JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Felix Burgstaller, MSc

Submitted at
Department for Business Informatics - Data & Knowledge Engineering

Supervisor and
First Examiner
o. Univ.-Prof. DI Dr. Michael Schrefl

Second Examiner
Univ.-Prof. Mag. Dr. Manuel Wimmer

Co-Supervisor
Mag. Dr. Bernd Neumayr

June 2019

Contextualized Business Rule Repositories: Business Rule Organization Through Contexts

Doctoral Thesis
to obtain the academic degree of
Dr. rer.soc.oec.
in the Doctoral Program
Social and Economic Sciences

Declaration of Authorship

I, Felix Burgstaller, MSc, hereby declare that the thesis “Contextualized Business Rule Repositories: Business Rule Organization Through Contexts” submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

This printed thesis is identical with the electronic version submitted.

Felix Burgstaller, MSc　　　　　　　　　　Date

Preface

This thesis, entitled "Contextualized Business Rule Repositories: Business Rule Organization Through Contexts", is a result of my research endeavors with Michael Schrefl and Bernd Neumayr at the Department of Business Informatics – Data & Knowledge Engineering of the Johannes Kepler University Linz from August 2014 to June 2019. Results of this research were published at various international conferences and workshops: [34–37, 40] regard Contextualized Business Rule Repositories (CBRs), [39, 41, 189, 190] regard SemanticNOTAMs: Ontology-based Representation and Semantic Querying of Digital Notices to Airmen (SemNOTAM) utilized as use case for CBRs, and [38, 111] regard publications not relevant to my dissertation where [111] was mainly written by Ilko Kovacic. CBR-related publications have been developed together with my supervisor Micheal Schrefl; colleagues Bernd Neumayr, Christoph Schütz, and Dieter Steiner from the Johannes Kepler University Linz; and Emanuel Sallinger from the University of Oxford. SemNOTAM-related publications have been developed mostly together with my supervisor Michael Schrefl; colleagues Dieter Steiner, Ilko Kovacic, and Bernd Neumayr; and Eduard Gringinger from Frequentis, where publications [189, 190] were mainly authored by Dieter Steiner. The research results from the SemNOTAM project form the content of Dieter Steiner's ongoing dissertation while we employ SemNOTAM as use case for CBRs. This thesis features significantly extended and revised versions of the results published regarding CBRs and utilizes our research results regarding SemNOTAM as use case.

The foundations of this thesis were laid while contributing to SemNOTAM which was funded by the Austrian Ministry of Transport, Innovation, and Technology in program TAKE OFF under grant FFG-839006. The development of a Vadalog-based proof-of-concept prototype for CBRs, an inheritance mechanism for rule modules, and insights into industry applications of Vadalog were partially supported by Erasmus+ and the University of Oxford enabling me to visit the Department of Computer Science at the University of Oxford, in particular, the Value Added Data Systems (VADA) team led by Georg Gottlob.

In addition to my co-authors, acknowledgements for my co-supervisor Manuel Wimmer, my colleagues Arjol Qeleshi and Median Hilal for their participation in discussions, and Margit Brandl for her administrative support are in order. Furthermore, I would like to thank Emanuel Sallinger, Yavor Nenov, Ruslan Fayzrakhmanov, Stephane Reissfelder, and Evgeny Sherkhonov from the VADA team at the University of Oxford for their valuable input and discussions.

Last but definitely not least, I would like to express my deepest gratitude to my family and friends for their everlasting support. Special thanks go to "(Therapie-) klettern" often responsible for counterbalancing time spent in the office or teaching and especially to my wife Birgit.

Felix Burgstaller
Linz, June 2019

Abstract

Today's businesses face an increasing number, complexity, and variability of business rules. Business rules, including the business vocabulary they rely on, mostly apply in specific situations or contexts only, that is, they have a scope limiting their validity. Hence, a business rule organization enabling effective and efficient maintenance, search, and execution of business rules along their contexts is vital to businesses.

Many research fields employ contexts to manage, organize, and reason about knowledge. A context comprises context knowledge, defining a scope, and contextualized knowledge, applying within the scope. Organizing knowledge by contexts facilitates faster knowledge entering and reasoning, reduced knowledge redundancy by organizing contexts in hierarchies, and easier browsing. However, current business rule management systems do not utilize contexts to organize business rules.

We propose *Contextualized Business Rule Repositories (CBRs)*, organizing business rules along their context of application. We introduce the generic CBR model which can be instantiated multi-level: for a specific domain, e.g., the aeronautical domain, and for a concrete application, e.g., ranking of aeronautical messages. Instantiated CBR models, realized in CBRs, explicitly define the application contexts of business rules. This enables, in addition to the benefits of contexts stated above, increased business rule readability and faster business rule execution.

To enable effective and efficient business rule maintenance in CBRs, we introduce modification operations for CBR models as well as CBR roles enabling effective separation of tasks and responsibilities. A subset of our modification operations is designed to ensure that CBRs remain consistent. We describe how our proposed modification operations and CBR roles support business rule management in CBRs.

We investigate a generic approach to rule set inheritance to foster reuse and to simplify adaptation of rule sets. Building on rule modules, i.e., rule sets with input and output schema, we precisely define inheritance of rule modules by incremental modification in multi-inheritance hierarchies. To prevent uncontrolled proliferation of modifications, we introduce modification restrictions which flexibly regulate the extent of modification. We integrate our approach to rule module inheritance into the generic CBR model resulting in the generic *Extended CBR with Rule Modules (ECBR)* model. To this end, we identify challenges arising from such an integration and discuss how we address these challenges in the generic ECBR model. Furthermore, we touch upon considerations for realizing instantiated ECBR models in ECBRs.

We demonstrate CBRs and their advantages by applying them to the real-world use case SemNOTAM addressing classification of aeronautical messages. Furthermore, we provide several proof-of-concept prototypes as well as a performance comparison with non-contextualized repositories. Regarding modification operations we show how CBRs ease maintenance of business rules compared to non-contextualized repositories. Concerning rule set inheritance we supply a proof-of-concept prototype employing Vadalog, an implementation of Warded Datalog$^{\pm}$. To demonstrate ECBRs and their benefits, we instantiate the generic ECBR model for two real-world use cases, namely, SemNOTAM and financial services in a large US tech-company.

Zusammenfassung

Heutige Unternehmen sind mit einer stetig steigenden Anzahl, Komplexität und Variabilität von Geschäftsregeln konfrontiert. Diese Geschäftsregeln, inklusive deren zugrunde liegendes Geschäftsvokabular, gelten meist nur in bestimmten Situationen oder Kontexten – sie haben einen eingeschränkten Anwendungsbereich. Folglich ist eine Organisationform für Geschäftsregeln, die eine effektive und effiziente Wartung, Suche und Ausführung von Geschäftsregeln hinsichtlich ihrer Anwendungsbereiche ermöglicht, essentiell für Unternehmen.

Viele Forschungsgebiete nutzen Kontexte um Wissen zu managen, zu organisieren und zu schlussfolgern. Ein Kontext besteht aus Kontextwissen, das einen Anwendungsbereich definiert, und kontextualisiertem Wissen, das in dem Anwendungsbereich gilt. Die Organisation von Wissen anhand von Kontexten ermöglicht schnellere Wissenserfassung, schnelleres automatisiertes Schlussfolgern, die Reduktion von redundantem Wissen durch Kontexthierarchien, und einfacheres Browsen. Trotz dieser Vorteile nutzen derzeitige Geschäftsregelmanagementsysteme Kontexte nicht zur Organisation von Geschäftsregeln.

Wir präsentieren *Contextualized Business Rule Repositories (CBRs)* die Geschäftsregeln anhand deren Anwendungskontexten organisieren. Wir stellen das generische CBR-Modell vor, welches sich mehrstufig instanziieren lässt: für eine spezifische Domäne, z.B., die aeronautische Domäne, und für eine bestimmte Anwendung, z.B., Reihung von Flugnachrichten. Instanziierte CBR-Modelle, realisiert in CBRs, definieren die Anwendungskontexte von Geschäftsregeln explizit. Dadurch werden, zusätzlich zu allgemeinen Vorteilen von Kontexten, die Lesbarkeit von Geschäftsregeln sowie die Performanz der Geschäftsregelausführung verbessert.

Um effektive und effiziente Wartung von Geschäftsregeln in CBRs zu ermöglichen, führen wir Operationen für CBR-Modelle sowie CBR-Benutzerrollen ein die eine effektive Trennung von Aufgaben und Verantwortlichkeiten erlauben. Ein Teil unserer Operationen ist in solcher Art und Weise gestaltet, dass CBRs von einem konsistenten Zustand in den nächsten überführt werden. Zusätzlich beschreiben wir wie unsere Operationen und CBR-Benutzerrollen Geschäftsregelmanagement unterstützen.

Wir untersuchen einen generischen Ansatz zur Vererbung von Regelsets um die Wiederverwendung und Anpassung von Regelsets zu fördern und zu vereinfachen. Aufbauend auf Regelmodulen, das sind Regelsets mit Eingabe- und Ausgabeschema, entwickeln wir eine präzise Definition für die Vererbung von Regelmodulen durch inkrementelle Modifikation in Multi-Vererbungshierarchien. Um unkontrollierte Modifikationen zu verhindern, führen wir Modifikationseinschränkungen ein, welche eine flexible Einschränkung des Modifikationsausmaßes ermöglichen. Wir integrieren unseren Vererbungsansatz in das generische CBR-Modell wodurch das generische *Extended CBR with Rule Modules (ECBR)* Modell entsteht. Hierzu identifizieren wir Herausforderungen, die sich durch diese Integration ergeben, und diskutieren wie wir diese im generischen ECBR-Modell lösen. Weiter beschreiben wir Überlegungen zur Realisierung von instanzierten ECBR-Modellen in ECBRs.

Wir demonstrieren CBRs und ihre Vorteile durch Anwendung im realen Anwendungsfall SemNOTAM, einem Projekt das sich mit der Klassifizierung von Flugnachrichten befasst. Weiters beschreiben wir mehrere CBR-Machbarkeits-Prototypen sowie einen Performanzvergleich mit nicht-kontextualisierten Repositories. Hinsichtlich Operationen zeigen wir die vereinfachte Wartung von CBRs im Vergleich zu nicht-kontextualisierten Repositories. Bezüglich unseres Vererbungsansatzes stellen wir einen Machbarkeits-Prototyp vor der Vadalog, eine Implementierung von Warded Datalog$^{\pm}$, als Regelsprache verwendet. Um ECBRs und ihre Vorteile zu zeigen, instanziieren wir das generische ECBR Modell für die realen Anwendungsfälle SemNOTAM sowie Finanzdienstleistungen in einem großen US Technologieunternehmen.

Contents

Chapter 1

Introduction

The increasing number, complexity, and volatility of business rules necessitates effective and efficient business rule organization supporting IT-business alignment. In this regard, current business rule management systems have limitations which we address with the proposed Contextualized Business Rule Repositories (CBRs). We motivate our research, describe our research method, present our use case, sketch our solution, and delineate our contributions.

This chapter is structured as follows: Section 1.1 delineates the motivation for CBRs and the goals for effective business rule organization. Section 1.2 gives a summary of the state-of-the-art and identifies limitations in current approaches to business rule organization. Section 1.3 describes the research method we employ, namely design-science, and how we apply it to develop a model for contextualized organization of business rules fulfilling the identified goals. Section 1.4 gives an introduction to the running example we employ throughout this thesis. Section 1.5 sketches the model for organizing business rules in contextualized business rule repositories. Section 1.6 delineates the contributions we make with this thesis. Section 1.7 gives an overview of the structure of this thesis.

1.1 Motivation

A business rule is, adapting rule definitions from dictionaries [170, 171], one of a set of explicit or understood regulations or principles governing conduct or procedure within a business. Another important aspect of business rules is that businesses can install, revise, and discontinue business rules as they see fit [146]. Different kinds of business rules exist. One of the most used classification schemata is by intended behavior [209]: structural assertions (definitions including facts and terms, i.e., vocabulary definitions) [75, 81, 83, 156], action assertions (constraints) [75, 81, 83, 133, 137, 138, 141, 156, 167, 168], and derivations (computations and inferences) [75, 81, 83, 133, 137, 138, 141, 156, 167, 168, 176]. Similarly, we consider business vocabulary definitions a specific class of business rules. Another classification, particularly relevant to IT-business alignment, discerns whether business rules can be automated in information systems or not [143, 146, 159, 167]. For instance, business rules such as "A hardhat must be worn at construction sites at all times" can hardly be automated.

Today's organizations face an increasing number of business rules [30, 87]. Most business rules have a specific scope, i.e., apply in specific situations or contexts only [32, 47, 67, 74, 115, 172, 175]. Furthermore, business rules are highly volatile [59, 176, 208], i.e., they frequently change due to increasingly rapid changes in their environments [81, 110, 208]. Moreover, due to ever more complex organizations, environments, governance, legislations, et cetera, the complexity of business rules increases. Effective and efficient management of business rules, capable of dealing with their number, volatility, and complexity, is vital [27, 43, 77, 81, 83, 156, 166]. Such management of business rules enables business management to flexibly and economically guide and control the organization and its operations [77, 201], thus promoting flexible and adaptable organizations [58, 75, 81, 135, 143, 156, 167, 176].

The number, volatility, and complexity of business rules poses several challenges to effective and efficient business rule management. First and foremost, business rule maintenance is affected – large business rule sets require effective organization mechanisms in order to keep the business rule set maintainable. Ideally, the organization mechanism facilitates IT-business alignment, i.e., facilitates applying IT "in an appropriate and timely way, in harmony with business strategies, goals and needs" [121, p. 3], a fundamental concern of business executives [121]. Large business rule sets also pose a challenge to business rule engines executing these business rules. The larger the number of business rules and/or the higher their complexity, the more computational resources are required, negatively affecting execution times. A challenge related to the maintenance of business rules is search within the business rule set. The larger the business rule set or the more complex the business rules are, the stronger the necessity for effective and efficient search mechanisms. Not particularly a challenge but an issue with many Business Rule Management Systems (BRMSs) is that they support automatable business rules only, i.e., non-automated business rules are not supported.

An effective business rule organization may simultaneously address all the above challenges and issues. The goals of an effective business rule organization are:

(1) to ease business rule maintenance,

(2) to improve business rule execution performance,

(3) to facilitate faster search for business rules and persons responsible for maintenance, and

(4) to support automated as well as non-automated business rules.

Another important aspect, regarding the situation-dependency of business rules, is the question how to determine the current situation and thus the business rule set(s) to be applied? In order to enable effective and efficient business rule management, and business rule execution in particular, an automated process should exist to this end. This process should accept information regarding a case, i.e., "a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome" [144, p. 5] (for instance, a legal case); determine the current situation from the provided case information; and execute the relevant business rule set(s). Since several business rule sets may be relevant, for example, due to inclusion hierarchies of business rule sets, strategies for merging business rule sets need to be defined.

1.2 State-of-the-Art

The *Business Rule Manifesto* [43] states ten principles of business rules essential for business-rule-based architectures and platforms. Main points are that business rules are discrete parts of business and technology models; that business rules should be explicitly and declaratively defined; that business rules should be managed in business-rule-based architectures which support continuous maintenance, business rule execution, and explanation of results; and that business rules are for the sake of business and should be managed by business and not IT.

Effective rule management is critical to business success [43, 67, 75, 77, 81, 110, 143, 167, 177]. *Business Rule Management (BRM)* combines methods, tools, and techniques [12, 77, 168, 186] to facilitate a systematic and controlled approach to manage business rules throughout their life cycle [12, 27, 43, 81, 88, 135, 146, 159, 161, 167, 186, 201, 208]. On a coarse level, BRM comprises the activities business rule acquisition, business rule maintenance, and business rule execution [163, 176]. Business rule acquisition includes tasks such as business rule elicitation, rule authoring, organization of business rules, and dissemination [27, 81, 139]. Business rule maintenance encompasses regular checks of business rules regarding their validity [12, 77, 81, 82, 156, 176], analyzing and simulating impacts of business rule changes [12, 27, 110, 156, 161, 186, 201], and ongoing refinements and improvements [201]. Business rule execution is composed of deployment, for example, forwarding business rule sets to a business rule engine, and the actual execution of business rules [27, 186, 208]. A task employed in many BRM activities and tasks is search [81, 86, 87, 135, 167, 176, 208], for instance, searching for specific business rules, a business rule's motivation, or the responsible rule developer. The effectiveness and efficiency with which these BRM activities and tasks can be performed are affected by the organization of business rules, for example, business rule execution can be sped up by proper organization of business rules.

Organizing business rules in a single rule set has major disadvantages for maintenance. Consequently, *current BRMSs and BRM approaches* often provide *inclusion hierarchies*, i.e., business rules may be organized into business rule sets which may include other business rule sets. This promotes business rule set reuse and consequently reduces redundancy and eases maintenance [27, 135, 139, 201]. Such inclusion mechanisms can be utilized to model hierarchical situations where business rules applying in a more general situation apply in its specific situations as well. Nevertheless, intended semantics are much clearer if a proper inheritance mechanism is employed where modifications to inherited business rules are allowed and may be restricted (for instance, an organizational unit may strengthen the condition for a business rule defined at organization level). Current approaches to business rule organization allow organization of business rules along single criteria (such as commercial BRMS, [82, 161]) often insufficient for a problem domain [27] or along fixed criteria (like [102, 177]). Fixed criteria do not allow for custom organization of business rules specific to an organization's needs. Also, explicit and structured description of situations is rarely supported. Consequently, IT-business alignment is only marginally supported. Moreover, none of the investigated approaches reported increased performance of business rule execution or improved search due to their organization of business rules. Nevertheless, some approaches provide means to bundle business rules for execution. Another drawback is that most BRMSs provide business rule organization techniques for executable business rules only.

Many research fields, such as data tailoring [23], semantic web [183], librarianship [71], artificial intelligence [18], or ubiquitous computing [57], employ *contexts* to organize knowledge. A context comprises context knowledge describing its scope and contextualized knowledge describing the knowledge valid within the described scope [57, 73, 115]. Contexts support faster knowledge entering and inference, factoring out of common knowledge and assumptions, easier browsing, and isolation of inconsistencies [15, 26, 32, 115]. Several context approaches, such as [127, 183], discuss inheritance of contextualized knowledge. Since an important aspect for understanding, applying, and communicating business rules is their situation- or context-dependency [32, 47, 67, 74, 115, 172, 175], contexts are an appropriate mechanism for organizing business rules.

Many current BRMS and BRM approaches lack a process to determine relevant business rule sets from cases – they assume that the business rule set(s) to be applied are known. A standard related to cases is the Case Management Model and Notation (CMMN) [144]. It is a graphical notation suited to represent the handling of cases requiring activities to be performed in a dynamic and unpredictable order in response to evolving situations [200]. Therefore, it can represent less structured processes compared to the Business Process Model and Notation (BPMN). CMMN defines a case as "a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome" [144, p. 5]. For instance, cases are employed in medicine comprising a patient's treatment, medical history, and current state or situation with the goal of improving the patient's health. The information for a specific case is stored in a single case file. For instance, a medical case may be stored as medical record within an IT-System. Such a case file may be modified by actions within the case or from the outside. Regarding a process determining the current situation or relevant business rules from a case, business case files in the sense of CMMN may be provided as inputs to the process.

1.3 Research Approach

The aim of our research is to construct artifacts serving the purpose of organizing business rules. To this end, we employ the *design-science* (also science of the artificial) research paradigm. Design-science research addresses relevant and unsolved problems by constructing innovative or unique artifacts or addresses solved problems by constructing more efficient or more effective artifacts [89, 124, 184] by employing design as research method [197]; creativity and trial-and-error are characteristic of such research [89].

A framework for design-science research in information systems has been proposed by March and Smith [124]. This framework comprises two dimensions: research artifacts (outputs) and research activities. *Research artifacts* of design-science research are constructs, models, methods, instantiations, and better theories [124, 197]. *Constructs* provide the conceptual vocabulary to describe and communicate problems within a domain. *Models* employ constructs to represent a situation, i.e., the design problem and its solution space; they express relationships between constructs. *Methods* provide, utilizing the constructs and models, guidance or even formal algorithms on how to perform certain tasks. *Instantiations* are operationalizations of constructs, models, and methods demonstrating feasibility of models and methods. *Better theories* result from reflection and abstraction of design artifacts as well as the employed methodology to construct them. Each research artifact can exist at different levels of abstraction, for instance, constructs at instantiation level or models contributing to the general advancement of knowledge in the domain [197].

Elementary *research activities* are build and evaluate [124]. Further research activities have been identified by Peffers, Tuunanen, Rothenberger, and Chatterjee [154] who combined them into a complete design-science research process model depicted in Figure 1.1. The first activity in this process model (problem identification and motivation) defines the specific research problem and motivates the necessity for a solution. Subsequently, quantitative and qualitative objectives for a solution are derived from the problem definition (defining objectives for solution). Once the objectives are defined, design artifacts are generated during the next activity (design

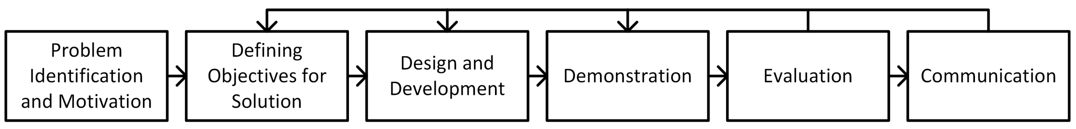

Problem Identification and Motivation
Defining Objectives for Solution
Design and Development
Demonstration
Evaluation
Communication

FIGURE 1.1: The process model for the design-science research method as proposed by Peffers, Tuunanen, Rothenberger, and Chatterjee [154]. Any of the first four activities may serve as starting activity.

and development). Design artifacts are then applied to solve one or more instances of the defined problem to demonstrate the artifacts suitability (demonstration). Subsequently, the artifact's effectiveness and efficiency as problem solution is evaluated regarding the defined objectives (evaluation). Depending on the evaluation results and gained knowledge, researchers may decide to iterate back to previous activities. The final activity is the dissemination of the problem, the artifact, demonstrations, evaluation results, et cetera (communication).

The described process may start with problem identification and motivation, with defining objectives for solution, with design and development, or with demonstration, depending on the initial reason for design-science research [154]. A problem-centered initiation starts with problem identification and motivation, for instance, research is caused by observations made. An objective-centered initiation begins with defining objectives, for example, research is triggered by a research need and can be addressed by a research artifact. A design- and development-centered approach opens with design and development, for instance, an artefact exists which has not been formally thought through as solution for the domain and problem at hand. A client-/context initiation starts with demonstration, for example, researchers retroactively apply research rigor to the design process.

Applying this framework to our problem, *constructs* encompass concepts from the business rule domain, from the organization domain, and from the context domain. Our research artifacts add further concepts to these constructs, such as rule modules and rule module inheritance. *Models* describe the relations between constructs, for instance, possible business rule organization strategies. In our case, the research artifacts generic CBR model and generic Extended CBR with Rule Modules (ECBR) model are models describing our solution for contextualized organization of business rules. *Methods* in our case encompass guidelines on how to use the proposed models and constructs as well as operations that can be applied to the models. Regarding *instantiations*, we instantiated the proposed models for different use cases to demonstrate their usefulness and implemented prototypes realizing the proposed models. We did not consider *better theories* in our research.

For each research artifact we oriented on the process depicted in Figure 1.1. Here we describe it for the generic CBR model presented in Chapter 3. First, we identify the problem and motivate the relevance of the problem as we have done in the beginning of this chapter. Based on a broad review of background literature we derive requirements for an effective and efficient contextualized business rule organization. Subsequently, we develop the generic CBR model, concretize it for the use case SemanticNOTAMs: Ontology-based Representation and Semantic Querying of Digital Notices to Airmen (SemNOTAM), and demonstrate its usefulness. Results have been presented at international conferences [35, 37, 40]. Feedback from demonstration, evaluation and communication has been used to improve the developed artifacts.

Building on the framework described previously [124], Hevner, March, Park, and Ram [89] proposed detailed guidelines for effective design-science research:

1. Design as an artifact: The output of design-science research is a purposeful artifact, be it a construct, model, method, instantiation, or better theory.
2. Problem relevance: The design artifact is a technical solution to important and relevant business problems.
3. Design evaluation: The design artifact must be evaluated regarding its utility, quality, and efficacy to solve a problem of relevance using well-executed evaluation methods.
4. Research contributions: Contributions resulting from design-science research have to be clear and verifiable.
5. Research rigor: Design-science research relies on rigorous methods in both construction and evaluation of design artifacts.
6. Design as search process: Design-science is an iterative process, in its most basic form it can be described as a generate/test or build/evaluate cycle [124, 184]. Within this cycle, design alternatives are designed and subsequently evaluated regarding the specific problem. Depending on the evaluation results, further design alternatives may have to be generated.
7. Communication of research: Outputs of design-science research have to be presented to technology-oriented as well as management-oriented audiences.

Table 1.1 depicts how we apply these guidelines in our research.

1.4 Use Case – SemNOTAM[1]

In the following we sketch the project SemanticNOTAMs: Ontology-based Representation and Semantic Querying of Digital Notices to Airmen (SemNOTAM)[2] and the resulting SemNOTAM prototype.

A Notice to Airmen (NOTAM) is a safety- and time-critical announcement of alterations to aeronautical conditions which might be of relevance to flight operations personnel. Many of the published NOTAMs are relevant only to specific flight operations and situations [64, 66], for instance, a helicopter pilot is not interested in closed gates at his/her destination aerodrome. Furthermore, the NOTAMs relevant to a specific flight operation are of different importance, for example, an aerodrome closure is critical whereas a taxiway closure is not. However, current NOTAM systems perform unsatisfactory in this regard [92] incurring information overload. Consequently, an effective intelligent filter and query system is indeed needed [66, 92].

SemNOTAM was a cooperative research project with partners from industry (Frequentis and AustroControl) and academia aiming to exploit semantic technologies to overcome the identified lacks. The designed prototype provides fine-grained semantic filtering, classification, and ranking of NOTAMs for flight operations personnel and their situations. Flight operations personnel and their situation are defined in

[1]Section based on publications F. Burgstaller, D. Steiner, and M. Schrefl. "Modeling Context for Business Rule Management". In: *2016 IEEE 18th Conference on Business Informatics (CBI)*. 2016, pp. 262–271 and F. Burgstaller, B. Neumayr, C. G. Schuetz, and M. Schrefl. "Modification Operations for Context-Aware Business Rule Management". In: *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. 2017, pp. 194–203

[2]Supported by the Austrian Ministry of Transport, Innovation, and Technology in program TAKE OFF under grant FFG-839006.

TABLE 1.1: Applying the research guidelines for design-science [89] to this thesis.

Research Guidelines	**Application in this Thesis**
Design as an artifact	Artifacts designed: the generic CBR model [35, 40], modification operations for CBR maintenance [34], rule module inheritance [36], prototypes [34, 36], and the generic ECBR model
Problem relevance	Concerning a motivation for the generic CBR model refer to Chapter 3, concerning a motivation for modification operations to Chapter 5, concerning a motivation for rule module inheritance to Chapter 6, and concerning a motivation for the generic ECBR model to Chapter 7
Design evaluation	Research artifacts are evaluated by prototypes, performance evaluations, informed arguments, and descriptive evaluation methods
Research contributions	Refer to Section 1.6 and Chapter 9 for summarized research contributions of this thesis
Research rigor	We make effective use of established notations and formalisms, such as Unified Modeling Language (UML) for modelling or mathematical formalisms for defining rule inheritance
Design as a search process	See the design-science research methodology process described above and its exemplary instantiation for the generic CBR model
Communication of research	Key results were presented at international conferences and workshops [34–37, 40]

so called interest specifications. An interest specification combines simple interests by conjunctions, disjunctions, and negation. Each simple interest either specifies a time, area, aircraft, or attribute of interest. For instance, an interest specification may define that a pilot is interested in a pre-flight briefing for a flight from Linz to Vienna using a small aircraft departing at noon. An interest specification may further contain information about required classifications and rankings of NOTAMs. A more detailed description of interest specifications can be found in Chapter 8 and [41].

The SemNOTAM prototype utilizes business rules to determine relevance, classes, and rankings of NOTAMs. These business rules are tacit knowledge of flight operations personnel and consequently need to be elicited. Therefore, to construct a small prototype, we have conducted several workshops with flight operations personnel, in particular, pilots. The elicited business rules are defined using aeronautical vocabulary. Besides relevant business rules, workshops with flight operations personnel revealed different business rules depending on the personnel's situation and the NOTAM's class, for example, NOTAMs of class heliport closure are irrelevant to landplane pilots but of importance to helicopter pilots. Such NOTAM classes are defined by aeronautical authorities, for instance, the Federal Aviation Administration (FAA) currently defines about 80 NOTAM classes, such as runway closure, called event scenarios [68].

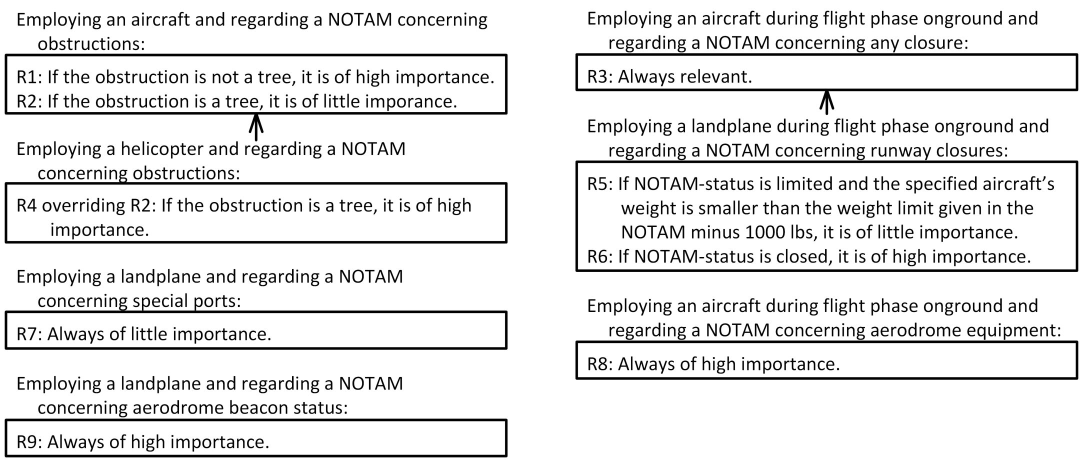

Employing an aircraft and regarding a NOTAM concerning obstructions:
R1: If the obstruction is not a tree, it is of high importance.
R2: If the obstruction is a tree, it is of little imporance.
Employing a helicopter and regarding a NOTAM concerning obstructions:
R4 overriding R2: If the obstruction is a tree, it is of high importance.
Employing a landplane and regarding a NOTAM concerning special ports:
R7: Always of little importance.
Employing a landplane and regarding a NOTAM concerning aerodrome beacon status:
R9: Always of high importance.
Employing an aircraft during flight phase onground and regarding a NOTAM concerning any closure:
R3: Always relevant.
Employing a landplane during flight phase onground and regarding a NOTAM concerning runway closures:
R5: If NOTAM-status is limited and the specified aircraft's weight is smaller than the weight limit given in the NOTAM minus 1000 lbs, it is of little importance.
R6: If NOTAM-status is closed, it is of high importance.
Employing an aircraft during flight phase onground and regarding a NOTAM concerning aerodrome equipment:
R8: Always of high importance.

FIGURE 1.2: Natural language business rules elicited for SemNOTAM.

Example 1.1. Figure 1.2 depicts a subset of the business rules elicited for SemNOTAM. We employ this subset of business rules as running example. Each business rule is identified by a unique prefix. We have grouped the business rules by the personnel's situation(s) in which they apply. The text outside a box describes the personnel's situation while the text within the box describes the applying business rules in natural language. An arrow between two boxes indicates an inclusion relationship, i.e., one rule set includes all rules of the other rule box. For instance, the rule set containing rule `R4` also contains rules `R1` and `R2`.

The situation-dependent application of business rules is but one issue to be addressed by SemNOTAM. Further difficulties arise from the complexity and continued evolution of the aeronautical domain, the legal requirement that all relevant NOTAMs must be presented to flight operations personnel, the large and increasing number of business rules, and the computational complexity due to the number of business rules, situations, NOTAMs, and vocabulary terms.

