64% and 41.46%, the *correctness arithmetic means* of the DT and PSP groups, which are both purely textual, are between about 4 to 10% higher than those of the DG (37.79%) and DGT (34.13%) groups. Also the *median correctness* values of the PSP (34.44%) and DT (33.61%) groups are between about 7% to 12% higher than those of the DG (21.67%) and DGT (26.76%) groups. According to the mean and median values, the *response times* appear to be slightly (about 2-3 minutes) faster in the PSP and DGT groups than in the DT and DG groups. Interestingly, many participants achieved a rather low level of correctness while the response times are overall far below the 90 minutes limit in all experiment groups. That is, time was no limiting factor and cannot be the cause of the low achieved correctness scores. The results show large differences in range between the minimum and maximum correctness. We commonly observed such large ranges in course exercises over the past years. Consequently, the results of this study in that regard are aligned with these past observations. Almost all *skew* values are positive, which indicates a right-tailed distribution. With a small negative skew value of -0.01, the PSP response time distribution is rather symmetric. *Kurtosis* is another measure for the shape of a distribution which focuses on the general tailedness. Positive kurtosis values indicate skinny tails with a distribution toward the mean whereas negative kurtosis values indicate fat tails. The majority of the kurtosis values of the correctness variable are negative. The sole exception is the DGT kurtosis value of 0.53, which clearly indicates a steeper distribution than in the other groups. With kurtosis values close to zero, the DT (-0.03) and DGT (-0.08) response time distributions are normal-tailed. In contrast, the DG response time distribution has a positive value (0.49) indicating skinny tails, and the PSP response time distribution has a negative value (-0.62) indicating fat tails.

In Figure 8.15, kernel density plots and box plots of the dependent variables *correctness* and *response time* are shown. The correctness distribution of the DGT group is steeper than those of the other groups (cf. Figure 8.15 (a)). There are three outliers in the DGT group indicating that only a few participants were able to achieve high levels of correctness in this group (cf. Figure 8.15 (b)). The outlier at 80.37% correctness had prior knowledge of graphical and textual behavioral constraint approaches while the other two outliers with correctness values of 97.2% and 94.4% did not have any prior knowledge of graphical and textual behavioral constraint approaches. According to the kernel density plot in Figure 8.15 (c), both the DGT and PSP response time distributions appear to be steeper than those of the other groups. In total there are four response time outliers, all of them in the *Declare* groups (two in DGT and one each in DT and DG).



(a) DT (b) DG (c) DGT

(d) PSP

Figure 8.16.: Scatter plots of response time vs. correctness with linear trend lines, 95% confidence regions, and coefficients of determination (r^2)

Table 8.3.: Kendall's rank correlation tau of the dependent variables *correctness* and *response time*

	DT	**DG**	**DGT**	**PSP**
z	-0.8067	0.2804	0.0939	-1.1395
p	0.4198	0.7792	0.9252	0.2545
tau	-0.106	0.0401	0.0124	-0.1447

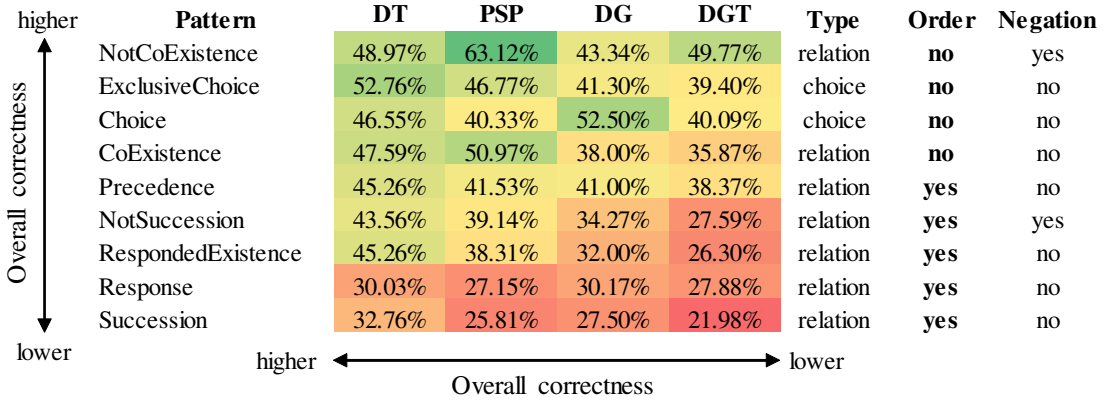


Figure 8.17.: Color scale plot of mean correctness per pattern and representation (the greener the more correct, the redder the less correct)

The scatter plots of the dependent variables *correctness* and *response time* do not show any clear signs of correlation (cf. Figure 8.16). Moreover, the results of all Kendall's rank correlation tau tests are non-significant (cf. Table 8.3). Consequently, there appears to be no correlation between those dependent variables.

For a more detailed analysis of the correctness variable, we make use of a color scale plot (cf. Figure 8.17) to identify potentially problematic language elements. Interestingly, relations in which the order of the involved states is of importance (e.g., succession) result in lower mean correctness values than patterns in which the order is not of importance (e.g., choice). Especially, the response and succession patterns show a low level of correctness in all groups.

Figure 8.18 shows diverging stacked bar charts (cf. Heiberger & Robbins [198]) of all Likert responses. In the following, we will discuss them:

- **Statement 1: Studying the behavioral constraint representation approach has been difficult.** 48% of the PSP participants *strongly agree* or *agree* that the approach is difficult to learn, followed by DT with 41%, and DGT with 38%. The share of *strongly agree* answers is about two times higher in the PSP group than in the DGT and DG groups. The purely graphical DG approach has the lowest percentage of *(strongly) agree* answers (36%). Interestingly, none of the DT participants answered with *strongly agree*. According to the bar chart, the purely graphical DG approach appears to be perceived slightly less difficult to learn than the other approaches.

- **Statement 2: Applying the knowledge about the behavioral constraint representation approach has been difficult.** According to the gathered data, it appears to be more difficult to apply the approaches than learning them. With 81%, the share of *(strongly)*

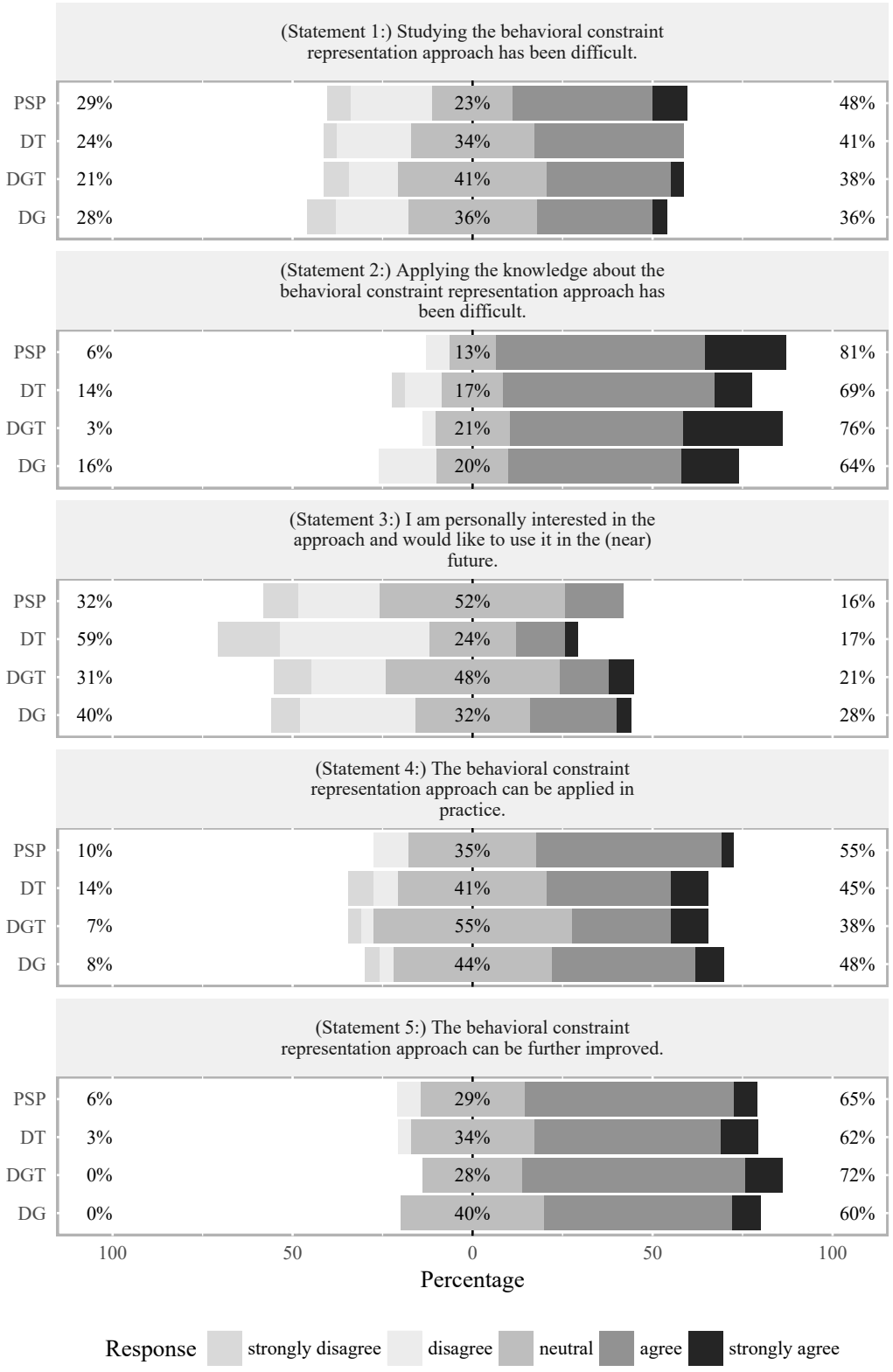


Figure 8.18.: Diverging stacked bar charts of participants' Likert responses

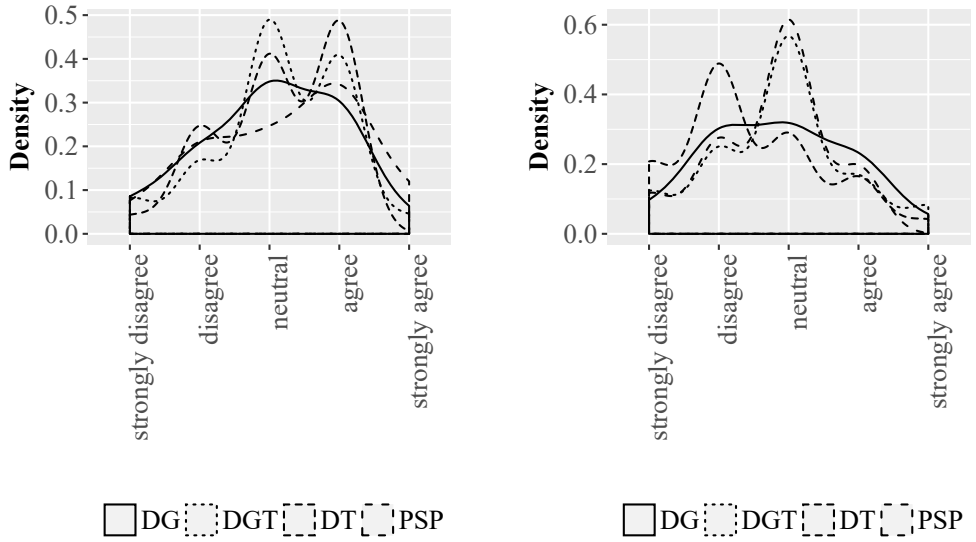
agree answers is higher in the PSP group than in the other groups, followed by the DGT group with 76%. The DG (with 16% *disagree*ing and 64% *(strongly) agree*ing) and DT approaches (with 14% *(strongly) disagree*ing and 69% *(strongly) agree*ing) are overall perceived a little less difficult to apply than PSP (with 6% *disagree*ing) and DGT (with 3% *disagree*ing).

- **Statement 3: I am personally interested in the approach and would like to use it in the (near) future.** With 59%, the majority of DT participants does not show any interest in using the approach in the future. The share of *neutral* answers is largest in the PSP and DGT groups (52% and 48%), which indicates that the participants of this group are rather undecided. There is, however, a tendency towards *(strongly) disagree*ing with a share of about $1/3$ of the given answers in those groups. Also, with a share of 40% *(strongly) disagree* answers, the DG group shows a rather negative attitude towards adopting the approach.

- **Statement 4: The behavioral constraint representation approach can be applied in practice.** With 55% *(strongly) agree*ing, the PSP group has the largest share of positive answers. Interestingly, the share of *strongly agree* answers is smaller than in the other three groups, and the PSP group has no *strongly disagree* answers at all. DG is second with 48% positive and only 8% negative answers, followed by DT with 45% *(strongly) agree*ing and 8% *(strongly) disagree*ing. 55% of the DGT participants are undecided, but there is a clear tendency towards *(strongly) agree*ing (38%).

- **Statement 5: The behavioral constraint representation approach can be further improved.** The share of *(strongly) agree* answers is large ($\geq 60\%$) in all groups. At the same time, there are no *strongly disagree* answers and the share of *disagree* answers is very low (6% in PSP and 3% in DT). The largest share of *(strongly) agree* answers is present in the DGT group (72%).

In addition to the visualization by diverging stacked bar charts in Figure 8.18, we are interested in the shape of the distributions as well, as they are important for testing model assumptions of statistical tests. Figure 8.19 shows kernel density plots of the Likert responses with especially striking differences in distribution shape. In Figure 8.19 (a) the PSP and DG distribution shapes are less steep than those of the other two approaches, and the PSP group has its peak at *agree* whereas the the DG group has its peak at *neutral*. In Figure 8.19 (b), the DG distribution shape is rather flat in comparison to the remaining distribution shapes. The DT group has its peak at *disagree* while the DGT and PSP approaches show a similar distribution shape and location with a peak at *neutral*.



(a) Statement 1 ("difficult to learn")

(b) Statement 3 ("personally interested in using the approach")

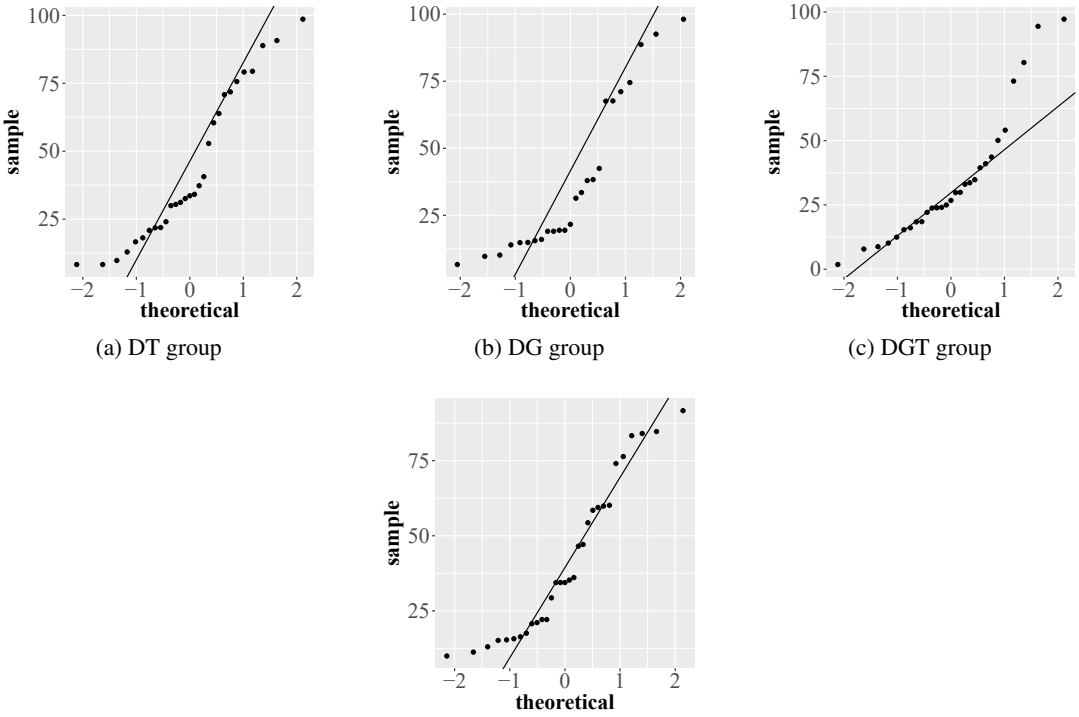
Figure 8.19.: Selected kernel density plots of Likert responses

Table 8.4.: Shapiro-Wilk test of normality (* for $\alpha = 0.05$, ** for $\alpha = 0.01$)

	DT	**DG**	**DGT**	**PSP**
Correctness	$W = 0.90205$ $p = 0.008112$ **	$W = 0.84027$ $p = 0.001158$ **	$W = 0.90796$ $p = 0.01528$ *	$W = 0.86744$ $p = 0.001775$ **
Response Time	$W = 0.98841$ $p = 0.9783$	$W = 0.95337$ $p = 0.2982$	$W = 0.95585$ $p = 0.2587$	$W = 0.93301$ $p = 0.06584$