Consequently, effective business rule organization, easing business rule maintenance, improving business rule execution performance, and facilitating faster search, is vital to the SemNOTAM prototype. We employ the rule bases of SemNOTAM systems, such as the rule base for the SemNOTAM prototype, as running example throughout this thesis to demonstrate the usefulness of our proposed CBRs and ECBRs regarding effective business rule organization.

Details regarding SemNOTAM are given in Chapter 8. Details on the SemNOTAM prototype's architecture can be found in [41, 190], on the employed modelling techniques in [39], on the classification of NOTAMs in [190], and on its employment for improving communication between flight operations centers and aircrafts in [189].

1.5 Solution Sketch

In Section 1.1 we argued the need for a business rule organization easing business rule maintenance, speeding up business rule execution, facilitating faster search in business rule sets, and supporting executable as well as non-executable business rules; Section 1.2 discussed the organization mechanisms of current BRMS and BRM approaches and their shortcomings in this regard.

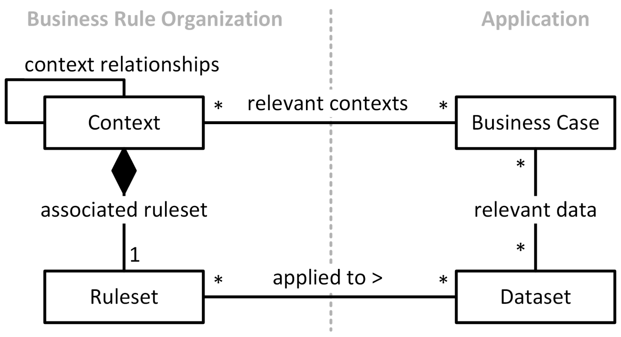


FIGURE 1.3: Overview of the proposed solution for CBRs.

In this thesis we design two generic, implementation-independent, and context-based organization models which are used to organize business rules in Contextualized Business Rule Repositories (CBRs) and Extended CBRs with Rule Modules (ECBRs) respectively. In particular, we utilize contexts to represent the hierarchical situation-dependency of business rules as well as the relationship between cases and business rule sets in the generic CBR model and the generic ECBR model. Repositories realizing a concretized CBR model or a concretized ECBR model for organizing business rules are called CBRs and ECBRs respectively.

Figure 1.3 depicts an overview of the proposed generic CBR and generic ECBR model. We distinguish two related parts: one regards the organization of business rules and the other regards the application. The business rule organization part represents contexts, i.e., context knowledge describing the scopes or classes of situations, and contextualized knowledge consisting of rule sets. Each *Context* describes the class of situations within which the associated rule set applies. Contexts may be related to each other, for instance, a context may describe a more specific class of situations than another one represented by a subsumption relationship. A *Ruleset* contains a set of business rules. A rule set is assigned to exactly one context, if a context is deleted so is its associated rule set.

The application part consists of business cases and datasets. The information comprised in a *Business case* may be employed to determine relevant contexts for it. As such, business cases are quite similar to case files in CMMN [144]. A *Dataset* contains data relevant to a business case. Each dataset may be assigned to several business cases and each business case may refer to multiple datasets.

The business rule organization and the application part are related. Several contexts and thus rule sets may be relevant to a specific business case. Each context may be relevant to several business cases. The contexts relevant to a specific business case may be determined automatically; their associated rule sets may be applied to the business case's relevant datasets.

Below we apply the overview model depicted in Figure 1.3 to different use cases to demonstrate its versatility. Example 1.2 applies the model to our use case SemNOTAM; Examples 1.3 and 1.4 show further applications in the law and the ubiquitous computing domain respectively.

Example 1.2. Considering our use case SemNOTAM, business cases correspond to interest specifications of flight operations personnel. For now, we regard a business case as describing a simple interest such as a time frame. Each context describes the class of situations in which its associated rule set applies, for instance, during flight phase enroute using a landplane. Furthermore, a context may specialize another one, for example, a context regarding landplanes specializes a context regarding aircrafts.

A rule set consists of business rules determining the relevance, classes, and rankings of NOTAMs. The dataset comprises published NOTAMs. The contexts relevant to a specific simple interest are automatically determined by the SemNOTAM CBR and their rule sets applied to the NOTAMs associated with the simple interest, i.e., the NOTAMs are classified, ranked, and filtered with respect to the simple interest.

Example 1.3. The presented solution overview can be applied to criminal law as well. A defence lawyer evaluates for a given trial possible courses of action to be taken during the hearing. Information on the represented party, governing laws, details on the case, et cetera, render some courses of action ineffective while other courses of action may crystallize as particularly useful. Considering the overview model in Figure 1.3, a business case describes a trial. Since trials may differ quite much by the accusation, for instance, theft or murder, different business case types for different accusations may be useful. A context describes the class of situations in which the associated rule set, i.e., constraints on possible courses of actions as well as guidelines and recommendations regarding courses of action, applies, for example, weak evidence and inexperienced prosecutor. The dataset comprises possible courses of action. Contexts relevant to a trial are determined and the corresponding rule sets applied to the courses of action associated with the trial. Thus, ineffective courses of action and guidelines and recommendations regarding the remaining courses of action are determined.

Example 1.4. A research field where contexts are extensively used is pervasive/ubiquitous computing [3, 5, 13, 50, 132, 138, 155, 191, 207]. Consider an adaptive User Interface (UI): depending on the end device, user, location, et cetera, the UI of an application shall be adapted in order to optimize user experience. A business case describes environmental information of an application call such as the user's current location or his/her experience. A context describes the class of situations for which UI adaptions are defined. A rule set contains the adaptations (in the form of rules) to be made to the UI when the associated context is relevant to the business case (the environment information of the application call). The dataset contains the UI configuration. Based on the environmental information of an application call, relevant contexts are determined and the corresponding adaptation rule sets applied to the UI configuration, effectively adapting the UI.

The generic CBR model and its extending generic ECBR model are designed with the goals of effective business rule organization in mind. Comparing CBRs and ECBRs to non-contextualized business rule repositories we make four claims:

(1) Eased business rule maintenance

(2) Improved business rule execution performance

(3) Faster search for business rules and persons responsible for maintenance

(4) Support for executable as well as non-executable business rules

1.6 Contributions

In this thesis we make three main contributions: the generic CBR model and CBR prototypes, rule module inheritance, and the generic ECBR model.

1.6.1 The Generic CBR Model and CBR Prototypes

We propose a generic multi-level CBR model for CBRs enabling multi-dimensional hierarchical organization of business rules reflecting situations as well as enabling explicit and structured description of classes of situations. This enables automatic determination of contexts, and thus rule sets, relevant to a given business case. Furthermore, faster search compared to non-contextualized repositories is enabled as the search space can be quickly narrowed using the explicit and structured description of classes of situations. Our CBRs support the determination of the current situation from business cases.

We provide proof-of-concept prototypes employing Vadalog, $\mathcal{FLORA}$-2, SPARQL Inferencing Notation (SPIN), and $\mathcal{FLORA}$-2 in combination with PostgreSQL as rule and implementation languages. We show that rule execution with our $\mathcal{FLORA}$-2-ProstgreSQL protoype outperforms non-contextualized rule execution for all but very small repositories.

In order to facilitate effective and efficient maintenance of business rules, we provide modification operations ensuring that a concretized CBR model moves from one consistent state to another. For these modification operations we show that their majority is more or equally efficient compared their semantic equivalent operations in non-contextualized repositories. Furthermore, we introduce four CBR-roles, namely, repository administrator, rule developer, user, and domain expert, enabling effective separation of tasks and responsibilities.

Claims (1) through (4), eased business rule maintenance, improved business rule execution, faster search, and support for non-executable as well as executable business rules are satisfied by CBRs.

1.6.2 Rule Module Inheritance

We propose a generic approach for inheritance of rule modules based on Datalog$^{\pm}$ as underlying rule language family. We define rule modules as rule sets with input and output interface definitions.

For such rule modules, we precisely define downward single-inheritance and multi-inheritance of rule module elements, i.e., their rule set and interface definitions. Inherited rules and interface definitions may be arbitrarily modified by default. We provide a definition for multi-inheritance conflicts which can be employed for detection. The inheritance mechanisms presented in this thesis do not provide guarantees regarding global restrictions imposed on rule sets by rule languages. For instance, a program in a concrete Datalog$^{\pm}$ language with stratified negation fulfilling all global restrictions may become invalid by adding rules introducing a dependency cycle with negation.

To prevent unconstrained proliferation of variations, we define modifications restrictions restricting the allowed modifications to inherited rule sets and interface definitions. For these modification restrictions we provide corresponding conformance checks.

We implement the rule module inheritance approach in a proof-of-concept prototype. This prototype employs Vadalog, a Datalog$^{\pm}$ implementation.

Since rule module inheritance is a generic approach it does not contribute by itself to our claims.

1.6.3 The Generic ECBR Model

We integrate the generic CBR model with the rule module inheritance approach, resulting in the generic ECBR model for ECBRs. To this end, we identify and address challenges stemming from this integration. We demonstrate the generic ECBR model by concretizing it for the real world use cases SemNOTAM and financial services provided by a large US tech-company.

ECBRs further contribute to claim (1), eased business rule maintenance.

1.7 Overview

Chapter 2 – Background gives a broad overview of background literature, i.e., business rules, contexts, and Context-Aware System or Application (CASA) development and deployment. For these topics we identify and discuss important aspects and attributes. Based on this section an appropriate context model for CBRs is designed in Chapter 3.

Chapter 3 – The Generic CBR Model: Modeling Context for Business Rule Organization employs the background literature from Chapter 2 to determine and justify our decisions regarding a context model for CBRs. Based on these decisions we design the generic CBR model and demonstrate its usefulness by applying it to SemNOTAM.

Chapter 4 – CBR Prototypes We present a number of proof-of-concept prototypes employing different rule languages and conduct feasibility studies. Furthermore, we introduce a database-backed CBR prototype enabling efficient contextualized rule execution and conduct a performance evaluation thereof.

Chapter 5 – Modification Operations for CBR Maintenance proposes atomic modification operations for concretized CBR models as well as CBR roles enabling effective maintenance in CBRs. We delineate composed modification operations employing atomic operations and CBR roles to ensure that concretized CBR models move from one consistent state to another. Furthermore, we show eased maintenance of business rules in CBRs compared to non-contextualized repositories.

Chapter 6 – Rule Module Inheritance with Modification Restrictions defines rule modules and single-inheritance as well as multi-inheritance in rule modules. Furthermore, we discuss restricting modifications to inherited rules and interface definitions.

Chapter 7 – The Generic ECBR Model: Extending the Generic CBR Model with Rule Module Inheritance discusses the integration of the generic CBR model with rules modules, rule module inheritance, and modification restrictions.

Chapter 8 – Use Cases of ECBRs demonstrates the usefulness of the generic ECBR model by concretizing it for two real world use cases, namely, SemNOTAM and financial services provided by a large US tech-company.

Chapter 9 – Conclusion concludes each of the main contributions, namely, the generic CBR model and modification operations for concretized CBR models, rule module inheritance, and the generic ECBR model, and summarizes their contributions regarding the goals of effective business rule organization.

Chapter 2

Background

To design a context model for CBRs, we first conduct a broad review of background literature regarding business rules (including vocabularies) and regarding contexts. For business rules, we identify important aspects and attributes, discuss their relations to business vocabularies, delineate their embedding in organizations, review their organization strategies, and discuss their management. Regarding contexts, we investigate different notions of context and determine aspects and options of contexts. Furthermore, we discuss the development and deployment process for context-aware systems or applications. This review of background literature is subsequently used to design an appropriate context model for CBRs.

This chapter is structured as follows: Section 2.1 discusses rules in general; Section 2.2 focuses on business rules, for instance, their definitions and principles, semantic definition of business terms in business vocabularies, business rule representation forms, business rule classes, as well as the embedding of business rules in organizations and their operations. Section 2.3 discusses various approaches to organize generic resources and business rules in particular. Section 2.4 regards management of business rules, in particular we discuss knowledge management and Business Rule Management (BRM). Concerning BRM we investigate the activities vital to an effective BRM. Section 2.5 delineates various definitions and notions of contexts and subsequently discerns and reviews attributes and aspects of context employed in the literature such as the components of context and context modeling. Section 2.6 describes the development and deployment process of a Context-Aware System or Application (CASA). Section 2.7 concludes the chapter.

2.1 Rules

A *rule* is an accepted principle or instruction stating what is allowed and what not [170, 171] – a prescribed guide for conduct or action [173]. Following these definitions, rules tend to remove degrees of freedom and thus need to be considered for making judgements and decisions [146, 167]. A more detailed definition is given by the Oxford dictionary: “One of a set of explicit or understood regulations or principles governing conduct or procedure within a particular area of activity. A principle that operates within a particular sphere of knowledge, describing or prescribing what is possible or allowable” [172]. Analyzing this definition we note that rules usually are part of a set having a larger impact than the sum of its individual rules [75, 81, 135]. Furthermore, rules apply in specific areas of activity or within a certain context, i.e., universal rules, if such exist at all, are an exception. Concerning information

This chapter extends the background given in F. Burgstaller, D. Steiner, and M. Schrefl. “Modeling Context for Business Rule Management”. In: *2016 IEEE 18th Conference on Business Informatics (CBI)*. 2016, pp. 262–271 systems, rules are employed in various areas of activity [135, 138], such as knowledge management [67, 135], businesses [81, 83, 146, 167], service composition [165], data bases [138], or expert systems [75, 81, 208].

This abundance of activity areas lead to various distinct rule modeling languages and representation techniques. Rule modeling distinguishes the meaning of a rule and its expression as rule statement [146, 167, 193]. Rules can be expressed using natural language [77, 81, 110, 138, 146], decision tables [138, 145], structured/controlled language [75, 81, 135, 138, 146, 176, 193], logical formulas [83, 138], procedural implementation [81], score cards [138], languages for encoding and storing (for example, Jess or Drools), formalized knowledge representation (such as Description Logics or F-Logic) [138], or standards/recommendations for rule interchange and translation with different levels of formal rigidity [135].

Here we shortly discuss three important standards/recommendations for generic rules, namely, Rule Markup Language (RuleML) [6, 174], Rule Interchange Format (RIF) [103, 136], and Production Rule Representation (PRR) [141]; standards specific to business rules are discussed in Section 2.2. RuleML [6, 174] is an open standard effort by RuleML Inc. aiming at an Extensible Markup Language (XML)-based uniform representation and interchange of rules across different logic formalisms and platforms. Its specification constitutes a system for web knowledge representation which provides bridges between, for instance, RIF, Semantic Web Rule Language (SWRL), common logic, and LegalRuleML. RuleML defines various rule language features clustered into three language families: deliberation, reaction, and consumer. An interesting family is the reaction family concerning reaction rules – derivation rules, knowledge representation calculi for reasoning, Event-Condition-Action (ECA) rules, production (condition-action) rules, triggers, and rule-based complex event processing – enabling syntactic serialization and semantics-preserving interchange.

RIF [103, 136] is an XML-based standard for exchanging rules between mainly web rule engines and facilitating rule set integration and synthesis. To this end, RIF provides different extensible families of languages (dialects) for which their syntax and semantics are clearly defined. The three defined families are RIF-Core, RIF Basic Logic Dialect, and RIF Production Rule Dialect, where custom dialects can be developed. The basic idea is that for exchange of rules a syntactic and semantics-preserving mapping is created.

PRR [141] is, other than RuleML and RIF, a platform-independent standard production rule and rule set representation compatible with rule engine vendor's definition of production rules enabling interchange of rules.

Whichever form of rule expression is chosen, rule statements should be defined using a vocabulary. A vocabulary provides unambiguous definitions [77, 146, 201] of all terms (concepts), names, facts (relating terms), et cetera that a given community, such as a company, uses when communicating. The importance of such vocabularies is underlined by the introduction of ISO standards, like ISO 1087-1 for English, and the Object Management Group (OMG) specification Semantics of Business Vocabulary and Business Rules (SBVR) [146].

2.2 Business Rules

There is no consensus yet on what exactly a business rule is [27, 87, 97, 138]. Refining the rule definitions given above, a business rule is one of a set of explicit or understood regulations or principles governing conduct or procedure within a business; a principle that operates within a business, describing or prescribing what is possible or allowable. This definition is similar to many business rule definitions in literature [43, 58, 77, 81, 83, 143, 146, 159, 166–168, 193, 208] where some take a more business others a more technical perspective. Furthermore, it highlights the importance of rule sets as does Gottesdiener: "Individual business rules make a business understandable, but it's their chemistry that brings it to life" [75, p. 2]. A business rule according to the SBVR standard is a "rule that is practicable and that is under business jurisdiction" [146, p. 98]. Under business jurisdiction means that the business can enact, revise, and discontinue its business rules as it sees fit [146]. Regardless the concrete definition employed, business rules support business operations by providing guidance, shaping business judgements, implementing courses of actions to be taken, and supporting or even making business decisions [143, 146, 167]. Thus, business rules are a vital asset [43, 67, 75, 77, 81, 110, 143, 156, 167].

An important document regarding business rules is the *Business Rule Manifesto* [43]. It states the Business Rule Group (BRG)'s view of business rules as essential and independent part of business and technology models and lays the foundation for business-rule-based architectures and platforms. They stipulate ten principles which we summarize together with references to additional literature making identical or similar statements below:

1. Business rules are primary requirements and are an essential and discrete part of business and technology models [12, 43, 75, 81, 110, 201].

2. Business rules are to be separated from business processes and procedures as business rules are neither processes nor procedures but constrain/govern them [12, 43, 81, 88, 97, 110, 135, 137, 143, 167, 193, 201]. Nevertheless, synergies between business processes, business rules, and their respective management should be utilized [97, 128, 137, 138, 164, 167, 201, 208, 209]. The externalization of business rules (for instance, decisions, pre-conditions, computations, restrictions on roles, or routing logic) enables simpler, more stable, and consequently improved business process models [81, 97, 138, 166, 167, 201].

3. Business rules are to be deliberate knowledge, i.e., explicitly defined, based on business vocabulary. They need to be nurtured, protected, and managed [27, 43, 77, 81, 83, 156, 167].

4. This goes hand in hand with the fourth principle: business rules should be expressed declaratively in natural language and not procedurally [43, 75, 81, 83, 97, 135, 156, 177]. Furthermore, business rules should be separated from their enforcement level, i.e., kinds of consequences if a business rule is violated, which may range from strict to guideline [43, 135, 146, 167, 176]. Strict enforcement means immediate sanction whereas guideline informs about violation but does not sanction.

5. Business rule expressions must be well formed, i.e., they can be verified (are they correct?) and validated (do we have the correct business rules?) [43, 167].

6. The sixth principle describes business-rule-based architectures supporting continuous business rule change, direct business rule execution, explanation of results, transparency of actual business rule evaluation, and the many-to-many relationship of business rules and events [43, 143].

7. Business rules define the boundary between acceptable and unacceptable business activity; the handling of these two cases should be separate [43].

8. Business rules are for the sake of business [27, 43, 75, 81, 143], i.e., they are about business guidance and thus motivated by business goals and objectives. Furthermore, each business rule comes with costs, thus its enforcement must be balanced against risks and opportunities. Often a small set of "good" business rules is more effective and efficient than a large set of business rules.

9. Business rules should be formulated, validated, verified, and managed by business people and not IT-people [12, 43, 75, 81].

10. Business logic/rules should be managed rather than their hardware or software platforms [43, 146], for example, business rules should be stored in such a way that they can easily be redeployed to other platforms. In particular, business rules and the ability to change them are essential to improving business adaptability [43]. The frequency with which business rules change, i.e., their volatility, may be different from rule to rule [59, 176, 208]. Since not all business rules can be automated [88, 143, 146, 159, 167], like "A helmet must be worn at all times when in the factory", focusing on hardware or software platforms means ignoring non-automatable business rules.

Various *other principles* have been proposed: Following from the Manifesto, business rules must be practicable, i.e., business people know what is allowed and can decide whether a business rule was complied to [81, 146, 167]. An often stated principle is that business rules should be atomic, i.e., they cannot be broken down anymore [75, 83, 135]. The SBVR states three further principles for business rules [146]: severability, i.e., the meaning of a business rule may be expressed separately from any other business rules; accommodation, i.e., conflicting business rules are taken as such, if no conflicts should occur the business rules must be adjusted accordingly; and wholeness, i.e., a business rule means exactly what is says and nothing more. Another important principle, especially if many distributed rule sets are used, is the rule of origin, i.e., each business rule is defined exactly once and reused if needed elsewhere [156, 167]. Other found principles include unambiguity, precision, clarity, traceability, compactness, consistency, completeness, and reusability [135, 193, 201].

Following these principles for business rules, business-rule-based architectures and platforms entail many *benefits*, the key benefits being flexibility and agility [12, 58, 75, 81, 135, 143, 156, 167, 176]. These key benefits are supported by the separation of business rules from processes and data; by atomic, explicit, declarative, well-formed, and technology-independent definition of business rules; and their direct execution, for instance, in a Business Rule Engine (BRE). Benefits for business processes by externalizing business rules are simpler, more stable, and consequently improved process models [97, 138, 167, 201]. Furthermore, business rules are managed by business people understanding the domain. Consequently, business-rule-based architectures and platforms enable cost savings due to, for example, shorter development times and faster application development [58, 135, 156, 193]. Other benefits include understandability for business people [27, 135, 167], facilitating the process of knowledge management [135], consistency and control of implemented and specified requirements [59, 156], increased maintainability and transparency of information systems [58, 156], technical independence [75, 156], and promoting reuse [27, 135, 156].

Two *standards* regard business rules: Semantics of Business Vocabulary and Business Rules (SBVR) and Decision Model and Notation (DMN). *SBVR* [146] is designed to unambiguously and formally define business vocabulary and behavioral guidance (including business rules and business policies). Furthermore, it defines a schema for interchange of business vocabulary and behavioral guidance. The difference to RIF, PRR, or RuleML is its support of business vocabularies as well as its focus on business. It is designed for business people to define unambiguous and translatable business rules as well as the necessary business vocabulary from their business language [146, 208]. As such, it follows most of the business rule principles described above. The business vocabulary described in [146] uses natural language expressions. Thus, it is designed for business community ownership and management, allowing the business vocabulary to be used and defined by the community relying on it. Another OMG specification regarding business rules aiming at business users is *DMN* [145]. A decision is regarded an act of determining an output from inputs using logic defining how to determine the output from the inputs [145]. Business decisions can be regarded as condition-action business rules or as inferences or computations. Existing modeling standards address decision-making from different perspectives, for example, by defining specific tasks/activities in which decisions are made like in BPMN or by defining decision logic like in PRR. DMN is intended to bridge these two perspectives. It provides a common notation understandable by business users responsible for eliciting decision requirements or for modeling, automating, managing, or monitoring decisions.

In the following we discuss business vocabularies, the foundation of business rules. Subsequently, we summarize different representation forms for business rules and present business rule classifications identified in the literature. Finally, we delineate the organizational embedding of business rules in businesses and their operations on the business as well as the technical level.

2.2.1 Business Vocabularies

Since business rule statements are a specific form of rule statements, they should be expressed using a vocabulary as well, in particular a *business vocabulary*. In fact, business rule statements are only of value if interpreted and defined against an unambiguously defined and well managed business vocabulary [77, 201]. The distinction between business vocabulary and business rule is not a strict one [135]; some researchers consider business vocabulary to be a specific class of business rules [75, 81, 83, 156] (see Section 2.2.3). A good practice is to keep concepts of fundamental essence in the business vocabulary whereas any business knowledge that might change should be treated as business rule [135, 167].

Business vocabularies are based on the semiotic/semantic triangle relating concepts (meanings), representations (expressions), and real-world things [146]. Vocabularies usually model concepts, representations, and their relationships. The relationship between representations and concepts is in general n:m, i.e., a representation may refer to several concepts; in a specific context, such as a speech community, it is n:1, i.e., a representation refers to exactly one concept. Based on these linguistic foundations, a business vocabulary contains terms (designating business concepts) and facts relating terms (also called verb concepts) [43, 135, 146, 167]. Terms are nouns expressing concepts essential to business, for instance, *pilot* or *NOTAMs* in SemNOTAM. Terms are to be unique within the context of consideration, countable, non-procedural, and atomic, i.e., they cannot be derived or computed from other terms [167]. Facts are expressed as verbs relating terms, for example, pilot *queries* NOTAMs. If related by facts, terms can be assigned roles, for instance, flight departs from *[origin]* airport [146, 167]. Facts can further be distinguished into properties (close relation of terms) and compositions (part-of relationship) [146]. Additionally, structural relationships between terms are possible: categorization (specialization of terms), classification (a term is an instance of a general term), characterization (a term incorporates a specific characteristic expressed as fact) [146]. In practice, business vocabularies are often represented using UML class diagrams [135] or ontologies.

Benefits of business vocabularies are, among others, unambiguous and formal definition of business terms and their relationships within a certain context, separation of meaning and representation allowing for example to express the same concept in different languages, the use of templates for terms and facts, thesauri capabilities by relating synonym or similar concepts and relationships, readily understandable for business users, and basis for integration of separately defined vocabularies [146].

2.2.2 Business Rule Representation

The described benefits of business rules depend on their appropriate representation. Business rules can be represented in many forms, such as natural language [77, 102, 110, 135, 146, 193], controlled language (for instance, employing patterns) [27, 75, 77, 81, 91, 135, 146, 165, 176, 193], integration in graphical models like UML class diagrams [75, 77, 159, 166], decision table or trees [27, 138, 145, 177], or logical formulas [138, 146, 159, 177], where graphical representations of business rules are suitable for smaller business rule sets [88].

These forms of business rule representation can be grouped into levels of formal rigidity. We identified several sets of levels: (a) business conversation piece, natural language version, rule specification language, and rule implementation language [81, 102]; (b) business policy, business rule statement, atomic business rule, and formal rule statement [83, 166] closer to Business Motivation Model (BMM) (see Section 2.2.4); or (c) simply into declarative business rules and their operational transformation (imperative shape) [58]. Ideally, transformation of a business rule from one to another level can be automated.

Besides different representations forms, two flavors of business rules can be distinguished: if-then (condition-action) and ECA or ECA-Alternative (ECAA) business rules [27]. ECA(A) business rules are basically if-then(-else) rules stating when to evaluate them.

2.2.3 Business Rule Classes

A critical prerequisite for formulating business rule statements is a clear understanding of which class of rule you are dealing with [193]. Various classification schemata have been proposed which Zoet [208] groups into domain-based, implementation-based, intended-behavior-based, and representation-based classification schemata. Domain-based classification schemata regard the application or focus area as classifying criterion [102, 209], for example, business function [77], application type [27, 193], or domain (like core rules or productivity rules) [27, 77, 82, 88]. Implementation-based classification schemata classify by implementation [209], for instance, in a BRMS or in databases [58, 88, 166]. One of the most used classification schemata is by intended behavior [209]: structural assertions (definitions including facts and terms, i.e., vocabulary definitions) [75, 81, 83, 156], action assertions (constraints) [75, 81, 83, 133, 137, 138, 141, 156, 167, 168], and derivations (computations and inferences) [75, 81, 83, 133, 137, 138, 141, 156, 167, 168, 176]. For representation-based classification schemata see the different representation forms in Section 2.2.2.

Further groups of business rule classification schemata identified are: by (1) violation, by (2) source, by (3) automatability, and (4) business-process-oriented. By violation (1) distinguishes definitional/structural rules and behavioral/operative rules [87, 146, 167, 176, 193]. Definitional rules supplement the business vocabulary; vocabulary definitions regard the essence of concepts while business rules provide the exact discrimination [167]. Definitional rules cannot be violated. Behavioral rules on the other hand, govern activity, i.e., they shape business conduct [167]. Behavioral rules can be violated, thus must have an enforcement level [167]. A classification of business rules by source (2) distinguishes external business rules, such as legislations, and internal business rules, established within the business [86, 208]. Classification by automatability (3) discerns whether business rules can be automated in information systems or not [143, 146, 159, 167]. The last group of classification schemata (4) aims at a better integration and alignment of business rules and business processes [102, 110, 209], for example, by distinguishing business rules into structural sequencing rule, actor inclusion rules, transactional sequencing rules, data condition rules, and outcome control rules [209].

2.2.4 Organizational Embedding of Business Rules

Business rules in isolation are useless, they need to be embedded in the business and its operations [135]. This embedding needs to be achieved on the business as well as the technical/IT level, ideally achieving a high degree of IT-business alignment. In the following we first discuss embedding on the business level and subsequently embedding on the technical level.

Ross [167] focuses on the business level, organizing business operations into three interrelated components: structure, the business vocabulary; power, the processes operating on the structure; and control, business rules constraining processes. Regardless the concrete embedding, successful businesses should be able to motivate their business rules and processes, i.e., they should be able to explain why they are in place [81, 143, 167, 176].

Therefore, OMG introduced the Business Motivation Model (BMM) [143] providing a schema or structure for developing, managing, and communicating business plans in an organized manner. An overview of BMM is given in Figure 2.1. An important assumption is that an enterprise's actions are driven by its decision to react to change and not by change itself, revealing why SBVR only considers rules under business jurisdiction as business rules. The foundation of a business' motivation are, at a high level, its aspirations (Vision) and its plans to realize them (Mission). The vision is broken down into goals and objectives (desired results) where objectives are quantified goals; the mission is broken down into strategies for approaching goals and tactics for achieving objectives (courses of action). Strategies may devolve to tactics and tactics may evolve to strategies over time. The courses of action can be realized by business processes governed by business policies and potentially guided by business rules. Each business is subject to internal and external influences which may provide opportunities or threats. Thus, assessments are necessary, judging an influencers impact on means and ends and identifying potential rewards and risks. Based on the assessments, directives (business policies and business rules) can be installed to govern and guide the courses of action, i.e., to keep the business on track towards its desired results [75, 143]. Business policies guide and govern strategies and tactics but tend to be, compared to business rules, less structured, less formally articulated, not atomic, and not directly enforceable or practicable [143, 146]. Furthermore, a business policy may include other business policies. Business rules are derived from business policies [83, 143, 146, 161]. All components employed in a

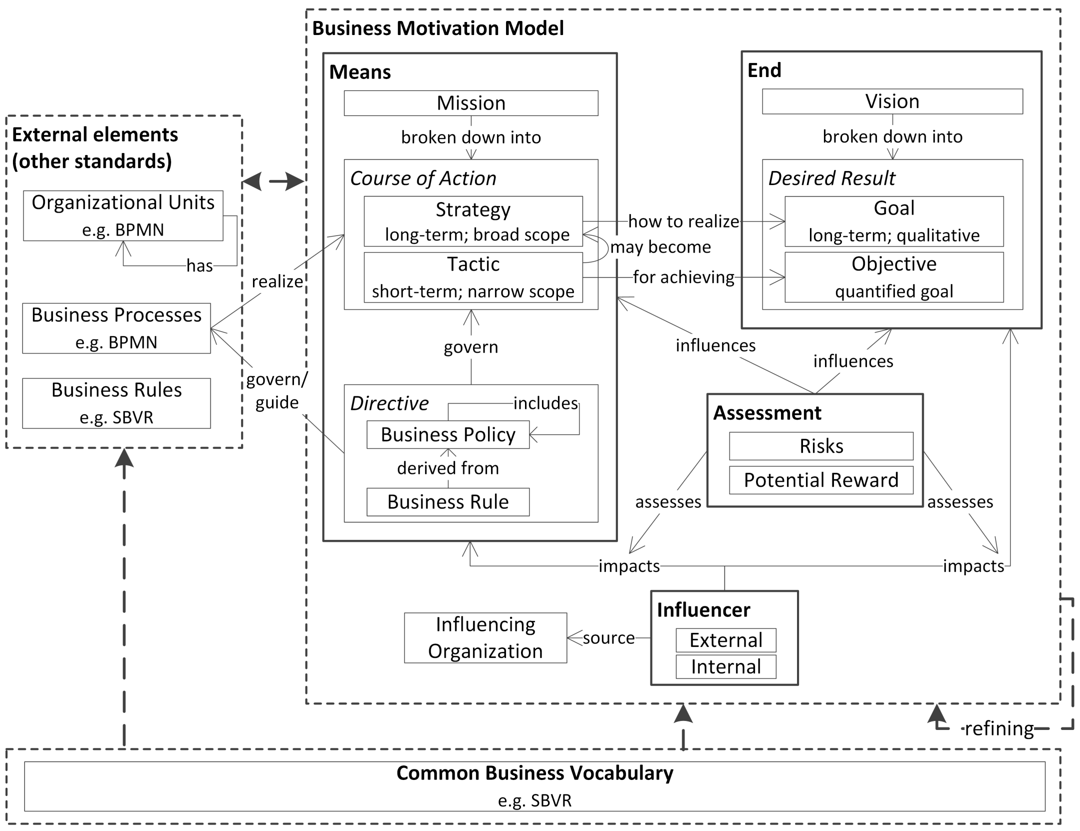


FIGURE 2.1: Overview of the BMM adapted from [143].

BMM, such as vision, strategies, business rules, et cetera, should be based on a common business vocabulary. In large businesses each organizational unit may have its own BMM refining the BMM of higher-level organization units. Other research [81, 110, 135] proposes business models/context very similar to BMM.

The Rule Maturity Model (RMM) [201] has a similar goal as BMM, i.e., aligning business objectives with the optimal BRM practices to achieve the business objectives. It provides six maturity levels, from zero, unawareness of business rules, to five, stewardship of business rules and their utilization for proactive change and prediction. Furthermore, the model orthogonally distinguishes three dimensions: business value, technical state, and business control. Business value describes the business value gained at each level. Technical state describes how business rules are managed and where they reside, for instance, in a business rule repository. Business control describes the role of business rule stewards at each level. This model can be used to assess the current state, defining a target state, certification, and assessing the business value of employing business rules in a business.

BMM takes a business perspective on the organizational embedding of business rules whereas RMM takes both a business and a technical perspective. A technical perspective is also taken by Gottesdiener who discusses the role of business rules in information system layering. Gottesdiener [75] describes a 3-tier architecture where business rules are separated from the data and presentation layer. Morgan [135] state that business rules may reside at any tier in an n-tier architecture. Zoet [208] proposes BRMS alongside workflow management systems, database management systems, and applications with user interface management systems on top, where BRMSs constrain and guide the other systems. Boyer and Mili [27] note that business rules can be implemented in any of these systems but BRMS are the best choice. A technical perspective is also assumed by Schacher and Patrick [176] proposing different business rule architectures: business-rules-as-a-service architecture, layered architecture (like [75, 208]), business-rules-in-database architecture, and generator-architecture where corresponding code is generated from business rules.

2.3 Approaches to Organizing Business Rules

We review approaches which can be employed to organize business rules, for instance, in a business rule repository. An appropriate organization of business rules is essential to enable effective and efficient business rule maintenance, business rule execution, as well as BRM. We review organization approaches independent of business rules, for example, polyhierarchies to organize generic resources, and organization approaches specific to business rules, for instance, organizing business rules into business-specific rule sets.

2.3.1 Organization Approaches Independent of Business Rules

A relevant research field is *organization* addressing the creation and maintenance of intentionally arranged collections of physical or virtual resources and any interactions with them on individual, collective, or institutional level [71, 74]. A discipline regarding organization is library and information science. The goal of organizing resources is to enhance existing resources and to retrieve and manage these resources. For this purpose, resources are described using a unique identifier and a set of resource descriptors [74]. Resource descriptors should be drawn from a designed language, i.e., a shared vocabulary. A schema specifies the set of resource descriptors for a specific resource type. Resources and their descriptors may have an effectiveness, i.e., their validity may be limited. Relationships between resources and resource sets may exist, potentially necessary to satisfy specific user requirements. Activities to be supported by organizing resources include search, browsing, and berrypicking [71]. When searching, the target is known; when browsing, we try to learn what we are looking for; and when berrypicking, the search target may evolve during search. Success of retrieval is measured by retrieving all and only relevant resources (recall and precision). Principle design questions when organizing are what is being organized, why, how much, when, and by whom [74]? Exemplary organization strategies are: (hierarchical) classification, polyhierarchies, faceted classification (orthogonal facets being composed of flat or hierarchical classification schemata), metadata, or indexing [71, 74]. Once an organization system is created, it must be maintained [74]. Predictable maintenance activities regard updates to vocabularies or schemas as well as upgrading technology; challenging maintenance activities regard changes to principles of organization or implementation technology [74].

A relevant concept employed in software engineering, database design, and ontologies is *modularization* [138, 151]. In its most generic meaning modularization "denotes the possibility to perceive a large knowledge repository (be it an ontology or a database) as a set of modules, i.e. smaller repositories that, in some way, are parts of and compose the whole thing" [151, p. 5]. The term modularization can denote the process as well as the result. The goals of modularization are scalability regarding querying and reasoning as well as evolution and maintenance, complexity management, understandability, context-awareness and personalization, and reuse of knowledge [53, 151]. In a nutshell, modularization aids controlled and structured development of large knowledge repositories, distributed knowledge authoring, and reuse of knowledge [151].

The two approaches to modularization are partitioning, splitting a large knowledge base into modules (possibly linked), and extraction, extracting relevant concepts from a large knowledge base into modules [52, 151]. The latter approach is interesting if context-dependent knowledge is to be organized – a modularization may describe different conceptualizations of the same domain [151] or organize knowledge along relevant dimensions [18, 31]. Irrespective of the concrete approach, created modules should be syntactically, semantically, and logically correct: (a) syntactically, the knowledge specification should adhere to the rules of the underlying language and data model, (b) semantically, a module should make sense for the users, and (c) logically, the knowledge inferred by the module is the same as could be inferred in the original repository [151]. Strategies for modularization include modularization creating disjoint modules, modularization resulting in overlapping modules, modularization driven by semantics of the application domain, modularization by graph decomposition, modularization employing machine learning algorithms, and composition of existing repositories [151]. Whichever strategy or mixture of strategies is appropriate depends on the requirements of the application at hand [53, 151]. Besides strategies for modularization, monitoring and evolution of existing modularizations is important to take appropriate maintenance actions, for instance, retiring unused modules. Criteria for evaluation of modularizations are application-dependent and include, amongst others, logical correctness, local completeness, module size, intra-module distance, module cohesion, domain coverage, connectedness of modules, redundancy of knowledge, and performance [151].

2.3.2 Organization Approaches Specific to Business Rules

Of particular interest to this thesis is the organization of business rules in a business rule repository. The foundation of any organization strategy is the grouping of business rules into *rule sets* which can be treated as units [135] (also rule books or rule populations [135, 167]). Various organization strategies for rule sets have been proposed [33, 82, 102, 119, 135, 138, 161, 177, 201], such as result-oriented, condition-oriented, performance-oriented, business-specific grouping (for instance, by business lines or business policies), business rule class oriented, divide and conquer, or situation-oriented. An issue with situation-oriented grouping is that many business rules apply in more than one situation [135]. The rule sets/modules created by a concrete organization strategy may be related to each other, for instance, by rule sets dependencies [201] or by hierarchical relationships [135]. Regardless the organization strategy, rule sets should be self-consistent, complete, non-overlapping, and managed [135].

The optimal organization of business rules depends on the problem domain, effective division of labor, and mapping to rule execution (typically by rule set extraction) [27]. Many BRM solutions provide hierarchical organization of rule sets where business rules of a higher-level rule set apply in lower-level rule sets as well [27, 135, 139, 201]. Often, such an organization along a single parameter or criteria is insufficient for a problem domain [27]. Employing metadata allows to introduce orthogonal dimensions to rule set hierarchies [27]. This approach implies a more sophisticated algorithm to extract rule sets for deployment. Consequently, major differences between development organization and runtime organization may exist, which may complicate maintenance and determination of actually applied business rules [27].

Issues particularly important when managing business rules are their increasingly large number [30, 86, 87, 138, 208], their increasing complexity, and their volatility [59, 176, 208]. Consequently, business rules need to be elicited incrementally [27, 43, 201] and, drawing on modularization, be organized in logical structures [81, 82, 102, 135, 138, 161, 176, 177, 208] promoting effective division of labor, incremental rule elicitation, and business rule and business rule set reuse [27, 201]. Moreover, the chosen organization strategy must aid decision making [161] and support the activities of BRM. When choosing an organization strategy, dependencies [82, 161, 167] and relationships [138] between business rules and within and between business rule sets need to be taken into account. Furthermore, the size of business rule sets needs to be considered, for instance, sets comprising more than 25 rules are typically less manageable [81].

2.4 Managing Business Rules

Business rules as vital asset are part of a business's knowledge [201]. Consequently, *knowledge management* may provide important clues for BRM. In this section we review knowledge management in general and BRM in particular.

2.4.1 Knowledge Management

Knowledge is context-dependent, i.e., knowledge needs to be contextualized to be shared and reused [32, 67]. The same holds for business rules which are part of a business' knowledge. The knowledge in a business needs to be managed and nurtured, for instance, it needs to be elicited from domain experts or distributed to the people needing it. Evans, Dalkir, and Bidian [67] summarize knowledge management as comprising systematic processes for knowledge acquisition, organization, sustainment, application, sharing, reuse, and renewal, designed to enhance a business' performance and to create value. Knowledge may be organized in modules [138], for example, by knowledge sources [135]. Furthermore, knowledge may be automatized, for instance, in rule-based systems or Business Process Management (BPM) systems, by knowledge engineers [138].

Based on their summary of knowledge management, Evans, Dalkir, and Bidian [67] developed the knowledge management cycle depicted in Figure 2.2: Once a request for knowledge is received, a searcher must check whether the knowledge exists or whether it must be created (identify). Once knowledge deemed valuable to the request for knowledge has been identified or created, it is stored and organized within the organizational memory. Knowledge is retrieved from the organizational memory to be shared and disseminated within the organization and with external organizations. Two principles for sharing can be followed, push and pull. Once shared, knowledge can be utilized to solve problems, make decisions, perform tasks, et cetera. The shared and used knowledge can form the basis for new or improved knowledge (learn). This knowledge management cycle can be tailored to a BRM cycle (c.f. Section 2.4.2).

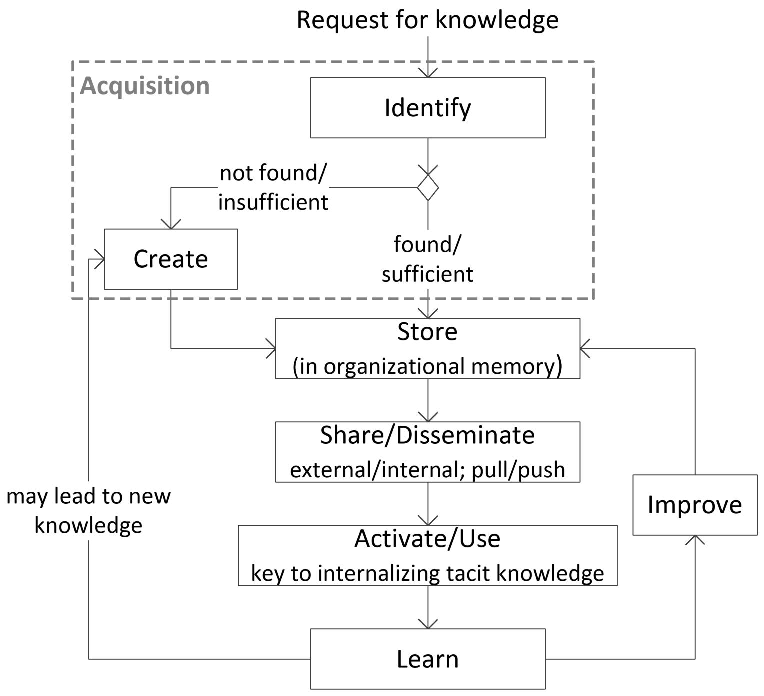


FIGURE 2.2: Knowledge management cycle adapted from [67].

2.4.2 Business Rule Management

Effective business rule management is critical to business success [43, 67, 75, 77, 81, 110, 143, 167, 177]. Thus, a systematic and controlled approach is necessary to manage business rules throughout their life cycle [12, 27, 43, 81, 88, 135, 146, 159, 161, 167, 186, 201, 208]. Such an approach, combining methods, tools, and techniques, is called BRM [12, 77, 168, 186]. BRM implementations are influenced by a number of factors [159, 208] such as the intended value proposition (guidance, communication, decisioning), business or IT orientation, business rule volatility, project management technique, users, or business rule representation. Since business rules are based on business vocabulary, business rules and business vocabulary should be integrated and managed together [81, 146, 167, 176]. Software systems providing BRM are called BRMS, supporting business ownership [27, 81, 135]. BRMS as well as BRM need to be integrated with the business' organization and (IT) architecture [77].

BRM needs to support various activities. On a coarse level acquisition, maintenance, and execution are necessary [163, 176]. In the literature the following activities, forming a process quite similar to the knowledge management process in Figure 2.2, are identified:

Capturing/Elicitation [12, 27, 77, 81, 83, 102, 138, 139, 156, 176, 186, 193, 201, 208] includes identification of resources [12, 186], for instance, business policies/strategy [81, 83, 139, 143, 167, 176, 201] or documents [135], and choosing an appropriate elicitation technique [116].

Authoring/Specification [12, 27, 77, 81, 102, 156, 186, 201, 208] Some authors differentiate capturing and authoring, where the former is the identification and the latter the formal specification of business rules. This activity includes creation of relevant facts and terms, i.e., business vocabulary management [77].

Organization/Structuring [27, 81, 138, 139] This includes organizing the elicited business rules in some form, frequently a business rule repository. This activity is often viewed as part of authoring.

Testing [12, 27, 75, 77, 81, 83, 87, 135, 139, 193, 201, 208] This activity includes unit testing, component testing, functional testing, system testing, performance testing, user acceptance testing, and/or continuous/regression testing [27, 135]. Furthermore, validation and verification of business rules [86, 186, 208] as well as semantic consistency testing [27] are assigned to this activity.

Sharing/Dissemination [81, 139, 156, 167, 176, 193] Making elicited business rules available to all stakeholders, usually employing a repository [44, 102, 156, 161, 193] as single point of entry [81, 208]. Some, like Nelson, Rariden, and Sen [139] and Schacher and Patrick [176], include deployment in this activity.

Realization/Automation [12, 27, 75, 77, 81, 83, 86, 102, 135, 138, 139, 156, 193, 201] Based on different influencing factors, such as volatility or automation of business rule, an implementation technique is chosen and the business rules implemented. This activity is sometimes viewed as sub-activity of deployment, for instance, by Smit, Zoet, and Matthijs [186].

Deployment [27, 77, 81, 135, 138, 186, 201, 208] This includes actually deploying the realized business rules. For BRMSs, this can include rule set extraction, i.e., a query is posed to the business rule repository and the resulting business rules packaged into an executable rule set [27].

Execution [27, 186, 208] The actual execution of business rule deployments. BRMSs often support ruleflows allowing fine-grained control over rule execution/evaluation within a rule set.

Orthogonal to this process, i.e., applying across the above listed activities, we identified these activities:

Security and Authorizations [102, 176, 193, 201] Access, responsibility, and modification permits/restrictions of stakeholders regarding business rules need to be controlled.

Maintenance / Change Management [12, 27, 77, 81, 110, 139, 176, 193] This activity includes regular checks of business rules and whether their motivation still holds [12, 77, 81, 82, 156, 176] as well as analysing and simulating the impact of business rule changes on stakeholders [12, 27, 110, 156, 161, 186, 201]. Besides checks, ongoing refinements and improvements are part of this activity [201].

Search [81, 86, 87, 135, 167, 176, 208] Typically, users search business rules in certain situations [167, 193, 201, 208], where a business rule is implemented [167], or a business rule's motivation [167].

Governance [27, 77, 135, 186, 208] Change necessitates coordination of activities to maintain a consistent system [135, 143]: Business rule governance is key for successful BRM – it disciplines communication and change management of business rules [27]. This activity includes versioning [27, 77, 102, 135, 167, 201, 208], traceability of business rules (from motivation to implementation) [81, 86, 102, 110, 167, 201], validity management, and metadata management [27, 81, 87, 135, 167, 193].

2.5 Contexts and Context Modeling

In this section we conduct a broad review of the concept of context and its various interpretations, components, operations, and modeling options. Based on our review of context, we develop our particular understanding of context for business rule organization in Chapter 3, utilizing our review of business rules. Below, we investigate the utility of context and various definitions for it.

Context is essential to resources and knowledge – they become useful only in particular contexts [31, 32, 47, 67, 74, 115, 175], for instance, knowledge about fishing is only useful if fishing grounds are nearby. Furthermore, contexts may be utilized as mechanism for managing, organizing, and reasoning about knowledge [15, 31, 149, 157]. As such, contexts enable [15, 26, 32, 115]: (1) faster knowledge entering and knowledge inference, (2) isolation of inconsistencies in contexts, (3) factoring out of shared assumptions when considering several contexts, and (4) easier browsing.

Efficient employment of context requires to understand the concept of context [2]. Most people have a tacit understanding of what context is although they find it difficult to explicate this understanding [2, 115]. Dictionaries [48, 49] usually provide two definitions: one regarding linguistics – parts of discourse surrounding a particular phrase or text helping to explain its meaning – and a general one – the situation/conditions within something exists or occurs. From these definitions follows that context is essential for understanding information and actions as well as for communication [31, 47, 60, 115, 153, 175]. Thus, it has received attention in different research fields, such as linguistics [5, 15, 29, 31], (cognitive) psychology [5, 15, 29], philosophy [15, 29, 207], or computer science [29, 57, 105, 138, 207]. These research fields regard similar, overlapping, and complementary aspects of context [15, 29]. Especially in computer science diverse research on contexts has been conducted, for example, contexts in the semantic web [5, 93, 183, 207], pervasive/ubiquitous computing [3, 5, 13, 50, 132, 138, 155, 179, 191, 207], context-aware computing [5, 24, 25, 51, 99, 114, 129, 130, 133, 134, 138, 195, 198, 199, 202, 206], or symbolic Artificial Intelligence (AI) [15, 29, 31, 32, 207]. A consensual and precise definition of context is missing [2, 3, 26, 29, 32, 50, 57, 60, 85, 138, 179, 191] – most likely because it is itself context-dependent, i.e., different research fields have different foci [15, 31, 32, 207]. Nevertheless, a widely acknowledged definition has crystallized over the years although not without criticism [107]:

Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [57, p. 5]

More general, entities have a specific focus [15, 29, 157], such as an objective, a step in problem solving, or a decision to make, for which context enables us to separate relevant from irrelevant knowledge [31, 32, 47, 67, 74, 115, 157, 175]. Pomerol and Brézillon [157] describe context as comprising, for a given focus, context knowledge, external knowledge, and proceduralized knowledge. Context knowledge is knowledge relevant, external knowledge is knowledge unknown/irrelevant, and proceduralized context is knowledge effectively used for the task at hand (focus) [157, 198]. Similarly, Bradley and Dunlop [29] distinguish: (1) focal information in the focus of attention and context information processed outside the focus of attention; (2) meaningful context relevant to user's goal(s) and incidental context irrelevant to user's goal(s); and (3) external context being the situation or the user's environment

TABLE 2.1: Summary of identified interpretations of context.

Aspect	**Options**
Notion of context	Representational / interactional
Objective	Viewpoints / separate concerns

TABLE 2.2: Representational and Interactional Context [15, 26, 31, 32, 60, 107, 149].

Representational context	**Interactional context**
Context as information	Context as process
Context is discrete/delimited	Context is continuous; particular to each activity occurrence
Context is static/stable	Context is dynamic; unknown beforehand
Activity happens within contexts	Context arises from activity; it is actively produced, maintained and enacted

and internal context being the knowledge/mechanisms of users' cognitive processes. Thus, context can be utilized to guide the focus of attention [31].

In the following we discuss interpretations of context, context components, operations on contexts, and modeling of contexts. Context-awareness and Context-Aware Systems or Applications (CASAs) employing context and context models are discussed in Section 2.6.

2.5.1 Interpretations of Context

Regarding interpretation of context we identified two distinct notions of context and two different approaches to modularizing knowledge into contexts. A summary is given in Table 2.1.

Due to the different research fields in which context is employed, two *notions of context* developed, detailed in Table 2.2: a positivism/representational notion assumed in engineering and a phenomenological/interactional notion assumed in cognitive science [15, 26, 31, 32, 60, 107, 149]. The *representational notion* views context as being comprised of a set of encodable features of the environment surrounding an activity or information which is made available alongside the activity or information. This viewpoint is often assumed in computer science: in software development and databases context is represented by views, aspects, roles, or workspaces; in machine learning context is environmental information used for classification; in symbolic AI context is a means to partition knowledge or facilitate reasoning; and in knowledge representation context is an abstraction mechanism for partitioning into possibly overlapping parts [107]. The *interactional notion* views context as an interaction problem focusing on how and why people can achieve and maintain a common understanding of the context for their actions. For instance, context is regarded interactional when constructing or working with human computer interfaces [31].

Context can be employed with different *objectives*. On the one hand, contexts can be seen as different viewpoints of a single object where all views need to be integrated to get an overall idea. An example are orthographic projections where each projection shows a different view of the object but only their integration reveals the 3-D object. On the other hand, contexts can consider separate concerns, i.e., each context considers different objects.

Organizing knowledge along contexts is supported by the modularization approaches knowledge extraction and partitioning [138, 151] presented in Section 2.3. Knowledge extraction can be utilized to create contextualized knowledge by extracting knowledge from a larger knowledge base [5, 18, 31, 52, 151, 183]. Thus, it may be used for both context objectives, viewpoints and separate concerns. Knowledge partitioning can be utilized to perform partitioning of a larger knowledge base into contextualized knowledge in the strict mathematical sense, i.e., each context corresponds to a separate concern [26, 52, 151].

2.5.2 Components of Context

In order to discuss components of context, we assume the context-as-a-box metaphor [73, 115]. This metaphor assumes a representational notion of context (Section 2.5.1). Nevertheless, some aspects of interactional context like dynamicity can be covered as well.

The *context-as-a-box metaphor*, employed by Giunchiglia and Bouquet [73] and Cyc [115], considers a context as box containing a collection of formulae (knowledge) valid under the same circumstances. These boxes may be hierarchically ordered. Each box is described by a set of parameters, delimiting the boundaries of the context. A specific context is identified by values for these parameters. This results in two layers of knowledge: contextualized knowledge (inside the box) is separated from context knowledge (outside the box) [3, 5, 19, 21, 32, 73, 115, 149, 155, 175, 183, 198]. Thus, a context generalizes a collection of assumptions [127] expressed in parameter values – it provides unarticulated knowledge about contextualized knowledge [5, 31]. For instance, knowing that we speak about the country France, the term Paris identifies France's capital. Not knowing about France, Paris could also be a first name or a city in the US. A key issue with the box metaphor is to identify context parameters, allowed values, their relations, and actual contexts [29]. In the following we assume the context-as-a-box metaphor for our analysis of context. Context components, their aspects, and options are summarized in Table 2.3.

Regarding the *contextualized knowledge* within a context box we may encounter different volatility levels, for instance, in some contexts the contained knowledge may not change, while in others it may change frequently. The knowledge within a context may have arbitrary relationships [18, 90, 115, 127], some suggest that knowledge in other contexts or complete contexts may be referenced [5, 127, 183]. For systems where contexts contain information which can be reasoned about, reasoning local to specific contexts and uniform reasoning have been discussed [18, 26, 32]. With local reasoning each context specifies the reasoning algorithm to be employed for its contextualized knowledge. Uniform reasoning applies the same algorithm to all contextualized knowledge. Whichever form of reasoning is employed, it can be utilized to detect conflicts of knowledge. These may occur within a context or when knowledge of another context is imported, for example, by inheritance. Once conflicts are detected, resolution strategies should be employed [3]. Various resolution strategies, like prioritization, most-specific knowledge, or manual resolution, have been proposed (c.f. [20, 56, 61, 104, 113, 181, 192]) – on a higher level we can distinguish manual, (semi-)automated, and no resolution.

Regarding *contexts* themselves, some propose predefined sets of contexts [155, 202] while others argue for unlimited number of contexts [15]. Nevertheless, even

TABLE 2.3: Summary of reported context components and their aspects based on the context-as-a-box metaphor.

Aspect	**Options**
Contextualized Knowledge	
Volatility	High / medium / low
Referencing contextualized knowledge	Relevant / irrelevant
Reasoning	Uniform / local / none
Conflict resolution	(Semi-)automatic / manual / none
Contexts	
Set of contexts	Dynamic / static
Context relationships	Independent / related
Semantics of context relationships	Inheritance / other / none
Context Classes	Relevant / irrelevant
Context Parameters	
Parameters of context	Uniform / individual
Set of parameters	Dynamic / static
Parameter relationships	Independent / related
Semantics of parameter relationships	Individual
Type of parameter	Primary / secondary
Parameter Values	
Set of parameter values	Dynamic / static
Value relationships	Independent / related
Semantics of value relationships	Subsumption / other / none

for predefined sets of contexts modifications may become necessary. Thus, we differentiate static and dynamic sets of contexts. Relationships between contexts, such as hierarchical context relationships [5, 21, 26, 32, 115, 127, 183], relationships relating different abstraction levels of contexts [5, 85], or general relationships [5, 155], are often discussed. Relationships of contexts often have implications on the contexts related, i.e., different context relationships may have different semantics, for instance, context hierarchies often imply knowledge inheritance [5, 115, 127, 183]. Orthogonal to knowledge inheritance one may employ context classes [93] which can be applied to contexts outside a context hierarchy.

Assuming the box metaphor, contexts are described by *parameters*. While many researchers propose different sets of parameters and parameter values [2, 3, 18, 29], for example, the twelve parameters used in Cyc [115], others argue that contexts can never be completely described and thus the set of parameters is infinite [18, 32, 73, 127]. These two views are also known as poor context (enumerable parameters) and rich context (infinite parameters) [115, 127]. Nevertheless, to employ contexts in computer science it is necessary to focus on a limited amount of parameters which can be represented in some software or hardware system. Other discussions regard whether the set of parameters is the same for each context, whether they may differ [18], or whether the set of parameters is modifiable. Furthermore, the relationships of parameters with contextualized knowledge, contexts, other parameters, and parameter values as well as their effects are discussed [18]. A relationship often discussed is whether a parameter's values are directly measured (primary) or derived from other parameters and their values (secondary) [2, 21, 155].

For each parameter, its type or range of allowed *parameter values* needs to be defined. Similar to contexts, these may be fixed or changed later on if necessary. Furthermore, parameter values may be related with other parameter values or context components. For instance, statistical data can assume different levels of measurement, namely nominal, ordinal, and cardinal, based on the relationship between values. Nominal corresponds to no relationships between values while ordinal introduces a ranking of values and cardinal allows arithmetical operations on values. An interesting relationship is subsumption [115, 183] where a parameter value subsumes another one. Subsumption hierarchies of parameter values can be used to construct context hierarchies [115, 183]. Analogously to context and parameter relationships the semantics of the relationships need to be defined.

2.5.3 Modeling Context

A context model is used to represent an explicit and coherent picture of contexts and their components as well as their relationships (i.e., context knowledge in the context-as-a-box metaphor) and context knowledge sources to be exploited in a CASA [32, 84, 133, 183, 207]. Challenges in modeling context are, amongst others, heterogeneous context knowledge, dynamicity, and quality of context [105]. Once these challenges are dealt with, context models promote explanation [32], separation of context concepts and domain/application concepts [3, 155, 175, 198], reduction of complexity in CASAs improving performance [21, 133], improvements regarding maintainability and evolvability of CASAs [21], support of exchange and access to context knowledge as well as reasoning about context knowledge [207], and support of context management [207]. We discuss requirements regarding context modeling and identify decisions necessary concerning context modeling. A summary of the decision-space is given in Table 2.4

Requirements regarding context modeling are mostly application- and domain-specific, also depending on the interpretation of context assumed. General requirements identified are: context models should be human- and machine-readable [133, 134] as they are often defined by hand [84]; evolvability/flexibility of context models [84, 191]; and appropriate level of formality, mainly depending on the type of formalism employed (see below). Other requirements regarding context modelling include representation of variable context granularities, context constraints, and heterogeneous context knowledge, as well as richness and quality indication, partial validation of context models, distributed composition, and application to existing environments [21, 24, 191].

Choosing a suitable *type of formalism* for context modeling requires a deep understanding of the context problem at hand [24]. Various types of formalisms have been proposed and analysed. Many of them can be represented using *graphical modeling* formalisms such as Resource Description Framework (RDF) graphs or UML [21, 79, 85, 191]. Some can be employed in a Model-Driven Development (MDD) fashion [94, 101, 175]. Types of formalisms often discussed are:

Key-value [21, 31, 138, 191, 198] and Markup [21, 191, 198] are the simplest types of formalisms. Key-value models employ key-value pairs to describe contexts whereas markup models utilize a variety of markup languages like XML. Both

TABLE 2.4: Summary of relevant aspects to modeling contexts.

Aspect	**Options**
Type of formalism	Key-value / logic-based / object-oriented / ontological / other
Graphical representation	Relevant / irrelevant
Meta/Generic model	Relevant / irrelevant
Level of abstraction	Conceptual / physical

have been shown to be limited in capturing relationships and heterogeneous context knowledge, consistency checking, and reasoning on contexts.

Logic-based [32, 138, 191, 198] types of formalisms are often employed due to automated reasoning and strong formalization enabling verification and validation of context models. Nevertheless, the flexibility of such models may be limited.

Object-oriented [138, 191, 198] types of formalisms intend to utilize concepts from object-orientation such as encapsulation, reusability, interfaces, and inheritance for context modeling. Since object-orientation is a widely used paradigm, object-oriented context models can be used to represent context at programming level.

Ontological [21, 31, 79, 138, 191, 198] types of formalisms are probably the most researched and best suited ones for modeling context. The standardized languages support reuse and sharing, definition of taxonomies and relationships, as well as different reasoners.

Other types of formalisms discussed are process-oriented [138], spatial models [21], topic maps [198], rule-based models [32], or hybrids [21].

Context models proposed in literature are often specific to certain domains [207]. To overcome this problem Bradley and Dunlop [29] proposed a multidisciplinary model although they encountered difficulties in representing context in a single model covering the viewpoints and interpretations of the various disciplines. *Generic or meta context models* are an interesting approach to this problem [191] although practicability and usability decrease with the generality of the model [24]. Context meta models form the basis for specific context models [14, 198]. Using this basis, designers are supported in deciding which context knowledge to include for a particular CASA [14, 198], i.e., they are supported in developing a specific context model based on the meta model. The development of a specific context model can be conducted in an incremental fashion. Bauer [14] analyzed 13 context meta models, identifying 3,742 distinct context parameters. They were categorized into physical world, individual, social groups, activity, technology, and change over time. About 20 % of the identified context parameters could not be assigned to any of the six categories, for example, confidentiality, usefulness, or quality aspects. An issue with meta-models is that it is difficult for designers to identify context parameters relevant to a specific CASA from such generic categories [14].

Independent of the previous aspects, we distinguish two *levels of abstraction*. Since a context model is used to represent an explicit and coherent picture of contexts in a specific CASA, a context model can represent the actual implementation in the CASA or the conceptual idea of context employed in the CASA.

TABLE 2.5: Summary of operations on context discussed in [18].

Operation	Options
Localized reasoning	Refer to Table 2.3
Push/Pop	Relevant / irrelevant
Shifting	Relevant / irrelevant

2.5.4 Operations on Contexts

A high-level classification of context operations identified in literature is depicted in Table 2.5. Benerecetti, Bouquet, and Ghidini [18] analyzed proposed mechanisms for contextual reasoning and classified them into three basic forms:

Local reasoning [18] describes reasoning local to a context where also references to knowledge in other contexts are taken into account. To this end, contexts must be entered, i.e., their assumptions must be assumed [127]. Once a context is entered, inferences can be made which then hold within this context. Leaving a context means discharging the assumptions of the context [127].

Push [18] moves contextualized knowledge to context knowledge, i.e., it adds a parameter to the box and removes the corresponding contextualized knowledge encoded inside the box. *Pop* [18] is the inverse operation where a parameter is moved inside the box to contextualized knowledge. Entering/leaving contexts by McCarthy can be seen as instances of push/pop [18].

The basic idea of *shifting* [18] is that changes in the parameter values of a context shift the interpretation of the contextualized knowledge inside the box. For instance, we know the sun is shining today. This knowledge is contained in a context having a parameter with today's date. Shifting the context's parameter value to tomorrow's date the contextualized knowledge will be yesterday the sun was shining. Thus, shifting requires knowledge about the relationships between parameter values. Other examples include viewpoint changes, categorizations, or the lifting mechanism introduced by McCarthy [18].

The three basic forms of context operations fit the three dimensions along which context dependent representation may differ according to Benerecetti, Bouquet, and Ghidini [18]: localized reasoning supports partial representations, push and pop support varying degrees of abstraction in representations, and shifting supports different perspectives in representations.