8.5. Statistical Inference

For proper hypothesis testing, it is important to select the most suitable method. Particularly, it is preferable to choose the method with the greatest statistical power given the properties of the data. Specific model assumptions must be met. A crucial model assumption of parametric testing is normality. The graphical analysis by normal Q-Q plots (cf. Figure 8.20) and Shapiro-Wilk tests of normality (cf. Table 8.4) suggest that the normality assumption does not hold in multiple cases. Specifically, the normality assumption does not hold for the correctness variables of all groups. Since there are indications of non-normality in the metric dependent variables *correctness* and *response time* (cf. Section 8.4.3), the model assumptions for parametric testing are violated. Therefore, parametric testing is ruled out. The non-parametric Kruskal-Wallis test



(a) DT group

(b) DG group

(c) DGT group

(d) PSP group

Figure 8.20.: Normal QQ plots of correctness data

Table 8.5.: Cliff's d of the dependent variables *correctness* and *response time*, two-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151])

		DT vs. **DG**	**DT** vs. **DGT**	**DT** vs. **PSP**	**DG** vs. **DGT**	**DG** vs. **PSP**	**DGT** vs. **PSP**
Correctness	$p_1 = P(X > Y)$	0.5807	0.5898	0.515	0.4966	0.4335	0.4171
	$p_2 = P(X = Y)$	0	0.0024	0.0001	0.0013	0	0.0034
	$p_3 = P(X < Y)$	0.4193	0.4078	0.4849	0.5021	0.5665	0.5795
	d	−0.1614	−0.1819	−0.03	0.0055	0.1329	0.1624
	s_d	0.1599	0.151	0.1521	0.1629	0.16	0.1498
	z	−1.0089	1.2049	−0.1974	0.0339	0.8309	1.0839
	$CI\ low$	−0.4516	−0.4557	−0.3188	−0.3057	−0.1851	−0.1386
	$CI\ high$	0.1597	0.1233	0.2639	0.3156	0.4257	0.436
	p	0.3177	0.2333	0.8442	0.9731	0.4098	0.2829
Response Time	$p_1 = P(X > Y)$	0.5103	0.5565	0.5551	0.5421	0.5471	0.5006
	$p_2 = P(X = Y)$	0	0	0	0	0	0
	$p_3 = P(X < Y)$	0.4897	0.4435	0.4449	0.4579	0.4529	0.4994
	d	−0.0207	−0.113	−0.1101	−0.0841	−0.0942	−0.0011
	s_d	0.1597	0.1547	0.1521	0.1634	0.1605	0.1513
	z	−0.1296	−0.7299	−0.7242	−0.5149	−0.5867	−0.0073
	$CI\ low$	−0.3239	−0.399	−0.3919	−0.3875	−0.3917	−0.291
	$CI\ high$	0.2864	0.1932	0.1905	0.2357	0.2211	0.2889
	p	0.8974	0.4685	0.4719	0.6088	0.5598	0.9942

Table 8.6.: Cliff's d of Likert scale responses, two-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Cliff [150] and Rogmann [151])

		DT/DG	DT/DGT	DT/PSP	DG/DGT	DG/PSP	DGT/PSP
Statement 1	$p_1 = P(X > Y)$	0.3779	0.3424	0.327	0.3255	0.3187	0.3382
	$p_2 = P(X = Y)$	0.3007	0.3163	0.287	0.2938	0.2594	0.2658
	$p_3 = P(X < Y)$	0.3214	0.3413	0.386	0.3807	0.4219	0.396
	d	−0.0566	−0.0012	0.059	0.0552	0.1032	0.0578
	s_d	0.1525	0.1451	0.1438	0.1524	0.1492	0.1451
	z	−0.3709	−0.0082	0.4099	0.3621	0.6919	0.3987
	$CI\ low$	−0.3443	−0.2802	−0.2223	−0.2419	−0.1919	−0.2255
	$CI\ high$	0.2409	0.278	0.3311	0.3429	0.3812	0.3322
	p	0.7122	0.9935	0.6834	0.7187	0.492	0.6915
Statement 2	$p_1 = P(X > Y)$	0.3255	0.2224	0.2047	0.2386	0.2297	0.3203
	$p_2 = P(X = Y)$	0.349	0.3507	0.3926	0.3228	0.3509	0.3716
	$p_3 = P(X < Y)$	0.3255	0.4269	0.4027	0.4386	0.4194	0.3081
	d	0	0.2045	0.198	0.2	0.1897	−0.0122
	s_d	0.1479	0.1361	0.1302	0.1452	0.1422	0.1386
	z	0	1.5028	1.5205	1.3774	1.3338	−0.0883
	$CI\ low$	−0.2846	−0.0732	−0.0673	−0.096	−0.0993	−0.2787
	$CI\ high$	0.2846	0.4528	0.4372	0.4635	0.4491	0.2559
	p	1	0.1385	0.1338	0.1743	0.1878	0.9299
Statement 3	$p_1 = P(X > Y)$	0.2621	0.2592	0.2681	0.36	0.3755	0.3604
	$p_2 = P(X = Y)$	0.2579	0.2414	0.2569	0.2648	0.2839	0.3281
	$p_3 = P(X < Y)$	0.48	0.4994	0.475	0.3752	0.3406	0.3115
	d	0.2179	0.2402	0.2069	0.0152	−0.0348	0.0489
	s_d	0.1486	0.1443	0.1443	0.1543	0.1534	0.1407
	z	1.4665	1.6649	1.4336	0.0983	−0.2271	−0.348
	$CI\ low$	−0.0865	−0.057	−0.0873	−0.282	−0.3258	−0.3164
	$CI\ high$	0.4851	0.4983	0.4679	0.3097	0.2621	0.2257
	p	0.1485	0.1015	0.1571	0.9221	0.8212	0.7291
Statement 4	$p_1 = P(X > Y)$	0.3103	0.3377	0.2959	0.3517	0.3006	0.2781
	$p_2 = P(X = Y)$	0.3338	0.3389	0.3348	0.3642	0.3691	0.3448
	$p_3 = P(X < Y)$	0.3559	0.3234	0.3693	0.2841	0.3303	0.3771
	d	0.0455	−0.0143	0.0734	−0.0676	0.0297	0.099
	s_d	0.1485	0.1433	0.1419	0.147	0.1459	0.141
	z	0.3065	0.0996	0.5174	−0.4597	0.2034	0.702
	$CI\ low$	−0.2438	−0.2891	−0.2053	−0.3451	−0.2534	−0.1801
	$CI\ high$	0.3274	0.2627	0.3411	0.2208	0.3081	0.3633
	p	0.7605	0.921	0.6069	0.6477	0.8396	0.4855
Statement 5	$p_1 = P(X > Y)$	0.3021	0.2354	0.3026	0.2152	0.2852	0.3348
	$p_2 = P(X = Y)$	0.4151	0.4269	0.4093	0.4414	0.4232	0.4472
	$p_3 = P(X < Y)$	0.2828	0.3377	0.2881	0.3434	0.2916	0.218
	d	−0.0193	0.1023	−0.0145	0.1283	0.0065	−0.1168
	s_d	0.1437	0.1358	0.1359	0.1403	0.1405	0.1302
	z	−0.1344	0.7529	−0.1063	0.9144	0.0459	−0.8969
	$CI\ low$	−0.2949	−0.1674	−0.276	−0.1523	0.2651	−0.3615
	$CI\ high$	0.2592	0.3577	0.2491	0.3897	0.2771	0.143
	p	0.8936	0.4547	0.9156	0.3647	0.9635	0.3735

assumes that the distribution shapes do not differ apart from their central locations. The relevant descriptive statistics of the data (cf. Section 8.4.3), namely the skew/kurtosis values and the kernel density plots, suggest differences in the shape of distribution between the groups. Due to the properties of the data, we make use of *Cliff's delta* (cf. Cliff [150] and Rogmann [151]), a robust non-parametric test that is unaffected by change in distribution and non-normal data. Neither of the test results is significant at the $\alpha = 0.05$ level (cf. Table 8.5 and Table 8.6). Consequently, the null hypotheses $H_{0,1}$ to $H_{0,6}$ (cf. Section 8.3.4) cannot be rejected.

The statistics software *R* was used for all statistical analyses.[8] In particular, we used the following libraries in the course of our statistical evaluations: *biotools* [153], *car* [154], *ggplot2* [155], *mvnormtest* [156], *mvoutlier* [157], *orddom* [151], *psych* [158], *usdm* [159], and *likert* [170].

8.6. Analysis of Free Text Answers

In addition to the 18 experiment tasks and the Likert scale-based questionnaire, we asked three free text questions to capture the thoughts of the participants regarding the studied and applied behavioral constraint representation. These questions focused on the personal opinion of the participants regarding positive ("likes") and negative ("dislikes") aspects of the assigned behavioral constraint approach as well as suggestions for improvement. In particular, the following three questions were asked:

- What do you like about the behavioral constraint representation approach?
- What do you dislike about the behavioral constraint representation approach?
- How would you improve the behavioral constraint representation approach?

Our analysis of the textual answers of the participants has been inspired by the summative content analysis approach [160]. Since the majority of answers given by the participants is very short and in note form, running a full-blown summative content analysis, which usually focuses on journal manuscripts or specific content in textbooks, is impossible. Nevertheless, it is possible to use the core idea of the technique, namely the counting of occurrences of specific content and the interpretation of the context associated with its use. In the following, we present the results of this analysis:

- 17.5% of all participants (13.8% in DT, 20% in DG, 17.2% in DGT, and 19.4% in PSP) have shown interest in practical examples and case studies to deepen the knowledge and to

[8] `https://www.r-project.org/`

grasp the full potential of their assigned approach, especially when applied in real-world scenarios.

- 16.7% of all participants (20.7% in DT, 8% in DG, 27.6% in DGT, and 9.7% in PSP) stated that they prefer a more formal definition of the available patterns (and scopes) of the assigned behavioral constraint representation to alleviate ambiguities that are inherently present in natural language.

- Three participants (9.7%) of the PSP group and one participant (3.4%) of the DT group reported issues regarding understanding truth values, especially the difference between temporary and permanent states, whereas one participant (3.4%) of the DGT group mentioned truth values positively and another one negatively. Neither positive nor negative aspects regarding truth values were mentioned by any participant of the DG group.

- Two participants (6.9%) of the DGT group and one participant (3.4%) of the DT group stated that they would have wanted access to the learning material during the experiment, because they had problems memorizing the meaning of the behavioral constraint patterns.

- 40% of the DG group participants reported problems regarding the graphical notation while 8% mentioned positive aspects. Two DG participants mentioned that the exclusive choice and choice symbols are hard to differentiate. Another participant stated that the `not_succession` pattern was difficult to grasp and remember. One participant reported that it was difficult to remember the order associated with relation patterns. Making use of more symbols, rather than using combinations of symbols, was proposed by one participant. The feedback of the remaining participants was more general in nature (e.g., "syntax is confusing" or "meaning of signs is hard to understand"). Only a single participant stated that the shapes used in the graphical representation are clear and easy-to-read. Another participant mentioned his personal preference towards graphical approaches in general.

- The share of DGT participants reporting problems with their assigned representation is 24.1%. Similar to the feedback of one DG participant, one participant would prefer using more shapes for a better differentiation of the patterns. Another participant found the naming of the patterns unclear. In this regard, another participant suggested using terms present in Boolean algebra (e.g., "or" instead of "choice"). The rest of the comments are more general in nature (e.g., "not intuitive"). Two participants (6.9%) mentioned the graphical operators positively ("I liked the graphic representation, as the graphics contained some semantic information about the constraint" and "easy to understand representation in form of simple symbols").

- 20.7% of the DT participants mentioned negative aspects about their assigned textual representation while 10.3% mentioned positive aspects. Like one of the DGT participants, one DT participant desires clearer naming of the patterns. Another participant mentioned that the naming of the patterns is appropriate. Interestingly, one participant reported problems with understanding the meaning of the direction of statements (e.g., whether `succession(`A`,` B`)` means A succeeds B or B succeeds A). Originally, we had assumed that the direction would be understood implicitly by the reading direction. However, as only a single participant reported this issue, it is highly questionable whether this is a general issue. Two participants reported difficulties with using the approach due to "a lot of similarities between the constraints", which makes them hard to distinguish. Two participants suggested adding a negation operator to the constraint language to support negations for each of the available patterns. Another participant liked that there exists a specific pattern for "every case", but at the same time criticized that the number of constraints is growing rapidly if the implementation of new scenarios becomes necessary. One participant liked the function-like style of the approach, which is familiar to programmers.

- 19.4% of the PSP participants criticized some aspects of their assigned representation while 9.8% mentioned positive aspects. One participant would prefer a "more sophisticated visualization" instead of the textual representation. Another participant was fond of the natural language approach, but criticized the lack of syntax highlighting in the experiment. We wanted to present all four tested representations by similar means to avoid bias towards a specific representation, so syntax highlighting was intentionally omitted in the PSP task descriptions. In actual implementations of the PSP approach, syntax highlighting or similar techniques are usually supported (e.g., Czepa et al. [199]). One participant liked the use of Boolean connectors, but would have wanted to see the actual symbols instead of words (e.g., $\wedge$ for `and`). A similar comment was made by another participant who suggests "writing in a more mathematical way". Another participant would have wanted to see the `xor` operator in use. The difference between the `between` and `after-until` scopes was mentioned as "difficult to understand" by one participant. Another participant mentioned that the scopes and patterns are clearly understandable.

8.7. Discussion

8.7.1. Evaluation of Results and Implications

While the descriptive statistics and the results of the analysis of free text answers appear to be slightly in favor of the textual approaches, the results of the inference statistics do not indi-

cate any significant difference between the tested representations. That is, the following null hypotheses cannot be rejected:

- $H_{0,1}$: There is no difference in terms of *understandability* between the representations.
- $H_{0,2}$: There is no difference in terms of *perceived learning difficulty* between the representations.
- $H_{0,3}$: There is no difference in terms of *perceived application difficulty* between the representations.
- $H_{0,4}$: There is no difference in terms of *personal interest in using the approach* between the representations.
- $H_{0,5}$: There is no difference in terms of *perceived practical application potential* between the representations.
- $H_{0,6}$: There is no difference in terms of *perceived improvement potential* between the representations.

However, it is striking that the achieved correctness is rather low on the average. From prior studies on the understandability of textual behavioral constraint approaches (cf. Chapter 6), it is evident that higher correctness values (about 70% on the average in PSP) are achievable if access to learning material and other material (e.g., hand-written notes) is granted during the experiment session. That is, it appears to be difficult to deduce the meaning of pattern-based behavioral constraints from their textual and/or graphical representations without additional support. As no additional support was provided to any of the groups, we do not think that this aspect could have influenced the relative outcomes of the experiment. In this regard, it might be possible that both approaches could benefit from a greater degree of additional support in a similar fashion. The analysis of the given free text answers regarding positive/negative aspects and suggestions for improvement with regard to the achievable correctness levels (cf. Section 8.6) provides additional evidence. Consequently, there are two angles of approach for improvement, namely the representation itself (i.e., finding better graphical and/or textual representations) and the support provided (i.e., supportive means provided by a behavioral constraint modeling tool).

8.7.2. Threats to Validity

All known threats that might have an impact on the validity of the results are discussed in this subsection.

Threats to Internal Validity

Threats to internal validity can be described as unobserved variables that might have an unwanted influence on an experiment's result by disturbing the causal relationship of independent and dependent variables. There exist several threats to the internal validity of this experiment, which must be discussed:

- *History effects* refer to events that occur in the environment and change the conditions of a study. The short duration of the study limits the possibility of changes in environmental conditions, and we did not observe any, but we cannot entirely rule out any such effect prior to the study taking place. However, in such a case, it is extremely unlikely that the scores of one group are more affected than another, because of the random allocation of participants to groups.

- *Maturation effects* refer to the impact that time has on an individual. Since the duration of the experiment was short, maturation effects are considered to be of minor importance, and we did not observe any such effects.

- *Testing effects* comprise learning effects and experimental fatigue. *Learning effects* were avoided by testing each person only once. *Experimental fatigue* is concerned with occurrences during the experiment that exhaust the participant either physically or mentally. The participants did not report, and we did not observe, any signs of fatigue.

- *Instrumental bias* occurs if the measuring instrument (i.e., a physical measuring device or the actions/assessment of the researcher) changes over time during the experiment. We tried to avoid instrumental bias by using an experimental design that enables an automated and standardized processing of the test results.

- *Selection bias* is present if the experimental groups are unequal before the start of the experiment (e.g., severe differences in relevant experience, age, or gender). Usually, selection bias is likely to be more threatening in quasi-experimental research. By using an experimental research design with the fundamental requirement of randomly assigning participants to the different groups of the experiment, we can avoid selection bias to a large extent. Moreover, our evaluation of the composition of the groups (regarding age, gender, experience/education in different dimensions) did not indicate any major differences.

- *Experimental mortality* is only likely to occur if the experiment lasts for a long time because the chances for dropouts increase (e.g., participants moving to another geographical location). Due to the short time frame of this study, experimental mortality was not an issue at all.

- *Diffusion of groups* occurs if a group of the experiment is contaminated in some way. We tried to mitigate this threat by asking the participants not to disclose or discuss anything related to the experiment before the experiment session. Since the participants share the same social group, and they are interacting outside the research process as well, we cannot entirely rule out a cross-contamination between the groups.