2.6 Context-Aware Systems or Applications (CASAs) and their Development and Deployment

A system or application is called *context-aware* if "it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [57, p. 5]. However, since there is no consensus on context, the term context-awareness is fuzzy, for example, context-awareness of humans radically differs from context-awareness of machines [3]. Nevertheless, an interpretation often encountered is that Context-Aware Systems or Applications (CASAs) can derive/sense the context, i.e. situational information, of their use and adapt/react accordingly without explicit user intervention [13, 57, 153, 155]. By this interpretation, our proposed CBRs are CASAs: business cases contain information, CBRs derive relevant contexts from this

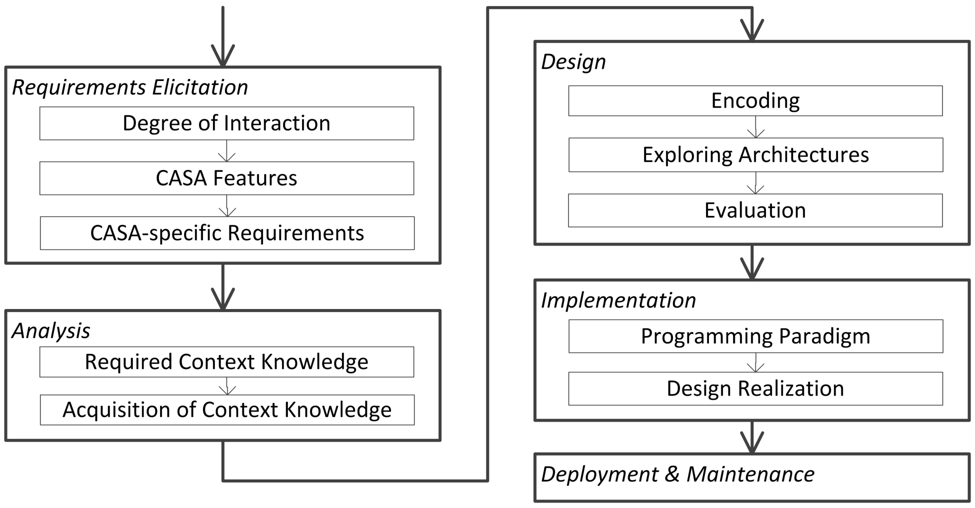


FIGURE 2.3: CASA Development and Deployment according to [3].

information, and utilizing the derived contexts provide relevant business rules only. Kirsch-Pinheiro, Mazo, Souveyet, and Sprovieri [105] define the broader term context-oriented systems as any system exploring contexts in their execution behavior in any way.

CASAs may employ context models (Section 2.5.3) to explicitly and coherently represent their context understanding, context components, relationships between context components, knowledge sources, et cetera. Since CASAs are a specific type of systems, their development and deployment should follow the common phases for system development and deployment.

Alegre, Augusto, and Clark [3] analyzed the support for the most common phases of a development process – requirements elicitation, analysis, design, implementation, and deployment and maintenance – in the development of CASAs and found only little support. Below we describe the development phases of CASAs based on [3], focusing on aspects relevant to CASA development only. The CASA-specific process is depicted in depicted in Figure 2.3; a summary of relevant aspects for the process is given in Table 2.6.

Requirements Elicitation An important decision regarding requirements is the intended *degree of interaction* with the CASA influencing other requirements. The purpose/goal of the intended CASA needs to be determined, in particular, the purpose of context knowledge, for example, adaption purposes, information characterization, or decision-making [105, 175]. Alegre, Augusto, and Clark [3] identify two degrees of interaction: active and passive. Active describes systems which autonomously conduct executions or configurations (see below) while passive systems require user involvement [3, 50, 60].

Going hand in hand with the degree of interaction are the required *features* of a CASA. For instance, Lenat [115] identifies modification of context knowledge and determining the context of a given context knowledge piece as features. Others analyze past research and propose quite similar categorizations of features [2, 149, 152, 178]. Alegre, Augusto, and Clark [3] identify four CASA-features:

1. Context-dependent *presentation* of information and services to various stakeholders [2, 3, 32, 57, 60, 105, 133, 152, 155, 175, 178]

TABLE 2.6: Summary of aspects identified for the development of CASAs.

Aspect	**Options**
Requirements Elicitation	
Degree of interaction	Active / passive
Features	Presentation / execution / configuration / tagging
Uncertainty handling	Relevant / irrelevant
Context history	Relevant / irrelevant
Versioning	Relevant / irrelevant
Context prediction	Relevant / irrelevant
Analysis	
Acquisition	Sensed / derived / manual
Dissemination	Query / subscription / none
Type of devices	Virtual / physical / combined
Interpreting raw data	Relevant / irrelevant
Design architecture	Middleware / client-server / ... / layered
Implement. paradigm(s)	Object-oriented / ... / model-driven

2. Context-dependent *execution* of services (active or passive) [3, 15, 31, 32, 57, 60, 133, 152, 155, 175, 178]

3. Context-dependent *configuration* of services (active or passive) [3, 15, 31, 32, 60, 105, 152, 175, 178]

4. Contextual *augmentation/tagging*: augmentation of digital data with context knowledge [2, 3, 32, 57, 152, 155]

Once the expected features and degree of interaction are determined, further requirements need to be elicited. We distinguish general and CASA-specific requirements. General CASA requirements we identified include, amongst others, robustness and adaptability [133, 138, 198], efficiency [21, 138], scalability and extensibility [3, 13, 132, 155], reusability [13, 115, 132, 183], and privacy/security [13, 138, 155]. Regarding CASA-specific requirements we identified the application-dependent requirements uncertainty handling [21, 24, 85, 138, 191], access to past contexts [13, 21], versioning, predicting future contexts [13, 21], and obligatory requirements such as usability of the context modeling mechanism employed [21], support of heterogeneous and mobile context knowledge sources [21], and understandability [133, 138]. Further CASA-specific requirements may be elicited using a number of different requirements elicitation techniques specialized for CASA development [3].

Uncertainty handling in CASAs may be necessary if context is derived or sensed. Often, the sources considered for context sensing/derivation are sensors and measurements which may not be 100 % accurate [85, 123, 155]. Thus, depending on the use case, it may be necessary to explicitly consider their quality to achieve good performance of CASAs [123]. This can be achieved by providing Quality of Context (QoC) metrics, such as reliability, timeliness, completeness, or significance, with sensed and measured values [123, 155]. CASAs may utilize the knowledge about QoC to improve their efficiency and effectiveness [123] or to perform validation and conflict resolution on derived/sensed context(s), for example, values out of range or two sensors report conflicting values [155].

Analysis Once the requirements are elicited, the *necessary context knowledge* needs to be identified [3, 105, 175] for the use case at hand [57, 105]. For instance, using the box metaphor, the relevant context parameters and parameter values need to be identified. To this end, decisions regarding the aspects in Tables 2.1 to 2.4 are required.

Once the context knowledge is identified, the *acquisition* of context knowledge needs to be decided, i.e., how to determine the context knowledge and how provide it to the CASA [24, 105]. To this end, observing devices (such as sensors) and their management need to be identified and defined [13, 132]. Regarding the acquisition of context knowledge various aspects need to be considered [21, 138, 155]: is context knowledge sensed, derived, or manually provided? Is context knowledge *disseminated*? Is it disseminated on a subscription basis or does it have to be queried [3, 5, 21, 155, 207]? Is the *type of device* physical, virtual, or combined? What is the frequency of knowledge acquisition?

The context knowledge acquired by sensors may be quite fine-granular or error-prone. Therefore, reasoning or *interpretation* may be used to process raw data acquired by observing devices [13, 15, 18, 21, 24, 26, 31, 85, 93, 138, 155, 157, 207]. Technologies employed to this end include supervised and unsupervised machine learning, rules, fuzzy logic, probabilistic logic, ontological technologies, or any combination thereof [155]. Whether such reasoning is necessary depends on the use case.

Design Designing a CASA based on elicited requirements and analysis comprises (based on [3]): encoding, defining the behavior of the CASA; unifying, exploring the design space for possible designs including context and context models; and evaluation, choosing a solution satisfying identified requirements and constraints. An essential step of design is the choice of a high-level system *architecture* [105]. Architectures employed for CASAs are for instance blackboard model (publish-subscribe), middleware, distributed, networked services, centralized, or client-server, where middleware is the most common architecture for CASAs [3, 155]. Besides these specific architectures, a generic architecture for CASAs has been proposed [13, 132]. This generic architecture divides functionalities into layers [13, 50, 132, 155]: observation devices, acquisition, preprocessing (interpretation/reasoning), storage/management of acquired context knowledge, and application. CASAs adapting this generic architecture may not use all layers, for instance, middleware solutions usually do not provide an application layer.

CASA Implementation Once requirement elicitation, analysis, and design are accomplished, the CASA needs to be implemented. For this purpose, a suitable programming paradigm or paradigm combination needs to be chosen, for instance, object-, aspect-, feature-, service-, agent-, context-oriented, or model-driven [3]. Using the generic architecture discussed above, this decision may be different for each layer. Once the programming paradigm(s) is/are defined, the observing devices, any required interpretation mechanisms, the context model using the chosen formalism and the vocabulary designed, storage/management, and dissemination/application are implemented [3].

Deployment and Maintenance Following the implementation of a CASA is its deployment and continuous maintenance [3]. Usually this phase is accompanied by such activities as user training or documentation. Maintenance describes modifications to a system to correct errors or to improve the system in various regards after its delivery [3]. Lientz [117] identified four distinct maintenance activities: corrective, adaptive, perfective, and preventive.

2.7 Conclusion

In this chapter we investigated the two key-concepts for CBRs, namely, business rules and contexts. Therefore, we discussed literature regarding business rules and contexts and identified important aspects and attributes for both. Furthermore, we gathered information about business rule organization, management of business rules, as well as the particularities of CASA development and deployment.

The identified aspects and attributes as well as the information about business rule organization and management of business rules are used to determine our context understanding and our context model for business rule organization along contexts in Chapter 3. Based on our context model, we develop several prototypes for CBRs in Chapter 4. For these prototypes, we oriented ourselves on the development and deployment process of CASAs.

Chapter 3

The Generic CBR Model: Modeling Context for Business Rule Organization

Many companies are confronted with an increasingly large number of business rules. Consequently, means for efficient and effective management of these business rules are necessary. However, Business Rule Management Systems (BRMS) frequently provide only simple business rule organization techniques such as collecting business rules into rule sets according to a single criterion. Many research fields organize information by contexts which may be hierarchically structured. Similarly, we propose a context model organizing business rules along their contexts – the generic CBR model. We call repositories realizing the generic CBR model or concretized CBR models for organizing business rules CBRs.

We determine and argue the appropriate choices for a context model organizing business rules along contexts. Based on these choices we present the generic CBR model for business rule organization that can be instantiated multi-level: for a specific domain and for a concrete application. We demonstrate the usefulness of the generic CBR model by relating it to a use case in the aeronautical domain, the semantic filtering of digital Notices to Airmen. This real world use case addressed in the SemNOTAM project involves thousands of business rules. These business rules need to be organized along multiple hierarchical context parameters (for example, aircraft type and mode of operation) such that they become manageable with regard to definition, maintenance, extension, as well as with regard to determination of the business rules relevant for a specific business case, for example, a flight plan of a pilot.

3.1 Introduction

In Section 1.1 and 1.2 we motivated the need for context-based organization of business rules. We propose CBRs building on contexts and reflecting situation-dependency in the organization of business rules. The organization within a CBR is modeled by a generic context model – the generic CBR model which can be instantiated multi-level to a specific domain and subsequently to a specific application. An overview is given in Figure 1.3 in Section 1.5. Regarding this overview, the generic CBR model mainly considers contexts, rule sets, and business cases; data sets are assumed to be integrated with business cases. Rule sets are applied to business cases.

This chapter is mainly based on the publications F. Burgstaller, D. Steiner, and M. Schrefl. "Modeling Context for Business Rule Management". In: *2016 IEEE 18th Conference on Business Informatics (CBI)*. 2016, pp. 262–271 and F. Burgstaller. "Employing Aviation-Specific Contexts for Business Rules and Business Vocabularies Management in SemNOTAM". in: *On the Move to Meaningful Internet Systems: OTM 2016 Workshops*. 2017, pp. 315–325

TABLE 3.1: The interpretation of context (i.e. its notion and objective) adopted for the generic CBR model.

Aspect	Chosen Option
Notion of context	Representational
Objective	Separate concerns

Employing the generic CBR model to organize business rules promises several advantages such as faster business rule entering, smaller business rule sets for execution, reduced redundancy through context hierarchies, and easier browsing. Further advantages regarding business rules include increased readability of business rules definitions as the class of situations does not need to be encoded, automatic construction of a business rule set containing all relevant business rules given a business case, and easier debugging. These benefits support our claims stated in Section 1.5: (1) efficient and effective maintenance of business rules, (2) increased performance of business rule execution, (3) faster search for business rules of interest, and (4) support for executable as well as non-executable business rules. In order to develop the generic CBR model, we employ the design-science paradigm, described in Section 1.3, and requirements elicitation and analysis of the CASA development process, presented in Section 2.6.

The remainder of this chapter is organized as follows: Section 3.2 builds on Chapter 2 to determine and argue our choices for the generic CBR model organizing business rules through contexts. Section 3.3 presents the generic CBR model implementing the choices and demonstrates its usefulness by instantiating it for SemNOTAM. Section 3.4 discusses the generic CBR model regarding our claims stated in Section 1.5 and the background reviewed in Chapter 2. Section 3.5 discusses related work and Section 3.6 concludes the chapter.

3.2 Context Model Requirements for Business Rule Organization

Before we develop the generic CBR model, we need to determine our understanding of context [24] and clarify the requirements for business rule organization. For this purpose, we choose for the aspects identified for contexts in Section 2.5 and for CASAs in Section 2.6 the options relevant to context-based organization of business rules based on the background review for business rules in Section 2.2, for business rule organization in Section 2.3, and for management of business rules in Section 2.4. In the following, we delineate our choices.

3.2.1 Interpretations of Context

In Section 2.5.1 we discussed two distinct notions of context and two modularization approaches (summarized in Tables 2.1 and 2.2). In the following, we delineate our choice regarding the notion of context we employ for the generic CBR model as well as the modularization approach we adopt. A summary is given in Table 3.1.

Considering the *notion of context* (summarized in Table 2.2) we find: Business rules are to be separated from business processes and procedures as well as their execution. Furthermore, they are to be explicitly and declaratively defined. Consequently, business rules are considered pieces of knowledge or information. Since business rules are employed in organizations which pursue specific goals, the number of situations/contexts in which they may apply can be enumerated, i.e., contexts can be regarded delimited. Nevertheless, the specific contexts as well as their number may change over time. The execution of business rules happens within contexts and may modify the current situation and consequently other contexts may become relevant. Hence, contexts for business rules have some dynamic aspects. Consequently, the relevant notion of context for business rules organization is representational context although also some aspects of interactional context are of interest. This also fits nicely with the context-of-a-box metaphor assumed to discuss components of context in Section 2.5.2.

Regarding the *objective* of contexts, we know that business rules are situation-dependent, i.e., business rules usually apply in different classes of situations. Furthermore, business rules applying in different classes of situations do not need to be integrated to achieve the expected results. Consequently, the objective of contexts regarding business rule organization is to separate concerns.

3.2.2 Components of Context

In Section 2.5.2 we present, assuming the context-as-a-box metaphor, the context components and their aspects we have identified in the literature (summarized in Table 2.3). In the following, we delineate our decisions regarding context components and their aspects for the generic CBR model; a summary is given in Table 3.2.

In our case, *contextualized knowledge* are sets of business rules. Since businesses need to constantly adapt to changes in areas such as markets, technologies, legislations, or governance, so need their business rules and business rule sets. Consequently, *volatility* of contextualized knowledge usually is medium to high, for example, in some particular insurance company rules for insurance products change 35 times a week [208]. *Referencing context knowledge*, such as a business rule or a term, in another context may be useful in certain cases. Nevertheless, for such cases other means, most importantly, inheritance, are available and consequently referencing is not required. Most business rule engines employ a single mode of *reasoning*. Different reasoning modes in contexts would complicate the accountability as well as explanation of results, especially if inheritance of contexts is supported as well. Thus, the mode of reasoning should be uniform across all contexts. Regarding *conflict resolution* the concrete choice depends on the use case at hand.

Regarding *contexts* as organizing units of business rules, frequent changes to the set of contexts are probable due to the volatility of business rules. Thus, the *set of contexts* is rather dynamic. In order to be able to represent the hierarchical organization of businesses and specialization hierarchies of classes of situations, we employ a hierarchy *relationship* between contexts. Further kinds of relationships may be useful depending on the use case. Business rules established by higher levels in a business organization usually apply for all lower levels as well. Furthermore, business rules applying for a general class of situations apply in their specific classes of situations as well. Consequently, and to foster business rule reuse as well as simplifying adaptation [36], we employ an inheritance *semantics* for the context hierarchy relationship. Similar semantics are employed in other approaches where child rule sets inherit from their parent rule sets [1, 4, 150]. Other relationships may have custom semantics appropriate for the use case at hand. Although *context classes* may be useful in certain

TABLE 3.2: Options chosen for the aspects of context components in the generic CBR model.

Aspect	**Chosen Option**
Contextualized Knowledge	
Volatility	High & medium
Referencing contextualized knowledge	Irrelevant
Reasoning	Uniform
Conflict resolution	Use case dependent
Contexts	
Set of contexts	Dynamic
Context relationships	Related
Semantics of context relationships	Inheritance & other
Context Classes	Irrelevant
Context Parameters	
Parameters of context	Uniform
Set of parameters	Dynamic
Parameter relationships	Independent
Semantics of parameter relationships	Irrelevant
Type of parameter	Primary & secondary
Parameter Values	
Set of parameter values	Dynamic
Value relationships	Related
Semantics of value relationships	Subsumption & other

cases we do not consider them essential for CBRs. If needed, they can be easily introduced as they are an orthogonal concept to context inheritance.

Employing the context-as-a-box metaphor, each context is described by a number of *parameters*. To simplify repository organization and maintenance, we choose *parameters of contexts* to be uniform, i.e., each context is described by the same set of parameters. For the same reason, and because parameters are rarely related, we consider different parameters to be independent of each other, i.e., they do not have *relationships* with each other. Consequently, we regard *semantics* for parameter relationships irrelevant. Similar to contextualized knowledge and its organizing contexts, the *set of parameters* may vary, i.e., it is dynamic. If the set of parameters is modified, it is modified for all contexts. Regarding the *type of parameter* both primary and secondary parameters are relevant.

Each parameter has a set of *parameter values* from which it can, for a particular context, assume values. Analogously to contexts and parameters, the *set of parameter values* is dynamic. Regarding *relationships of parameter values*, most parameter values form hierarchies, for example, the organizational units in a business. These hierarchies usually have a subsumption *semantics*, for instance, a marketing department includes public relations and market research. Other relationships of parameter values with custom semantics may be necessary depending on the use case.

TABLE 3.3: Options chosen regarding the aspects identified in Table 2.4 for modeling context in the generic CBR model.

Aspect	**Chosen Option**
Type of formalism	Object-oriented
Graphical representation	Relevant
Meta/Generic model	Relevant
Level of abstraction	Conceptual

3.2.3 Modeling Context

Context modeling is vital to effective and efficient organization of business rules in a CBR as the context model will be used to present the rule organization strategy to end users. In Section 2.5.3 we delineate challenges and requirements of context modeling, describe various types of formalisms for modeling context, and discuss further aspects of context modeling (summarized in Table 2.4). In the following, we discuss our choices regarding the modeling of context in the generic CBR model. A summary of our choices is given in Table 3.3.

Even though ontological *types of formalisms* are the most researched and recommended ones, we opted for an object-oriented formalism. We do so as the context nature being representational suggests to model contexts as objects with attributes, the context-as-a-box metaphor fits nicely with object-orientation, and useful object-oriented concepts, such as encapsulation, inheritance, reusablility, and interfaces, are available. Encapsulation is essential since the objective of our contexts is to separate concerns, i.e., each context is an individual unit containing business rules. Inheritance is useful as we want to model inheritance of contexts promoting reuse of business rules. Since a CBR interfaces other applications, the concept of interfaces is helpful. Other advantages of object-orientation are its wide acceptance and usage as well as the modeling formalisms and graphical representations available.

The aim of our context model is to represent the contextual organization of business rules. Since a picture is worth a thousand words, we decide for a *graphical* context model. This is supported by the many graphical modeling languages available for object-orientation, such as UML class diagrams. Furthermore, a graphical model may be directly employed to present the organization structure to end users.

Regarding the remaining two aspects, we decide for a *meta* and *conceptual* model. The meta-model allows us to present our underlying abstract idea of contexts. This model can be instantiated (adapted) to models of specific use cases which in turn can be instantiated to applications. Since our model is a conceptual one, it is independent of a concrete implementation and thus is not overloaded with implementation details. Furthermore, different use cases may require different implementations depending on the technologies and systems available in the business as well as the use case itself.

3.2.4 Operations on Contexts

In Section 2.5.4 we identified and described a high-level classification of context operations (summarized in Table 2.5). In the following, we discuss our choices regarding context operations relevant to the generic CBR model. A summary of these choices is given in Table 3.4. Localized reasoning is irrelevant as discussed in Section 3.2.2.

TABLE 3.4: Relevance of context operations for the generic CBR model.

Operation	Chosen Option
Localized reasoning	Irrelevant
Push/Pop	Relevant
Shifting	Relevant

Push and pop are essential context operations for organizing business rules in CBRs. Reorganization operations like adding and removing parameters from the context model require push and pop. Adding and removing parameters without push and pop respectively, the contextualized knowledge would apply in a different class of situations. Furthermore, business rule statements should be self-explanatory, i.e., every statement can be taken at face value (even if taken out of context) [146, 167]. Consequently, a context operation for decontextualization, namely pop, is required.

Shifting is, like push and pop, a necessary context operation for business rule organization in a CBR. Reorganization operations often require to move statements within the context hierarchy, for example, when splitting a context. Furthermore, if parameter values are arranged in trees, an alternative implementation of decontextualization is to shift a context's parameter values to the corresponding root parameter values.

3.2.5 CASAs and their Development and Deployment

In Section 2.6 we discussed CASAs and their development phases (summarized in Figure 2.3 and Table 2.6). Contextualized Business Rule Repositories (CBRs) are CASAs which realize the generic CBR model or a concretized CBR model. CBRs need to, first and foremost, fulfill the claims identified in Section 1.5. Furthermore, for a given business case relevant contexts and consequently business rules need to be determined and provided to the user. The actual execution of the business rules is not necessarily part of a CBR but can be performed externally. Definitely required of CBRs are means to add, eliminate, and modify contexts and their business rules, for instance, addition of business rules to specific contexts must be supported.

In the following, we delineate our decisions for CASA aspects regarding the generic CBR model. Since we design a conceptual and thus implementation-independent model, we consider development phases requirements elicitation and analysis only, as these may influence the generic CBR model; the implementation- and use-case-dependent phases design, implementation, and deployment are not considered. A summary of our decisions is given in Table 3.5.

Considering the requirements for a CBR above, the *degree of interaction* to be supported by the generic CBR model is active as relevant contexts and business rules should be determined automatically for business cases provided. The CASA-*features* we derive from the above requirements are: presentation, due to required determination of relevant business rules for given business cases, and tagging, due to required management and maintenance of business rules. Whether execution is a feature or not depends on the concrete implementation, i.e., if the CBR has a built-in rule engine executing determined relevant business rules, execution is featured. Whether *uncertainty handling*, *context history*, *versioning*, and *context prediction* are

TABLE 3.5: CASA development options relevant to the generic CBR model.

Aspect	Chosen Option
Requirements Elicitation	
Degree of interaction	Active
Features	Presentation & tagging & (execution)
Uncertainty handling	Use case dependent
Context history	Use case dependent
Versioning	Use case dependent
Context prediction	Use case dependent
Analysis	
Acquisition	Derived
Dissemination	Query
Type of devices	Virtual
Interpreting raw data	Relevant

necessary depends on the specific use case. These aspects are not considered in the generic CBR model.

We distinguish context knowledge from knowledge about business cases. Relevant context knowledge, i.e., the relevant classes of situations, are derived from knowledge about a provided business case. Thus, context knowledge is *acquired* by derivation from knowledge about business cases. Furthermore, since business cases are provided to a CBR and the CBR returns relevant business rules and contexts, *dissemination* is by querying. As business cases are an artificial knowledge object/unit, the *type of device* is virtual. Since we derive higher-level context knowledge from knowledge about specific business cases, CBRs need to support *interpretation of raw data*.

3.3 Modeling Context for Business Rule Organization

In this section we present the generic and conceptual (implementation independent) CBR model and its instantiation levels based on the choices made in the previous section. This conceptual model can be realized in various ways, for example, using knowledge-based or expert system techniques where a concretized CBR model describes the organization of the knowledge base and a reasoning engine is used for context determination and business rule execution.

We decided, see Section 3.2.3, to employ an object-oriented and graphical formalism for the conceptual and generic CBR model. A widely used graphical modeling language for object-oriented and conceptual modeling is UML, in particular, UML class diagrams. Since UML class diagrams are also used to represent meta-models [7, 54], they are suitable to represent the generic CBR model, a meta-model for business rule organization in CBRs. Employing standard UML class diagrams to conceptually model contexts, limitations regarding the adaptability of the resulting context model arise. To overcome these limitations, non-strict multi-level modeling adapting De Lara [54] is employed, resulting, adhering to the design-science process model presented in Section 1.3, in the three levels summarized in Table 3.6.

TABLE 3.6: The three levels of CBR models for context-aware organization of business rules.

Level	Model
M2	Generic context model for business rule organization
M1	Domain-specific context model for business rule organization
M0	Application-specific contexts, business rules, and business cases

Level M2 defines the generic CBR model independent of any application domain; it is a meta context model for context-aware business rule organization. As such, the generic CBR model describes the realization of all choices made in the previous section, in particular, the realization of the purposes presentation and tagging. It describes contexts, parameters, and business cases. Models of level M1 instantiate the generic CBR model regarding an application domain and thus constrain and refine the generic CBR model, that is, they define the parameters describing the domain-specific contexts, restrict available relationships, and define the business case classes relevant in the domain. Additionally, the semantics of relationships as well as the semantics of dynamism, such as the consequences of removing a parameter, need to be specified. Models of level M0 instantiate domain-specific CBR models and thus contain concrete contexts and their business rules as well as concrete business cases.

In the following, we delineate multi-level modeling and subsequently detail the three levels of an CBR model, employing SemNOTAM introduced in Section 1.4 as use case.

3.3.1 Multi-level Modeling

A multitude of multi-level modeling approaches exists in the literature [42]. The common idea of multi-level modeling approaches is to allow arbitrary numbers of meta-levels when modeling a problem [54]. Consequently, model elements have a dual aspect as types and instances. They are instances to the meta-level above and types to the meta-level below and hence are often called clabjects [7]. For modeling such clabjects, meta- as well as instance-modeling facilities can be employed [54]. In certain situations, multi-level models are simpler than two-level modelling. Furthermore, multi-level models are a promising technology for domain-specific modeling languages [54].

De Lara, Guerra, and Cuadrado [54] distinguish two combinable techniques regarding multi-level modeling: potency-based multi-level modeling [7, 9] and orthogonal classification architecture [8, 10]. *Potency-based multi-level modeling* identifies the need to control the instantiation depth of model elements and their features. This is necessary as model elements and features may not be instantiated at the adjacent level below but at any level below. To this end, the concept of potency is introduced. A potency for a model element specifies the number of levels until it is ultimately instantiated. As such it is decremented by one at each level until it reaches zero and can be instantiated; it cannot be instantiated at any level further below. Various notations for potencies exist [9, 54], we employ a superscript notation, for example, $\texttt{resolved}^2$ in the generic CBR model in Figure 3.2 indicates that `resolved` is instantiated two levels below on the application level. To ease readability, intermediate model elements having a potency greater than zero, may be hidden [54].

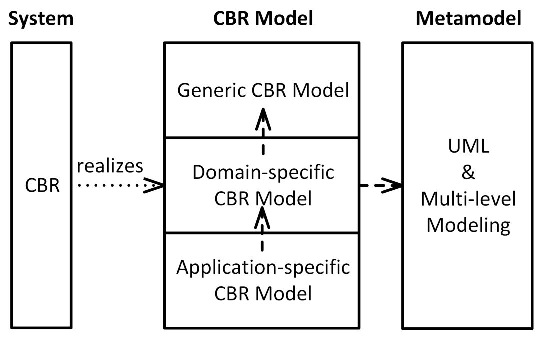

System
CBR Model
Metamodel
CBR
realizes
Generic CBR Model
Domain-specific CBR Model
Application-specific CBR Model
UML & Multi-level Modeling

FIGURE 3.1: Overview of the orthogonal classification architecture for CBR models drawing on Atkinson and Kühne [11].

Orthogonal classification architecture distinguishes ontological and linguistic typing of model elements. Ontological typing refers to instantiation of a model element in its domain, for example, the ontological type of Felix the Cat is cat. Linguistic typing on the other hand refers to the meta-modeling facility employed to construct the model element, for example, the linguistic type of cat is class.

In addition to basic multi-level modeling, we employ the relation configurator pattern [54]. The relation configurator pattern is useful if on-demand redefinition of reference types which can be instantiated requires to: (a) configure the cardinality or the (b) referenced class of the redefined reference. An example for the configurator pattern can be found in Figure 3.3 where association `interest` redefines association `defBy` to cardinality one and the class `Interest`.

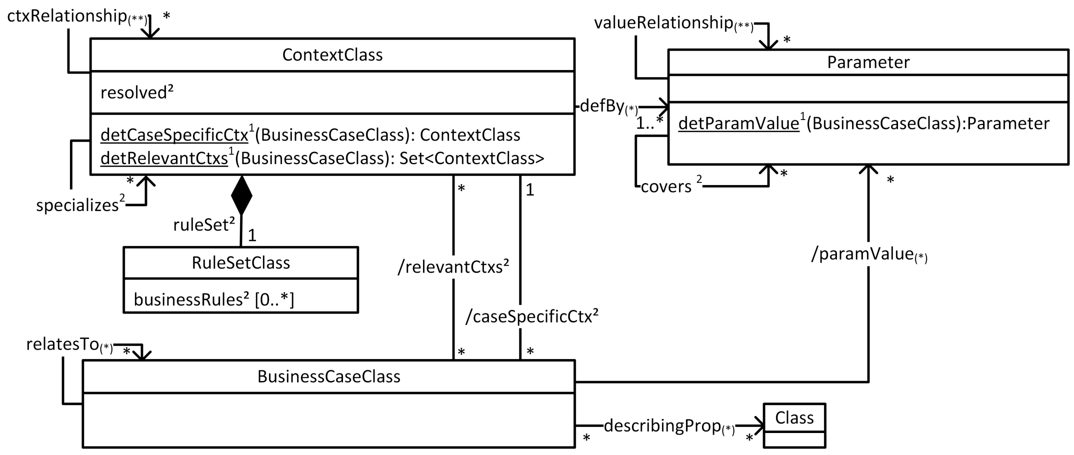
Furthermore, we extend potencies to methods, i.e., each method can be assigned a potency defining at which level the method has to be implemented. Furthermore, the parameter types of methods are to be refined to instances of those types on the lower levels. For example, `detCaseSpecificCtx`^{1}`(BusinessCaseClass):ContextClass` in Figure 3.2 is implemented on the next level, depicted in Figure 3.3. Its parameter types are refined to `SemNOTAMCase` and `AIMCtx` respectively.

For the generic CBR model we utilize orthogonal classification architecture as well as potency-based multi-level modeling as depicted in Figure 3.1. The CBR model employs ontological typing for its three levels presented in Table 3.6. This ontological typing employs potency-based multi-level modeling. The model elements of CBR models are constructed by linguistic typing of meta-model elements for UML and multi-level modeling (as described above), for example, typing contexts as UML classes. A system realizing a fully-instantiated conceptual CBR model is called a CBR.

3.3.2 Generic Model (M2)

The generic CBR model is designed to support the choices made in Section 3.2. The choices regarding the interpretation of context, detailed in Section 3.2.1, are modeled by considering context an object described by parameter values for its set of parameters and relating relevant business rules. Parameter values specify when the context is relevant, i.e., the class of situations in which the contained business rules apply. Thus, the parameter values also determine whether a context is relevant for a business case.

These objects, contexts, parameters, parameter values, and business cases, are modeled by the classes `ContextClass`, `Parameter`, and `BusinessCaseClass` in the generic



(*) Needs to be configured on the next level using the relation configurator pattern.
(**) Needs to be reflexively configured on the next level, e.g., :ctxRelationship needs to be a reflexive relationship of AIMCtx.

FIGURE 3.2: The generic CBR model for context-aware organization of business rules.

CBR model (Figure 3.2); parameter values are not represented by a separate class at level M2 but instantiate domain-specific parameter classes at level M0. The set of contexts, parameters, parameter values, and business cases can be modified by utilizing object-orientation to construct or deconstruct instances. Depending on the concrete implementation, other features may be used. A detailed discussion of operations modifying these sets, as well as context operations supported, is given in Chapter 5.

Each specific `ContextClass`, not to be confused with context classes by Homola, Serafini, and Tamilin [93], defines a set of specific `Parameters` describing it by employing the relationship `defBy`, i.e., all contexts instantiating the same `ContextClass` are described by the same set of `Parameters`. Each context instance at level M0 specifies parameter values for its parameters modeled by the relationship `defBy`. The attribute `resolved` is true if it is a context for which the semantics of any relationships as well as potentially necessary conflict resolutions have been applied (a case-specific context); otherwise it is false. These properties, `defBy` and `resolved`, uniquely identify a context. Each context further specifies a `RuleSet` containing business rules (`businessRules`) relevant to it. Since we do not support heterogeneous reasoning nor referencing context knowledge in other contexts, the class `RuleSet` is sufficient as modeled in Figure 3.2. Relationships between contexts, for instance, relationships of hierarchical nature, inclusion relationships, or relationships indicating similarity, can exist. These are modeled by the relationships `ctxRelationship` and `specializes` respectively. Since `specializes` is considered an important relationship, it is modeled explicitly in the generic CBR model. Similarly, the hierarchical relationship of parameter values, `covers`, is modeled explicitly. Other parameter value relationships configure `valueRelationship` at level M1. Since we assume that `Parameters` are independent of each other, we do not model any relationship between parameters.

Each business case (`BusinessCaseClass`) specifies properties describing it, indicated by the association `describingProp`. These properties are used to derive the parameter values of a business case regarding `Parameters`, using the respective `detParamValue` method. These derived parameter values are depicted by the relationship `/paramValues` between `BusinessCaseClass` and `Parameter`. Based on the derived associations `/paramValues`, the relevant contexts for a business case are determined using method `detRelevantCtxs`. The set of relevant contexts for a business case is represented by the derived relationship `/relevantCtxs` between `BusinessCaseClass` and `ContextClass`. From these relevant contexts, a case-specific context is created using method `detCaseSpecificCtx`, resolving inheritance relations, other context relationships, and potential conflicts between all relevant contexts (see below). The case-specific context is referred to by the derived relationship `/caseSpecificCtx`.

The described derived relationships are determined by methods, in particular class methods. Class methods are employed as we want to enable domain-specific implementations as well as domain-specific parameter types for our methods. Due to these restrictions methods cannot be implemented at level M2. Since methods can only be defined at class level, we need to model them as class methods at level M1. The class method `detParamValue` determines the parameter values of a business case for the corresponding `Parameter`. Implementations of `detParamValue` range from directly returning a business case property value to complex derivations over a set of business case properties. Thus, we support primary parameters (direct measurements) and secondary parameters (derived measurements) as required in Section 3.2.2. The class method `detRelevantCtxs` of `ContextClass` uses `/paramValues` to determine the set of contexts relevant to a given business case. The concrete implementation depends on the use case. Class method `detCaseSpecificCtx` computes, for a given business case, a single case-specific context for which all relationship semantics and conflict resolution strategies have been applied and the attribute `resolved` is set `TRUE`. To this end, it employs method `detRelevantCtxs` to determine the relevant contexts, combines them into a case-specific context, and performs necessary conflict resolutions. The concrete combination algorithm depends on the use case, for instance, specific inheritance semantics may be required. Analogously, the conflict resolution is domain-specific, for example, to resolve the conflict of a NOTAM being rendered highly important by a business rule and irrelevant by another business rule can be resolved by reporting highly important to prevent omission of important NOTAMs.

In Section 3.2.5 we discussed CASA development options relevant to the generic CBR model. In particular, we considered requirements elicitation and analysis. Regarding requirement elicitation, the required active degree of interaction is supported by the methods `detCaseSpecificCtx` and `detRelevantCtxs` which automatically determine relevant business rules and contexts respectively for a given business case. The required CASA features to be supported by the generic CBR model are presentation, tagging, and optionally execution. The methods `detCaseSpecificCtx` and `detRelevantCtxs` and the corresponding derived relationships `/caseSpecificCtx` and `/relevantCtxs` enable CASA-feature presentation. The support of CASA-feature tagging is discussed in Chapter 5. If CASA-feature execution is to be supported, the case-specific context has to be executed on the concrete business case potentially resulting in derived properties for the business case (not modelled). Regarding analysis of CASAs, the required acquisition of context knowledge by derivation, the virtual type of device for sensing context knowledge, and the interpretation of raw data are supported by the class method `detParamValue` of `Parameter`. The required dissemination by querying is given by the methods `detCaseSpecificCtx` and `detRelevantCtxs` which return for a given business case the case-specific context or the relevant contexts respectively.

3.3.3 Domain-specific Model (M1)

Any domain-specific CBR model constrains and refines the generic CBR model by instantiating it regarding a domain. Relevant context relationships other than `specializes` need to be defined using relation configuration on `ctxRelationship`. Domain-specific parameters are instantiated from `Parameter` requiring appropriate relationship configuration of `defBy`, such as multiplicities of one, to ensure that each context is described by the same parameters. Additional relationships between parameter values besides `covers` can be defined by configuring relationship `valueRelationship`. Besides context and parameter instantiations, the methods' signatures need to be refined and their logic defined. Similarly to `ContextClass` and `Parameter`, the business case class needs to be refined by instantiation. This includes definition of business case properties describing the domain-specific business case, i.e., configuring relationship `describingProp`. The presented generic model supports dynamic sets of contexts, parameters, and parameter values by instantiation; a detailed description is given in Chapter 5.

Besides structural aspects, the methods `detRelevantCtxs` and `detCaseSpecificCtx` need to be defined. While the implementations of `detRelevantCtxs` will not vary much across domains, the implementation of `detCaseSpecificCtx` may. In most cases, implementations of `detRelevantCtxs` will look for the context(s) whose parameter values are identical to or are the closest ancestor parameter values to the ones derived for the business case (`paramValue`). The found context(s) as well as all its ancestors in the context hierarchy are returned. Regarding the implementation of `detCaseSpecificCtx`, the semantics of hierarchical relationships of contexts and of parameter values are of interest (c.f. Table 3.2). A semantics of hierarchies often implemented in programming languages is inheritance. The question is: What does inheritance mean with respect to business rules in the application domain? Two common options are additive and most-specific inheritance. Using the former option, all business rules of the context hierarchy relevant to a concrete business case are relevant. With the latter option business rules can be refined, i.e., only the most specific business rules of the context hierarchy are relevant. A detailed discussion of inheritance regarding rules including multi-inheritance and conflict resolution is given in Chapter 6.

Example 3.1. We instantiate the generic CBR model for our use case SemNOTAM as depicted in Figure 3.3. For this purpose, we need to specify the business case class and the parameters that influence relevance of business rules. Relevant parameters in our use case are simple interests, flight phases, and event scenarios. For these parameters we need to derive parameter values from our SemNOTAM case, our domain-specific business case. In the following we discuss the domain-specific CBR model instantiation in Figure 3.3.

Our domain-specific `ContextClass` is called Aeronautical Information Management (AIM) Context (`AIMCtx`). It is described by the domain-specific parameters `Interest`, `FlightPhase`, and `EventScenario` which instantiate `Parameter`. Therefore, `defBy` is configured to one relationship for each parameter using the relationship configurator pattern. Regarding the context and parameter value relationships, only relationships establishing hierarchies, i.e., `specializes` and `covers` are of interest. In SemNOTAM, we derive the hierarchy of contexts (`specializes`) from the hierarchies of parameter values (`covers`). Business rules propagate along the `specializes` relationship. This mechanism needs to be implemented in `detCaseSpecificCtx`.

`SemNOTAMCase` is our domain-specific `BusinessCaseClass`. As such, it relates an `UserSituation` with a `NOTAM`. A `UserSituation` is a combination of an atomic interest

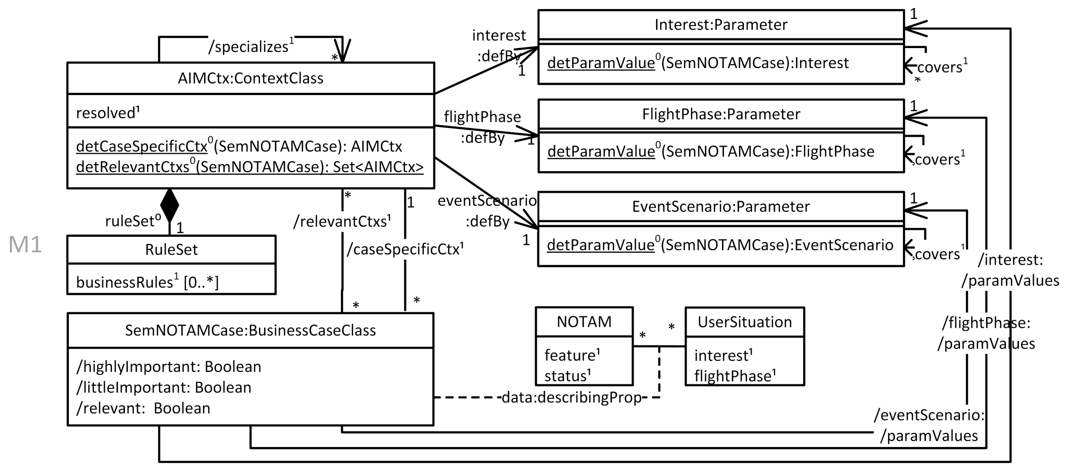


FIGURE 3.3: A domain-specific context model for context-aware organization of business rules and business vocabulary in the aeronautical domain, specifically SemNOTAM.

(`interest`) with a specific `flightPhase`. Applying relevant rules to a SemNOTAM case determines the associated NOTAM's relevance and its rank, i.e., it determines the values for the attributes `highlyImportant`, `littleImportant`, and `relevant`. Class `UserSituation` is not able to represent the full interest specification introduced in Section 1.4. In order to support full interest specifications including complex interests, each atomic interest has to be submitted as `UserSituation`. The determined attribute values then have to be combined as defined by the complex interests in the interest specification.

Each parameter defines a `detParamValue` method. For instance, `detParamValue` of `Interest` derives the corresponding parameter value from `UserSituation.interest`. The concrete code depends on the specific implementation. Executions of certain `detParamValue` methods can be cached, for example, the event scenario of NOTAMs may be materialized.

The semantics of hierarchical relationships in SemNOTAM are similar to Homola, Serafini, and Tamilin [93] where the hierarchy of parameter values determines the hierarchy of contexts. A context specializes another context if it has at least one more specific parameter value for a parameter and the parameter values for the other parameters are equal or more specific. Thus, `specializes` at level M1 is prefixed with a "/" to indicate its derivation. The semantics of a context specializing another one is the inheritance of its `RuleSet` instance. In SemNOTAM, additive inheritance semantics are applied. The only exception are defaults whose value can be refined in sub-contexts, for instance, a default spatial buffer of 70 nautical miles (nm) at the uppermost context can be refined to 50 nm at a lower context. Since such defaults are supported, `detCaseSpecificCtx` needs to define conflict resolution strategies for multi-inheritance issues. In case of spatial buffers, the resolution strategy is to use the largest buffer. The concrete code for `detCaseSpecificCtx` depends on the implementation.

3.3.4 Application-specific Model (M0)

Models of level M0 instantiate domain-specific CBR models and consequently contain concrete context and business case instances. Business rules need to be elicited from business personnel and stakeholders, organized into contexts, and the describing

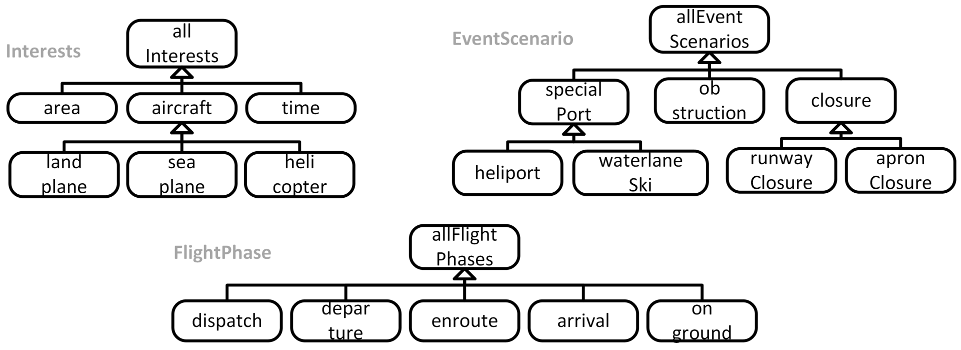
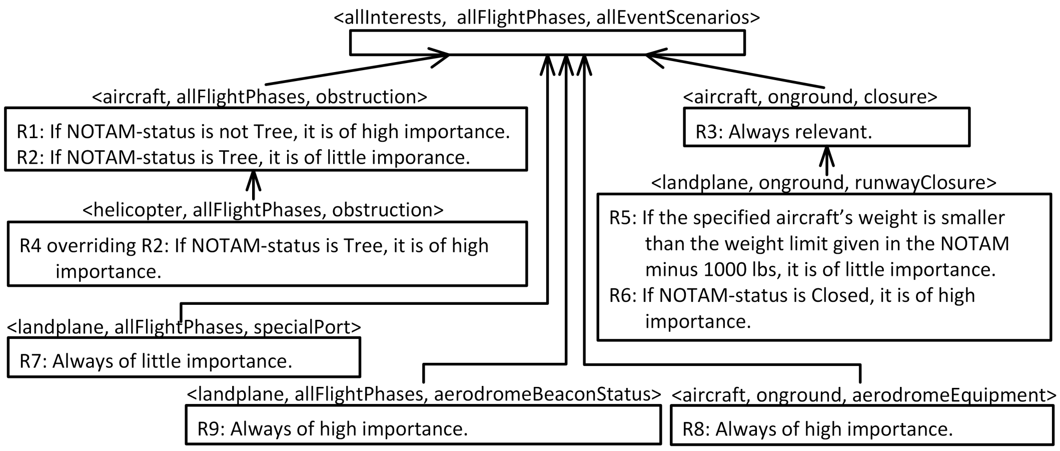


FIGURE 3.4: The parameter values and their hierarchies identified for our application-specific CBR model in SemNOTAM.


<allInterests, allFlightPhases, allEventScenarios>

<aircraft, allFlightPhases, obstruction>
R1: If NOTAM-status is not Tree, it is of high importance.
R2: If NOTAM-status is Tree, it is of little imporance.

<helicopter, allFlightPhases, obstruction>
R4 overriding R2: If NOTAM-status is Tree, it is of high importance.

<landplane, allFlightPhases, specialPort>
R7: Always of little importance.

<aircraft, onground, closure>
R3: Always relevant.

<landplane, onground, runwayClosure>
R5: If the specified aircraft's weight is smaller than the weight limit given in the NOTAM minus 1000 lbs, it is of little importance.
R6: If NOTAM-status is Closed, it is of high importance.

<landplane, allFlightPhases, aerodromeBeaconStatus>
R9: Always of high importance.

<aircraft, onground, aerodromeEquipment>
R8: Always of high importance.

FIGURE 3.5: Excerpt of contextualized business rules elicited from business personnel.

parameters and parameter values determined. Consequently, level M0 realizes CASA-feature tagging, i.e., business rules are assigned to concrete contexts. Furthermore, feature presentation is realized by determining the contexts relevant to a concrete business case (`/relevantCtxs`). The application of relevant business rules to a concrete business case corresponds to the CASA-feature execution.

Example 3.2. We determine the parameter values and their hierarchies for the three parameters `Interest`, `FlightPhase`, and `EventScenario` (depicted in Figure 3.4). Each parameter has a root parameter value. Parameter `Interest` has the seven parameter values. For this example we consider the three atomic interests `area`, `aircraft`, and `time` as `Interest` parameter values. Furthermore, we detail `aircraft` into the specific aircrafts `landplane`, `seaplane`, and `helicopter`. Parameter `FlightPhase` has six parameter values designating different flight phases covering an intended flight from preparation (`dispatch`) to arrival: `dispatch`, `onground`, `departure`, `enroute`, `arrival`. Parameter `EventScenario` has eight parameter values covering several event scenarios defined in [68]: `heliport`, `waterlaneSki`, `runwayClosure`, and `apronClosure`. Event scenarios `heliport` and `waterlaneSki` are subsumed under `specialPort` and all closure event scenarios are subsumed under `closure`.

Following from our defined parameter values we allow $7 * 6 * 8 = 336$ different classes of situations, i.e., contexts, under which business rules may apply. Subsequently, we elicit and organize business rules into these contexts (CASA-feature tagging). In particular, we organize the business rules presented in Section 1.4 into

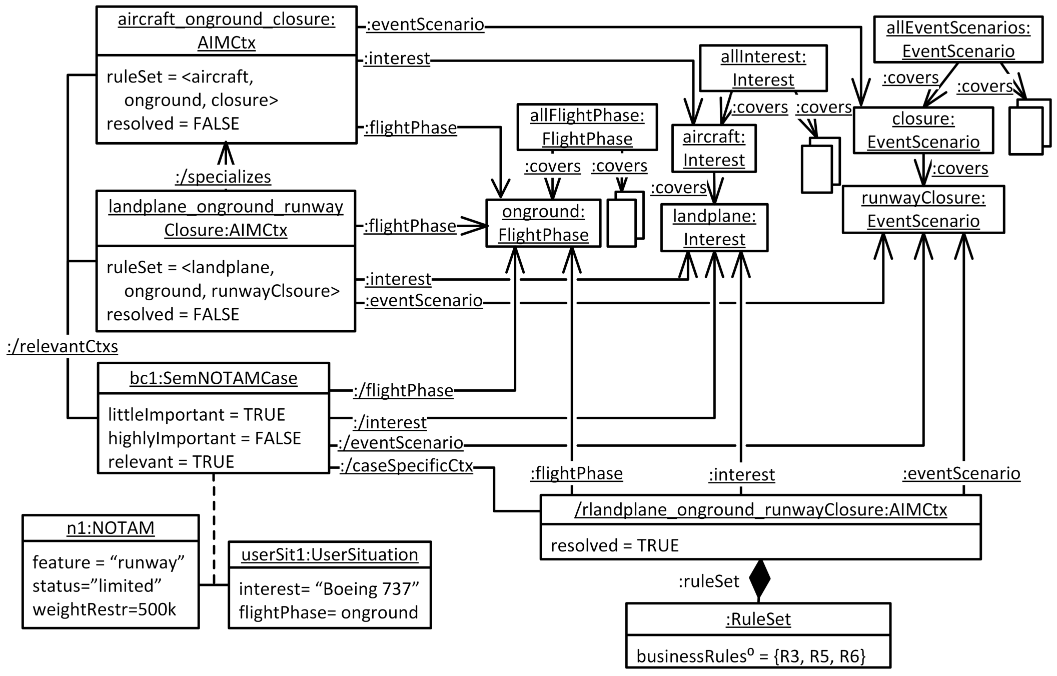


FIGURE 3.6: Detail of an application-specific CBR model for SemNOTAM.

these contexts. The resulting organization is depicted in Figure 3.5. For instance, rule R4 defines that it overrides rule R2 in context ⟨helicopter, allFlightPhases, obstruction⟩.

Figure 3.6 depicts a detail of the application-specific CBR model for SemNOTAM, in particular context `aircraft_onground_closure` with rule R3 and its specializing context `landplane_onground_runwayClosure` with rules R5 and R6. The attribute `ruleSet` refers to the rule sets presented in Figure 3.5.

SemNOTAM case `bc1` relates user situation `userSit1` specifying interest `"Boeing 737"` and flight phase `onground` with NOTAM `n1` reporting status `limited` and a weight restriction of `500k` lbs for a runway. Using `detParamValue`, the corresponding parameter values, `landplane`, `onground`, and `runwayClosure`, are determined. Method `detRelevantCtxs` returns the three relevant contexts ⟨landplane, onground, runwayClosure⟩, ⟨aircraft, onground, closure⟩, and ⟨allInterests, allFlightPhases, allEventScenarios⟩, where the latter is not depicted. Method `detCaseSpecificCtx` returns the new case-specific context `rlandplane_onground_runwayClosure` for which all semantics of relationships have been applied, i.e., inheritance has been resolved (R3, R5, and R6). Applying these rules to `bc1`, `n1` is determined to be `relevant` by rule R3 and to be `littleImportant` by rule R5.

3.4 Discussion

Regarding the claims made in Section 1.5 the proposed generic CBR model satisfies claims (1) eased business rule maintenance, (2) improved business rule execution performance, (3) faster search, and (4) support for executable as well as non-executable business rules.

Claim (1), improved maintenance, is mainly supported by (a) the explicit, structured, and multi-dimensional organization of business rules along their domain- and business-specific contexts as well as by (b) the automatic determination of relevant business rules from given business cases. As each context describes the class of situations in which their contained business rules apply (a), the number of conditions needed to define business rules decreases. This in turn increases the readability of business rule definitions. Furthermore, business rule redundancy is decreased by allowing inheritance of contexts and thus business rule sets. The multi-dimensional organization improves search as the search space can be quickly narrowed down using the parameters and parameter values in a CBR model. Furthermore, organization along domain- and business-specific contexts allows to customize business rule organization to the specific needs of a business, for instance, business rules may be organized along generic parameters, such as by their business rule class (c.f. Section 2.2.3), their representation (c.f. Section 2.2.2), or whether they define acceptable or unacceptable activity (c.f. Section 2.2), and/or along any domain- or business-specific parameters, such as flight phases, interests, and/or event scenario in the case of SemNOTAM. The automatic determination of relevant business rules (b) supports explanation and maintenance through case-specific contexts as for a specific business case only a fraction of contexts and thus business rules in the CBR need to be considered. Furthermore, business rules as well as their organization can be modified using the object-oriented mechanisms construction and destruction to add or remove elements from domain- and application-specific CBR models or by using the modification operations introduced in Chapter 5.

Claim (2), improved business rule execution performance is mainly achieved by the organization along contexts, in particular, by the partitioning nature of our organization along contexts, and the automatic determination of relevant contexts from business cases. By these features only a fraction of the business rules within a CBR have to be executed. Furthermore, due to the decreased number of conditions in business rule definitions, performance is increased. Thus, the generic CBR model for organizing business rules is advantageous for both business as well as IT.

Claim (3), improved search has already been addressed above. Claim (4), support for executable as well as non-executable business rules, is satisfied as no constraints on business rules or their representation in `RuleSet` are made.

Regarding usability, the presented generic CBR model for business rule organization is rather simple with only a few classes and relationships. Nevertheless, instantiated CBR models demand additional effort for managing contexts, context parameters, and parameter values. However, as long as the number of parameters and parameter values is reasonable this effort is outweighed by the improvement in manageability of business rules.

Although the presented generic CBR model is designed for business rules, it can also be used to organize other kinds of rules, for instance, rules for knowledge graph management [16] or web data extraction [72]. Furthermore, the generic CBR model can be used to organize any kind of information by simply exchanging the class `RuleSetClass` in Figure 3.2 by a class corresponding to the kind of information to be organized.

A CBR is a CASA realizing a CBR model for business rule organization. To enable the required CASA-features identified in Section 3.2.5 for CBRs, namely, presentation and tagging, the generic CBR model needs to support them. Context-aware presentation is supported by the methods `detCaseSpecificCtx` and `detRelevantCtxs`; tagging of business rules is supported by construction and destruction as well as the modification operations introduced in Chapter 5. Whether a CBR needs to support CASA-feature execution enabling direct execution of business rules depends on the use case – domain-specific and application-specific CBR models can be extended or instantiated to represent the effects of business rule execution like in Figure 3.3. This influences whether non-executable business rules are supported by a CBR as well as the ease of redeployment of business rules to other hardware/software platforms.

3.5 Related Work

In this section we delineate related work. Section 3.5.1 discusses related approaches not explicitly addressing rules or vocabulary. Section 3.5.2 considers related work from the domain of business rule organization. Section 3.5.3 discusses different commercial BRMSs. Section 3.5.4 delineates related approaches regarding contexts and context models. A summary of discussed related work is given in Table 3.7 regarding aspects of the proposed CBR model. Aspects considered are: whether a generic model for (rule) organization is supported, whether rules are at the center of organization and thus rule/term definitions and rule management are simplified, whether classes of situations are described in an explicit and structured way, and whether organizational aspects such as multi-dimensionality of organization, inheritance, custom relationships and semantics, automatic determination of relevant knowledge, modification operations, or conflict resolutions are supported.

Section 3.5.5 reviews rules in CASAs. Since none of the found approaches employs context to organize the rules we refrain from a comparison to CBRs.

3.5.1 Information Organization and Context-oriented Knowledge Management

Related approaches not explicitly addressing rules are faceted classification; Cyc, a knowledge base of human common sense and knowledge; and context-oriented knowledge management.

Faceted classification is a generic approach to information/knowledge organization from organization research respectively library and information science [71, 74]. As such it may be employed to organize business rules as well. Each resource is described with properties from orthogonal facets. A facet may be an enumeration, a boolean, a hierarchical classification, or a set of numerical ranges. Thus, multidimensional organization of resources is enabled. When searching for a particular resource not all facets need to be considered. A similar organization can be constructed using metadata management tools which regard an organization's management of its data and information assets [55]. Comparing faceted classification and metadata management tools with the generic CBR model, we find that specific advantages of organizing business rules utilizing faceted classification or metadata management are not discussed. Regarding description of classes of situations, faceted classification and metadata management provide structured description by facets and metadata respectively, but whether classes of situations are described explicitly depends on the facets and metadata employed. Regarding organization we find that neither faceted classification nor metadata management does not support further aspects, such as inheritance or custom relationships, apart from multi-dimensional organization.

The Cyc knowledge base of human common sense and common knowledge [115] employs contexts to organize knowledge (assertions and rules) along twelve mostly independent dimensions. These define a twelve-dimensional space of assumptions. Regions in this space correspond to contexts. Knowledge definitions become simpler as assumption covered by the twelve-dimensional space need not be encoded in the definitions. Contexts may be hierarchically ordered where assertions of more

TABLE 3.7: Summary of related work regarding aspects of the proposed generic CBR model. "+" indicates support, "o" restricted support, and "-" no support of the respective aspect.

	Faceted Classification [71, 74]	**Cyc [115]**	**Context-oriented Knowledge Management [185]**	**KnowledgeScope [112]**	**Modeling Contextual Knowledge [187]**	**Generic Business Rule Org. [33, 44, 82, 87, 102, 161, 188]**	**Hierarchical Organization [27, 119, 138, 161]**	**MCRDR [100]**	**Clustering of Rules [140]**	**Commercial BRMS [27, 70, 77, 95, 148, 196]**	**Context-oriented Model [198]**	**Context-dependent Goal Models [114]**	**Context-aware Application Adaption [202]**	**CDT-based [25, 51, 149]**	**CKR [183]**	**Framework for Representing/ Managing Context Info. [199]**
Generic Model for (Rule) Org.	+	-	o	o	-	o	o	o	+	o	+	o	-	-	+	+
Contextualized Knowl. Rules	+	+	-	o	+	+	+	o	o	o	o	-	+	-	o	-
Simplified Rule/Term Def.	-	+	-	-	+	-	-	+	-	-	-	-	-	-	o	-
Simplified Rule Managem.	-	+	-	-	+	o	-	+	o	+	o	-	-	-	-	-
Descr. of Classes of Situations																
Explicit	o	+	-	+	+	+	o	-	-	+	+	+	+	+	+	+
Structured	+	+	-	+	+	+	o	-	-	o	+	+	+	+	+	+
Organization																
Multi-dimensional	+	+	-	+	+	+	o	-	-	+	o	+	+	+	+	+
Inheritance	-	+	-	-	o	-	+	+	o	o	o	+	-	o	+	-
Custom Rel./Semantics	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-
Autom. Determ. of Knowl.	-	-	+	-	-	-	-	+	-	+	+	-	+	o	-	-
Modification Operations	-	o	-	-	+	o	-	o	-	+	-	-	-	+	-	+
Conflict Resolutions	-	+	-	-	-	-	-	-	-	o	-	o	-	-	-	-

general contexts apply in more specific ones by default – nevertheless, inherited assertions may be contradicted or overridden. Besides specialization, general lifting of assertions is supported. The contexts are key to faster knowledge entering – due to easier search along the dimensions and simpler knowledge specification – and faster reasoning – due to limiting the search to one relevant context and a small number of imported assertions from other contexts. Moreover, they ease browsing by employing fixed dimensions, reduce inconsistencies as they mostly separate them in different contexts, and factor out shared assumptions which may be ignored. Nevertheless, a trade-off between too-coarse and too-fine-grained contexts needs to be found, especially regarding reasoning as additional reasoning for dimensions, meta-level information, inter-context relations, and so forth is necessary. The basic idea of contexts and their relationships is similar to the one employed in the generic CBR model. The generic CBR model, different than Cyc, is a generic context model which can be instantiated to various domains and applications. Thus, the generic CBR model is not restricted to certain context dimensions (parameters) but allows free definition of parameters relevant to the problem at hand. Furthermore, the generic CBR model explicitly provides custom relationships and semantics of relationships, in particular for specialization, as well as determination of relevant contexts for a specific business case.

A context-oriented knowledge management for decision support in business networks is proposed in [185]. Knowledge is described in an application ontology comprising: a domain ontology, containing conceptual knowledge about the application domain; and a task ontology, describing problems in the application domain as well as problem-solving methods. Environment knowledge resources provide information of any changes in the domain. The context model represents knowledge about a decision situation, i.e., its settings and problems. Context is modelled at two levels: abstract and operational. Abstract context reduces the application ontology to knowledge relevant to the current decision situation and reduces the environment resources to the ones needed to instantiate knowledge specified in the abstract context (contextual resources). Operational context populates the abstract context ontology with knowledge from contextual resources. It reflects recent changes in the environment and embeds a constraint-based specification of the problem at hand. Constraint satisfaction is employed to derive feasible and satisfactory solution(s) in the current situation. Decision making then regards choosing one of these solutions. This is an altogether different approach to context-oriented knowledge management compared to the generic CBR model.

KnowledgeScope [112] is a knowledge management system defined around an organizational process, employing a process meta-model (Knowledge-in-Context model) to organize knowledge and context in a repository. Context in KnowledgeScope accommodates different perspectives of various roles in a process. Kwan and Balasubramanian employ four perspectives, namely, functional, informational, organizational, and behavioral perspective to cater to the knowledge needs of different roles. These four perspectives may be regarded context information, for instance, in the context of a strategist user the functional perspective is relevant. Thus, contexts (classes of situations) may be described explicitly and structured as defined in the Knowledge-in-Context model. Since they consider several perspectives, their organization can be considered multi-dimensional. The behavioral perspective enables definition of procedures under different organizational contexts and recording of process instances and their context. In order to ease knowledge and context acquisition and to present relevant knowledge and context information at the right time and place, Kwan and Balasubramanian propose to incorporate knowledge repositories and workflow support. Compared to the generic CBR model, Kwan and Balasubramanian recognize the importance of context for knowledge management but do not employ it to organize the knowledge. Furthermore, they do not discuss rules/terms as specific form of knowledge. Regarding organization, aspects apart from multi-dimensional organization are not supported.

3.5.2 Organization of (Business) Rules

Various approaches regarding business rule organization have been proposed. Modularization has been discussed in Section 2.3. In the following, we review an organization approach based on the context-as-a-box metaphor [187], various generic approaches for business rule organization [33, 44, 82, 87, 102, 161, 188], organization of rules and vocabulary in hierarchies [27, 119, 138, 161], multiple classification ripple down rules [100], and grouping rules into clusters and decision units [140]. Furthermore, we compare the reviewed approaches with the generic CBR model.