- *Compensatory rivalry* is present if participants of a group put in extra effort when they have the impression that the representation of another group might lead to better results than their own. This treat is mitigated by insisting on nondisclosure.

- *Demoralization* could occur if a participant is assigned to a specific group that she/he does not want to be part of. We did not observe any signs of demoralization such as increased dropout rates or complaints regarding group allocation.

- *Experimenter bias* refers to undesired effects on the dependent variables that are unintentionally introduced by the researcher. The experiment was designed in a way that limits chances for this kind of bias. In particular, all participants received a similar preparation and worked on the same set of tasks (i.e., only the constraint representation differs). Moreover, the results of the controlled experiment were processed automatically in a standardized procedure.

- *Self-selection bias*: The possibility of self-selection bias appears to be negligible as merely three students participated in alternative activities.

- *Impact of preparation*: We designed the preparation material in such a way to keep the necessary effort involved in learning the patterns at a doable level for each participant. In accordance with Miller's law [194], at most nine language elements were presented in each experiment group. Consequently, the learning effort was minimal, which strongly mitigates the risk of insufficient preparation. Moreover, instead of directly asking the degree of effort spent on preparation, which might lead to insincere answers (i.e., a participant might expect to be punished for not preparing well), we tried to check indirectly by asking how difficult the studying of the approach was. We assumed that a participant who did not prepare himself/herself will not have a strong opinion on that topic and tick the neutral item or abstain. Subsequently, we removed the data of these participants from the data set and performed hypotheses testing. As in the full data set, no test result was significant. That is, even if participants who possibly did not prepare themselves are not considered, the results still apply.

Threats to External Validity

The external validity of a study focuses on its generalizability. In the following, we discuss potential threats that hinder a generalization. Different types of generalizations must be considered:

- *Generalizations across populations*: By statistical inference, we try to make generalizations from the sample to the immediate population. We do not intend to claim generalizability across populations without further empirical evidence. This study has a strong focus on the understandability of the tested representations from the viewpoint of novice software designers. We acknowledge that expert users who are familiar with *Declare* and/or *Property Specification Patterns* potentially perform better.
- *Generalizations across groups*: Since the experiment groups are equivalent to specific behavioral constraint representations, options for variation are limited. Nevertheless, in future studies, it might be interesting to introduce new or amended representations (e.g., a graphical representation that is based on DGT, but using just a single relation shape).
- *Generalizations across settings/contexts*: The participants of this study are students who enrolled in computer science courses at the University of Vienna, Austria. Apparently, a majority of the students are Austrian citizens, but there is a large presence of foreign students as well. Surely, it would be interesting to repeat the experiment in different settings/contexts to evaluate the generalizability in that regard. For example, repeating the experiment with native English-speakers might lead to different (presumably better) results since English terms are used in the textual/hybrid constraint representations.
- *Generalizations across time*: In general, it is hard to predict whether the results of this study hold over time. For example, if teaching of graphical or textual behavioral constraint approaches is intensified in the computer science curricula at the University of Vienna, then the students would bring in more expertise, which likely would have an impact on the results.

Threats to Construct Validity

There are potential threats to the validity of the construct that must be discussed:

- *Inexact definition & Construct confounding*: This study has a primary focus on the construct *understandability*, which is measured by the dependent variables *correctness* and *response time*. This construct is exact and adequate. Several existing studies use this construct and its variables (cf. Feigenspan et al. [143] and Hoisl et al. [144]).

- *Mono-method bias*: To measure the correctness of answers, the evaluation by an automated method appears to be the most accurate measure as it does not suffer from experimenter bias or instrumental bias. Keeping time records was the personal responsibility of each participant due to organizational reasons. The participants were instructed extensively on how to keep time records, and they were informed that accurate time record keeping will have a positive impact on the final grading. We also made clear that the overall response time has no influence on the grading to avoid time stress. We did not detect any irregularities (e.g., overlapping time frames or long pauses) in those records. Nonetheless, this measuring method leaves room for measuring errors, and an additional or alternative measuring method (e.g., performing the experiment with an online tool that handles record keeping) would reduce this threat. The additional task of keeping accurate time records might have had a negative impact on performing the actual experiment tasks, but no participant reported any such effect.

- *Reducing levels of measurements*: Both the correctness and response time are continuous variables. That is, the levels of measurements are not reduced. The Likert scales (also called agree-disagree rating scales) used in this study offer 5 answer categories rather than 7 or 11, because the latter produce data of lower quality according to Revilla et al. [168].

- *Group-sensitive factorial structure*: In some empirical studies a specific assigned experiment group might sensitize participants to develop a different view on a construct. Since we did not ask questions regarding the subjective level of understandability, but tried to measure the actual level of understandability objectively, this threat appears to not be present at all. The questionnaire at the end of the question sheet is neither meant nor used to measure the understandability construct, but used to measure other aspects. Here, we tried to mitigate this threat by focusing on one-dimensional constructs (i.e., the multi-dimensional construct *perceived difficulty* is split up into *perceived learning difficulty* and *perceived application difficulty*).

Threats to Content Validity

Content validity is concerned with the relevance and representativeness of the elements of a study for the construct that is measured:

- *Relevance*: All tasks of this study are based on recurring behavioral constraint patterns that are present in existing graphical and textual behavioral constraint approaches (cf. [23], [27], [42], [121]).

- *Representativeness*: A representative subset of existing behavioral constraint patterns was used for designing the tasks of the experiment. In this study we focused on a set of commonly used binary relations (cf. [23], [27], [42], [121]). Unary constraints are common as well, but we decided to omit them due to their simplicity. It is also worth noting that some behavioral constraints in *Declare* are not covered in the Property Specification Patterns, and vice versa. In particular, the scopes of PSP are not part of Declare, the chain patterns have a different meaning in *Declare* and PSP, and *Declare* supports additional behavioral constraints (i.e., alternate, negative relation and choice patterns). Nevertheless, some of the *Declare* patterns which are not explicitly covered by PSP can be realized by combinations of Property Specification Patterns. Others, like the alternate patterns of *Declare* are not covered by the originally proposed PSP collection. Naturally, it would have been possible for us to extend both *Declare* and PSP with new patterns including new graphical and textual elements, but proposing new pattern representations was not the goal of this empirical study, so the established as-is state of these approaches was covered.

Threats to Conclusion Validity

Retaining outliers might be a threat to conclusion validity. However, all outliers appear to be valid measurements, so deleting them would pose a threat to conclusion validity as well. We performed a thorough investigation of model assumptions before applying the most suitable statistical test with the greatest statistical power, given the properties of the acquired data. That course of action is considered to be extremely beneficial to the conclusion validity of this study.

8.8. Related Work

We are not aware of any existing empirical studies that investigate the differences in understandability of representative graphical and textual behavioral constraint languages in a similar way and depth as the presented study does.

We also would like to mention that there exists a huge body of studies on understandability of models that are merely remotely related to our work. For example, a study by Reijers and Mendling [200] investigates the understandability of classical flow-driven business process models. Interestingly, professionals could not be distinguished from the students of two participating universities, and the students of one university performed even better than professionals. However, since that study considers flow-driven business processes only, the results are hardly transferable to behavioral constraints that are declarative in nature.

In the following, we focus on studies that are highly related to the presented work, namely studies that are concerned with the understandability of behavioral constraints.

8.8.1. Empirical Studies on the Understandability of Behavioral Constraint Representations in Software Architecture and Software Engineering

There exist only very few studies on analyzing and comparing different behavioral constraint languages in the field of software architecture and engineering.

An eye-tracking experiment with 28 participants by Sharafi et al. [30] on the understandability of graphical and textual software requirement models did not reveal any statistically significant difference in terms of correctness of the approaches, which is in line with the results of our study. However, software requirement models and behavioral constraints are merely distantly related. The study also reports that the response times of participants working with the graphical representations were slower. Interestingly though, the participants preferred the graphical notation.

Czepa et al. [131] compared the understandability of three languages for behavioral software architecture compliance checking, namely the Natural Language Constraint language (NLC), the Cause-Effect Constraint language (CEC), and the Temporal Logic Pattern-based Constraint language (TLC), in a controlled experiment with 190 participants. The NLC language is simply using the English language for software architecture descriptions. CEC is a high-level structured architectural description language that abstracts the Event Processing Language [104] and enables nesting of cause parts, that observe an event stream for a specific event pattern, and effect parts, that can contain further cause-effect structures and truth value change commands. TLC is a high-level structured architectural description language that abstracts behavioral patterns. Interestingly, the statistical inference of this study suggests that there is no difference in understandability of the tested languages. This could indicate that the high-level abstractions employed bring those structured languages closer to the understandability of unstructured natural language architecture descriptions. Moreover, it might also suggest that natural language leaves more room for ambiguity, which is detrimental for its understanding. Potential limitations of that study are that its tasks are based on common architectural patterns/styles (i.e., a participant possibly recognizes the meaning of a constraint more easily by having knowledge of the related architectural pattern) and the rather small set of involved behavioral constraint patterns (i.e., only very few behavioral constraint patterns were necessary to represent the architecture descriptions).

Hoisl et al. [144] conducted a controlled experiment on three notations for scenario based model tests with 20 participants. In particular, they evaluated the understandability of a semi-structured natural language scenario notation, a diagrammatic scenario notation, and a fully-structured textual scenario notation. According to the authors, the purely textual semi-structured

natural language scenario notation is recommended for scenario-based model tests, because the participants of this group were able to solve the given tasks faster and more correctly. That is, the study might indicate that a textual approach outperforms a graphical one for scenario based model test, an effect that our study did not discover for behavioral constraints. However, the validity of the experiment is limited by its sample size and the lack of statistical hypothesis testing.

A controlled experiment carried out by Heijstek et al. [29] with 47 participants focused on finding differences in understanding of textual and graphical software architecture descriptions. Interestingly, participants who predominantly used textual architecture descriptions performed significantly better, which suggests that textual architectural descriptions could be superior to their graphical counterparts. In our study, which has a focus specifically on textual and graphical behavioral constraints instead of software architecture descriptions, such an effect was not measurable.

8.8.2. Empirical Studies on the Understandability of Behavioral Constraint Representations in Business Process Management

In the field of business process management, there exist studies that evaluate the understandability of declarative business processes which are composed of a set of behavioral constraint patterns. These studies are highly related to our work since they investigate the understandability of pattern-based behavioral constraints in the context of declarative business processes.

Weber et al. [191] carried out a controlled experiment (with 25 and 16 participants) on the impact of varying the levels of pattern-based behavioral constraints in planning and executing a journey. In particular, one group was exposed to only 2 behavioral constraints while another had to take 12 behavioral constraints into account. Interestingly, their statistical analysis does not show any significant difference in understanding. That might indicate that potential users handle varying constraint numbers well, but there also might not be enough measurable difference between 2 and 12 constraints. It would be interesting to evaluate how users cope with larger numbers of constraints (e.g., 25, 50, 100) as well. Moreover, the small sample sizes are a threat to validity of this study.

Zugal et al. [172] investigate the understandability of hierarchies in declarative business processes in an experiment with 9 participants. The results of their research indicate that hierarchies must be handled with care. While information hiding and improved pattern recognition are considered to be positive aspects of hierarchies since the mental effort for understanding a process model is lowered, the fragmentation of processes by hierarchies might lower overall understandability of the process model. Another important finding of their study is that users appear to approach declarative process models in a sequential manner, even if the user is definitely not bi-

ased through previous experiences with sequential business process models (e.g., BPMN [20]). They conclude that the abstract nature of declarative process models does not seem to fit the human way of thinking. Moreover, they observed that the participants of their study tried to reduce the number of constraints to consider by putting away sheets that describe irrelevant sub-process or by using the hand to hide parts of the process model that are irrelevant. The validity of this study is strongly limited by the extremely small sample size.

Haisjackl et al. [173] investigate the users' understanding of declarative business process models that are composed of a set of ten behavioral constraint patterns with 9 participants. The evaluation seems to be based on the same experimental data as in the work by Zugal et al. [172]. Similar to the latter work, they point out that users tend to read such models sequentially, despite the declarative nature of the approach. The larger a model, the more often are hidden dependencies overlooked, which indicates that increasing numbers of constraints lower understanding. Moreover, they report that individual constraints are overall well understood, but there seem to be problems with understanding the *precedence* constraint. As the authors point out, this kind of confusion could be related to the graphical arrow-based representation of the constraints, where subtle differences decide on the actual meaning. That is, the arrow could be confused with a sequence flow as present in flow-driven, sequential business processes. As with the work by Zugal et al. [172], the validity of this study is possibly strongly affected by the small sample size.

Haisjackl & Zugal [28] investigated differences between textual and graphical declarative workflows using the *Declare* notation in an empirical study with 9 participants. The evaluation seems to be based on the same experimental data as in the work by Zugal et al. [172] and Haisjackl et al. [173]. This study is highly related to our work presented in this chapter. The authors state that the results of their study indicate that the graphical representation are advantageous in terms of perceived understandability, error rate, duration, and mental effort, but this conclusion seems to be based merely on descriptive statistics (i.e., arithmetic means and counting occurrences). The lack of hypothesis testing and the small number of participants are severe threats to the validity of this study.

An approach by De Smedt et al. [174] tries to improve the understandability of declarative business process models by revealing hidden dependencies. They conduced an experiment with 95 students. The result suggests that explicitly showing hidden dependencies enables a better understanding of declarative business process models.

A study by Pichler et al. [175] compares the understandability of imperative and declarative business process modeling notations. This study indicates that imperative process models are significantly more understandable than declarative models, but the authors also state that the participants had more previous experience with imperative process modeling than with declar-

ative process modeling. Moreover, the sample size (28 participants) is rather small, which is a threat to validity of this study.

Mendes Cunha et al. [192] try to improve declarative business process modeling by taking the comments of 4 persons into consideration. The resulting language is based on the same behavioral constraint patterns, but it proposes different graphical notations. Obviously, the small number of participants and the lack of evaluation of the proposed alternative graphical elements are serious threats to validity.

8.9. Conclusion and Future Work

8.9.1. Summary

The results of this controlled experiment study with 116 participants did not reveal any significant difference in understandability, nor in any other tested aspects (i.e., perceived learning difficulty, perceived application difficulty, personal interest in using the representation, perceived practical applicability, perceived potential for further improvement of the behavioral constraint representations), between graphical, textual, and hybrid behavioral constraint representations. Merely the descriptive statistics and the results of the analysis of free text answers are slightly in favor of the textual behavioral constraint approaches tested. The achieved correctness is rather low on average in all experiment groups. Prior studies on the understandability of textual behavioral constraint approaches (cf. Chapter 6) yielded higher correctness values (about 70% on the average in PSP) when access to learning material and other material (e.g., hand-written notes) was granted during the experiment session. That is, it appears to be difficult to deduce the meaning of pattern-based behavioral constraints from their textual and/or graphical representations without additional support. The analysis of the given free text answers regarding positive/negative aspects and suggestions for improvement (cf. Section 8.6) provides additional evidence in that regard.

8.9.2. Impact

Since there appears to be no significant difference in understandability of textual and graphical behavioral constraint approaches, the results of this empirical study might indicate that the tested representations can be used interchangeably. However, a major obstacle in this regard could be the overall low level of achieved correctness, which must be further investigated. In response to the low level of achieved correctness, this study indicates two angles of approach for further research and improvement of textual and graphical behavioral constraint representations, namely the representation itself (i.e., finding better graphical and/or textual representations) and the

technology support provided (i.e., the support provided by a behavioral constraint modeling tool or by analysis, refactoring, and debugging tools).

Our carefully designed and conducted empirical study can work as a solid foundation for further empirical evaluations of pattern-based behavioral constraint representations and their future development.

8.9.3. Future Work

The experiment could be repeated with different user groups (e.g., industrial practitioners) to gain further insights in the understandability of the representation from different perspectives. Other experiments could be run to further investigate the results. Experiments with different symbols in graphical behavioral constraint representations and variations of the terms used in textual approaches are also opportunities for future research. For example, a new representation could be introduced to the current experimental setup that streamlines the hybrid *Declare* approach (DGT) by reducing the number of available connector shapes to a single shape, just like relations in an ontology. For the evaluation of the understandability of interrelated behavioral constraint collections or the creation process of such, qualitative studies that are based on eye-tracking and think-aloud protocols [201] would further evolve the body of knowledge. Moreover, the presented study focuses on the understandability of already given textual and graphical behavioral constraints, so conducting an experiment on the understandability related to authoring textual and graphical constraints would be another interesting possibility for future research. In this regard, adequate tool support is assumed to be a major topic. The results of this experiment indicate that behavioral constraints in which the order of the involved states is of importance are particularly difficult to understand, so these elements should receive special attention. To improve the understandability of these constraints, a behavioral constraint editor could, for example, provide a tooltip with an animation that illustrates the temporal order of the involved elements (e.g., showing sample traces with the corresponding truth value state). Also, the textual terms and/or graphical elements of the representations should be revisited. For example, the response pattern (both in the textual and/or graphical form) can be easily misunderstood as a strict sequence (e.g., as known from procedural/imperative modeling languages). Such ambiguities must be avoided to achieve higher levels of understanding. An alternative textual representation of the response pattern, which might leave less room for misunderstandings by emphasizing the temporal order, could be A `at time` ta `requires` B `at time` $tb > ta$. The corresponding amended *Declare* notation of the response pattern is shown in Figure 8.21. Such amendments can be the starting point for further empirical evaluations with the goal to improve the understandability of behavioral constraint representations.