Sordo, Tokachichu, Vitale, Maviglia, and Rocha [187] propose an approach whose core idea is similar to CBRs. They suggest to separate the conditions restricting the scope of rules from the actual logic of a rule. The scope-restricting conditions form the context of the rule. They base their approach on the box metaphor [18, 73] describing a context by a set of parameters and values for these parameters. A special parameter value is ANY which includes all possible values. Thus, a simple form of inheritance is supported. Contexts in [187] must not overlap to prevent ambiguity. Sordo, Tokachichu, Vitale, Maviglia, and Rocha utilize context reasoning operations to manipulate contextualized knowledge and context knowledge. Supported are adding/removing contextualized knowledge, push and pop, and context shifting. Besides rules, contexts may contain value specifications, like `threshold = 40`. This approach facilitates rule authoring and maintenance, rule reuse, atomic rules, and knowledge discovery. Compared to the generic CBR model, Sordo, Tokachichu, Vitale, Maviglia, and Rocha's approach utilizes the box metaphor as well but a more detailed meta-model of context-based organization of rules is missing. Furthermore, they do not mention defined or derived hierarchical relationship of parameter values (except ANY) or contexts, nor is inheritance of rules or vocabulary explicitly discussed. Furthermore, no custom relationships can be defined. The approach does neither discuss automatic determination of relevant knowledge nor conflict resolution.

Besides Sordo, Tokachichu, Vitale, Maviglia, and Rocha's approach, various strategies for organizing business rules have been proposed in literature, form very simple ones organizing along single attributes [82, 161], to multi-dimensional approaches [33, 44, 87, 102, 188]. Here we consider multi-dimensional approaches in more detail. Brown and Pomykalski [33] propose two approaches to rule base partitioning. One partitions rule bases into sets of possibly overlapping rule chains leading from inputs to outputs. The other partitions the rule base into sequences of rule groups where the output of one group is the input for the next group. A parameter driven approach, storing rules in a repository along with values for various attributes is proposed by Butleris and Kapocius [44]. The presented repository model is designed for the Ross Method. Furthermore, rules may be grouped. A similar approach is taken by Herbst and Myrach [87] where ECA-Alternative (ECAA) rules are stored alongside their embedding within the business/organization in a repository. From this repository various (graphical) views can be generated. The meta-model for the repository comprises a business rule submodel, containing business rules, events, conditions, and actions; and a process submodel, containing ECAA chains, where the action of one ECAA rule causes the triggering event of another ECAA rule, representing business processes. Abstraction and refinement of business rules is introduced to reduce complexity [87, 88]. A business rule can be refined to an ECA chain with the same input event(s) and output action(s). Another repository model [102] takes rule collection, transition to design and implementation of information systems, and support of business change into account. Each business rule is either an intentional rule (natural language rule from business perspective), an operational rule (formal rule statement from business process perspective), or an information system architecture rule (rules from implementation perspective) where intentional rules lead to operational rules and operational rules to architecture rules. Depending on the kind of rule, different viewpoints are assumed. In the repository schema for intentional rules, each business rule is collected in a rule collection session, has a number of attributes (such as type, body, version, or expiry date), and hinders or creates an opportunity for a goal. Operational rules in the repository can be structured along event, condition, and action as well as referenced relationships between activities and actors, activity enablers, and information objects. Multi-dimensional contextualized rules are also found in comparative data analysis in data warehouses [188]. Rules are assigned to hierarchically ordered contexts, i.e., the comparative cubes specified by comparative concepts, in which they apply. The hierarchy of contexts has a most-specific inheritance semantic, i.e., only the most specific rules are executed. Supported rules are judgement and analysis rules which are restricted to one and two conditions respectively. Consequently, this approach is not suitable for arbitrary rules. Compared to the generic CBR model, none of the above approaches provides a full generic model for rule organization or discusses improvements in readability of rule and term definitions. Only a few of the reviewed approaches detail modification operations and their side-effects. None of the approaches allows custom relationships or custom semantics of relationships supported. Furthermore, they do not support free definition of parameters and parameter values for organization or automatic determination or relevant rules. Conflict resolutions are not discussed.

A concept employed in the generic CBR model, which has been introduced in some form before, are hierarchies of rules and rule sets. For instance, data mining often results in a large number of rules which need to be organized and summarized in a way easy to analyze [119]. For this purpose, Liu, Hu, and Hsu [119] distinguish general rules giving the big picture and exception rules defining exceptions. These exception rules may be arbitrarily nested with summaries being provided. A different approach is taken by Rai and Anantaram [161] who support hierarchical groupings of rules to enable efficient reuse (single point of source). Organization of knowledge and thus business rules by situations/contexts has been proposed before [67, 135, 141, 201]. Evans, Dalkir, and Bidian [67] state the necessity of knowledge being contextualized in order to be shared and reused. The same holds for business rules. Similarly, Boyer and Mili [27] suggest organizing business rules by their application area (jurisdiction) either using rule sets or dynamic queries. All of these approaches consider rules and their inheritance, Nalepa [138] also discusses relationships between groups of rules where such relationships are used to optimize the inference process. Compared to the generic CBR model, these approaches do not provide a full generic model for rule organization nor do they discuss improvements in rule and term readability or BRM. A few approaches support limited description of classes of situations but not as flexible and customizable as the multi-level CBR model. Regarding organization, multi-dimensionality has been discussed in a few approaches, although not in detail. Neither of the approaches tackles automatic determination of relevant knowledge, modification operations, or conflict resolutions.

Multiple Classification Ripple Down Rules (MCRDR) [100] are an incremental knowledge acquisition method dealing with multiple independent classifications. A knowledge base is organized into hierarchical levels of classification rules, where rules on a lower level refine specific rules on the adjacent higher level. This may be considered a form of inheritance. Furthermore, rule definitions become simpler by hierarchically refining them. For a certain object with some attributes, first the top level rules are executed. For each rule satisfied, the refining rules on lower levels are executed until no more refinement rules exist or no more rules are satisfied. Consequently, for a given object, relevant classification rules are found and executed. If an object is classified wrongly, a new classification rule is added where appropriate. This and the limitation to classification rules simplifies BRM for this specific approach. Compared to the generic CBR model, MCRDR is restricted to classifications rules only – in cases were classifications and exceptions to classifications are not sufficient MCRDR is not suitable while the generic CBR model is. Since all knowledge is encoded in rules, explicit or structured description of classes of situations is not supported. Regarding organization, multi-dimensionality is not supported, neither are custom relationships or semantics nor conflict resolutions.

Nowak and Wakulicz-Deja [140] propose organizing knowledge bases (often containing rules) into groups of similar rules using cluster analysis, into decision units, or into a combination thereof. Rule clusters are designed for forward chaining. To cluster rules, rules are transformed into a vector space model to which agglomerative clustering is applied. Thus, inference is sped up as only specific rule clusters need to be executed as well as search is improved. A hierarchical clustering algorithm is employed as it is a simple and natural form for representing large data sets. Thus, inheritance is partially supported. Decision units are designed for backward chaining and for knowledge bases with rules which might create deep inference paths. A decision unit comprises rules with the same attribute and attribute value in the conclusion. Consequently, a decision unit net depicts the dependencies in the knowledge base. Extending decision units by petri nets enables modeling the dynamics of inference processes. Nowak and Wakulicz-Deja furthermore propose organizing rules within clusters into decision units. This supports both forward chaining (employing clusters) and backward chaining (using decision units). By these organization strategies, BRM is simplified as usually only groups of knowledge need to be considered instead of the complete knowledge base. Different to our approach, an instantiable generic model is not supplied although the approach can be applied to any knowledge repository to organize knowledge for various forms of chaining. An organization along business criteria is not supported. Improvements in readability of knowledge definitions or BRM are not discussed. Furthermore, no means to describe knowledge groups are provided. Regarding organization, only inheritance is implicitly considered.

3.5.3 Commercial Business Rule Management Systems

BRMSs often organize automatable business rules only, i.e., rules that they can execute [135]. Furthermore, many BRMSs provide hierarchical organization of rule sets where business rules of a higher-level rule set apply in lower-level rule sets as well [27, 135, 139, 201]. We regard four commercial BRMSs, namely, IBM ILOG JRules, FICO Blaze Advisor, JBoss Drools, and Oracle Business Rules. All of them support, at least in basic form, the BRM activities identified in Section 2.4.2 [27, 77, 95, 148, 196]. Furthermore, each of them supports business vocabularies and different rule representations, such as decision tables, decision trees, natural language, or rule patterns. In the following we detail each of the four commercial BRMSs and compare them to CBRs.

Regarding rule repository organization, *JRules* [27, 95] provides the most sophisticated tools: Repositories have a development organization and a runtime organization. The development organization is into rule projects and these rule projects into hierarchically organized rule packages. Rule projects can depend on other rule projects which make their business rules and business vocabulary accessible. A best practice is to separate the business vocabulary, development organization, and runtime organization into different rule projects. Similarly, common rules in different projects should be extracted to a separate project. The runtime organization is into rule sets. Rule sets are constructed by rule queries on rule projects extracting rules based on metadata and concerned business objects at runtime. The execution order within a rule set can be controlled by ruleflows. Since organization along a single criteria is often insufficient [27], rule metadata can be used to introduce orthogonal (hierarchical) dimensions. Furthermore, JRules allows overriding of rules, i.e., for each rule one can specify which other rules it overrides. For change management an external BPM product can be used.

Using *Blaze Advisor* [70, 77] rules can be grouped into functional rule sets; so called ruleflows control the execution order of rule sets. Similar to JRules, rules and rule sets can be assigned metadata which can be queried. Regarding inheritance of rules, Blaze Advisor's approach is rule subsumption, i.e., a rule can refer to another rule and will inherit this rule's condition. *Drools* [196] allows to organize rules in rule set files, decision trees, decision tables, et cetera. These units may be organized in hierarchical packages where rules can be assembled from multiple sources. Similar to JRules and Blaze Advisor, Drools supports ruleflows as well as metadata (tagging) for rules. Furthermore, rules can be extended, i.e., the extended rule's condition is inherited. *Oracle Business Rules* [148], similar to Blaze Advisor, views rule sets as units of execution where ruleflows specify the order in which rule sets are executed. The documentation neither mentions support for overriding nor for inheriting rules.

Many commercial BRMS provide extensive features – also regarding business rule organization although mostly exclusively to automatable business rules. The generic CBR model supports all kinds of rules, whether automatable or not. None of the investigated BRMS solutions supports modeling of multi-dimensional organization schemes as are necessary for organizing by situation/context. JRules, with its support for hierarchical metadata properties, could implement such a scheme but the formed rule sets and their relationships would not be first-class citizens of JRules making it difficult to obtain an overview of the repository organization. Furthermore, a simplification of rule definitions cannot be achieved. CBR models on the other hand support comprehensive overviews of the business rule organizations as well as simpler business rule definitions. JRules further supports determination of relevant rule sets using JRules's runtime rule set construction. Major differences between development organization and actual runtime organization may be the case [27]. Depending on the concrete realization of a CBR model in a CBR, development and runtime organization may be different considering contexts and their hierarchy to be development organization and case-specific contexts to be runtime organization. The generic CBR model supports determination of all relevant contexts and thus business rules for business cases in development and runtime organization while BRMSs usually do not provide such functionality. Commercial BRMS provide rule inheritance from higher-level rule sets to lower level rule sets, rule extension, or rule overriding where the relationship needs to be explicitly stated. CBR models on the other hand support custom inheritance mechanisms, for instance, where more specific contexts inherit from more general contexts. The inheritance hierarchy of contexts is derived from the hierarchies of parameter values. The semantics of this inheritance mechanism can be adjusted as needed. Regarding available rule set or rule project relationships, commercial tools provide dependency, important for rule reuse, and ruleflow. The meaning of these relationships is fixed. Furthermore, adding relationships is hardly possible. The generic CBR model allows to specify the meaning of such relationships, for example, inclusion, as well as the addition of relationships if necessary. Conflict resolutions may be defined in BRMSs, often by defining priorities of rules.

3.5.4 CASAs and Context Models

Many CASAs and context models for various domains have been published. In the following, we review CASAs and context models relevant to the generic CBR model and compare them with our proposed approach. In particular, we consider: the Context-oriented Model [198], a top-down modelling approach serving as base context model for CASAs; context-aware adaption of applications in general [202] and goal models in requirements engineering [114] in particular; Context Dimension Tree (CDT)-based frameworks, methodologies, and tailoring [25, 51, 149]; Contextualized Knowledge Repositories in the semantic web [183]; and a general framework for representing and managing context information for adaptive web information systems [199].

The Context-Oriented Model (COM) [198] abstracts generic context concepts from domain- and application-specific concepts to create a top-down modelling approach which can be used as base context model for generic CASAs. Their approach is based on context definitions by Dey [57], Pomerol and Brézillon [157], and Brézillon [32]: Contexts are related to foci. A focus may be an objective, task, or decision making and as such enables distinction of relevant and irrelevant knowledge. Context, in a given focus, comprises context knowledge (relevant knowledge), external knowledge (unknown/irrelevant knowledge), and proceduralized context (knowledge effectively used in the focus, supporting the task at hand). For instance, in the generic CBR model, the focus is specified by a business case for which relevant business rules are determined (context knowledge) – external knowledge are all business rules not relevant. The COM comprises, similar to CBR models, three layers: the upper layer contains generic context management related concepts (meta-layer), the middle layer defines domain-specific context-related concepts within the generic ones, and the lower layer instantiates the middle layer for an application. Concepts on the upper level are: entities, real world things relevant to the domain; contextual elements, anything characterizing the context of entities, like location or duration, which may be hierarchically organized; rules and actions, deriving context information from data and building proceduralized knowledge; and foci. Derived concepts on the upper level are: the set of contextual elements relevant in a focus, the set of relevant rules to be activated in a focus, and proceduralized knowledge constructed from contextual elements and the set of rules. This organization of knowledge may ease knowledge management. Compared to the generic CBR model, COM is based on ontological/tree-map modeling focusing on determining proceduralized context for different foci. The generic CBR model on the other hand, is a conceptual model focusing on organization of business rules, determining relevant business rules/terms for given business cases, and is designed to the peculiarities of business rules. Rules in COM are organized by foci and contextual elements (kind of forming a multi-dimensional organization), while in the generic CBR model rules are organized by contexts. Improved rule/term readability is not reported. The semantics of context element hierarchies in COM is fixed whereas CBRs allow custom semantics. Furthermore, the generic CBR model explicitly allows custom relationships of contexts and parameter values. The generic CBR model enables the detection of relevant contexts from a business case, whereas COM requires a defined relationship between foci and contextual elements not allowing to derive this relationship. Regarding organization, neither modification operations nor conflict resolutions are discussed.

The work by Lapouchnian and Mylopoulos [114] is close to our approach in the sense that it proposes context-dependent visibility of elements in goal models (employed in requirements engineering) and enables displaying goal models for certain circumstances. A context is defined as "an abstraction over a set of environment assumptions" [114, p. 117]. Not all elements of a goal model are affected by context. Things in the domain influencing the elements in a goal model are called context entities, they are the source of domain variability. These context entities may be related to one or more context variability dimensions which are aspects of a domain along which the domain changes. A dimension defines a set of values. A context is a specific value for an entity-dimension combination. Each context has a contextual tag (label) and a definition (boolean expression) which defines when the context is active. Negated contexts are allowed, enabling detection of context conflicts. Contexts may be hierarchically organized with a simple form of non-monotonic multi-inheritance supported, enabling reuse of defined contexts and incremental development of models. Contextual tags are then used to annotate elements in the goal model defining the circumstances of their visibility. Additionally, the process for deriving and analyzing context-aware goal models is described. Their notion of contexts corresponds to parameter values and their notion of dimensions to parameters in the generic CBR model. A main difference to the generic CBR model are the objects contextualized and the goal pursued: the generic CBR model contextualizes business rules and is designed for active usage and deployment whereas [114] contextualizes visibility of model elements and is designed for requirements engineering. Regarding context dimensions/parameters, the generic CBR model follows a uniform approach whereas [114] proposes an individual approach, allowing different dimensions for model elements. Regarding context and parameter value detection [114] allows a more fine-grained definition than the generic CBR model which defines detection at parameter/dimension level. The generic CBR model's inheritance mechanism is not restricted to parameter values only. Furthermore, custom relationships and their semantics, automatic determination of relevant knowledge, and modification operations are not discussed for Lapouchnian and Mylopoulos's approach.

An approach similar to Lapouchnian and Mylopoulos is presented by Wang, Mehta, Chung, Supakkul, and Huang [202]. They consider context-aware adaptive applications, i.e., applications adapting their behavior based on context. For such systems they present a goal-oriented approach to selecting alternative adaptations taking into consideration a particular contextual event as well as important non-functional requirements. By taking into account context and non-functional requirements a goal-oriented systematic decision process is enabled. Context comprises static or dynamic information around or about entities. Such information regard objects, time, space, physical environment, and scenario. Thus, classes of situations may be described in an explicit and structured way. Wang, Mehta, Chung, Supakkul, and Huang propose a rule-based context-aware adaptive system, where detection of predefined context-changes (events) triggers rules to be executed causing adaptation. Their notion of context corresponds to a parameter value in the generic CBR model. Compared to the generic CBR model, contexts are more rigid whereas the generic CBR model allows instantiation of contexts, parameters, and parameter values as needed. Simplified rule/term definitions or BRM are not reported. Furthermore, no hierarchies nor any other relationships of contexts are introduced. Consequently only simple organization possibilities are offered. Regarding organization, Wang, Mehta, Chung, Supakkul, and Huang provides multi-dimensional organization and enables detection of contexts which triggers adaption rules adapting the software system; other aspects are not supported.

A general framework for the design of CASAs employing the concept of Context Dimension Tree (CDT) for context modelling and Datalog$^{\pm}$ for their implementation is proposed in [149]. It describes two life-cycle loops: design and run-time loop. The former contains the tasks context modelling and application domain modelling, which can be performed independently of each other, as well as the critical task of designing the relationships between the two models. The latter contains tasks such as context sensing, recognition, validation and exploitation. A CDT describes the dimensions, dimension values, potential parametrized values, and their relationships for contexts in a tree-based model. Furthermore, CDTs allow for constraints to prevent specific combinations of values for contexts. The Datalog$^{\pm}$ implementation is split into several programs, one containing application-independent vocabulary, one the application-specific context model, and one the context instances. A data-tailoring application similar to the one presented in [25] is given as use case. Compared to the generic CBR model, Orsi and Tanca's [149] framework does not provide an implementation-independent meta-model for contexts. Rules/terms and the advantages of CDTs for them are not discussed. The generic CBR model allows custom relationships of contexts and parameter values with custom semantics, whereas [149] provides only a fixed one. Conflict resolutions are not discussed.

The CARVE (Context-aware Relations View Definition) methodology [25] derives context-aware views of relational data to reduce the amount of data presented to users depending on their provided context. Context is modeled in a CDT comprising dimensions. Each dimension represents an aspect of context and comprises hierarchically organized context elements. For each context element a partial view determining data relevant to the context element is specified. A hierarchical relationship of two context elements corresponds to the partial view of the upper-level element to be further reduced in the lower-level element. These partial views are then combined, using predefined operators, to context-aware (materialized) views for contexts. The CDT may be modified using predefined operations [160]. The CARVE methodology's design goal is tailoring relational data while the generic CBR model focuses on business rule organization. Depending on the concrete business rules, the generic CBR model can be instantiated for information tailoring, like for SemNOTAM. Regarding relationships, the generic CBR model allows custom relationships as well as customized semantics of relationships whereas CARVE provides a fixed specialization relationship of context elements only. Furthermore, users need to supply their context whereas CBR models derive relevant context(s) from environment data contained in business cases.

The Context-Aware Data Design, Integration, Customization, and Tailoring (context-ADDICT) [51] project employs ontology-based context and domain models for context-aware data integration and tailoring in distributed mobile environments. It proposes a design methodology, dynamic hooking and integration of new information sources, and data-tailoring of beforehand unknown data sources. Similar to Bolchini, Quintarelli, and Tanca [25] they employ CDTs to define context dimensions (aspects of context) and allowed dimension values. A configuration chunk is a set containing exactly one dimension value for each dimension. Each potential configuration chunk has to be related to concepts in the domain ontology, thus defining a set of relevant data from the data sources. The approach allows restriction on dimension value combinations as well as wildcards for dimension values. Similar to CARVE, context-ADDICT is designed for data tailoring and employs context for this aim while the generic CBR model employs contexts for organization of business rules, enabling, amongst other use cases, data tailoring. Inheritance or custom relationships of contexts are not mentioned, nor is the semantics of hierarchically ordered dimension values discussed. Furthermore, determination of the relevant configuration chunks for a user in a specific situation is not described.

Contextualized Knowledge Repositories (CKRs) [183] for the semantic web enable organization of ontological concepts by employing description-logic based, hierarchically ordered contexts. A context is described by a set of values from different but fixed number of context dimensions. These values may be hierarchically ordered. The value hierarchies are used to derive a hierarchy of contexts. Concepts propagate along these hierarchies from general contexts to more specific contexts. Concepts may assume different meanings in different contexts. They differentiate a vocabulary for knowledge (object vocabulary) and a vocabulary for meta-knowledge, i.e., context description. The object vocabulary is shared between all contexts but the symbols may have different interpretations. CKRs are designed for concepts and the semantic web whereas the generic CBR model is designed for business rules/terms and a broader area of application. Consequently, simplified rule definitions and simplified BRM are not discussed for CKRs. Furthermore, CKRs do not provide context model operations besides shifting and push/pop nor has conflict resolution been discussed. Automatic determination of relevant contexts is missing in CKR as well as possibilities to define context and parameter value relationships other than inheritance and subsumption or custom semantics of relationships. Conflict resolutions are not discussed.

Virgilio and Torlone [199] sketch a general framework supporting representation and management of a variety of context information for adaptive web information systems. This framework comprises a middleware data model facilitating representation and maintenance of heterogeneous context data. The provided context meta-model provides basic primitives such as dimensions and attributes where a dimension is described by a set of attributes. These primitives can be used to represent classes of situations in an explicit and structured way. Besides basic primitives for context modeling Virgilio and Torlone propose a set of basic primitives for manipulation. Compared to the generic CBR model, they do not provide a generic model which can be instantiated to models more detailed. Contextualized rules/terms are not discussed for Virgilio and Torlone's approach. Regarding organization, only multi-dimensionality and modification operations are reported.

3.5.5 Rules in CASAs

CASAs adapt their behavior to the current context [138]. Consequently, for two different contexts a context-aware system may provide different services, information, or information presentation. Rules in context-aware systems are mostly employed for adaptation based on context [96, 99, 129, 130] but also to derive higher-level context information [206], associating contexts with services [195], or context matching and processing [133, 134]. None of the found approaches employs contexts to organize rules, thus we summarize the approaches but refrain from a comparison with CBRs.

The Context-aware Software Entity Adaptation (CASEA) [99] system considers context-aware adaptation of software entities constituting enterprise applications. A context is any information affecting a software entity. Two types of rules are introduced: context dependency rules, defining context ranges affecting a software entity, and adaptation rules, initiating appropriate adjustments of software entities. Lower-level software entities may inherit context dependency rules from upper level software entities. Adaptation rules are expressed using ECA rules where the condition is formed by context ranges derived by context dependency rules.

The Hierarchical Context-aware Computing (HCAC) [130] architecture supports context gathering, management, and distribution as well as service adaptation. It is hierarchical in the sense that it provides four stacked and loosely coupled layers and supports context data on different semantic levels from raw to formalized. The lowest layer is the physical layer bridging the physical world (for example, user, environment, or sensors) and the computing system. The second layer, context handling, collects, integrates, and shares context data from the physical layer. The third layer is the context-aware service layer responsible for composing services customized to context. How to customize services to contexts is defined in rules [129, 130]. These rules are handled in the fourth layer, the rule generation layer. This layer comprises a rule library and a rule generation module employing context history to extract new rules or to revise existing ones for service adaption [129, 130].

Ibrahim and Mouel [96] describe a context-aware mechanism that dynamically specializes general strategies for the use of services based on the current context in a pervasive environment. General strategies are represented as a set of ECA rules. Specialization means specialization of the condition and action parts of ECA rules. The framework has several layers: device layer, containing available devices; service layer, containing services of available devices; and strategy layer, describing the strategies applied and executing services based on context. A predicate-based context model is employed (`predicate(subject, value)`) where the predicate is the type of context, for instance, location, subject is an actor or object, and value is the value of the subject. Each predicate is described by an ontology comprising generic and specialized concepts. Similarly, an action ontology exists. Once the user's context is defined, the general strategy is specialized accordingly, i.e., its rules' condition is expressed using context predicates and the actions are expressed using concrete action concepts linked to services.

Xue, Pung, and Sen [206] propose a scalable context data management system (CDMS) which works as middleware between context-aware systems and different operating spaces. An operating space is any entity having multiple context sources to provide context data. A context attribute is a specific kind of context supplied by an operating space, like a person's heart beats per minute. A context schema describes all context attributes of an operating space. A context domain is a class of operating spaces with similar context attributes whose context schemas are integrated into a common domain schema by a schema matcher. The architecture consists of a system server (the middleware) and Operating System Gateways (OSG). An OSG is a uniform interface to acquire data from operating spaces and provides components to manage and manipulate context data within the space. The system server receives queries which the context query engine parses using the global set of domain schemas maintained in the schema manager. The query engine identifies the relevant operating spaces from context domains and context attributes used in the query. Based on this information the query engine creates a query plan distributing the query onto OSGs for partial local evaluation. Subsequently, the partial results are merged and returned. A simple rule-based context reasoner can be used to deduce events of interest across operating spaces and domains or to derive more abstract context attributes in OSGs.

The Rule-based Framework for Managing Context-aware Services (RuCAS) [195] defines context-aware services as automatically detecting changes in context and performing appropriate actions. A context is viewed as situational information which is derived from system or sensor information. Contexts can be atomic or compound (based on other contexts). In RuCAS, every context-aware service is described by an ECA rule. Event is some context which became true, condition a set of contexts to be satisfied, and action a set of web services to be executed.

'Intelligent Context' proposed in [133, 134] introduces a system providing an effective, portable, implementation-independent approach which enables context-aware service provision, data processing from various differing sources, handling of complexity of context, context management, ability to manage constraint satisfaction, and ability to realize predictable decision support under uncertainty. The ability to process contextual information is vital to Intelligent Context. Moore, Hu, Jackson, and Wan distinguish input context (the problem) and output context (a potential solution). Determining whether an output context is an acceptable match for an input context is called context matching. The system comprises: a context application responsible for data capturing; a rule base; a database for contextual information; and three context management modules, namely, context definition repository, context matching, and context processing. The context definition repository models the context-space holding input and output contexts for context matching. The rule base is a repository for (fuzzy) ECA-based context processing rules. These are employed in context matching and context processing. Context processing concerns the creation, access and maintenance of context definitions. Context reasoning is key to Intelligent Context, enabling context processing which in turn facilitates context matching. To this end, a context reasoning ontology defining core and domain concepts and their relationships is defined.

3.6 Conclusion

In this chapter we have considered the context (model) aspects identified in Section 2.5 regarding business rules organization and determined and argued our choices. Based on these choices we have presented the generic conceptual CBR model for business rules organization along contexts. This generic CBR model can be instantiated multi-level for a specific domain and for concrete applications within this domain. Notable choices include: representational notion of contexts; relationships of contexts and thus of business rule sets; custom semantics of relationships such as custom inheritance mechanisms; modifiable sets of contexts, parameters, and parameter values; and automatic determination of relevant contexts for a given business case. To demonstrate the usefulness of the generic CBR model we have instantiated it for SemNOTAM.

On the one hand, instantiated CBR models have several advantages such as increased business rule readability, reduced business rule redundancy, and explicit definition of a business rule's applicable classes of situations. On the other hand, additional effort is necessary to manage and maintain instantiated CBR models. Consequently, for each potential use case the additional costs need to be compared to the gained benefits.

The generic CBR model satisfies all claims made in Section 1.5, i.e., eased business rule maintenance, improved performance of business rule execution, faster search, and support for executable and non-executable business rules.

Chapter 4

CBR Prototypes

The generic CBR model presented in the previous chapter is an implementation-independent model. In order to demonstrate that CBRs realizing CBR models are practicable, we describe several CBR prototypes. We utilize rule languages as implementation languages and for representing the organized business rules, thus enabling direct rule execution. We conduct feasibility studies on the rule execution performance of our prototypes and based on their results construct a database-backed CBR. For this database-backed CBR we show that the rule execution outperforms that of a database-backed Non-contextualized Business Rule Repository (NCBR) for all but very small repositories.

4.1 Introduction

The generic CBR model is a conceptual model, i.e., it does not describe a concrete implementation. To demonstrate its practicability we implemented three proof-of-concept prototypes realizing CBR models. For this purpose, we oriented ourselves on the CASA development and deployment process described in Section 2.6.

First, we need to concrete the decisions regarding requirements elicitation listed in Table 3.2.5 for CBRs. Regarding CASA-features, we additionally decide for execution. This enables us to demonstrate that CBRs satisfy claim (2), improved business rule execution performance. Regarding the aspects uncertainty handling, context history, versioning, and context prediction, we decide not to support them – we focus on basic CBRs. Nevertheless, these four aspects can be supported in CBRs. Regarding the design architecture, we decide for a client-server model: users (clients) create and submit business cases to CBRs (servers). Regarding the implementation paradigm, we chose rule languages as rules have been shown to be an efficient and intelligible way of representing context knowledge [138]. Furthermore, the contextualized knowledge, i.e., business rules, can be expressed using the same rule language. Thus, maintenance is simplified and at the same time business rule execution can be performed using the same engine.

Our proof-of-concept prototypes, employing $\mathcal{FLORA}$-2, Datalog$^{\pm}$/Vadalog, and SPARQL Inferencing Notation (SPIN), respectively, are described in Section 4.2 together with the results of some feasibility studies regarding their business rule execution performance. The feasibility studies of our proof-of-concept prototypes revealed that it is far from trivial to realize the theoretical speed-up in business rule execution in CBRs due to the potential performance overhead of contextualized rule execution, for example, determining relevant business rules. We develop a CBR prototype realizing the speed-up by employing relational databases, in particular PostgreSQL, in combination with $\mathcal{FLORA}$-2. We compare the performance of our CBR prototype with the performance of a $\mathcal{FLORA}$-2- and PostgreSQL-based Non-contextualized

Business Rule Repository (NCBR). The prototype and the accompanying performance study are discussed in Section 4.3. Section 4.4 concludes the chapter.

4.2 Proof-of-concept Prototypes

We present three proof-of-concept prototypes. Each prototype employs a different rule language, namely: $\mathcal{F}LORA$-2 [104], an object-oriented knowledge representation and reasoning system building on Frame-logic allowing a realization close to UML CBR models; Datalog$^{\pm}$/Vadalog [45], an expressive and efficient Datalog derivative with a wide range of built-ins like database access or inclusion of machine learning algorithms; and SPIN [106], a SPARQL Protocol and RDF Query Language (SPARQL)-based rule and constraint language, integrating concepts from rule-based systems, query languages, and object-oriented languages. For the $\mathcal{F}LORA$-2-based and the SPIN-based proof-of-concept prototypes, we provide feasibility studies comparing their business rule execution performance to non-contextualized business rule repositories using the same rule language. For the Datalog$^{\pm}$/Vadalog prototype we did not perform feasibility studies due to limited resources.

Section 4.2.1 presents our $\mathcal{F}LORA$-2-based proof-of-concept prototype realizing our CBR model for SemNOTAM; Section 4.2.2 presents our Vadalog-based CBR realization; and Section 4.2.3 presents our SPIN-based CBR realization for the semantic web.

4.2.1 $\mathcal{F}LORA$-2[1]

This proof-of-concept prototype closely resembles the generic CBR model and its instantiation for SemNOTAM by utilizing the frame-based language $\mathcal{F}LORA$-2. $\mathcal{F}LORA$-2 [104] is an object-oriented knowledge representation and reasoning system building on Frame-logic. It supports a multitude of different concepts and features such as meta-programming, logical updates, or defeasible reasoning. Therefore, $\mathcal{F}LORA$-2 is a suitable language for experimental prototypes or research projects.

In the following, we discuss excerpts of our $\mathcal{F}LORA$-2 realization[2]; a summary of relevant $\mathcal{F}LORA$-2 syntax is displayed in Table 4.1. Furthermore, we conduct a feasibility study comparing the performance of rule execution in our $\mathcal{F}LORA$-2-based CBR with a $\mathcal{F}LORA$-2-based NCBR.

Realization

Our $\mathcal{F}LORA$-2-based proof-of-concept prototype is based on the multi-level instantiation and schema definition features of $\mathcal{F}LORA$-2. We organize our $\mathcal{F}LORA$-2 realization into two modules: context knowledge, i.e., contexts, parameters, and parameter values, is contained in module `ctxModel`; business cases and their describing properties in module `bc`. Rules applying in a specific context are stored in $\mathcal{F}LORA$-2 files. The module `bc` may contain only one specific business case at a time.

An excerpt of the realization of level M2 of the CBR model is depicted in Listing 4.1. These statements define the schema of CBR model elements at level M1, and also the schema for CBR model elements at level M0 where already defined at level

[1]Previously published in F. Burgstaller, B. Neumayr, C. G. Schuetz, and M. Schrefl. "Modification Operations for Context-Aware Business Rule Management". In: *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. 2017, pp. 194–203

[2]The full implementation is available at `https://github.com/burgst/CBRM`

TABLE 4.1: *$\mathcal{F}$LORA*-2 syntax elements used in this chapter.

Syntax	Description
`o:C. C:M.`	`o` is an instance of `C` which is an instance of `M`.
`p(e,f).`	A fact of predicate `p` with arguments `e` and `f`.
`a(?e), (b(?e); f(?e)).`	True if there exists a fact `?e` for predicate `a` which also holds for predicate `b` or `f`.
`C[|p{i..j} => R|].`	Method/property `p` at class `C` with cardinality constraint `i` to `j` has range `R`.
`C[|m(A) => R|]@u.`	Method `m` of class `C` with argument of class `A` has range `R` at module `u`.
`o[p->e]`	`o` has property `p` with value `e`.
`o[m(a)->e]`	Method `m` with argument `a` applied on object `o` returns object `e`.
`%`	Prefix `%` indicates that the predicate/method is non-tabled, i.e., (partial) results are not cached.
`a(?x) :- b[m->?x].`	A rule $\forall_x : b(m,x) \rightarrow a(x)$.
`forall(?X)^(p(?X)~~>q(?X))`	A statement $\forall_x : p(x) \rightarrow q(x)$.
`(<>[+'f.flr'>>m])`	Adds *$\mathcal{F}$LORA*-2 program `f.flr` to module `m`.

M2. Line 2 states that each `ContextClass` has to specify a non-empty set of parameters (`|defBy{1..*} => Parameter|`) and a set of `ctxRelationships` referencing instances of `ContextClass`. The attributes and associations for domain-specific context classes are defined in Line 3: any `ContextClass` instance (bound to variable `?model`) has property `file` of type `String` (referencing a rule set) and boolean property `resolved`. Line 3 further defines that each instance of `ContextClass` has an association `specializes` where the domain and range are the respective `ContextClass` instance bound to `?model`. Further associations defined in Line 3 are one association per concrete parameter named after the respective concrete parameter and referencing instances of the respective parameter (`?param1..1 => ?param`). The method signatures of domain-specific context classes are defined in Lines 4 and 5, for example, each domain-specific instance of `ContextClass` provides a method `detRelevantCtxs` accepting instances of the respective domain-specific `BusinessCaseClass` and returning a context instance of the respective domain-specific `ContextClass` (Line 4). Similarly, Lines 8 and 9 define the schemata for domain-specific `Parameter`. Line 8 defines the optional relationships `covers` and `valueRelationship` between instances of domain-specific parameters. Line 9 defines the method signature for `detParamValue` accepting domain-specific business cases and returning parameter values. The schema for domain-specific instances of `BusinessCaseClass` is defined in Line 12.

The instantiation of the generic CBR model to level M1 is split into structural instantiation and method implementation. Listing 4.2 depicts the structural instantiation for our use case SemNOTAM. We define `AIMCtx` to be our domain-specific `ContextClass` (Line 1) with the three parameters `Interest`, `FlightPhase`, and `EventScenario` (Lines 2–3). We define `SemNOTAMCase` as our domain-specific `BusinessCaseClass`. Each `SemNOTAMCase` instance has to provide an interest specification as well as a NOTAM as describing properties.

LISTING 4.1: Excerpt of the generic CBR model realized with $\mathcal{FLORA}$-2.

```
% ContextClass
ContextClass[|defBy{1..*} => Parameter, ctxRelationship => ContextClass|].
?model[|file{1..*} => \string, resolved => \boolean, specializes => ?model, ?param{1..1}
↪  => ?param|]:- ?model:ContextClass, ?model[defBy->?param].
?model[detRelevantCtxs(?domainCase) => ?model]:- ?model:ContextClass,
↪  ?domainCase:BusinessCaseClass@bc.
?model[detCaseSpecificCtx(?domainCase,?,?) => ?model]:- ?model:ContextClass,
↪  ?domainCase:BusinessCaseClass@bc.

% Parameter
?param[|covers => ?param, valueRelationship => ?param|]:- ?param:Parameter.
?param[detParamValue(?domainCase) => ?param]:- ?param:Parameter,
↪  ?domainCase:BusinessCaseClass@bc.

% BusinessCaseClass
BusinessCaseClass[|descProps => \object|].
```

LISTING 4.2: Excerpt of structural elements of the domain-specific CBR model for SemNOTAM realized with $\mathcal{FLORA}$-2.

```
AIMCtx:ContextClass.
{Interest, FlightPhase, EventScenario}: Parameter.
AIMCtx[defBy->{Interest, FlightPhase, EventScenario}].
SemNOTAMCase:BusinessCaseClass[|userSituation{1..1} => UserSituation, notam{1..1} =>
↪  NOTAM|].
```

The second part of instantiating the generic CBR model is method implementation. Listing 4.3 presents generic implementations for determining `specializes` from parameter values, `detRelevantCtxs`, and `detCaseSpecificCtx`, as well as an excerpt of domain-specific `detParamValue` methods. Lines 2 and 3 derive the reflexive-transitive closure of `covers`. Lines 6 and 7 derive relationship `specializes`. Line 10 shows one of the many rules implementing `detParamValue` for the parameter `Interest`. It derives from a given `SemNOTAMCase`, employing the predicate `getAircraftType`, the parameter value for parameter `Interest`. Lines 13 and 14 define the derivation of relevant contexts and case-specific contexts. Method `detRelevantCtxs` (line 13) returns for a given business case the relevant contexts, i.e., it determines all contexts which have for every parameter the same or more general parameter values as the parameter values derived for the business case. Non-tabled method `%detCaseSpecificCtx` (line 14) creates for a given business case `?bc` a new module `?mTarget` with all rules combined; `?ctxf` returns the rule files of relevant contexts. To resolve conflicts, `%detCaseSpecificCtx` can be extended by corresponding predicates or, employing defeasible logic, the appropriate statements need to be loaded into `?mTarget`.

Domain-specific CBR models are instantiated to models of level M0 by providing instances for the defined classes. An excerpt of level M0 is shown in Listing 4.4. The parameter value hierarchy of parameter `FlightPhase` is depicted in Lines 1–2. The definition of context ⟨landplane, onground, runwayClosure⟩ as depicted in Figure 3.6 is shown in Line 4; the SemNOTAM case `bc1` (Figure 3.6) is represented in lines 6–8.

For the $\mathcal{FLORA}$-2 prototype we assume that all business rules are defined in $\mathcal{FLORA}$-2 as well. Thus, determined `caseSpecificCtxs` can be immediately executed on business cases. Listing 4.5 shows $\mathcal{FLORA}$-2 representations for rules `R5` and `R8` from Figure 3.5.

LISTING 4.3: Excerpt of methods defined for the domain-specific CBR model for SemNOTAM implemented with $\mathcal{F}LORA$-2.

```
% Reflexive-transitive closure of covers
?superVal[covers->?subVal]:-?_param:Parameter, ?superVal:?_param[covers->?_val],
↪  ?_val:?_param[covers->?subVal:?_param].
?val[covers->?val]:- ?val:?_param, ?_param:Parameter.

% Context relationship specializes
?parentCtx[paramCovers(?param)->?childCtx]:-?model:ContextClass,
↪  {?parentCtx,?childCtx}:?model, ?parentCtx[?param:Parameter->?_v1],
↪  ?childCtx[?param:Parameter->?_v2], ?_v1[covers->?_v2].
?childCtx[specializes->?parentCtx]:- ?model:ContextClass, {?parentCtx,?childCtx}:?model,
↪  ?parentCtx[paramCovers(?)->?childCtx], true{forall(?param)^(?model[defBy->?param]~~>
↪  ?parentCtx[paramCovers(?param)->?childCtx])}.

% detParamValue derivation
Interest[detParamValue(?bc)->?val]:- ?val:Interest, (?bc:SemNOTAMCase[userSituation ->
↪  ?[interest->?aircraft]])@bc, getAircraftType(?aircraft,?val).

% relevantCtxs and caseSpecificCtx derivation
AIMCtx[detRelevantCtxs(?bc)->?ctx]:-?model:ContextClass, ?ctx:?model,
↪  true{forall(?p,?v)^(?ctx[?p:Parameter->?v] ~~> (?p:Parameter[detParamValue(?bc) ->
↪  ?val:?p],?v:?p[covers -> ?val]))}.
AIMCtx[%detCaseSpecificCtx(?bc, ?mTarget, ?ctxf)]:- AIMCtx[detRelevantCtxs(?bc) -> ?ctx],
↪  ?ctx[file -> ?ctxf], (<>[+?ctxf>>?mTarget]).
```

LISTING 4.4: Excerpt of the application-specific CBR model for SemNOTAM realized with $\mathcal{F}LORA$-2.

```
{allFlightPhases, onground, departure, enroute, arrival, dispatch}: FlightPhase.
allFlightPhases[covers -> {onground, departure, enroute, arrival, dispatch}].

landplane_onground_runwayClosure:AIMCtx[Interest -> landplane, FlightPhase -> onground,
↪  EventScenario -> runwayClosure, file->'ctxs/ctxLORwycl.flr'].

userSit1:UserSituation[interest -> Boeing_737:Aircraft, flightPhase -> onground].
n1:NOTAM[feature->runway, status->limited, weightRestriction->500000].
bc1:SemNOTAMCase[userSituation->userSit1, notam->n1].
```

Feasibility Study

In order to confirm the expected performance gains by using a $\mathcal{F}LORA$-2-based CBR over a $\mathcal{F}LORA$-2-based NCBR, we conduct a small feasibility study. To this end, we define NCBRs first and subsequently conduct the feasibility study.

Non-contextualized Repository During the SemNOTAM research project we designed a first prototype. For this first prototype, we stored the elicited business

LISTING 4.5: $\mathcal{F}LORA$-2 representations for rules R5 and R8 from Figure 3.5.

```
/*R5*/ littleImportant:- (?BC:SemNOTAMCase[userSituation -> ?[interest ->
↪  ?Aircraft:Aircraft]], ?Aircraft[weight -> ?W], ?n:NOTAM[status -> limited,
↪  weightRestriction -> ?WLimit])@bc, ?W < (?WLimit-1000).

/*R8*/ highlyImportant.
```

LISTING 4.6: Rule R5 and R8 from Listing 4.5 as defined in our $\mathcal{F}LORA$-2-based NCBR.

```
1     % Rules
2     /*R5*/ ?BC[littleImportant -> ?N] :-
3       ?BC[userSituation->?[interest->?I, flightPhase->?FP], notam->?N]@bc,
        ↪  ?I:AircraftOfInterestTypeLandplane, ?FP:onground, ?N:RunwayClosure,
        ↪  ?I[weight->?W]@bc, ?N[status->limited, weightRestriction->?WLimit]@bc, ?W <
        ↪  (?WLimit-1000).
4
5     /*R8*/ ?BC[highlyImportant -> ?N] :-
6       ?BC[userSituation->?[interest->?I, flightPhase->?FP], notam->?N]@bc,
        ↪  ?I:AircraftOfInterestTypeAircraft, ?FP:onground, ?N:AerodromeEquipement.
7
8     % Concepts
9     ?I:Interest :- (?:SemNOTAMCase[userSituation-> ?US], ?US[interest->?I])@bc.
10    ?I:AircraftInterest :- ?I:Interest, ?I:Aircraft.
11    ?I:AircraftTypeLandplane :- ?I:AircraftInterest, ?I[type -> landplane].
12    AircraftInterest::Interest.
13    AircraftTypeLandplane::AircraftTypeAircraft.
14
15    ?FP:?FP :- (?:SemNOTAMCase[userSituation-> ?US], ?US[flightPhase->?FP])@bc.
16    {FlightPhaseOnground, FlightPhaseDeparture, FlightPhaseEnroute, FlightPhaseArrival,
      ↪  FlightPhaseDispatch}:FlightPhase.
17
18    ?N:RunwayClosure :- (?N:NOTAM, ?N[feature->runway,status->closed])@bc.
19    ?N:RunwayClosure :- (?N:NOTAM, ?N[feature->runway,status->limited])@bc.
20    RunwayClosure::Closure.
21    ?N:AerodromeEquipment :- (?N:NOTAM, ?N[feature->aerodromeILS])@bc.
22    {SpecialPort, Obstruction, Closure, AerodromeBeaconStatus,
      ↪  AerodromeEquipment}::EventScenario.
23
24    Boeing_737:Aircraft[type->landplane,weight->194700].
```

rules in non-contextualized form in rule sets [39, 41, 190]. We employ the business rule style and representation employed in this first prototype to construct the non-contextualized repository for our comparison. Non-contextualized business rules are split into a scope-part and an actual-rule-part. The scope-part limits the applicability of the actual-rule-part. Both, scope-part and actual-rule-part employ terms from the business vocabulary defined in the rule set. Business vocabulary and its terms are considered a specific class of business rules.

Example 4.1. Our NCBRs are based on the assumption that they employ the same business case representation as CBRs do. Figure 1.2 presents non-contextualized business rules elicited for SemNOTAM in natural language. The text outside a box describes the scope-part while the text within a box describes actual-rule-parts. Listing 4.6 depicts the non-contextualized representation of rule R5 (Lines 2 and 3) and R8 (Lines 5 and 6). The scope-part and the actual-rule-part of business rule R5 are shown in Figure 4.1 outside and inside the box respectively. The definition of the business rule's scope-part and the actual-rule-part employ the concepts defined in Lines 9–24. The concepts defined in Lines 9–11 derive from a given business case whether its interest considers an aircraft and whether this aircraft is a landplane. Lines 12–13 establish term hierarchies, for instance, an aircraft interest is also an interest. Lines 15 and 16 define concepts for flight phases specified in the given business case. Lines 18–22 derive event scenarios for NOTAMs given in business cases. Line 24 defines some information about Boeing 737s.

Compared to the CBR representation in Listing 4.5, the rule definitions in Listing 4.6 are longer. For instance, rule R8 is a simple fact in Listing 4.5 while a longer rule in Listing 4.6. This is the case as the context knowledge needs to be encoded

```
?BC[userSituation->?[interest->?I, flightPhase->?FP], notam->?N]@bc,
  ?I:AircraftOfInterestTypeLandplane, ?FP:onground, ?N:RunwayClosure
```

```
?BC[littleImportant -> ?N] :- ?I[weight->?W]@bc, ?N[status-
  >limited, weightRestriction->?WLimit]@bc, ?W < (?WLimit-1000).
```

FIGURE 4.1: The scope-part (outside the box) and actual-rule-part (inside the box) of business rule R5 using $\mathcal{FLORA}$-2.

in the rule – in Listing 4.6 we do so by specifying the scope-part for rules R5 and R8 employing our defined concepts.

Results To estimate the feasibility of CBRs, we compare the performance of our simple $\mathcal{FLORA}$-2-based CBR prototype with the performance of a $\mathcal{FLORA}$-2-based NCBR prototype concerning our running example. We employ an Intel Core i5-6200U (2.30 GHz) with 2×8 GB DDR4. The application for performance evaluation is implemented in Java. We utilize `ProcessBuilder` to run the $\mathcal{FLORA}$-2 version 1.2 command line interface (CLI) as process. Streams are used to pass commands to the $\mathcal{FLORA}$-2 CLI and to read information printed by the $\mathcal{FLORA}$-2 CLI.

First, we compare the times for loading the respective prototypes and SemNOTAMCases. Loading the CBR prototype into a clean $\mathcal{FLORA}$-2 instance takes on average 1.368 s while loading the NCBR prototype takes about 1.116 s (n=200). This difference is explained by the nature of the prototypes: the CBR prototype requires more code as it is written to work with arbitrary numbers of parameters and parameter values while the NCBR prototype works only for the specific use case. The time required for deleting the current SemNOTAMCase and adding a new SemNOTAMCase is on average less than 6 ms (n=7200).

Second, we compare the time it takes to determine and apply all relevant business rules to a given SemNOTAMCase employing the CBR and the NCBR prototype. The application of relevant business rules may result in facts `relevant`, `littleImportant`, and/or `highlyImportant`. We define 24 SemNOTAMCases covering all contexts in our use case. Each of these 24 SemNOTAMCases is submitted 100 times to the respective prototype. Thus, we are able to determine and record reliable measurements of the time needed to retrieve the derived information.

For the CBR prototype we need to distinguish: (1) cold start, i.e., context files have not been compiled yet, (2) precompiled, i.e., all context files have been compiled beforehand, and (3) materialized, i.e., contexts are resolved to case-specific contexts and loaded in $\mathcal{FLORA}$-2 modules. Cold start takes on average 138 ms, precompiled 55 ms, and materialized 27 ms per SemNOTAMCase (n=2400). The differences between these variants are statistically significant (confidence = 0.999).

We compare the NCBR prototype with the precompiled and the materialized CBR prototype. Since the NCBR prototype is compiled when loaded to $\mathcal{FLORA}$-2, this establishes the same preconditions for both the CBR and the NCBR prototype. A boxplot of the measured times is given in Figure 4.2. The NCBR prototype needs on average 7 ms to retrieve `relevant`, `littleImportant`, and `highlyImportant` for a given SemNOTAMCase (n=2400). Therefore, it is significantly faster (confidence = 0.999) than the CBR precompiled (on average 48 ms slower) and the CBR materialized (on average 20 ms slower) prototype.

Analysing the time is spent in the CBR prototype we find: Determining the derived results `relevant`, `littleImportant`, and `highlyImportant` from a case-specific context ($\mathcal{FLORA}$-2 module) takes on average 2 ms. Determining the contexts relevant

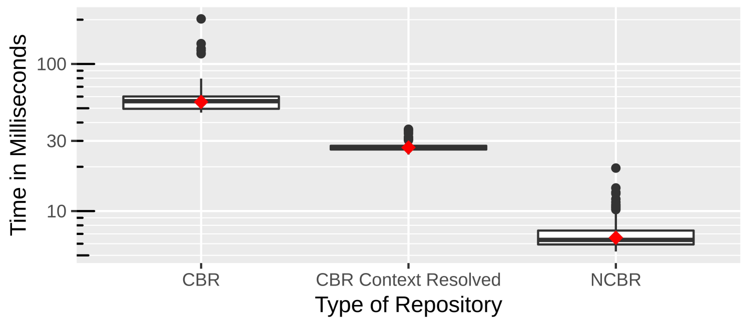


FIGURE 4.2: Comparing the execution times of our $\mathcal{F}LORA$-2-CBR prototype including construction of case-specific contexts (CBR), of the $\mathcal{F}LORA$-2-CBR prototype assuming that case-specific contexts have been resolved to $\mathcal{F}LORA$-2 modules (CBR Context Resolved), and of the $\mathcal{F}LORA$-2-NCBR prototype. The diamonds within boxes represent the respective mean.

to a given SemNOTAMCase takes on average 25 ms; determining relevant contexts and building a case-specific context module takes on average 53 ms. The long time required for relevant context determination is caused by the generic code working with arbitrary numbers of parameters and parameter values. The NCBR prototype on the other hand only works with those defined in the use case. Exchanging the generic code for one specific to the use case is expected to increase the performance significantly. The longer time required by case-specific context construction is mainly due to file operations (loading files into $\mathcal{F}LORA$-2).

The results of this feasibility study show that our simple $\mathcal{F}LORA$-2-based CBR prototype is insufficient to actually realize the benefit of improved performance over NCBRs for our use case. An efficient CBR realization is required. In Section 4.3 we present an efficient database-backed CBR prototype.

4.2.2 Datalog$^{\pm}$/Vadalog

In addition to the $\mathcal{F}LORA$-2-based prototype we developed a prototype employing Vadalog, a concrete language of the Datalog$^{\pm}$ rule language family. Datalog$^{\pm}$ [45] is a versatile rule language family, extending plain Datalog with features such as negative constraints, existentials to create values, and equality-generating dependencies. Vadalog [17] is an implementation of Warded Datalog$^{\pm}$ providing a wide range of built-ins, such as machine learning, aiming at commercial use. Datalog$^{\pm}$/Vadalog is quite expressive, for instance, it encompasses Datalog with no restriction on recursion while still being efficient. As such, Datalog$^{\pm}$/Vadalog is a versatile rule language family also employed for big data wrangling [109] and knowledge graph management [16].

Due to the supported built-ins and performance, we consider Vadalog a well-suited rule and implementation language for CBRs. The prototype presented here is split into two parts: (1) context representation and determination of relevant contexts and (2) determination of a case-specific context by resolving inheritance and conflicts. Here we discuss the first part, the second part is detailed in Section 6.5. A feasibility study, as for the other prototypes, has not been conducted due to limited resources. Nevertheless, functionality tests have been performed.

Since Vadalog does not support multi-level instantiation, we realize CBR models as follows: The classes specified at the generic CBR model define the language, LISTING 4.7: Excerpt of structural elements of the domain-specific CBR model for SemNOTAM realized with Datalog$^{\pm}$/Vadalog.

```
1   contextClass("aimCtx").
2   parameter("Interest"). parameter("FlightPhase"). parameter("EventScenario").
3   defBy("aimCtx","Interest"). defBy("aimCtx","FlightPhase").
      ↪ defBy("aimCtx","EventScenario").
4   businessCaseClass("semNOTAMCase").
```

i.e., level M2 defines the predicates available and to be used. Classes are represented by unary predicates whereas associations are represented by binary predicates. For instance, `ContextClass` with relationship `defBy` is represented by predicates `contextClass/1` and `defBy/2`. Derived associations are not explicitly represented as predicates, they are implicitly available by realizing the respective method deriving them. Checks for correct domain and range as well as multiplicity have been omitted. Thus, level M2 is only implicitly realized in our Vadalog prototype. Models of level M1 are instantiated by populating the language predicates.

The instantiation of the generic CBR model to models of level M1 with the Datalog$^{\pm}$/Vadalog prototype can be split in two parts: structural instantiation and method implementation. In Listing 4.7 we show the structural instantiation for our use case SemNOTAM corresponding to the UML diagram shown in Figure 3.3. To this end, we employ the predicates from our language. The statements have the same meaning as in Listing 4.2, for example, Line 1 defines `aimCtx` as our domain-specific context class. A difference is in Line 4, where the $\mathcal{F}LORA$-2 statement defines that each `SemNOTAMCase` relates one `UserSituation` and one `NOTAM`. For Vadalog, we need to extend our language by domain-specific language predicates, for example, `hasNOTAM` and `hasUserSituation` to relate `SemNOTAMCase` with NOTAMs and `UserSituation` respectively. An instantiation of these predicates is shown in Listing 4.9.

The second part of instantiating the generic CBR model to models of level M1 is method implementation. In Listing 4.8 we present generic implementations for determining `specializes` from parameter values and `detRelevantCtxs`, as well as an excerpt of domain-specific `detParamValue` methods. Lines 2–4 define transitive and reflexive-transitive closures for `covers`. These are employed to derive the context relationship `specializes` from the parameter value hierarchies (Lines 7–9). An excerpt of the implementation of `detParamValue` is shown in Lines 12 and 13. These access the NOTAM and user situation associated with a given business case and employ their describing properties (`hasDescProp`) to derive the parameter value for the respective parameter. For instance, value `runwayClosure` is derived for parameter `EventScenario` if the associated NOTAM concerns a runway which has status closed. Derivation of relevant contexts is depicted in lines 16–18. Employing the parameter values derived from a business case (`detParamValue`), the relevant contexts are determined. The implementation of method `detCaseSpecificCtx` is discussed in Chapter 6.

Models of level M1 are instantiated to models of level M0 by providing facts for the predicates in the domain-specific language. An excerpt of level M0 in our use case SemNOTAM is depicted in Listing 4.9. Lines 2–3 specify the two parameter values `allInterests` and `area` and their relationship for parameter `Interest`. Line 6 shows a context definition. The internal ID of this context is `ctx0`, its name is `allInterests_allFlightPhases_allEventScenarios`, and it is an instance of `aimCtx`. The `RuleSet` contained in the context is stated using the predicate `ruleSet`. The parameter value for each parameter of `aimCtx` are specified using `hasParamValue`. Lines

LISTING 4.8: Excerpt of methods defined for the domain-specific CBR model for SemNOTAM implemented with Datalog$^{\pm}$/Vadalog.

```
% Transitive and reflexive-transitive covers
tCovers(SuperVal,SubVal) :- tCovers(SuperVal,X), covers(X,SubVal).
tCovers(SuperVal,SubVal) :- covers(SuperVal,SubVal).
trCovers(SuperVal,SubVal) :- tCovers(SuperVal,SubVal). trCovers(Val,Val):-
↪ paramValue(_,Val).

% Context hierarchy derivation
paramCover(ParentCtx,ChildCtx,Param):- hasParamValue(ChildCtx,Param,SuperVal),
↪ hasParamValue(ParentCtx,Param,SuperVal2), trCovers(SuperVal2,SuperVal).
notParamCover(ChildCtx,ParentCtx,Param):- context(ChildCtx),
↪ hasContextClass(ChildCtx,CtxCl), defBy(CtxCl,Param), context(ParentCtx), not
↪ paramCover(ChildCtx,ParentCtx,Param).
specializes(ChildCtx,ParentCtx) :- paramCover(ParentCtx,ChildCtx,_), not
↪ notParamCover(ParentCtx,ChildCtx,_).

% detParamValue derivation
detParamValue(BC,"Interest",Val) :- businessCase(BC), hasUserSituation(BC,I),
↪ hasDescProp(I,"interest",A), aircraft(A), hasDescProp(A,"type",Val).
detParamValue(BC,"EventScenario","runwayClosure") :- businessCase(BC), hasNOTAM(BC,N),
↪ notam(N), hasDescProp(N,"feature","runway"), hasDescProp(N,"status","closed").

% relevantCtxs derivation
bcParamCover(BC,Ctx,Param) :- hasParamValue(Ctx,Param,PVal),
↪ detParamValue(BC,Param,PVal2), trCovers(PVal,PVal2).
notBcParamCover(BC,Ctx,Param) :- businessCase(BC), context(Ctx),
↪ hasContextClass(Ctx,CtxCl), defBy(CtxCl,Param), not bcParamCover(BC,Ctx,Param).
relevantCtxs(BC,Ctx) :- bcParamCover(BC,Ctx,_), not notBcParamCover(BC,Ctx,_).
```

LISTING 4.9: Excerpt of the application-specific CBR model for SemNOTAM realized with Datalog$^{\pm}$/Vadalog.

```
1   % Parameter value hierarchies
2   paramValue("Interest","allInterests"). paramValue("Interest","area").
3   covers("allInterests", "area").
4
5   % Context definition
6   context("ctx0"). hasName("ctx0","allInterests_allFlightPhases_allEventScenarios").
    ↪ hasContextClass("ctx0","aimCtx"). ruleSet("ctx0","module0"). hasParamValue("ctx0",
    ↪ "Interest","allInterests"). hasParamValue("ctx0", "FlightPhase", "allFlightPhases").
    ↪ hasParamValue("ctx0", "EventScenario","allEventScenarios").
7
8   % Business case specification
9   businessCase("bc1"). hasBusinessCaseClass(BC,"semNOTAMCase").
    ↪ hasUserSituation("bc1","userSit1"). hasNOTAM("bc1","n1").
10  interestSpec("userSit1"). hasProp("userSit1","interest","Boeing737").
    ↪ hasProp("userSit1","flightPhase","onground").
11  notam("n1"). hasProp("n1","feature","runway"). hasProp("n1","status","limited").
    ↪ hasProp("n1","weightRestriction","500000").
```

9–11 define business case `bc1`. Line 9 specifies that `bc1` is of class `semNOTAMCase` and has NOTAM `n1` as well as user situation `userSit1` assigned. Lines 10 and 11 depict the specification of the user situation and NOTAM as shown in Figure 3.6.

LISTING 4.10: SPARQL representation of rule R2 in context ⟨helicopter, allFlightPhases, obstruction⟩ which derives NOTAMs to be of little importance if they have status `tree` .

```
    CONSTRUCT { spin:_this aim:littleImportance ?n. }
    WHERE { ?n aim:status obstacle:tree. }
```

4.2.3 SPARQL Inferencing Notation[3]

We construct this prototype to demonstrate that the generic CBR model is also applicable to the semantic web. We focus on domain- and application-specific CBR models only, not dealing with multi-level instantiation in ontology languages. Furthermore, for simplicity we regard rules only, vocabulary is considered future work. A detailed description is given in [37].

In the following, we discuss our SPIN-based prototype. Furthermore, we conduct a feasibility study comparing our contextualized SPIN-prototype with a non-contextualized SPIN-prototype.

Realization

SPIN is a SPARQL-based rule and constraint language integrating concepts from rule-based systems, query languages, and object-oriented languages [106]. SPIN enables formal description of object behavior, i.e., SPIN statements are assigned to classes and apply to individuals of their assigned classes only. SPIN [106] comprises the SPIN modeling vocabulary and an RDF representation of SPARQL. The SPIN modeling vocabulary supports inference rules (`spin:rule`), constraints on classes (`spin:constraints`), and constructors (`spin:constructor`) specified using SPIN's RDF representation of SPARQL. For instance, the SPARQL representation of rule R2 from our use case is shown in Listing 4.10. SPIN rules can be organized into rule libraries. Furthermore, the SPIN modeling vocabulary supports user-defined execution orders of SPIN rules, user-defined rule templates, and user-defined functions. The latter two features increase rule readability and promote reuse of rules.

In order to realize a CBR model using SPIN, CBR model elements need to be represented in RDF(S) or Web Ontology Language (OWL). Consequently, appropriate representations of context classes and contexts, their parameters and parameter values, as well as business case classes and business cases need to be found: Each context, utilizing SPIN's object orientation, is represented as an OWL class with its rules assigned. Parameters and their parameter values form subsumption hierarchies. The assignment of parameter values to contexts can be represented in two ways. Regardless the variant, the hierarchy of contexts can be derived from the parameter value hierarchies using a reasoner. The two variants are:

- employing object properties as modeled in Chapter 3 and
- subclassing, presumably faster in evaluation.

Each business case class is represented as an OWL class and defines its describing properties in the same fashion as the contexts, i.e., either by subclassing or using

[3]Previously published in F. Burgstaller, C. Schütz, B. Neumayr, and M. Schrefl. "Towards Contextualized Rule Repositories for the Semantic Web". In: *Joint Proceedings of the Web Stream Processing workshop (WSP 2017) and the 2nd International Workshop on Ontology Modularity, Contextuality, and Evolution (WOMoCoE 2017) co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 22nd, 2017*. Oct. 2017, pp. 98–109. URL: `http://ceur-ws.org/Vol-1936/paper-09.pdf`

object properties (cf. variants above). In the simplest configuration, describing properties instantiate or refer to the parameter values directly. In more complex configurations, parameter values must first be derived from describing properties using OWL or SPIN.

This approach enables a reasoner to determine the relevant contexts for a given business case. In particular, the reasoner derives that the business case is an individual of the relevant contexts. Since a business case can be an individual of multiple contexts, the rules of several contexts are evaluated when performing SPIN evaluation. Thus, conflicting conclusions can be derived, for example, a NOTAM could be classified both relevant and irrelevant. To resolve this, a cautious resolution rule could be defined: a NOTAM is relevant as soon as one rule classifies it as relevant. In other cases, most specific rules may have precedence.

CBRs also consider business vocabulary. Thus, SPIN-based CBRs should enable contextualized vocabulary as well. Creating one SPIN rule library per context would theoretically allow to define terms local to a context by defining them in the corresponding rule library. In practice, however, non-SPIN triples relevant to rule execution are not allowed in SPIN rule libraries. A different option is to represent the entire vocabulary using SPIN rules. In this case, however, a reasoner would have to first execute the vocabulary rules before making inferences. Modeling vocabulary explicitly, one ontology per context containing the context's vocabulary is necessary. These ontologies must be imported to enable their use in SPIN rules, i.e., all vocabularies are accessible in all contexts. Another option is to integrate contextualized knowledge repositories [183].