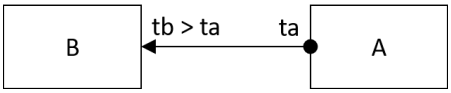


Figure 8.21.: Alternative `response` notation stating the temporal order of the involved elements explicitly by labels

Part IV.

Conclusions

9. Conclusions and Future Work

In this final chapter, we will discuss to what extent the research questions have been covered, the contributions made, and identify current limitationsBased on these limitations, we state opportunities for future work. In the future work section, we try to outline immediate next steps for conducting research based on the current results as presented in this thesis.

9.1. Research Questions Revisited

In Chapter 1, we formulated three main research questions and their refinements (or sub-questions) that were subsequently addressed in Chapters 2 to 8. In this section, we revisit the research questions in order to summarize the main contributions made by this thesis.

Research Question 1 (RQ 1)

How can behavioral consistency during case modeling be supported?

- RQ 1.1: *How can the behavior of case templates be captured as a state-transition system for model checking?*
- RQ 1.2: *Since model checking is known to be computationally expensive, how can case templates be model checked efficiently?*
- RQ 1.3: *Can model checking be applied to real-world case templates that are realistic in size and structure?*

RQ 1 has been addressed in Chapter 2.

The verification of case templates is realized by model checking, which is based on exhaustive state space exploration. This is the major difference to testing, which usually does not cover all possible ways of execution. To quote Dijkstra: “Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence”; that is, the major advantage of model checking over testing is its completeness in considering all possible

ways a case model may be executed.

To counteract the computational complexity of model checking, we suggested several novel state space reduction techniques for case templates. These reductions are carried out in advance of generating a state-transition system that models the execution behavior of a case template. Besides the reduction techniques, a major contribution are the transformation rules which describe how a case template can be transformed to a state-transition system that can be used by a model checker for exhaustive state space exploration. Eventually, we studied the approach in the context of two real-world case templates that are representative for semi-structured and highly-structured case templates, and realistic in size. This evaluation was carried out on off-the-shelf hardware, yet it resulted in fast to acceptable response times, depending on the structure of a case and the behavioral constraint under investigation.

Besides the technical contributions, the prototype was presented at several occurrences to ISIS Papyrus‘ employees, management, and customers. The great importance of behavioral consistency was acknowledged and triggered discussions regarding complementing or superseding the limited testing-based validation capabilities of the ISIS Papyrus ACM software.

Research Question 2 (RQ 2)

How can business domains and their terminology be considered during behavioral constraint modeling?

RQ 2 has been addressed in Chapter 3.

In an early prototyping phase of behavioral constraint authoring in which we used Xtext[1] and Xtend[2], we soon realized that a tighter business integration will be necessary to enable practical exploitability. Consequently, we created a new prototype that became fully integrated with a business ontology, and a front-end for behavioral constraint which is able to query the business ontology in combination with language elements stemming from behavioral constraint patterns. This prototype has drawn a great degree of attention both from customers and management of ISIS Papyrus, because it is able to combine two worlds, namely formal verification and business rules. The formal verification community tends to focus on technical, mathematical, and logic-based aspects, but it predominantly ignores practical applicability in a business context (cf. Elgammal et al. [23]). On the other hand, approaches such as RuleSpeak[3] and SBVR [24] have a strong business focus, but they do not or only insufficiently support formal behavioral verification. Our behavioral consistency approach contributes significantly to these currently

[1] https://www.eclipse.org/Xtext/
[2] https://www.eclipse.org/xtend/
[3] https://www.rulespeak.com/

existing shortcomings of existing approaches. The great business value of the approach has been acknowledged by ISIS Papyrus, and stimulated the development of a novel ACM solution called *Papyrus Converse* (cf. [202], [203]). Consequently, besides the foundational, technical, and empirical contribution of our work, a first industry adoption of a subset of our contributions has been achieved.

Research Question 3 (RQ 3)

How can behavioral consistency during case execution be supported?

- RQ 3.1: *How can prescriptiveness be avoided while providing support for behavioral consistency?*
- RQ 3.2: *How can implicit knowledge be leveraged for user guidance (i.e., the recommendation of next actions)?*

RQ 3 has been addressed in Chapter 4.

We presented a behavioral consistency framework for ACM that shifts the control over behavioral constraints from technical to business users. The presented approach aims at supporting business users in being consistent with existing behavioral constraints while not jeopardizing flexibility during case executions. Particularly, business users may take any action they consider appropriate or necessary to achieve a business goal. That is, the approach informs business users about existing or pending behavioral consistency issues, but it does not hinder or force any specific action. Nonetheless, behavioral consistency issues are properly documented, and recommendations are made to avoid or to resolve behavioral consistency issues. These recommendations are based on decisions made during other case executions. That is, implicit knowledge is captured automatically for user guidance by machine learning. Business users are also encouraged to explicitly document their knowledge by behavioral constraints which is another possibility for transferring knowledge to other business users.

Asides from these conceptual contributions of the proposed framework, the combination of business-driven behavioral constraints and machine learning has become one of the core concepts in *Papyrus Converse* (cf. [202], [203]).

Research Question 4 (RQ 4)

How can it be supported that a formal behavioral constraint specification matches the intent of its creator?

RQ 4 has been addressed in Chapter 5.

Linear Temporal Logic (LTL) is a widely used specification language for model checking and runtime verification. Despite the importance of correct specifications, the correctness of the LTL specification seems to be often taken for granted. Even long-existing pattern repositories are not free from incorrect LTL specifications (cf. Section 5.5). We contributed a novel approach for plausibility checking of LTL specifications and demonstrated its applicability in the business process management field. Nonetheless, the approach is generalizable to other domains that make use of LTL specifications. Consequently, the plausibility checking approach has the potential to improve the correctness of LTL specifications in various domain, which can contribute to better quality of program code and software in general.

Research Question 5 (RQ 5)

How understandable are existing representative behavioral constraint languages?

- RQ 5.1: *How understandable are existing behavioral constraints modeled in different major languages, and are there significant differences?*
  This research question has been addressed in Chapter 6.
- RQ 5.2: *How understandable are major behavioral constraint languages when applied for modeling, and are there significant differences?*
  This research question has been addressed in Chapter 7.
- RQ 5.3: *Is there a difference in understandability between graphical and textual behavioral constraint approaches?*
  This research question has been addressed in Chapter 8.

This research question and its refinements (i.e., sub-questions) resulted in several empirical studies. The empirical studies were concerned with the evaluation of major representations for the specification of behavioral constraints. As the results of these studies indicate, from the set of representative tested approaches, pattern-based behavioral constraints appear to be the most suitable approach as they provide the highest level of understandability, both for reading existing behavioral constraints and in the creation process of new behavioral constraints. Fur-

thermore, the results do not indicate any significant differences in understandability between graphical and textual pattern-based behavioral constraints. We opted to go with textual pattern-based behavioral constraints for supporting a structured natural language approach that enables business-driven behavioral constraint authoring in junction with business terminology delivered by the business ontology (cf. Chapter 3). While pattern-based approaches provide a high level of understandability, their expressiveness is limited to a set of available patterns. That is, it might be necessary to extend the pattern catalog. To support the formalization of new patterns, we proposed an approach for checking the plausibility of pattern formalizations in LTL by plausibility specifications in EPL (cf. Chapter 5). Our empirical studies suggest that EPL specifications are significantly more understandable than LTL specifications, which indicates that EPL can be a suitable language for writing plausibility specifications.

Beyond the scope of behavioral consistency in ACM, our empirical studies contribute to the general body of knowledge on the understandability of the tested representative behavioral constraint approaches. Our empirical groundwork has the potential to stimulate further empirical studies on behavioral constraint representations, to trigger new developments that improve the state of the art, and to help practitioners in choosing the right approach for a specific use case or area of usage.

9.2. Limitations

The work presented in this dissertation contributes significantly to supporting behavioral consistency in Adaptive Case Management, but some issues still remain. In this section, the main limitations of this thesis are summarized.

The proposed model checking approach (cf. Chapter 2) does not scale to large problem sizes due to the high computational complexity of model checking despite making use of the proposed reduction and abstraction techniques.

Despite standardization efforts in the case management domain (cf. CMMN [19]), so far, there is no unified modeling approach for ACM case templates, and structures and semantics of ACM case templates seem to be subject to continuous change. The proposed model checking approach (cf. Chapter 2) is oriented on current conceptual aspects of CMMN and ISIS Papyrus' ACM solution. As such, the state transition system is likely to change in reaction to evolutionary processes associated with case modeling. For example, the current model checking approach does not yet consider newly proposed conceptual developments such as flow dependencies [204], which enable a lightweight and seamless integration of process flows in case templates.

The controlled experiments conducted suggest that a pattern-based behavioral constraint approach can provide a high level of understandability, but the actual implementation of behavioral

constraint authoring as presented in Chapter 3 has not been evaluated (e.g., regarding usability or user acceptance) in any usability study thus far. Industrial case studies (cf. [75], [79]) as well as internal evaluations of the proposed behavioral constraint authoring approach carried out by ISIS Papyrus have shown promising results and eventually led to the development of a new product called *Papyrus Converse* (cf. [202], [203]).

Potential benefits associated with making use of the behavioral consistency support framework presented in Chapter 4 have not yet been sufficiently empirically evaluated. In this sense, it is also possible that such an evaluation reveals potential drawbacks of the approach that are yet unknown. The prototype implementation of the framework has been demonstrated to customers of ISIS Papyrus as a teaser of new feature developments at multiple *ISIS Papyrus Open House and User Conference* events. This kind of exposure gave us valuable insights in customer demands and has led to continuous improvements of the overall approach. Unfortunately, further exposure of the prototype to customers of ISIS Papyrus has been out of question. On the one hand, it would have been too risky for ISIS Papyrus to ship software to customers in a prototype state, while on the other hand, customers are interested in mature software that is stable and meant for use in production. In cooperation with ISIS Papyrus, case studies were conducted which indicated the practical applicability of the framework (cf. [75], [79]). Eventually, the combination of business-driven behavioral constraints and machine learning has been adopted by ISIS Papyrus in the new product *Papyrus Converse* (cf. [202], [203]).

Our empirical studies indicate that EPL, the language on which plausibility specifications are based, provides a higher level of understandability than LTL, which provides to some extent evidence for the applicability of the plausibility checking approach discussed in Chapter 5. However, the plausibility checking approach itself has not been tested empirically. Consequently, it is yet unknown to what extent the approach really can contribute to writing more correct LTL specifications.

The understandability of graphical and textual pattern-based behavioral constraint representations has only been studied on basis of already given constraints. That is, we do not yet know whether there are differences in understandability with regards to creating such constraints.

While the empirical study presented in Chapter 6 has been replicated by a second controlled experiment run, the studies presented in Chapter 7 and Chapter 8 have not yet been replicated. Also the former mentioned study should be replicated by other researchers in different environments, with different participants, and different populations, to test whether the results are generalizable.

9.3. Future Work

On basis of the stated limitations, we identify the following opportunities for future work.

Possible next steps in improving behavioral verification of case templates (cf. Chapter 2) can be concerned with finding supplementary methods that counteract current limitations regarding scalability and response times. In this regard, it would make sense to use computationally less expensive methods such as approximate model checking (cf. Owen & Menzies [54]) to get fast results or when traditional model checking cannot be applied due to high model complexities.

Behavioral consistency checking of case templates (cf. Chapter 2) must be aligned with new developments in case modeling which are yet to come. Parts of CMMN can be applied for representing ACM case templates (cf. Kurz et al. [37]), but this standard as well as (proprietary or specialized) models in (commercial) ACM software or academic inputs are ever-evolving. Consequently, new developments (e.g., seamlessly integrated process flows [204]) must be considered in behavioral consistency checking methods as well.

The proposed approach for enabling business users in behavioral constraint authoring (cf. Chapter 3) must be further evaluated. Testing the understandability of the behavioral patterns involved can be seen as a first step, but this evaluation did not consider the actual application of the software in a business environment. A potential next step could be a qualitative study (e.g., using interviews and thinking aloud protocols) with an adequately/reasonably large set of business professionals (e.g., $15 - 30$ compliance officers).

Valuable feedback by the management and customers of ISIS Papyrus has influenced the behavioral consistency framework presented in Chapter 4, but this framework has not yet been sufficiently empirically studied. It would be interesting to conduct a field test in that regard to understand to what extent the behavioral consistency can be improved by the proposed framework. If a field test is not applicable (e.g., due to high costs or risks involved in a productive environment), an alternative could be an evaluation in a lab (e.g., with student proxies). For example, an experiment group could be exposed to the implementation of the framework in the ISIS Papyrus ACM software while the control group uses the same software without the prototype implementation of the framework. An assumption to test could be that the overall behavioral consistency achieved by the experiment group is higher than in the control group.

While the empirical work presented in Chapter 6 and Chapter 7 provides evidence that EPL-based behavioral constraints provide a higher level of understandability than LTL-based behavioral constraints which could imply the suitability of EPL-based behavioral constraints as plausibility specifications, the plausibility checking approach (cf. Chapter 5) itself has not been directly empirically evaluated. Consequently, an opportunity for future work is directly studying the impact of plausibility checking by a controlled experiment in which the experiment group is

exposed to the plausibility checking approach while the control group has to work without the plausibility checking tool. Our expectation would be that the experiment group makes overall less errors than the control group.

Differences in understandability between graphical and textual pattern-based constraints have only been studied with regards to already given behavioral constraints. We would suggest to further study the understandability in the context of behavioral constraint modeling (i.e., behavioral constraint authoring) by another controlled experiment.

Another opportunity for future work is the replication of the empirical studies presented in this thesis. We also would like to invite other researchers to further study the understandability of behavioral constraint representations, either by replication of our studies or by designing and conducting their own experiments. Other kinds of experiment designs are conceivable. For example, we have not yet studied the cognitive load involved in understanding given behavioral constraints or authoring them. Monitoring brain waves could give us insights in the cognitive load involved in such tasks (cf. Majdic et al. [205]).

A. Appendix

A.1. Experiment on Graphical and Textual Behavioral Constraint Representations — Sample Solutions of Experimental Tasks

Table A.1.: Sample solution of Task 1

Group	Sample Solution
EPL	init ==> TS not 'Evaluate Loan Risk'.completed until 'Officially Sign Contract'.started ==> PV 'Evaluate Loan Risk'.completed ==> PS init ==> TS 'Evaluate Loan Risk'.role != 'Branch Office Manager' ==> PV init ==> TS 'Officially Sign Contract'.role != 'Branch Office Manager' ==> PV
LTL	! 'Officially Sign Contract'.started W 'Evaluate Loan Risk'.completed G! ('Officially Sign Contract'.role != 'Branch Office Manager') G! ('Evaluate Loan Risk'.role != 'Branch Office Manager')
PSP	'Evaluate Loan Risk'.completed precedes 'Officially Sign Contract'.started 'Officially Sign Contract'.role != 'Branch Office Manager' never occurs 'Evaluate Loan Risk'.role != 'Branch Office Manager' never occurs

Table A.2.: Sample solution of Task 2

Group	Sample Solution

EPL	`init ==> TS`
	`every('Check Customer Privilege'.completed -> 'Check Credit Worthiness'.started) ==> TS`
	`every 'Check Customer Privilege'.started ==> TV`
	
	`init ==> TS`
	`not 'Check Customer Privilege'.completed until 'Evaluate Loan Risk'.started ==> PV`
	`'Check Customer Privilege'.completed ==> PS`
	
	`init ==> TS`
	`not 'Check Credit Worthiness'.completed until 'Evaluate Loan Risk'.started ==> PV`
	`'Check Credit Worthiness'.completed ==> PS`

LTL	`G('Check Customer Privilege'.completed -> F 'Check Credit Worthiness'.started)`
	
	`! 'Evaluate Loan Risk'.started W 'Check Customer Privilege'.completed`
	
	`! 'Evaluate Loan Risk'.started W 'Check Credit Worthiness'.completed`

PSP	`'Check Customer Privilege'.completed leads to 'Check Credit Worthiness'.started`
	
	`'Check Customer Privilege'.completed precedes 'Evaluate Loan Risk'.started`
	
	`'Check Credit Worthiness'.completed precedes 'Evaluate Loan Risk'.started`

Table A.3.: Sample solution of Task 3

Group	Sample Solution
EPL	init ==> TS not 'Preoperative Screening'.completed until 'Laparoscopic Gastrectomy'.started ==> PV 'Preoperative Screening'.completed ==> PS init ==> TS not 'Preoperative Screening'.completed until 'Open Gastrectomy'.started ==> PV 'Preoperative Screening'.completed ==> PS init ==> TS 'Open Gastrectomy'.started leads-to 'Laparoscopic Gastrectomy'.started ==> PV 'Laparoscopic Gastrectomy'.started leads-to 'Open Gastrectomy'.started ==> PV init ==> TS every('Laparoscopic Gastrectomy'.completed leads-to 'Nursing'.started) ==> TS every 'Laparoscopic Gastrectomy'.completed ==> TV init ==> TS every('Open Gastrectomy'.completed leads-to 'Nursing'.started) ==> TS every 'Open Gastrectomy'.completed ==> TV
LTL	! 'Laparoscopic Gastrectomy'.started W 'Preoperative Screening'.completed ! 'Open Gastrectomy'.started W 'Preoperative Screening'.completed (F 'Open Gastrectomy'.started -> G! 'Laparoscopic Gastrectomy'.started) & (F 'Laparoscopic Gastrectomy'.started -> G! 'Open Gastrectomy'.started) G('Laparoscopic Gastrectomy'.completed -> F 'Nursing'.started) G('Open Gastrectomy'.completed -> F 'Nursing'.started)
PSP	'Preoperative Screening'.completed precedes 'Laparoscopic Gastrectomy'.started 'Preoperative Screening'.completed precedes 'Open Gastrectomy'.started after 'Open Gastrectomy'.started ['Laparoscopic Gastrectomy'.started never occurs] after 'Laparoscopic Gastrectomy'.started ['Open Gastrectomy'.started never occurs] 'Laparoscopic Gastrectomy'.completed leads to 'Nursing'.started 'Open Gastrectomy'.completed leads to 'Nursing'.started

Table A.4.: Sample solution of Task 4

Group	Sample Solution
EPL	`init ==> TS` `every('Lead Contamination identified' leads-to not 'Renovation'.completed until ['Cleaning'.running and not 'Presence of Certified Renovator'.running]) ==> PV`
LTL	`G('Lead Contamination identified' & ! 'Renovation'.completed -> (! ('Cleaning'.running & ! 'Presence of Certified Renovator'.running) W 'Renovation'.completed))`
PSP	`after 'Lead Contamination identified' until 'Renovation'.completed ['Cleaning'.running and not 'Presence of Certified Renovator'.running never occurs]`

Table A.5.: Sample solution of Task 5

Group	Sample Solution

EPL	`init ==> TS`
	`not b.finished until r.started ==> PV`
	`not r.started until b.finished ==> PS`
	
	`init ==> TS`
	`not d.finished until r.started ==> PV`
	`not r.started until d.finished ==> PS`
	
	`init ==> TS`
	`not p.started until [y < 1978 and (t = 'residential house' or t = 'apartment' or t = 'child-occupied facility') and renovation.started] ==> PV`
	`p.started ==> PS`

LTL	`!r.started W (b.finished & !r.started)`
	
	`!r.started W (d.finished & !r.started)`
	
	`!(y < 1978 & (t = 'residential house' | t = 'apartment' | t = 'child-occupied facility') & renovation.started)) W p.started`

PSP	`before r.started [ b.finished occurs ]`
	
	`before r.started [ d.finished occurs ]`
	
	`p.started precedes (y < 1978 and (t = 'residential house' or t = 'apartment' or t = 'child-occupied facility') and renovation.started)`

A.2. Experiment on Graphical and Textual Behavioral Constraint Representations — Evaluation of Normality Assumption & Parametric Testing by Welch's t-test

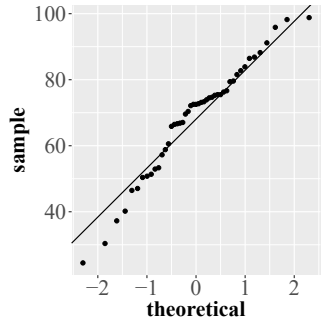
Since the dependent variables *syntactic correctness*, *semantic correctness*, and *response time* are interval-scaled, parametric methods would be the first choice, but the multivariate normality assumption appears to be violated according to the Shapiro-Wilk tests of multivariate normality in Table A.6, so multivariate parametric testing (MANOVA) is ruled out. According to Shapiro-Wilk tests of univariate normality in Table A.7, there are no indications of non-normality, but there are signs of non-normality in the descriptive statistics in Section 7.4.2. Also, normal QQ plots of the data show signs of non-normality (cf. Figure A.1). Due to the large sample sizes ($n > 30$) in SE2, it might be valid to assume that the Central Limit Theorem holds. In ASE, the sample size is not large enough to claim that. Since there is uncertainty regarding normality, we prefer to apply non-parametric testing (cf. Section 7.4.3). Nonetheless, in case of assumed normality, we also show that parametric testing yields similar results (cf. Table A.8 and Table A.9). This additional testing was performed since the violation of normality is based on the interpretation of plots only, which leaves room for subjectivity.

Table A.6.: Shapiro-Wilk test of multivariate normality (* for $\alpha = 0.05$, ** for $\alpha = 0.01$, * for $\alpha = 0.001$)

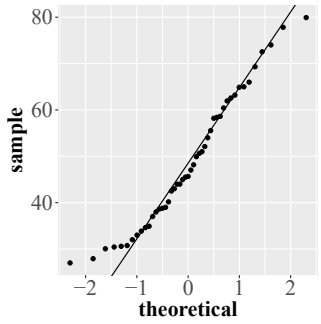
Group	**SE2**	**ASE**
LTL	$W = 0.96138$ $p = 0.09547$	$W = 0.89909$ $p = 0.03359$ *
PSP	$W = 0.94299$ $p = 0.02316$ *	$W = 0.96263$ $p = 0.6813$
EPL	$W = 0.96448$ $p = 0.1618$	$W = 0.91843$ $p = 0.05393$

Table A.7.: Shapiro-Wilk test of univariate normality (* for $\alpha = 0.05$, ** for $\alpha = 0.01$, * for $\alpha = 0.001$)

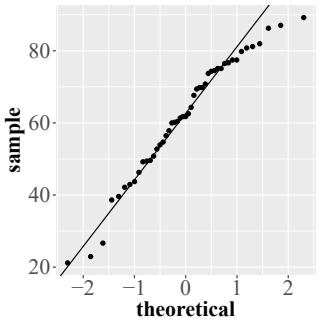
Group	Dependent Variable	SE2	ASE
LTL	Syntactic Correctness	$W = 0.97501$ $p = 0.3526$	$W = 0.951$ $p = 0.3558$
	Semantic Correctness	$W = 0.95487$ $p = 0.05047$	$W = 0.94524$ $p = 0.2761$
	Response Time	$W = 0.98169$ $p = 0.6127$	$W = 0.95759$ $p = 0.469$
PSP	Syntactic Correctness	$W = 0.96204$ $p = 0.1296$	$W = 0.91825$ $p = 0.138$
	Semantic Correctness	$W = 0.98311$ $p = 0.7232$	$W = 0.96835$ $p = 0.7889$
	Response Time	$W = 0.95661$ $p = 0.0789$	$W = 0.89976$ $p = 0.06731$
EPL	Syntactic Correctness	$W = 0.96063$ $p = 0.1139$	$W = 0.9358$ $p = 0.1314$
	Semantic Correctness	$W = 0.98412$ $p = 0.7652$	$W = 0.96757$ $p = 0.6075$
	Response Time	$W = 0.98163$ $p = 0.6606$	$W = 0.94779$ $p = 0.2425$



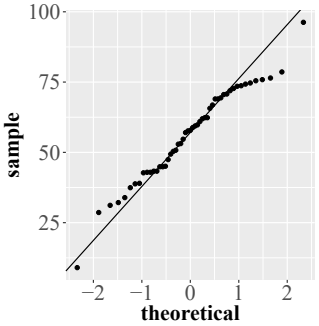
(a) Syntactic Correctness data of PSP group in SE2



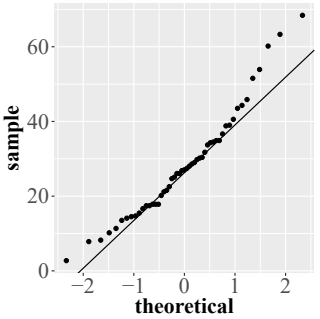
(b) Response time data of PSP group in SE2



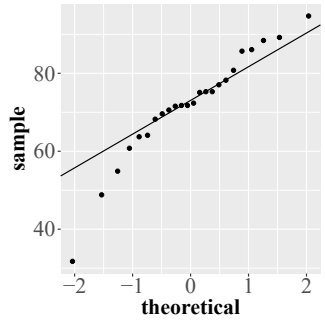
(c) Syntactic Correctness data of EPL group in SE2



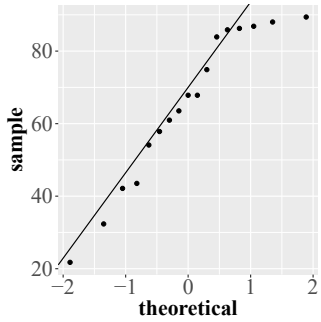
(d) Syntactic Correctness data of LTL group in SE2



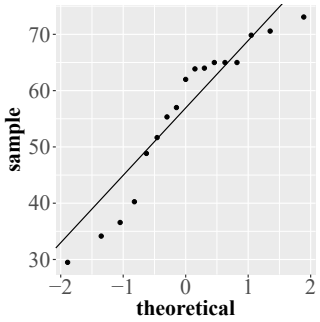
(e) Semenatic Correctness data of LTL group in SE2



(f) Syntactic Correctness data of EPL group in ASE



(g) Syntactic Correctness data of PSP group in ASE



(h) Response time data of PSP group in ASE

Figure A.1.: Normal QQ plots

Table A.8.: Welch's t-test of syntactic/semantic correctness and response time in SE2, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Welch [206]) and adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$]

			PSP/LTL	**PSP/EPL**	**EPL/LTL**
Software Engineering 2 (Bachelor-Level Course)	**Syntactic Correctness**	t	3.5867	1.9529	1.5761
		df	94.691	91.994	94.863
		$CI\ low$	0.0651	0.0102	-0.0029
		$CI\ high$	-	-	-
		$mean\ x$	0.6864	0.6864	0.6182
		$mean\ y$	0.5652	0.6182	0.5652
		p	0.0003	0.0269	0.0592
		$FDR\ adjusted\ p$	0.0013	0.0647	0.1109
		$level\ of\ significance$	**	-	-
	Semantic Correctness	t	7.0831	3.8143	3.2849
		df	93.444	91.596	95.061
		$CI\ low$	0.1661	0.0679	0.048
		$CI\ high$	-	-	-
		$mean\ x$	0.5019	0.5019	0.382
		$mean\ y$	0.2849	0.382	0.285
		p	1.3×10^{-10}	0.0001	0.0007
		$FDR\ adjusted\ p$	1.9×10^{-9}	0.0009	0.0027
		$level\ of\ significance$	***	***	**
	Response Time	t	1.861	1.2971	0.5009
		df	93.123	91.955	93.774
		$CI\ low$	-	-	-
		$CI\ high$	9.8170	8.6859	5.9519
		$mean\ x$	48.6769	48.6769	44.869
		$mean\ y$	43.4902	44.869	43.4902
		p	0.9671	0.9011	0.6912
		$FDR\ adjusted\ p$	0.9671	0.9655	0.7975
		$level\ of\ significance$	-	-	-

Table A.9.: Welch's t-test of syntactic/semantic correctness and response time in ASE, one-tailed with confidence intervals calculated for $\alpha = 0.05$ (cf. Welch [206]) and adjusted p-values (cf. Benjamini & Hochberg [152]) [Level of significance: * for $\alpha = 0.05$, ** for $\alpha = 0.01$, *** for $\alpha = 0.001$]

			PSP/LTL	**PSP/EPL**	**EPL/LTL**
Software Engineering 2 (Bachelor-Level Course)	**Syntactic Correctness**	t	1.3239	-1.1642	3.371
		df	28.887	25.573	40.268
		$CI\ low$	-0.023	-0.1671	0.0746
		$CI\ high$	-	-	-
		$mean\ x$	0.6513	0.6513	0.7191
		$mean\ y$	0.5701	0.7191	0.5701
		p	0.098	0.8724	0.0008
		$FDR\ adjusted\ p$	0.2449	0.9254	0.0062
		$level\ of\ significance$	-	-	**
	Semantic Correctness	t	3.1981	-0.5583	4.7839
		df	29.231	29.156	42.581
		$CI\ low$	0.0754	-0.1125	0.1223
		$CI\ high$	-	-	-
		$mean\ x$	0.4693	0.4693	0.4971
		$mean\ y$	0.3085	0.4971	0.3085
		p	0.0017	0.7095	10^{-5}
		$FDR\ adjusted\ p$	0.0083	0.8869	0.0002
		$level\ of\ significance$	**	-	***
	Response Time	t	0.7786	-0.6463	1.4701
		df	35.654	35.389	41.049
		$CI\ low$	-	-	-
		$CI\ high$	11.6186	4.5789	13.9503
		$mean\ x$	55.9853	55.9853	58.8236
		$mean\ y$	52.3191	58.8236	52.3191
		p	0.7793	0.2611	0.9254
		$FDR\ adjusted\ p$	0.8992	0.4352	0.9254
		$level\ of\ significance$	-	-	-

Bibliography

[1] W. M. P. van der Aalst, „Process-Aware Information Systems: Lessons to Be Learned from Process Mining“, in *Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems*, K. Jensen and W. M. P. van der Aalst, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26, ISBN: 978-3-642-00899-3. DOI: 10.1007/978-3-642-00899-3_1. [Online]. Available: https://doi.org/10.1007/978-3-642-00899-3_1.

[2] OMG, *Unified Modeling Language 2.4.1 Superstructure*, http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF, Last accessed: February 22, 2019.

[3] K. D. Swenson, *Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done*. Meghan-Kiffer Press, 2010, ISBN: 0929652126 9780929652122.

[4] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Aalst, „Process Flexibility: A Survey of Contemporary Approaches“, in *Advances in Enterprise Engineering I*, ser. Lecture Notes in Business Information Processing, J. Dietz, A. Albani, and J. Barjis, Eds., vol. 10, Springer Berlin Heidelberg, 2008, pp. 16–30, ISBN: 978-3-540-68643-9. DOI: 10.1007/978-3-540-68644-6_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68644-6_2.

[5] K. Swenson, *What is Case Management?*, http://adaptivecasemanagement.org/AboutACM.html, Accessed: 2014-08-21.

[6] W. M. P. van der Aalst and M. Weske, „Case Handling: A New Paradigm for Business Process Support“, *Data Knowl. Eng.*, vol. 53, no. 2, pp. 129–162, May 2005, ISSN: 0169-023X. DOI: 10.1016/j.datak.2004.07.003.

[7] M. J. Pucher, „Considerations for Implementing Adaptive Case Management“, in *Taming the Unpredictable Real World Adaptive Case Management: Case Studies and Practical Guidance*, L. Fischer, Ed., Future Strategies Inc., 2011.

[8] M. A. Marin, M. Hauder, and F. Matthes, „Case Management: An Evaluation of Existing Approaches for Knowledge-Intensive Processes“, in *AdaptiveCM'15*, Aug. 2015.

[9] H. R. Motahari-Nezhad and K. D. Swenson, „Adaptive Case Management: Overview and Research Challenges“, in *2013 IEEE 15th Conference on Business Informatics*, Jul. 2013, pp. 264–269. DOI: 10.1109/CBI.2013.44.

[10] C. Di Ciccio, A. Marrella, and A. Russo, „Knowledge-intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches“, *J. Data Semantics*, vol. 4, no. 1, pp. 29–57, 2015, ISSN: 1861-2032. DOI: 10.1007/s13740-014-0038-4.

[11] Michael G. Oxley, *H.R.3763 - Sarbanes-Oxley Act of 2002*, https://www.congress.gov/bill/107th-congress/house-bill/3763/text, Last accessed: February 22, 2019, 2002.

[12] Bank for International Settlements, *Basel III: International framework for liquidity risk measurement, standards and monitoring*, https://www.bis.org/publ/bcbs188.htm, Last accessed: February 22, 2019, 2010.

[13] United States Environmental Protection Agency, *EPA's Lead-Based Paint Renovation, Repair and Painting (RRP) Rule*, https://www.epa.gov/lead/lead-renovation-repair-and-painting-program-rules, Last accessed: February 22, 2019, 2011.

[14] M. Rovani, F. M. Maggi, M. de Leoni, and W. M. van der Aalst, „Declarative process mining in healthcare“, *Expert Systems with Applications*, vol. 42, no. 23, pp. 9236–9251, 2015, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2015.07.040. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741741500500X.

[15] R. Eshuis, „Symbolic Model Checking of UML Activity Diagrams“, *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 1–38, Jan. 2006, ISSN: 1049-331X. DOI: 10.1145/1125808.1125809.

[16] O. Kherbouche, A. Ahmad, and H. Basson, „Using model checking to control the structural errors in BPMN models“, in *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, May 2013, pp. 1–12. DOI: 10.1109/RCIS.2013.6577723.

[17] Z. Sbai, A. Missaoui, K. Barkaoui, and R. Ben Ayed, „On the verification of business processes by model checking techniques“, in *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, vol. 1, Oct. 2010, pp. V1-97-V1-103. DOI: 10.1109/ICSTE.2010.5608905.

[18] I. Raedts, M. Petković, Y. S. Usenko, J. M. van der Werf, J. F. Groote, and L. Somers, „Transformation of BPMN models for Behaviour Analysis“, in *MSVVEIS*, J. C. Augusto, J. Barjis, and U. U. Nitsche, Eds., INSTICC press, 2007, pp. 126–137.

[19] OMG, *Case Management Model and Notation (CMMN) Version 1.0*, http://www.omg.org/spec/CMMN/1.0/PDF/, Last accessed: 2015-03-16.

[20] ——, *BPMN 2.0*, http://www.omg.org/spec/BPMN/2.0/PDF, Last accessed: February 22, 2019, 2011.

[21] J. Koehler, G. Tirenni, and S. Kumaran, „From business process model to consistent implementation: a case for formal verification methods“, in *Enterprise Distributed Object Computing Conference, 2002. EDOC '02. Proceedings. Sixth International*, 2002, pp. 96–106. DOI: 10.1109/EDOC.2002.1137700.

[22] A. P. Sistla and E. M. Clarke, „The Complexity of Propositional Linear Temporal Logics“, *J. ACM*, vol. 32, no. 3, pp. 733–749, Jul. 1985, ISSN: 0004-5411.

[23] A. Elgammal, O. Turetken, W.-J. van den Heuvel, and M. Papazoglou, „Formalizing and appling compliance patterns for business process compliance“, *Software & Systems Modeling*, vol. 15, no. 1, pp. 119–146, 2016, ISSN: 1619-1374. DOI: 10.1007/s10270-014-0395-3. [Online]. Available: http://dx.doi.org/10.1007/s10270-014-0395-3.

[24] OMG, *Semantics of Business Vocabulary and Rules^{TM} (SBVR^{TM})*, http://www.omg.org/spec/SBVR/, Last accessed: February 22, 2019.

[25] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Berlin-Heidelberg: Springer, 2012.

[26] A. Pnueli, „The Temporal Logic of Programs“, in *18th Annual Symposium on Foundations of Computer Science*, ser. SFCS '77, Washington, DC, USA: IEEE Computer Society, Oct. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1977.32.

[27] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, „Patterns in Property Specifications for Finite-state Verification“, in *21st International Conference on Software Engineering (ICSE)*, ser. ICSE '99, Los Angeles, California, USA: ACM, 1999, pp. 411–420, ISBN: 1-58113-074-0. DOI: 10.1145/302405.302672. [Online]. Available: http://doi.acm.org/10.1145/302405.302672.

[28] C. Haisjackl and S. Zugal, „Investigating Differences between Graphical and Textual Declarative Process Models“, in *Advanced Information Systems Engineering Workshops: CAiSE 2014 International Workshops, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, L. Iliadis, M. Papazoglou, and K. Pohl, Eds. Cham: Springer International Publishing, 2014, pp. 194–206, ISBN: 978-3-319-07869-4. DOI: 10.1007/978-3-319-07869-4_17. [Online]. Available: https://doi.org/10.1007/978-3-319-07869-4_17.

[29] W. Heijstek, T. Kuhne, and M. R. V. Chaudron, „Experimental Analysis of Textual and Graphical Representations for Software Architecture Design“, in *2011 International Symposium on Empirical Software Engineering and Measurement*, Sep. 2011, pp. 167–176. DOI: 10.1109/ESEM.2011.25.

[30] Z. Sharafi, A. Marchetto, A. Susi, G. Antoniol, and Y. G. Guéhéneuc, „An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension“, in *2013 21st International Conference on Program Comprehension (ICPC)*, May 2013, pp. 33–42. DOI: 10.1109/ICPC.2013.6613831.

[31] S. T. March and G. F. Smith, „Design and natural science research on information technology“, *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, 1995, ISSN: 0167-9236. DOI: https://doi.org/10.1016/0167-9236(94)00041-2.

[32] A. R. Hevner, S. T. March, J. Park, and S. Ram, „Design Science in Information Systems Research“, *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004, ISSN: 0276-7783.

[33] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, „A Design Science Research Methodology for Information Systems Research“, *J. Manage. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, Dec. 2007, ISSN: 0742-1222. DOI: 10.2753/MIS0742-1222240302. [Online]. Available: http://dx.doi.org/10.2753/MIS0742-1222240302.

[34] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000, ISBN: 0-7923-8682-5.

[35] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, „NuSMV 2: An OpenSource Tool for Symbolic Model Checking“, in *Proceedings of the 14th International Conference on Computer Aided Verification*, ser. CAV '02, London, UK, UK: Springer-Verlag, 2002, pp. 359–364, ISBN: 3-540-43997-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=647771.734431.

[36] Forrester Research, *The Forrester Wave™: Dynamic Case Management, Q1 2016*.

[37] M. Kurz, W. Schmidt, A. Fleischmann, and M. Lederer, „Leveraging CMMN for ACM: Examining the Applicability of a New OMG Standard for Adaptive Case Management“, ser. S-BPM ONE '15, Kiel, Germany: ACM, 2015, 4:1–4:9, ISBN: 978-1-4503-3312-2.

[38] C. Czepa, H. Tran, U. Zdun, S. Rinderle-Ma, T. Tran, E. Weiss, and C. Ruhsam, „Supporting Structural Consistency Checking in Adaptive Case Management“, in *CoopIS'15*, Oct. 2015, pp. 311–319.

[39] E. M. Clarke, „The Birth of Model Checking“, in *25 Years of Model Checking: History, Achievements, Perspectives*, O. Grumberg and H. Veith, Eds. Springer, 2008, pp. 1–26.

[40] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, „NUSMV: a new symbolic model checker“, *International Journal on Software Tools for Technology Transfer*, vol. 2, p. 2000, 2000.

[41] E. M. Clarke and E. A. Emerson, „Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic“, in *Logic of Programs, Workshop*, Springer, 1982, pp. 52–71, ISBN: 3-540-11212-X.

[42] W. M. P. van der Aalst and M. Pesic, „DecSerFlow: Towards a Truly Declarative Service Flow Language“, in *WS-FM 2006*, M. Bravetti, M. Núñez, and G. Zavattaro, Eds. Berlin, Heidelberg: Springer, 2006, pp. 1–23, ISBN: 978-3-540-38865-4. DOI: 10.1007/11841197_1. [Online]. Available: http://dx.doi.org/10.1007/11841197_1.

[43] K. L. McMillan, „The SMV System“, in *Symbolic Model Checking*. Springer, 1993, pp. 61–85.

[44] *Property Pattern Mappings for LTL*, http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml, Last accessed: February 22, 2019.

[45] K. D. Swenson, P. Nathaniel, M. J. Pucher, C. Webster, and A. Manuel, „How Knowledge Workers Get Things Done“, in. Future Strategies Inc., 2012, pp. 155–164.

[46] N. Herzberg, K. Kirchner, and M. Weske, „Modeling and Monitoring Variability in Hospital Treatments: A Scenario Using CMMN“, in *BPM'14 Workshops*, F. Fournier and J. Mendling, Eds. Springer, 2015, pp. 3–15.

[47] A. Awad, G. Decker, and M. Weske, „Efficient Compliance Checking Using BPMN-Q and Temporal Logic“, in *6th International Conference on Business Process Management*, ser. BPM '08, Milan, Italy: Springer, 2008, pp. 326–341, ISBN: 978-3-540-85757-0. DOI: http:{\slash}{\slash}dx.doi.org{\slash}10.1007{\slash}978-3-540-85758-7_24.

[48] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. van der Aalst, „Compliance monitoring in business processes: Functionalities, application, and tool-support“, *Information Systems*, vol. 54, pp. 209–234, 2015, ISSN: 0306-4379.

[49] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, „Towards a Compliance Support Framework for Adaptive Case Management“, in *AdaptiveCM'16*, Sep. 2016. [Online]. Available: http://eprints.cs.univie.ac.at/4752/.

[50] A. Nigam and N. S. Caswell, „Business Artifacts: An Approach to Operational Specification“, *IBM Syst. J.*, vol. 42, no. 3, pp. 428–445, Jul. 2003, ISSN: 0018-8670.

[51] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculin, „Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles“, in *WS-FM 2010*. Springer, 2011, pp. 1–24.

[52] P. Gonzalez, A. Griesmayer, and A. Lomuscio, „Verifying GSM-Based Business Artifacts“, ser. ICWS '12, IEEE Computer Society, 2012, pp. 25–32.

[53] H. Tran, F. U. Muram, and U. Zdun, „A Graph-Based Approach for Containment Checking of Behavior Models of Software Systems“, in *2015 IEEE 19th International Enterprise Distributed Object Computing Conference*, Sep. 2015, pp. 84–93. DOI: 10.1109/EDOC.2015.22.

[54] D. Owen and T. Menzies, „Lurch: a Lightweight Alternative to Model Checking“, in *In The International Conference on Software Engineering and Knowledge Engineering*, 2003, pp. 158–165.

[55] D. Owen, T. Menzies, M. Heimdahl, and J. Gao, „On the advantages of approximate vs. complete verification: bigger models, faster, less memory, usually accurate“, in *28th Annual NASA Goddard Software Engineering Workshop, 2003. Proceedings.*, Dec. 2003, pp. 75–81. DOI: 10.1109/SEW.2003.1270728.

[56] M. Pesic and W. M. P. van der Aalst, „A Declarative Approach for Flexible Business Processes Management“, in *International Conference on Business Process Management Workshops*, Vienna, Austria: Springer-Verlag, 2006, pp. 169–180, ISBN: 3-540-38444-8, 978-3-540-38444-1. DOI: 10.1007/11837862_18.

[57] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. v. d. Aalst, „Monitoring Business Constraints with the Event Calculus“, *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 1, 17:1–17:30, Jan. 2014, ISSN: 2157-6904. DOI: 10.1145/2542182.2542199. [Online]. Available: http://doi.acm.org/10.1145/2542182.2542199.

[58] M. Montali, „The ConDec Language“, in *Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 47–75, ISBN: 978-3-642-14538-4. DOI: 10.1007/978-3-642-14538-4_3. [Online]. Available: https://doi.org/10.1007/978-3-642-14538-4_3.

[59] L. T. Ly, S. Rinderle-Ma, and P. Dadam, „CAiSE, Hammamet, Tunisia“, in, B. Pernici, Ed. Springer, 2010, ch. Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems, pp. 9–23.

[60] T. Hildebrandt, M. Marquard, R. R. Mukkamala, and T. Slaats, „OTM Workshops, Graz, Austria“, in, Y. T. Demey and H. Panetto, Eds. Springer, 2013, ch. Dynamic Condition Response Graphs for Trustworthy Adaptive Case Management, pp. 166–171.

[61] S. Schönig and M. Zeising, „The DPIL Framework: Tool Support for Agile and Resource-Aware Business Processes“, in *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015.*, F. Daniel and S. Zugal, Eds., ser. CEUR Workshop Proceedings, vol. 1418, CEUR-WS.org, 2015, pp. 125–129. [Online]. Available: http://ceur-ws.org/Vol-1418/paper26.pdf.

[62] A. Elgammal, O. Turetken, and W.-J. Van Den Heuvel, „Using patterns for the analysis and resolution of compliance violations“, *Int'l Journal of Cooperative Information Systems*, vol. 21, no. 01, pp. 31–54, 2012. DOI: 10.1142/S0218843012400023.

[63] G. De Giacomo, R. D. Masellis, and M. Montali, „Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness“, in *AAAI*, C. E. Brodley and P. Stone, Eds., AAAI Press, 2014, pp. 1027–1033, ISBN: 978-1-57735-661-5.

[64] A. Armando and S. E. Ponta, „Model Checking of Security-sensitive Business Processes“, in *6th Int'l Conference on Formal Aspects in Security and Trust (FAST)*, Eindhoven, The Netherlands: Springer-Verlag, 2010, pp. 66–80. DOI: 10.1007/978-3-642-12459-4_6.

[65] M. Zeising, S. Schönig, and S. Jablonski, „Towards a common platform for the support of routine and agile business processes“, in *CollaborateCom*, Oct. 2014, pp. 94–103.

[66] J. Yu, T. P. Manh, J. Han, Y. Jin, Y. Han, and J. Wang, „Pattern Based Property Specification and Verification for Service Composition“, in *Web Information Systems – WISE 2006: 7th International Conference on Web Information Systems Engineering, Wuhan, China, October 23-26, 2006. Proceedings*, K. Aberer, Z. Peng, E. A. Rundensteiner, Y. Zhang, and X. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 156–

168, ISBN: 978-3-540-48107-2. DOI: 10.1007/11912873_18. [Online]. Available: http://dx.doi.org/10.1007/11912873_18.

[67] R. Yan, C.-H. Cheng, and Y. Chai, „Formal Consistency Checking over Specifications in Natural Languages“, in *DATE*, Grenoble, France, 2015, pp. 1677–1682.

[68] A. Elgammal and T. Butler, „ICSOC 2014 Workshops“, in, F. Toumani, B. Pernici, D. Grigori, D. Benslimane, J. Mendling, N. Ben Hadj-Alouane, B. Blake, O. Perrin, I. Saleh Moustafa, and S. Bhiri, Eds. Springer, 2015, ch. Towards a Framework for Semantically-Enabled Compliance Management in Financial Services, pp. 171–184.

[69] N. A. Manaf, S. Moschoyiannis, and P. J. Krause, „Service Choreography, SBVR, and Time“, in *FOCLASA, Madrid, Spain*, 2015, pp. 63–77.

[70] J. M. E. M. Werf, H. M. W. Verbeek, and W. M. P. Aalst, „BPM, Tallinn, Estonia“, in, A. Barros, A. Gal, and E. Kindler, Eds. Springer, 2012, ch. Context-Aware Compliance Checking, pp. 98–113.

[71] W3C, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, https://www.w3.org/Submission/SWRL/, Last accessed: February 22, 2019.

[72] F. Yip, N. Parameswaran, and P. Ray, „Rules and Ontology in Compliance Management“, in *EDOC*, Oct. 2007, pp. 435–435.

[73] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, „Property Specification Patterns for Finite-state Verification“, in *Proceedings of the Second Workshop on Formal Methods in Software Practice*, ser. FMSP '98, Clearwater Beach, Florida, USA: ACM, 1998, pp. 7–15, ISBN: 0-89791-954-8. DOI: 10.1145/298595.298598. [Online]. Available: http://doi.acm.org/10.1145/298595.298598.

[74] J. Yu, T. P. Manh, J. Han, Y. Jin, Y. Han, and J. Wang, „Pattern Based Property Specification and Verification for Service Composition“, in *7th International Conference on Web Information Systems*, Wuhan, China: Springer, 2006, pp. 156–168.

[75] T. Tran, E. Weiss, C. Ruhsam, C. Czepa, H. Tran, and U. Zdun, „Embracing Process Compliance and Flexibility through Behavioral Consistency Checking in ACM: A Repair Service Management Case“, in *AdaptiveCM'15*, ser. Business Process Management Workshops 2015, Aug. 2015. [Online]. Available: http://eprints.cs.univie.ac.at/4409/.

[76] M. Leitner, J. Mangler, and S. Rinderle-Ma, „Definition and Enactment of Instance-Spanning Process Constraints“, in *International Conference of Web Information System Engineering*, ser. LNCS, Cyprus: Springer, 2012, pp. 652–658.

[77] A. Awad, A. Barnawi, A. Elgammal, R. Elshawi, A. Almalaise, and S. Sakr, „Runtime Detection of Business Process Compliance Violations: An Approach Based on Anti Patterns“, in *30th Symposium on Applied Computing*, ser. SAC'15, Salamanca, Spain: ACM, 2015, pp. 1203–1210, ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2699488. [Online]. Available: http://doi.acm.org/10.1145/2695664.2699488.

[78] C. D. Ciccio and M. Mecella, „On the Discovery of Declarative Control Flows for Artful Processes“, *ACM Trans. Manage. Inf. Syst.*, vol. 5, no. 4, 24:1–24:37, Jan. 2015, ISSN: 2158-656X. DOI: 10.1145/2629447.

[79] T. Tran, E. Weiss, C. Ruhsam, C. Czepa, H. Tran, and U. Zdun, „Enabling Flexibility of Business Processes by Compliance Rules: A Case Study from the Insurance Industry“, in *13th International Conference on Business Process Management 2015, Industry Track*, Aug. 2015. [Online]. Available: http://eprints.cs.univie.ac.at/4399/.

[80] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, „NuSMV: a new Symbolic Model Verifier“, in *11th Conf.on Computer-Aided Verification (CAV)*, Trento, Italy: Springer, Jul. 1999, pp. 495–499.

[81] I. Barba, B. Weber, C. D. Valle, and A. Jiménez-Ramírez, „User recommendations for the optimized execution of business processes“, *Data and Knowledge Engineering*, vol. 86, no. 0, pp. 61–84, 2013, ISSN: 0169-023X. DOI: http://dx.doi.org/10.1016/j.datak.2013.01.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169023X13000050.

[82] M. T. Gómez-López, L. Parody, R. M. Gasca, and S. Rinderle-Ma, „OTM 2014 Conferences, Amantea, Italy, October 27-31“, in, R. Meersman, H. Panetto, T. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T. Sellis, Eds. Springer, 2014, ch. Prognosing the Compliance of Declarative Business Processes Using Event Trace Robustness, pp. 327–344.

[83] T. Tran, C. Ruhsam, M. J. Pucher, M. Kobler, and J. Mendling, „Towards a pattern recognition approach for transferring knowledge in ACM“, in *3rd International Workshop on Adaptive Case Management and other non-workflow approaches to BPM*, Ulm, Germany, 2014.

[84] F. Fleuret, „Fast Binary Feature Selection with Conditional Mutual Information“, *J. Mach. Learn. Res.*, vol. 5, pp. 1531–1555, Dec. 2004, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1044711.

[85] H. Blockeel, L. D. Raedt, and J. Ramon, „Top-Down Induction of Clustering Trees“, in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 55–63, ISBN: 1-55860-556-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=645527.657456.

[86] M. M. Akhlagh, S. C. Tan, and F. Khak, „Temporal data classification and rule extraction using a probabilistic decision tree“, in *ICCIS'12*, vol. 1, Jun. 2012, pp. 346–351. DOI: 10.1109/ICCISci.2012.6297267.

[87] Q. Shi, Y. Zhao, and M. Liu, „Towards learning segmented temporal sequences: A decision tree approach“, in *ICMLC'15*, vol. 1, Jul. 2015, pp. 145–150. DOI: 10.1109/ICMLC.2015.7340913.

[88] S. Schönig, C. Cabanillas, S. Jablonski, and J. Mendling, „Mining the Organisational Perspective in Agile Business Processes“, in *BPMDS'15, Stockholm, Sweden, June 8-9*, 2015, pp. 37–52.

[89] G. Lakshmanan, D. Shamsi, Y. Doganata, M. Unuvar, and R. Khalaf, „A markov prediction model for data-driven semi-structured business processes“, English, *Knowledge and Information Systems*, pp. 1–30, 2013, ISSN: 0219-1377. DOI: 10.1007/s10115-013-0697-8. [Online]. Available: http://dx.doi.org/10.1007/s10115-013-0697-8.

[90] K. Y. Rozier, „Survey: Linear Temporal Logic Symbolic Model Checking“, *Comput. Sci. Rev.*, vol. 5, no. 2, pp. 163–203, May 2011, ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2010.06.002. [Online]. Available: http://dx.doi.org/10.1016/j.cosrev.2010.06.002.

[91] S. Salamah, A. Gates, S. Roach, and O. Mondragon, „Verifying Pattern-Generated LTL Formulas: A Case Study“, English, in *Model Checking Software*, ser. LNCS, vol. 3639, Springer, 2005, pp. 200–220, ISBN: 978-3-540-28195-5. DOI: 10.1007/11537328_17.

[92] J. Barnat, P. Bauch, and L. Brim, „Checking Sanity of Software Requirements“, English, in *Software Engineering and Formal Methods*, ser. LNCS, vol. 7504, Springer, 2012, pp. 48–62, ISBN: 978-3-642-33825-0.

[93] J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik, „Exploiting resolution proofs to speed up LTL vacuity detection for BMC“, English, *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 5, pp. 319–335, 2010, ISSN: 1433-2779. DOI: 10.1007/s10009-009-0134-1.

[94] A. Awad, M. Weidlich, and M. Weske, „Consistency Checking of Compliance Rules“, English, in *Business Information Systems*, ser. Lecture Notes in Business Information Processing, W. Abramowicz and R. Tolksdorf, Eds., vol. 47, Springer Berlin Heidelberg, 2010, pp. 106–118, ISBN: 978-3-642-12813-4. DOI: 10.1007/978-3-642-12814-1_10. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12814-1_10.

[95] T. Holmes, E. Mulo, U. Zdun, and S. Dustdar, „Model-aware Monitoring of SOAs for Compliance“, in *Service Engineering: European Research Results*. Vienna: Springer Vienna, 2011, pp. 117–136, ISBN: 978-3-7091-0415-6. DOI: 10.1007/978-3-7091-0415-6_5. [Online]. Available: https://doi.org/10.1007/978-3-7091-0415-6_5.

[96] M. Pešić, D. Bošnački, and W. M. P. van der Aalst, „Enacting Declarative Languages Using LTL: Avoiding Errors and Improving Performance“, in *Model Checking Software*, J. van de Pol and M. Weber, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 146–161, ISBN: 978-3-642-16164-3.

[97] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali, „Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces“, in *Business Process Management*, S. Sadiq, P. Soffer, and H. Völzer, Eds., Cham: Springer International Publishing, 2014, pp. 1–17, ISBN: 978-3-319-10172-9.

[98] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst, „Runtime Verification of LTL-Based Declarative Process Models“, in *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers*, S. Khurshid and K. Sen, Eds. Springer, 2012, pp. 131–146.

[99] G. De Giacomo and M. Y. Vardi, „Linear Temporal Logic and Linear Dynamic Logic on Finite Traces“, in *IJCAI*, ser. IJCAI '13, Beijing, China: AAAI Press, 2013, pp. 854–860, ISBN: 978-1-57735-633-2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2540128.2540252.

[100] A. Bauer, M. Leucker, and C. Schallhart, „Comparing LTL Semantics for Runtime Verification“, *J. Log. and Comput.*, vol. 20, no. 3, pp. 651–674, Jun. 2010, ISSN: 0955-792X. DOI: 10.1093/logcom/exn075. [Online]. Available: http://dx.doi.org/10.1093/logcom/exn075.

[101] Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem, „Runtime Verification of Component-based Systems in the BIP Framework with Formally-proved Sound and Complete Instrumentation“, *Softw. Syst. Model.*, vol. 14, no. 1, pp. 173–199, Feb. 2015,

ISSN: 1619-1366. DOI: 10.1007/s10270-013-0323-y. [Online]. Available: http://dx.doi.org/10.1007/s10270-013-0323-y.

[102] Y. Joshi, G. M. Tchamgoue, and S. Fischmeister, „Runtime Verification of LTL on Lossy Traces“, in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17, Marrakech, Morocco: ACM, 2017, pp. 1379–1386, ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019827. [Online]. Available: http://doi.acm.org/10.1145/3019612.3019827.

[103] J. Morse, L. Cordeiro, D. Nicole, and B. Fischer, „Model Checking LTL Properties over ANSI-C Programs with Bounded Traces“, *Softw. Syst. Model.*, vol. 14, no. 1, pp. 65–81, Feb. 2015, ISSN: 1619-1366. DOI: 10.1007/s10270-013-0366-0. [Online]. Available: http://dx.doi.org/10.1007/s10270-013-0366-0.

[104] EsperTech Inc., *EPL Reference*, http://www.espertech.com/esper/release-5.1.0/esper-reference/html/event_patterns.html, Last accessed: February 22, 2019.

[105] Wikiphoto, http://www.wikihow.com/Become-a-Software-Engineer, Licensed under: http://creativecommons.org/licenses/by-nc-sa/3.0/ Last accessed: February 22, 2019.

[106] EsperTech Inc., *Esper*, http://www.espertech.com/esper/, Last accessed: February 22, 2019.

[107] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, „Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar“, *Software Engineering*, vol. 41, no. 7, pp. 620–638, Jul. 2015, ISSN: 0098-5589. DOI: 10.1109/TSE.2015.2398877.

[108] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein, „Formal Correctness, Safety, Dependability, and Performance Analysis of a Satellite“, in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, Zurich, Switzerland: IEEE Press, 2012, pp. 1022–1031, ISBN: 978-1-4673-1067-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337354.

[109] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti, „Specification patterns from research to industry: A case study in service-based applications“, in *2012 34th International Conference on Software Engineering (ICSE)*, Jun. 2012, pp. 968–976. DOI: 10.1109/ICSE.2012.6227125.

[110] A. Post, I. Menzel, J. Hoenicke, and A. Podelski, „Automotive Behavioral Requirements Expressed in a Specification Pattern System: A Case Study at BOSCH“, *Requir. Eng.*, vol. 17, no. 1, pp. 19–33, Mar. 2012, ISSN: 0947-3602. DOI: 10.1007/s00766-011-0145-9. [Online]. Available: http://dx.doi.org/10.1007/s00766-011-0145-9.

[111] O. M. Kherbouche, A. Ahmad, and H. Basson, „Formal approach for compliance rules checking in business process models“, in *2013 IEEE 9th International Conference on Emerging Technologies (ICET)*, Dec. 2013, pp. 1–6. DOI: 10.1109/ICET.2013.6743500.

[112] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, „Reduction Techniques for Efficient Behavioral Model Checking in Adaptive Case Management“, in *The 32nd ACM Symposium on Applied Computing (SAC 2017)*, Apr. 2017. [Online]. Available: http://eprints.cs.univie.ac.at/4879/.

[113] S. Morimoto, „A Survey of Formal Verification for Business Process Modeling“, in *Computational Science – ICCS 2008: 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part II*, M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 514–522, ISBN: 978-3-540-69387-1. DOI: 10.1007/978-3-540-69387-15_8. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69387-15_8.

[114] A. Bucchiarone, H. Muccini, P. Pelliccione, and P. Pierini, „Model-Checking Plus Testing: From Software Architecture Analysis to Code Testing“, in *FORTE 2004*, M. Núñez, Z. Maamar, F. L. Pelayo, K. Pousttchi, and F. Rubio, Eds. Berlin, Heidelberg: Springer, 2004, pp. 351–365, ISBN: 978-3-540-30233-9. DOI: 10.1007/978-3-540-30233-9_26. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30233-9_26.

[115] E. Mulo, U. Zdun, and S. Dustdar, „Domain-specific language for event-based compliance monitoring in process-driven SOAs“, *Service Oriented Computing and Applications*, vol. 7, no. 1, pp. 59–73, Apr. 2013, ISSN: 1863-2394. DOI: 10.1007/s11761-012-0121-3. [Online]. Available: http://dx.doi.org/10.1007/s11761-012-0121-3.

[116] D. Knuplesch, M. Reichert, L. T. Ly, A. Kumar, and S. Rinderle-Ma, „On the Formal Semantics of the Extended Compliance Rule Graph“, Ulm University, Ulm, Technical Report UIB-2013 - 05, Apr. 2013. [Online]. Available: http://dbis.eprints.uni-ulm.de/1147/.

[117] L. de Silva and D. Balasubramaniam, „PANDArch: A Pluggable Automated Non-intrusive Dynamic Architecture Conformance Checker“, in *ECSA'13*, Montpellier, France: Springer, 2013, pp. 240–248, ISBN: 978-3-642-39030-2.

[118] S. Blom, J. van de Pol, and M. Weber, „LTSmin: Distributed and Symbolic Reachability“, in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 354–359, ISBN: 978-3-642-14295-6. DOI: 10.1007/978-3-642-14295-6_31. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_31.

[119] G. J. Holzmann, „The model checker SPIN“, *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–295, 1997.

[120] G. De Giacomo, R. De Masellis, and M. Montali, „Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness“, in *AAAI*, ser. AAAI'14, Canada: AAAI Press, 2014, pp. 1027–1033. [Online]. Available: http://dl.acm.org/citation.cfm?id=2893873.2894033.

[121] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, „DECLARE: Full Support for Loosely-Structured Processes“, in *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, ser. EDOC '07, Washington, DC, USA: IEEE Computer Society, 2007, pp. 287–, ISBN: 0-7695-2891-0. [Online]. Available: http://dl.acm.org/citation.cfm?id=1317532.1318056.

[122] E. Wu, Y. Diao, and S. Rizvi, „High-performance Complex Event Processing over Streams“, in *SIGMOD '06*, ser. SIGMOD '06, Chicago, IL, USA: ACM, 2006, pp. 407–418, ISBN: 1-59593-434-0. DOI: 10.1145/1142473.1142520. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142520.

[123] J. Boubeta-Puig, G. Díaz, H. Macià, V. Valero, and G. Ortiz, „MEdit4CEP-CPN: An approach for Complex Event Processing modeling by Prioritized Colored Petri Nets“, *Information Systems*, 2017, ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2017.11.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437917300108.

[124] S. Kunz, T. Fickinger, J. Prescher, and K. Spengler, „Managing Complex Event Processes with Business Process Modeling Notation“, in *Business Process Modeling Notation*, J. Mendling, M. Weidlich, and M. Weske, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 78–90, ISBN: 978-3-642-16298-5.

[125] M. Adam, C. Cordeiro, L. Field, D. Giordano, and L. Magnoni, „Real-time complex event processing for cloud resources“, *Journal of Physics: Conference Series*, vol. 898, no. 4, p. 042 020, 2017.

[126] L. Aniello, G. A. Di Luna, G. Lodi, and R. Baldoni, „A Collaborative Event Processing System for Protection of Critical Infrastructures from Cyber Attacks“, in *Computer Safety, Reliability, and Security*, F. Flammini, S. Bologna, and V. Vittorini, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 310–323, ISBN: 978-3-642-24270-0.

[127] G. Cugola and A. Margara, „TESLA: A Formally Defined Event Specification Language“, in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10, Cambridge, United Kingdom: ACM, 2010, pp. 50–61, ISBN: 978-1-60558-927-5. DOI: 10.1145/1827418.1827427. [Online]. Available: http://doi.acm.org/10.1145/1827418.1827427.

[128] T. Tran, E. Weiss, A. Adensamer, C. Ruhsam, C. Czepa, H. Tran, and U. Zdun, „An Ontology-Based Approach for Defining Compliance Rules by Knowledge Workers in Adaptive Case Management“, in *5th International Workshop on Adaptive Case Management and other Non-workflow Approaches to BPM (AdaptiveCM 16), 20th IEEE International Enterprise Computing Workshops (EDOCW 2016)*, Sep. 2016. [Online]. Available: http://eprints.cs.univie.ac.at/4753/.

[129] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor, „A Language and Environment for Architecture-based Software Development and Evolution“, in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE '99, Los Angeles, California, USA: ACM, 1999, pp. 44–53, ISBN: 1-58113-074-0. DOI: 10.1145/302405.302410. [Online]. Available: http://doi.acm.org/10.1145/302405.302410.

[130] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann, „Sustainable Architectural Design Decisions“, *IEEE Software*, vol. 30, no. 6, pp. 46–53, Nov. 2013, ISSN: 0740-7459. DOI: 10.1109/MS.2013.97.

[131] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, „On the Understandability of Semantic Constraints for Behavioral Software Architecture Compliance: A Controlled Experiment“, in *IEEE International Conference on Software Architecture (ICSA 2017)*, Apr. 2017. [Online]. Available: http://eprints.cs.univie.ac.at/5059/.

[132] V. R. Basili, G. Caldiera, and H. D. Rombach, „The Goal Question Metric Approach“, in *Encyclopedia of Software Engineering*, Wiley, 1994.

[133] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, „Preliminary Guidelines for Empirical Research in Software Engineering“, *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002, ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1027796. [Online]. Available: http://dx.doi.org/10.1109/TSE.2002.1027796.

[134] M. Höst, B. Regnell, and C. Wohlin, „Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment“, *Empirical Software Engineering*, vol. 5, no. 3, pp. 201–214, Nov. 2000, ISSN: 1573-7616. DOI: 10.1023/A:1026586415054. [Online]. Available: https://doi.org/10.1023/A:1026586415054.

[135] P. Runeson, „Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data“, in *Proceedings 7th International Conference on Empirical Assessment and Evaluation in Software Engineering*, 2003, pp. 95–102.

[136] M. Svahnberg, A. Aurum, and C. Wohlin, „Using Students As Subjects - an Empirical Evaluation“, in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '08, Kaiserslautern, Germany: ACM, 2008, pp. 288–290, ISBN: 978-1-59593-971-5. DOI: 10.1145/1414004.1414055. [Online]. Available: http://doi.acm.org/10.1145/1414004.1414055.

[137] I. Salman, A. T. Misirli, and N. Juristo, „Are Students Representatives of Professionals in Software Engineering Experiments?“, in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15, Florence, Italy: IEEE Press, 2015, pp. 666–676, ISBN: 978-1-4799-1934-5. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818836.

[138] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, „Reporting Experiments in Software Engineering“, in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 201–228.

[139] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*, 1st. Springer, 2010.

[140] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, „Robust Statistical Methods for Empirical Software Engineering“, *Empirical Software Engineering*, pp. 1–52, 2016.

[141] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008, ISBN: 026202649X, 9780262026499.

[142] C. Czepa and U. Zdun, *On the Understandability of Temporal Properties Formalized in Linear Temporal Logic, Property Specification Patterns and Event Processing Language [Data set]*, http://doi.org/10.5281/zenodo.891007, 2017.

[143] J. Feigenspan, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake, „Do background colors improve program comprehension in the #ifdef hell?“, *Empirical Software Engineering*, vol. 18, no. 4, pp. 699–745, 2013.

[144] B. Hoisl, S. Sobernig, and M. Strembeck, „Comparing Three Notations for Defining Scenario-Based Model Tests: A Controlled Experiment“, in *QUATIC'14*, Sep. 2014, pp. 95–104.

[145] H. Habiballa and T. Kmet, „Theoretical branches in teaching computer science“, *International Journal of Mathematical Education in Science and Technology*, vol. 35, no. 6, pp. 829–841, 2004. DOI: 10.1080/00207390412331271267. eprint: https://doi.org/10.1080/00207390412331271267. [Online]. Available: https://doi.org/10.1080/00207390412331271267.

[146] M. Knobelsdorf and C. Frede, „Analyzing Student Practices in Theory of Computation in Light of Distributed Cognition Theory“, in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER '16, Melbourne, VIC, Australia: ACM, 2016, pp. 73–81, ISBN: 978-1-4503-4449-4. DOI: 10.1145/2960310.2960331. [Online]. Available: http://doi.acm.org/10.1145/2960310.2960331.

[147] D. Carew, C. Exton, and J. Buckley, „An empirical investigation of the comprehensibility of requirements specifications“, in *2005 International Symposium on Empirical Software Engineering, 2005.*, Nov. 2005, 10 pp.-. DOI: 10.1109/ISESE.2005.1541834.

[148] M. Spichkova, „“Boring Formal Methods” or “Sherlock Holmes Deduction Methods”?“, in *Software Technologies: Applications and Foundations*, P. Milazzo, D. Varró, and M. Wimmer, Eds., Cham: Springer International Publishing, 2016, pp. 242–252, ISBN: 978-3-319-50230-4.

[149] F. Richardson and R. M. Suinn, „The Mathematics Anxiety Rating Scale“, vol. 19, pp. 551–554, Nov. 1972.

[150] N. Cliff, „Dominance statistics: Ordinal Analyses to Answer Ordinal Questions“, *Psychological Bulletin*, vol. 114, pp. 494–509, 1993.

[151] J. J. Rogmann, *Ordinal Dominance Statistics (orddom): An R Project for Statistical Computing package to compute ordinal, nonparametric alternatives to mean comparison (Version 3.1)*, Available online from the CRAN website http://cran.r-project.org/, 2013.

[152] Y. Benjamini and Y. Hochberg, „Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing“, *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[153] A. R. da Silva, G. Malafaia, and I. P. P. de Menezes, „biotools: an R function to predict spatial gene diversity via an individual-based approach“, *Genetics and Molecular Research*, vol. 16, gmr16029655, 2017.

[154] J. Fox and S. Weisberg, *An R Companion to Applied Regression*, Second. Thousand Oaks CA: Sage, 2011. [Online]. Available: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion.

[155] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009, ISBN: 978-0-387-98140-6. [Online]. Available: http://ggplot2.org.

[156] Slawomir Jarek, *mvnormtest: Normality test for multivariate variables*, https://CRAN.R-project.org/package=mvnormtest, Last accessed: February 22, 2019, 2012.

[157] Peter Filzmoser and Moritz Gschwandtner, *mvoutlier: Multivariate Outlier Detection Based on Robust Methods*, https://CRAN.R-project.org/package=mvoutlier, Last accessed: February 22, 2019, 2017.

[158] W. Revelle, *psych: Procedures for Psychological, Psychometric, and Personality Research*, R package version 1.7.5, Northwestern University, Evanston, Illinois, 2017. [Online]. Available: https://CRAN.R-project.org/package=psych.

[159] B. Naimi, N. a.s. Hamm, T. A. Groen, A. K. Skidmore, and A. G. Toxopeus, „Where is positional uncertainty a problem for species distribution modelling“, *Ecography*, vol. 37, pp. 191–203, 2014. DOI: 10.1111/j.1600-0587.2013.00205.x.

[160] H.-F. Hsieh and S. E. Shannon, „Three Approaches to Qualitative Content Analysis“, *Qualitative Health Research*, vol. 15, no. 9, pp. 1277–1288, 2005, PMID: 16204405. DOI: 10.1177/1049732305276687. eprint: http://dx.doi.org/10.1177/1049732305276687. [Online]. Available: http://dx.doi.org/10.1177/1049732305276687.

[161] S. Khoshafian, *Intelligent BPM: The Next Wave for Customer Centric Business Applications*. Pegasystems Incorporated, 2013, ISBN: 9780986052101. [Online]. Available: https://books.google.at/books?id=IYACnwEACAAJ.

[162] T. Tran, E. Weiss, C. Ruhsam, C. Czepa, H. Tran, and U. Zdun, „Enabling Flexibility of Business Processes Using Compliance Rules: The Case of Mobiliar“, in *Business Process Management. Cases*, J. vom Brocke and J. Mendling, Eds., Springer, 2017. [Online]. Available: http://eprints.cs.univie.ac.at/5094/.

[163] F. Ferri, E. Pourabbas, and M. Rafanelli, „The Syntactic and Semantic Correctness of Pictorial Configurations to Query Geographic Databases by PQL“, in *Proceedings of the 2002 ACM Symposium on Applied Computing*, ser. SAC '02, Madrid, Spain: ACM, 2002, pp. 432–437, ISBN: 1-58113-445-2. DOI: 10.1145/508791.508873. [Online]. Available: http://doi.acm.org/10.1145/508791.508873.

[164] M. Hindawi, L. Morel, R. Aubry, and J.-L. Sourrouille, „Description and Implementation of a UML Style Guide“, in *Models in Software Engineering*, M. R. V. Chaudron, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 291–302, ISBN: 978-3-642-01648-6.

[165] D. Harel and B. Rumpe, „Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff“, Jerusalem, Israel, Israel, Tech. Rep., 2000.

[166] N. Chomsky, Ed., *Syntactic structures*. Mouton & Co., 1957.

[167] C. Czepa, A. Amiri, E. Ntentos, and U. Zdun, *Modeling Compliance Specifications in Linear Temporal Logic, Event Processing Language and Property Specification Patterns [Data set]*, May 2018. DOI: 10.5281/zenodo.1246561. [Online]. Available: https://doi.org/10.5281/zenodo.1246561.

[168] M. A. Revilla, W. E. Saris, and J. A. Krosnick, „Choosing the Number of Categories in Agree–Disagree Scales“, *Sociological Methods & Research*, vol. 43, no. 1, pp. 73–97, 2014. DOI: 10.1177/0049124113509605.

[169] I. Lytra, P. Gaubatz, and U. Zdun, „Two Controlled Experiments on Model-based Architectural Decision Making“, *Information and Software Technology*, vol. 58, pp. 63–75, Jul. 2015, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.03.006. [Online]. Available: http://eprints.cs.univie.ac.at/4342/.

[170] Jason Bryer, Kimberly Speerschneider, *likert: Analysis and Visualization Likert Items*, https://CRAN.R-project.org/package=likert, Last accessed: February 22, 2019, 2016.

[171] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, „Declarative workflows: Balancing between flexibility and support“, *Computer Science - Research and Development*, vol. 23, no. 2, pp. 99–113, 2009, ISSN: 0949-2925. DOI: 10.1007/s00450-009-0057-9. [Online]. Available: http://dx.doi.org/10.1007/s00450-009-0057-9.

[172] S. Zugal, P. Soffer, C. Haisjackl, J. Pinggera, M. Reichert, and B. Weber, „Investigating expressiveness and understandability of hierarchy in declarative business process models“, *Software & Systems Modeling*, vol. 14, no. 3, pp. 1081–1103, Jul. 2015, ISSN: 1619-1374. DOI: 10.1007/s10270-013-0356-2. [Online]. Available: https://doi.org/10.1007/s10270-013-0356-2.

[173] C. Haisjackl, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber, „Making Sense of Declarative Process Models: Common Strategies and Typical Pitfalls“, in *Enterprise, Business-Process and Information Systems Modeling: 14th International Conference, BPMDS 2013, 18th International Conference, EMMSAD 2013, Held at CAiSE 2013, Valencia, Spain, June 17-18, 2013. Proceedings*, S. Nurcan, H. A. Proper, P. Soffer, J. Krogstie, R. Schmidt, T. Halpin, and I. Bider, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 2–17, ISBN: 978-3-642-38484-4. DOI: 10.1007/978-3-642-38484-4_2. [Online]. Available: https://doi.org/10.1007/978-3-642-38484-4_2.

[174] J. De Smedt, J. De Weerdt, E. Serral, and J. Vanthienen, „Improving Understandability of Declarative Process Models by Revealing Hidden Dependencies“, in *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, S. Nurcan, P. Soffer, M. Bajec, and J. Eder, Eds. Cham: Springer International Publishing, 2016, pp. 83–98, ISBN: 978-3-319-39696-5. DOI: 10.1007/978-3-319-39696-5_6. [Online]. Available: https://doi.org/10.1007/978-3-319-39696-5_6.

[175] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers, „Imperative versus Declarative Process Modeling Languages: An Empirical Investigation“, in *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, F. Daniel, K. Barkaoui, and S. Dustdar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 383–394, ISBN: 978-3-642-28108-2. DOI: 10.1007/978-3-642-28108-2_37. [Online]. Available: https://doi.org/10.1007/978-3-642-28108-2_37.

[176] R. D. Rodrigues, M. D. Barros, K. Revoredo, L. G. Azevedo, and H. Leopold, „An Experiment on Process Model Understandability Using Textual Work Instructions and

BPMN Models“, in *2015 29th Brazilian Symposium on Software Engineering (SBES)*, vol. 00, Sep. 2015, pp. 41–50. DOI: 10.1109/SBES.2015.12. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SBES.2015.12.

[177] G. Jošt, J. Huber, M. Heričko, and G. Polančič, „An empirical investigation of intuitive understandability of process diagrams“, *Computer Standards and Interfaces*, vol. 48, pp. 90–111, 2016, Special Issue on Information System in Distributed Environment, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2016.04.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548916300332.

[178] R. Kowalski and M. Sergot, „A logic-based calculus of events“, *New Generation Computing*, vol. 4, no. 1, pp. 67–95, Mar. 1986, ISSN: 1882-7055. DOI: 10.1007/BF03037383. [Online]. Available: https://doi.org/10.1007/BF03037383.

[179] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, Robby, and H. Zheng, „Bandera: Extracting Finite-state Models from Java Source Code“, in *ICSE '00*, Limerick, Ireland: ACM, 2000, pp. 439–448, ISBN: 1-58113-206-9.

[180] B. H. C. Cheng and J. M. Atlee, „Research Directions in Requirements Engineering“, in *Future of Software Engineering, 2007. FOSE '07*, May 2007, pp. 285–303. DOI: 10.1109/FOSE.2007.17.

[181] J. Hatcliff, X. Deng, M. B. Dwyer, G. Jung, and V. P. Ranganath, „Cadena: an integrated development, analysis, and verification environment for component-based systems“, in *25th International Conference on Software Engineering, 2003. Proceedings.*, May 2003, pp. 160–172. DOI: 10.1109/ICSE.2003.1201197.

[182] T. Krismayer, R. Rabiser, and P. Grünbacher, „Mining Constraints for Event-based Monitoring in Systems of Systems“, in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017, Urbana-Champaign, IL, USA: IEEE Press, 2017, pp. 826–831, ISBN: 978-1-5386-2684-9.

[183] Z. Li, J. Han, and Y. Jin, „Pattern-Based Specification and Validation of Web Services Interaction Properties“, in *Service-Oriented Computing - ICSOC 2005*, B. Benatallah, F. Casati, and P. Traverso, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 73–86, ISBN: 978-3-540-32294-8.

[184] P. Y. H. Wong and J. Gibbons, „Property Specifications for Workflow Modelling“, in *Integrated Formal Methods*, M. Leuschel and H. Wehrheim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 56–71, ISBN: 978-3-642-00255-7.

[185] K. Namiri and N. Stojanovic, „Using Control Patterns in Business Processes Compliance“, in *Web Information Systems Engineering – WISE 2007 Workshops*, M. Weske, M.-S. Hacid, and C. Godart, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 178–190, ISBN: 978-3-540-77010-7.

[186] W. Dou, D. Bianculli, and L. Briand, „OCLR: A More Expressive, Pattern-Based Temporal Extension of OCL“, in *Modelling Foundations and Applications*, J. Cabot and J. Rubin, Eds., Cham: Springer International Publishing, 2014, pp. 51–66, ISBN: 978-3-319-09195-2.

[187] R. L. Smith, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil, „Propel: an approach supporting property elucidation“, in *24th Intl. Conf. on Software Engineering*, ACM Press, 2002, pp. 11–21.

[188] A. Awad, M. Weidlich, and M. Weske, „Visually specifying compliance rules and explaining their violations for business processes“, *Journal of Visual Languages and Computing*, vol. 22, no. 1, pp. 30–55, 2011, Special Issue on Visual Languages and Logic, ISSN: 1045-926X. DOI: https://doi.org/10.1016/j.jvlc.2010.11.002.

[189] S. Goedertier, J. Vanthienen, and F. Caron, „Declarative business process modelling: principles and modelling languages“, *Enterprise Information Systems*, vol. 9, no. 2, pp. 161–185, 2015. DOI: 10.1080/17517575.2013.830340. eprint: https://doi.org/10.1080/17517575.2013.830340. [Online]. Available: https://doi.org/10.1080/17517575.2013.830340.

[190] W. M. P. van der Aalst, M. Westergaard, and H. A. Reijers, „Beautiful Workflows: A Matter of Taste?“, in *The Beauty of Functional Code: Essays Dedicated to Rinus Plasmeijer on the Occasion of His 61st Birthday*, P. Achten and P. Koopman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 211–233, ISBN: 978-3-642-40355-2. DOI: 10.1007/978-3-642-40355-2_15. [Online]. Available: https://doi.org/10.1007/978-3-642-40355-2_15.

[191] B. Weber, H. A. Reijers, S. Zugal, and W. Wild, „The Declarative Approach to Business Process Execution: An Empirical Test“, in *Advanced Information Systems Engineering: 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings*, P. van Eck, J. Gordijn, and R. Wieringa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 470–485, ISBN: 978-3-642-02144-2. DOI: 10.1007/978-3-642-02144-2_37. [Online]. Available: https://doi.org/10.1007/978-3-642-02144-2_37.

[192] L. Mendes Cunha, C. Cappelli, and F. M. Santoro, „Semiotic Engineering to Define a Declarative Citizen Language“, in *Human Interface and the Management of Information: Supporting Learning, Decision-Making and Collaboration: 19th International Conference, HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings, Part II*, S. Yamamoto, Ed. Cham: Springer International Publishing, 2017, pp. 503–515, ISBN: 978-3-319-58524-6. DOI: 10.1007/978-3-319-58524-6_40. [Online]. Available: https://doi.org/10.1007/978-3-319-58524-6_40.

[193] K. Figl and J. Recker, „Exploring cognitive style and task-specific preferences for process representations“, *Requirements Engineering*, vol. 21, no. 1, pp. 63–85, Mar. 2016, ISSN: 1432-010X. DOI: 10.1007/s00766-014-0210-2. [Online]. Available: https://doi.org/10.1007/s00766-014-0210-2.

[194] G. Miller, *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*, 1956.

[195] A. Bauer, M. Leucker, and C. Schallhart, „Runtime Verification for LTL and TLTL“, *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, 14:1–14:64, Sep. 2011, ISSN: 1049-331X. DOI: 10.1145/2000799.2000800. [Online]. Available: http://doi.acm.org/10.1145/2000799.2000800.

[196] C. Czepa and U. Zdun, *On the Understandability of Graphical and Textual Pattern-Based Behavioral Constraint Representations [Data set]*, Mar. 2018. DOI: 10.5281/zenodo.1209839. [Online]. Available: https://doi.org/10.5281/zenodo.1209839.

[197] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, „Monitoring Business Process Compliance Using Compliance Rule Graphs“, English, in *On the Move to Meaningful Internet Systems: OTM 2011*, ser. LNCS, vol. 7044, Springer, 2011, pp. 82–99, ISBN: 978-3-642-25108-5. DOI: 10.1007/978-3-642-25109-2_7.

[198] R. Heiberger and N. Robbins, „Design of Diverging Stacked Bar Charts for Likert Scales and Other Applications“, *Journal of Statistical Software, Articles*, vol. 57, no. 5, pp. 1–32, 2014, ISSN: 1548-7660. DOI: 10.18637/jss.v057.i05. [Online]. Available: https://www.jstatsoft.org/v057/i05.

[199] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, „Ontology-Based Behavioral Constraint Authoring“, in *2nd International Workshop on Compliance, Evolution and Security in intra- and Cross-Organizational Processes (CeSCoP 2016), 20th IEEE International Enterprise Computing Workshops (EDOCW 2016)*, Sep. 2016. [Online]. Available: http://eprints.cs.univie.ac.at/4754/.

[200] H. A. Reijers and J. Mendling, „A Study Into the Factors That Influence the Understandability of Business Process Models“, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 3, pp. 449–462, May 2011, ISSN: 1083-4427. DOI: 10.1109/TSMCA.2010.2087017.

[201] K. A. Ericsson and H. A. Simon, *Protocol analysis; Verbal reports as data*. Cambridge, MA: Bradford books/MIT Press, 1984.

[202] ISIS Papyrus, *Building New Business Applications - Experience A Mind-Shift!*, https://www.isis-papyrus.com/Download/TI/TI_Papyrus_Converse_E.pdf, Last accessed: 2018-08-24.

[203] The ISIS Times Online, *ISIS Papyrus finishes successful joint research project with the University of Vienna*, https://isistimes.wordpress.com/2017/10/20/isis-papyrus-finishes-successful-joint-research-project-with-the-university-of-vienna/, Last accessed: 2018-08-24.

[204] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, „Lightweight Approach for Seamless Modeling of Process Flows in Case Management Models“, in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17, Marrakech, Morocco: ACM, 2017, pp. 711–718, ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019616. [Online]. Available: http://doi.acm.org/10.1145/3019612.3019616.

[205] B. Majdic, C. Cowan, J. Girdner, W. Opoku, O. Pierrakos, and E. Barrella, „Monitoring brain waves in an effort to investigate student's cognitive load during a variety of problem solving scenarios“, in *2017 Systems and Information Engineering Design Symposium (SIEDS)*, Apr. 2017, pp. 186–191. DOI: 10.1109/SIEDS.2017.7937713.

[206] B. L. Welch, „The generalization of 'student's' problem when several different population variances are involved“, *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947. DOI: 10.1093/biomet/34.1-2.28. [Online]. Available: http://dx.doi.org/10.1093/biomet/34.1-2.28.