Feasibility Study

We expect a threshold to exist above which the average response time (the time between providing a business case and being returned the business case with rules applied) for our SPIN-based CBR is lower compared to SPIN-based NCBRs. For this purpose, we derive NCBR ontologies from the CBR ontologies in the same fashion as for the $\mathcal{F}LORA$-2-based NCBRs. Consequently, for each single fully instantiated CBR model we have a context-aware ontology using subsumption (*SCtx*) and a context-unaware ontology using subsumption (*SNoCtx*). Ontologies using object properties performed poorly compared to subsumption ontologies in initial tests and thus are excluded from the feasibility study.

A generator is needed to create context models of various sizes and complexity and their corresponding ontology variants. Furthermore, random interest specifications need to be generated in order to determine the suspected threshold. Consequently, we constructed a generator accepting five parameters determining the size and complexity of the generated ontologies: (1) number of parameters, i.e., instances of `Parameter`, (2) parameter value hierarchy depth, i.e., number of levels of `covers`, (3) parameter value hierarchy width, i.e., instances of `covers` at one level, (4) context density, i.e., the percentage of potential contexts actually instantiated, and (5) number of rules per context.

For the tests we used Intel Core i7-4770s with 16 GB RAM. We ran the generator for different configurations covering number of parameters (2-5), parameter value hierarchy depth (1-2) and width (2-5), context density (25 %, 50 %, 75 %), and rules per context (12, 25, 50, 100). The configurations were chosen in such a way that we tested up to about 12,000 contexts. Ontology metrics for two sample context models are depicted in Table 4.2. We measured the time for loading and for SPIN inference

TABLE 4.2: Ontology variant metrics for two sample context models using SPIN.

Ontology Variant	# Contexts	# Rules	# OWL Axioms	Size
SCtx	16	160	3,300	53.7 kB
SNoCtx			4,200	78.7 kB
SCtx	1,800	180,000	2,200,000	53 MB
SNoCtx			4,300,000	111 MB

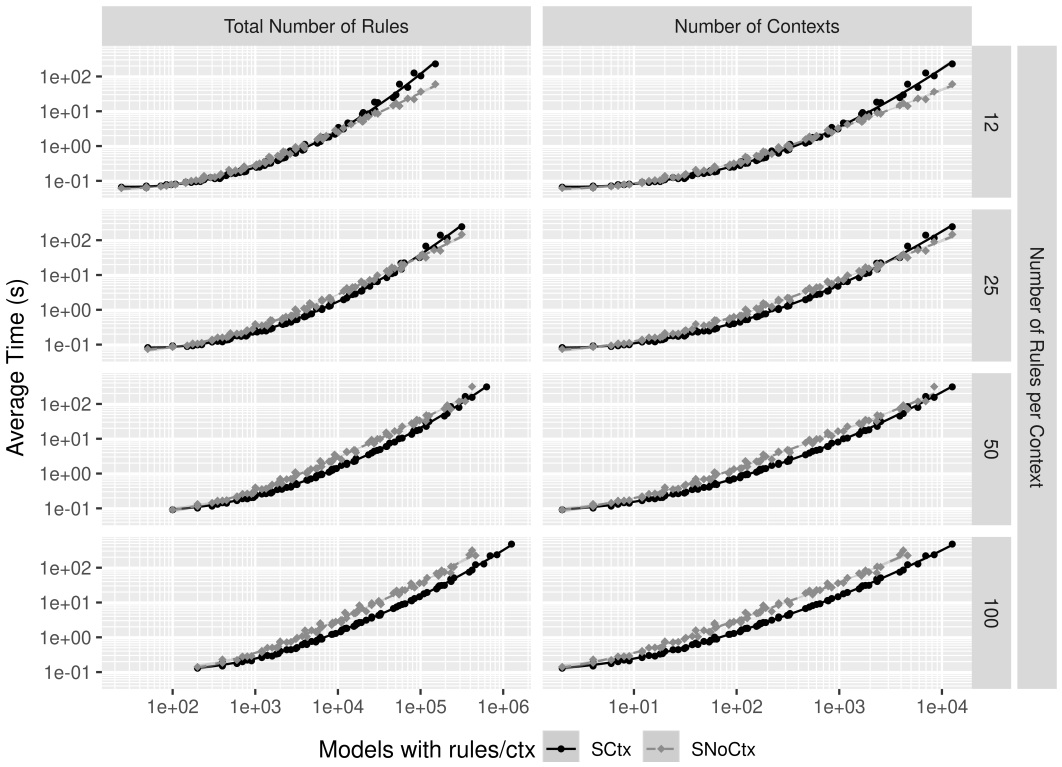


FIGURE 4.3: Average response time vs. total number of rules grouped by rules per context for our SPIN-based CBR prototype.

preparation as well as the average response time for 25 randomly generated interest specifications for SCtx and SNoCtx.

Regarding loading time and inference preparation times we expect SCtx to perform best as its ontology variants are the smallest and its rules are the simplest, i.e., they contain the fewest terms. Analyzing the loading times for the different ontology variants in detail showed that the average time mainly depends on the total number of rules ($totalRules$). The time (in seconds) necessary for loading an SCtx ontology variant can be predicted using $avgTime = 2.15 \times 10^{-4} \times totalRules$. This linear regression has an R^2 value of 0.99, i.e., 99 % of the variance in loading time is explained by it. SNoCtx takes about twice as long. The time needed for inference preparation also mainly depends on the number of rules: SCtx $avgTime = 1.6 \times 10^{-4} \times totalRules$ ($R^2 = .99$), SNoCtx about thrice as long. Consequently, the SCtx ontology variant is faster considering loading and inference preparation confirming our expectation.

Considering the average response time, we expect SCtx to outperform SNoCtx for larger number of rules and contexts. Figure 4.3 depicts the average response time versus the total number of rules (left column) and versus the number of contexts (right column) grouped by the number of rules per context. Note that due to the additional expressions in rules for SNoCtx, tests of it ran out of memory when using more than 450,000 rules. The left column in Figure 4.3 reveals that the average response time of context-unaware ontologies only slightly depends on the number of rules per context. For SCtx, on the other hand, there is a strong dependency. The threshold above which SCtx's average response time surpasses SNoCtx's is about 40 rules per context. For rules-per-context ratios below 40, the additional administrative effort seems to outweigh the benefits. Depicting the average response time versus the number of contexts (right column) we see that SCtx and SNoCtx for 25 rules per context have almost identical average response times up to approximately 3,000 contexts. Furthermore, we can see that the more contexts are employed the larger the gap between the average response time of SCtx and SNoCtx becomes.

In conclusion, SCtx outperforms SNoCtx in any case regarding loading time and inference preparation time. Regarding the average response time, SCtx outperforms SNoCtx when more than 40 rules per context are used.

4.3 Database-backed CBR Prototype[4]

In this section we will discuss a prototypical realization of database-backed CBRs for efficient and scalable contextualized rule execution.

It seems obvious that CBRs can speed-up business rule execution, because with contextualized rule execution, (1) the number of atoms in the body of rules gets smaller, (2) the number of rules to be executed (in one rule engine call) gets smaller, (3) the number of facts to be considered (in one rule engine call) gets smaller.

But, as the first experiments (see Section 4.2.1 and 4.2.3) have shown, it is far from trivial to realize this speed-up due to the potential performance overhead of contextualized rule execution.

In this section we will show how the potential performance gains of contextualized rule execution can be realized based on a mix of design choices. These design choices are the result of iterative experimentation and represent our knowledge of how to use $\mathcal{FLORA}$-2 for efficient contextualized rule execution (actually, we are using ErgoAI, the commercial variant of $\mathcal{FLORA}$-2, for this prototype):

- *Batch-enabled rules* are executed in batch, instead of separately for each case (single-case rules), in order to avoid the performance overhead for (i) determining a case-specific context for each case separately (ii) executing rules separately for each case.

- Facts are stored in a relational *database*, instead of in fact files, in order to avoid reading facts from files (which encompasses rather expensive F-Logic parsing and compilation as Prolog programs).

- Simple and context-independent reasoning tasks are realized in the database as (materialized) views using (recursive) SQL, for example, to determine parameter values, reflexive-transitive closure of value hierarchies, and context hierarchies.

[4]The database-backed CBR prototype was mainly developed by Bernd Neumayr; the performance evaluation mainly conducted by myself, Felix Burgstaller.

Materialized View userSituationCtx

userSit	interest	flightPhase	*interestVal*	*flightPhaseVal*
u1	Boeing_737	onground	*landplane*	*onground*
u2	Boeing_747	onground	*landplane*	*onground*
u3	Bell_Augusta	arrival	*helicopter*	*arrival*

Materialized View notamCtx

notam	feature	status	*eventScenarioVal*
n1	runway	limited	*runwayClosure*
n2	runway	closed	*runwayClosure*
n3	obstruction	tree	*obstruction*

Materialized View concreteCtx

data-related	situation-related	
eventScenarioVal	*interestVal*	*flightPhaseVal*
runwayClosure	*landplane*	*onground*
obstruction	*landplane*	*onground*
runwayClosure	*helicopter*	*arrival*
obstruction	*helicopter*	*arrival*

Table aircraft

interest	aircraftType	aircraftWeight
Boeing_737	landplane	194700
Boeing_747	landplane	735000
Bell_Augusta	helicopter	

Table weightRestr

notam	weightRestr
n1	500000

FIGURE 4.4: Sample relational representation of concrete contexts, user situations, data items (NOTAMs), and related data; derived parameter values are shown in slanted font.

- The schema of the database is *context-class-specific* and not generic. This makes database querying as part of rule execution (SQL queries are part of rule bodies) more efficient.
- Finally, we found that in many scenarios, including SemNOTAM, a case relates some data item to some user situation or task. Hence, we distinguish *data-related and situation-related parameters*. The system then separately derives parameter values for data items and for user situations. This allows to postpone the combination of user situations and data items to the execution of the rules for a concrete context.

The remainder of this section is organized as follows. First, we introduce database-backed CBRs along our running example. Second, as a prerequisite for the comparative evaluation, we describe database-backed NCBRs. Third, we discuss a preliminary performance comparison which will already give indications of performance gains. Fourth, we develop a test data generator and conduct and discuss performance studies.

4.3.1 Database-backed CBRs

We limit our attention to CBR use cases where *data items* (for example a NOTAM) and *user situations* (for example an aircraft pilot preparing for a specific flight) are to be matched. In terms of domain-specific CBR models, each combination of data item and user situation constitutes a business case (or *case* for short). We assume that the parameter values for one set of parameters (for instance `eventScenario`) can be derived from the data item and the values for another set of parameters (for instance `interest` and `flightPhase`) from the user situation. The former set of parameters is referred to as data-related parameters and the latter is referred to as situation-related parameters. A *concrete context* combines situation-related parameter values and data-related parameter values. When executing a concrete context, the system imports user situations and data items with these situation-related and data-related parameter values, respectively. Local to a context at execution time, data items and user situations are combined to cases. Every case belongs to exactly one concrete context. In contrast to the conceptual CBR model and the proof-of-concept prototype realizations in the previous section, business cases do not get an object identifier but are instead identified by data item and user situation.

LISTING 4.11: ErgoAI program for executing concrete context ⟨landplane, onground, runwayClosure⟩.

```
//---- DB IMPORT ----------------------------------
?n:NOTAM, ?n[feature->?f,status->?s] :-  pg1[query(q1,['SELECT notam, feature, status
↪  FROM notamCtx WHERE eventScenarioVal=''runwayClosure''''], [?n,?f,?s])]@\sql.
?u:UserSituation,?u[interest->?i,flightPhase->?f]:- pg1[query(q3,['SELECT ispec,
↪  interest, flightPhase FROM usersituationCtx WHERE interestVal=''landplane'' AND
↪  flightPhaseVal=''onground''''],[?u,?i,?f])]@\sql.
?i:Aircraft, ?i[type->?t,weight->?w] :- ?_[interest->?i],pg1[query(q1,['SELECT
↪  aircraftType, aircraftWeight FROM aircraft WHERE interest =
↪  ''',?i,''''],[?t,?w])]@\sql.
?n[weightRestriction->?v] :- pg1[query(q2,['SELECT notam, weightRestr FROM weightRestr
↪  where notam IN (SELECT notam FROM notamCtx WHERE
↪  eventScenarioVal=''runwayClosure'')'], [?n,?v])]@\sql.
//---- RULES ----------------------------------
/*R3*/ relevant(?i,?n) :- ?i:UserSituation, ?n:NOTAM.
/*R5*/ littleImportant(?i,?n) :- ?i:UserSituation[interest->?a:Aircraft], ?a[weight->?w],
↪  ?n:NOTAM[status->limited,weightRestriction->?wLimit], ?w < (?wLimit - 1000).
/*R6*/ highlyImportant(?i,?n) :- ?i:UserSituation, ?n:NOTAM, ?n[status->closed].
//---- QUERIES ----------------------------------
?- relevant(?i,?n).
?- littleImportant(?i,?n).
?- highlyImportant(?i,?n).
```

Example 4.2. Figure 4.4 shows a fragment from a relational representation of a CBR. There are three user situations, `u1`, `u2`, and `u3` in `userSituationCtx` with their asserted properties, `interest` and `flightphase`, together with derived parameter values, `interestVal` and `flightPhaseVal`. Furthermore, there are three data items, `n1`, `n2`, and `n3` in `notamCtx`, with asserted properties, `feature` and `status`, and derived parameter value `eventScenarioVal`. There are two parameter values for data-related parameter `eventScenario`, namely ⟨runwayClosure⟩ with data items `n1` and `n2` and ⟨obstruction⟩ with data item `n3` and two combinations of parameter values for situation-related parameters `interest` and `flightPhase`, namely ⟨landplane, onground⟩ with user situation `u1` and `u2` and ⟨helicopter, arrival⟩ with user situation `u3`. Combining data-related parameter values and situation-related parameter values in materialized view `concreteCtx` the system derives four concrete contexts. Within each concrete context the relevant user situations and data items are combined, resulting in nine cases in total (these are not shown in the relational representation, this combination only occurs at runtime): concrete context ⟨landplane, onground, runwayClosure⟩ with user situations `u1` and `u2` and data items `n1` and `n2` which combine to four cases `(u1,n1)`, `(u1,n2)`, `(u2,n1)`, `(u2,n2)`; concrete context ⟨landplane, onground, obstruction⟩ with user situation `u1` and `u2` and data item `n3` which combine to two cases `(u1,n3)` and `(u2,n3)`; concrete context ⟨helicopter, arrival, runwayClosure⟩ with user situation `u3` and data items `n1` and `n2` which combine to two cases `(u3,n1)` and `(u3,n2)`; concrete context ⟨helicopter, arrival, obstruction⟩ with user situation `u3` and data item `n3` which combine to case `(u3,n3)`.

Concrete contexts are the units of rule execution. For each concrete context the system derives the set of relevant contexts, includes their rules, and executes them to match the context's data items and user situations. Single-case rules, as employed in the previous sections, are meant to be executed on each case separately and, thus, do not come with variables for user situation and data item. For batch execution these rules are translated (by hand) to batch-enabled rules by adding variables for user situation and data item.

Example 4.3. For executing concrete context ⟨landplane, onground, runwayClosure⟩ the system first derives the set of relevant contexts (compare Figure 3.5): ⟨landplane, onground, runwayClosure⟩ with rules R5 and R6, context ⟨aircraft, onground, closure⟩ with rule R3, and empty context ⟨allInterests, allFlightPhases, allEventScenarios⟩. For batch-enabled rule execution, rules R3, R5, and R6 are translated to their batch-enabled counterparts (see Listing 4.11), lines 7 to 9.

These batch-enabled rules are inserted into the knowledge base together with rules for importing facts from the relational database. Import rules map the result of Structured Query Language (SQL) queries to F-Logic facts. These SQL queries are parameterized with the context's parameter values to only import relevant facts.

Example 4.4. Importing NOTAMs with parameter value `runwayClosure` from the database together with related facts specifying a weight restriction and the mapping to F-Logic facts is specified by rules (lines 2 and 5 in Listing 4.11). Importing user situations with parameter values `landplane` and `onground` together with related aircrafts is specified in a similar way (lines 3 and 4 in Listing 4.11). Rule execution is triggered by querying predicates `relevant`, `littleImportant`, and `highlyImportant` (lines 11–13 in Listing 4.11).

For each concrete context an ErgoAI program is generated and executed and the outputs (i.e., the results of the queries) are collected.

Example 4.5. For each concrete context, that is, for each tuple from materialized view `concreteCtx` in Figure 4.4, the system generates and executes an ErgoAI program similar to the one shown in Listing 4.11. The results of the queries on predicates `relevant`, `littleImportant`, and `highlyImportant` from the four programs are collected in main memory (and may be stored in the database for documentation or further processing or distribution).

4.3.2 Database-backed NCBRs

We want to demonstrate the performance gains in rule execution facilitated by database-backed CBRs. To this end, we first describe an efficient approach to database-backed NCBRs with which to compare.

For database-backed NCBRs we make the same design choices as for database-backed CBRs (see beginning of this section): batch-enabled rules, facts are stored in database in non-generic schemas, simple context-independent reasoning tasks are realized in the database, including the derivation of data-related and situation-related parameters.

Let us now describe non-contextualized rule execution by its difference to contextualized rule execution as described in the previous subsection.

The overall approach to non-contextualized database-backed rule execution is to have a single ErgoAI program which is executed once. The non-contextualized database-backed ErgoAI program imports facts from the same database as does contextualized rule execution. It imports all the facts at once and it imports the derived parameter values as well as the parameter value hierarchies. In this way, non-contextualized rule execution benefits from our design choice to realize simple reasoning tasks, namely the derivation of parameter values, in the database.

Example 4.6. The import of all NOTAMs together with derived `eventScenarioVal` and the import of all weight restrictions is specified in rules (lines 8 and 10 in Listing 4.12). In comparison to contextualized database imports the SQL queries in the rule bodies do not have a where-clause. The reflexive transitive closure of the value hierarchy

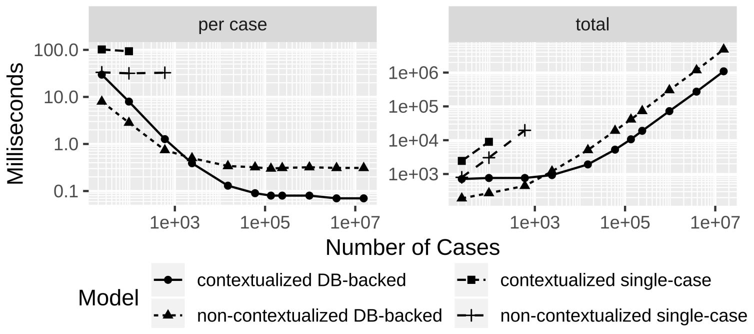


FIGURE 4.5: Results of a preliminary performance comparison, varying the number of cases from 24 to 15,360,000, with a fixed number of concrete contexts and rules based on the SemNOTAM scenario. Contextualized and non-contextualized single-case rule execution cease to produce valid results with 600 cases and 2400 cases, respectively. Starting at 2400 cases, contextualized database-backed rule execution outperforms non-contextualized database-backed rule execution.

of parameter `eventScenario` is imported from the database and mapped to predicate `eventScenarioTR` which is later used in rule bodies to limit the scope of rules.

Batch-enabled non-contextualized rules are similar to batch-enabled contextualized rules but have additional atoms in their body effectively limiting the scope of the rules as if they were executed only for some context (compare Section 4.2.1).

Example 4.7. Rule R3, `relevant.`, in context ⟨landplane, onground, runwayClosure⟩ and its batch-enabled form at line 7 in Listing 4.11 is made non-contextualized by adding parameter checks to the rule body (see line 15 in Listing 4.12). For the user situation, the `interestVal` must be the same as or a descendant of `aircraft` and the `flightPhaseVal` must be the same or a descendant of `onground`. For the NOTAM, the `eventScenarioVal` must be the same or a descendant of `closure`.

4.3.3 Preliminary Performance Comparison

For a preliminary performance comparison we take the CBR model, including context model, rules, and cases, from the feasibility study in Section 4.2.1. To scale the experiment in the number of cases we generate cases and additional facts from the existing ones. We then test the performance of contextualized database-backed rule execution as well as of non-contextualized rule execution. Performance is measured as the time of rule execution (per case and in total) including querying the database and including parsing the output of ErgoAI. We also test the performance of contextualized single-case and non-contextualized single-case rule execution (from the feasibility study in Section 4.2.1). For this purpose, we make slight adaptations to the $\mathcal{FLORA}$-2-based CBR prototype to make the executions comparable: Since database-backed rule execution has no precompiled code, we employ the cold-start variant of our $\mathcal{FLORA}$-2-based CBR prototype. Furthermore, the database-backed rule execution loads only cases relevant to a concrete context. Thus, instead of loading all user situations and NOTAMs to module `bc` and only exchanging the `SemNOTAMCase`, we erase module `bc` and load only the current NOTAM, user situation, and `SemNOTAMCase`. The time measured includes the exchange of `SemNOTAMCase`, rule execution, and parsing the results provided by the $\mathcal{FLORA}$-2 CLI. This is the cause for slower execution times reported in Figure 4.5 compared to Figure 4.2.

LISTING 4.12: Non-Conctextualized ErgoAI program.

```
//---- DB IMPORT ----------------------------------
eventScenarioTR(?c,?p) :- pg1[query(q1,['SELECT child, parent FROM
↪  eventScenarioT'],[?c,?p])]@\sql.
eventScenarioTR(?c,?c) :- pg1[query(q1,['SELECT concept FROM
↪  eventScenarioConcept'],[?c,?p])]@\sql.
interestTR(?c,?p) :- pg1[query(q1,['SELECT child, parent FROM interestT'],[?c,?p])]@\sql.
interestTR(?c,?c) :- pg1[query(q1,['SELECT concept FROM interestConcept'],[?c,?p])]@\sql.
flightPhaseTR(?c,?p) :- pg1[query(q1,['SELECT child, parent FROM
↪  flightPhaseT'],[?c,?p])]@\sql.
flightPhaseTR(?c,?p) :- pg1[query(q1,['SELECT concept FROM
↪  flightPhaseConcept'],[?c,?p])]@\s
?n:NOTAM, ?n[feature->?f,status->?s,eventScenarioVal->?e] :- pg1[query(q1,['SELECT notam,
↪  feature, status, eventScenarioVal FROM notamCtx'],[?n,?f,?s,?e])]@\sql.
?u:UserSituation, ?u[interest->?i,flightPhase->?f, flightPhaseVal->?fv, interestVal->?iv]
↪  :- pg1[query(q3,['SELECT ispec, interest, flightPhase, flightPhaseVal, interestVal
↪  FROM ispecCtx'],[?u,?i,?f,?fv,?iv])]@\sql.
?n[weightRestriction->?v] :- pg1[query(q2,['select notam, weightRestr from
↪  weightRestr'],[?n,?v])]@\sql.
?i:Aircraft, ?i[type->?t,weight->?w] :- pg1[query(q1,['select a.interest, a.aircraftType,
↪  a.aircraftWeight from aircraft a'],[?i,?t,?w])]@\sql.
//---- RULES --------------------------------------
/*R1*/ highlyImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM, ?n[status->?s],
↪  eventScenarioTR(?n.eventScenarioVal,obstruction),
↪  interestTR(?u.interestVal,aircraft), \naf ?s = tree.
/*R2*/ littleImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM, ?n[status->?s], ?s = tree,
↪  eventScenarioTR(?n.eventScenarioVal,obstruction),
↪  interestTR(?u.interestVal,aircraft).
/*R3*/ relevant(?u,?n) :- ?u:UserSituation, ?n:NOTAM,
↪  interestTR(?u.interestVal,aircraft), flightPhaseTR(?u.flightPhaseVal,onground),
↪  eventScenarioTR(?n.eventScenarioVal,closure).
/*R4*/ highlyImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM[status->tree],
↪  interestTR(?u.interestVal,helicopter),
↪  eventScenarioTR(?n.eventScenarioVal,obstruction).
/*R5*/ littleImportant(?u,?n) :-  ?u:UserSituation, ?n:NOTAM,
↪  interestTR(?u.interestVal,landplane), flightPhaseTR(?u.flightPhaseVal,onground),
↪  eventScenarioTR(?n.eventScenarioVal,runwayClosure), ?u[interest->?a:Aircraft],
↪  ?a[weight->?w], ?n[status->limited,weightRestriction->?wLimit], ?w < (?wLimit -
↪  1000).
/*R6*/ highlyImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM, ?n[status->closed],
↪  interestTR(?u.interestVal,landplane), flightPhaseTR(?u.flightPhaseVal,onground),
↪  eventScenarioTR(?n.eventScenarioVal,runwayClosure).
/*R7*/ littleImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM,
↪  interestTR(?u.interestVal,landplane),
↪  eventScenarioTR(?n.eventScenarioVal,specialPort).
/*R8*/ highlyImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM,
↪  interestTR(?u.interestVal,aircraft), flightPhaseTR(?u.flightPhaseVal,onground),
↪  eventScenarioTR(?n.eventScenarioVal,aerodromeEquipment).
/*R9*/ highlyImportant(?u,?n) :- ?u:UserSituation, ?n:NOTAM,
↪  interestTR(?u.interestVal,landplane),
↪  eventScenarioTR(?n.eventScenarioVal,aerodromeBeaconStatus).
//---- QUERIES ------------------------------------
// -- same as in Listing 2.11
```

Table 4.3 and Figure 4.5 show the results of these experiments. The results indicate that batch-enabled database-backed rule execution always (at least in this scenario) outperforms single-case file-based rule execution. In addition, for larger number of cases the two single-case variants do not produce valid results probably because of the many dynamic updates to the ErgoAI knowledge base. For the comparison between contextualized and non-contextualized rule execution the results are also intuitive: contextualized rule execution has some performance overhead because of multiple rule engine calls. With increasing number of cases contextualized rule execution outperforms non-contextualized rule execution.

TABLE 4.3: Preliminary performance comparison of rule execution in DB-backed CBRs and NCBRs.

Cases	NCBR DB-backed Total Time (ms)	NCBR Single-case Total Time (ms)	CBR DB-backed Total Time (ms)	CBR Single-case Total Time (ms)	NCBR DB-backed Time per Case (ms)	NCBR Single-case Time per Case (ms)	CBR DB-backed Time per Case (ms)	CBR Single-case Time per Case (ms)
24	191	798	713	2444	7.96	33.25	29.71	101.83
96	272	3044	763	8962	2.83	31.71	7.95	93.35
600	446	19578	764	∞	0.74	32.63	1.27	∞
2,400	1,199	∞	929	∞	0.50	∞	0.39	∞
15,000	5,123	∞	1,924	∞	0.34	∞	0.13	∞
60,000	19,097	∞	5,261	∞	0.32	∞	0.09	∞
135,000	41,133	∞	10,636	∞	0.30	∞	0.08	∞
240,000	73,242	∞	18,955	∞	0.31	∞	0.08	∞
960,000	302,611	∞	72,402	∞	0.32	∞	0.08	∞
3,840,000	1,177,008	∞	272,793	∞	0.31	∞	0.07	∞
15,360,000	4,817,084	∞	1,089,858	∞	0.31	∞	0.07	∞

4.3.4 Performance Studies

In this subsection we describe conducted performance studies, comparing contextualized and non-contextualized database-backed rule execution. We focus on database-backed rule execution as our preliminary comparative evaluation clearly indicated that database-backed outperforms single-case rule evaluation. Our preliminary comparative evaluation also indicated that contextualized database-backed rule execution comes with performance overhead (especially for small and simple rule sets) but scales much better than non-contextualized rule execution. In order to compare the performance of contextualized and non-contextualized database-backed rule-execution in more depth, we developed a CBR generator (a test data generator) which allows to scale the CBR's size in different dimensions. The generated CBR, including generated rules and facts, is stored in a database from which both the non-contextualized rule execution as well as the contextualized rule execution read.

Test Data Generation

The CBR test data generator produces CBRs with the following characteristics:

Every case relates a user situation with seven properties to a data item also with seven properties making a total of 14 properties. For each such property there is a class of 3100 objects, which acts as range of the property, and a concept hierarchy of 31 concepts forming a perfect binary tree with five levels (with one concept at level 0, two concepts at level 1, four concepts at level 2, eight concepts at level 3, and 16 leaf concepts at level 5). Each object refers to one of these concepts and has nine numeric attributes, which have integers 0..9 as range; values are randomly selected.

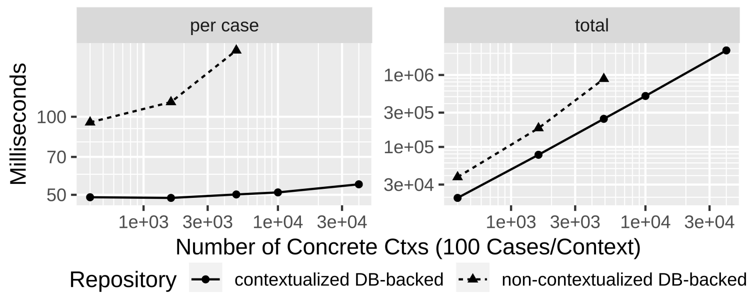


FIGURE 4.6: Varying the number of concrete contexts with a fixed number of 100 cases per concrete context in repositories with four data-related parameters and three user-related parameters. Non-contextualized rule execution runs out of memory with 100 concrete contexts (10,000 cases) as well as with 400 concrete contexts (40,000 cases). With 49 concrete contexts (4,900 cases) contextualized rule execution outperforms non-contextualized rule execution by a factor of 3.59.

Parameters are associated with properties; the parameter value for a given case is determined as the concept referred to by the object referred to by the property (that is, the above described concept hierarchies act as parameter value hierarchies). Data items and user situations refer via their properties only to objects that refer to a leaf concept, this means that concrete contexts (i.e., parameter value combinations of business cases) only have leaf parameter values.

For each concrete context there are four relevant contexts (i.e., parameter value combinations which act as scope of rules), one with parameter values at level 0, one with parameter values at level 1, one with parameter values at level 2, and one that corresponds to the concrete context (with leaf parameter values). Predicates are shared across contexts, there are twelve auxiliary predicates and three output predicates, the latter are queried and output during rule execution. Every relevant context contains one randomly generated rule per auxiliary predicate, matching data items and user situations based on equality of a numeric attribute of the object referred to by a property of the data item and a numeric attribute of the object referred to by a property of the user situation. Every relevant context further contains one randomly generated rule per output predicate, building the conjunction of two auxiliary predicates. In total, every concrete context has 60 rules (15 from each relevant context).

Experiments

In our experiments we scaled the rule execution task regarding (i) the number of contexts with a fixed number of cases per context thus increasing the total number of cases as well as the total number of rules, see Figure 4.6, 4.7, and 4.8, (ii) the number of cases per context with a fixed number of concrete contexts and a fixed number of parameters, thus increasing the total number of cases but with a fixed total number of rules, see Figure 4.9 (and also Figure 4.11), and (iii) the number of parameters, with a fixed number of cases and a fixed number of concrete contexts, thereby increasing the number of atoms in non-contextualized rule bodies see Figure 4.10. Increasing the number of contexts increases the total number of rules because our context-rule-case generator generates a fixed number of rules per context. An overview of the experiments and their results is given in Table A.1 and A.2 in Appendix A.

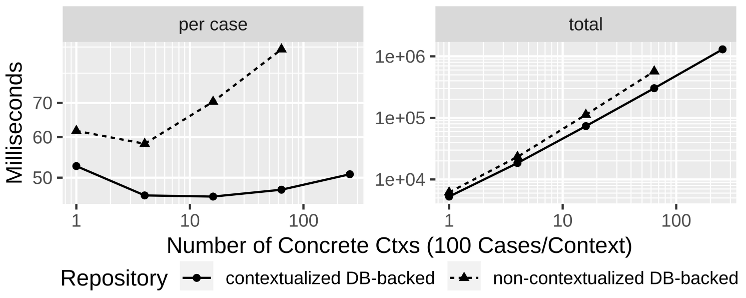


FIGURE 4.7: Varying the number of concrete contexts with a fixed number of 100 cases per concrete context in repositories with one data-related parameter and one user-related parameter. Non-contextualized rule execution runs out of memory with 256 concrete contexts (25,600 cases). With 64 concrete contexts (6,400 cases) contextualized rule execution outperforms non-contextualized rule execution by a factor of 1.88.

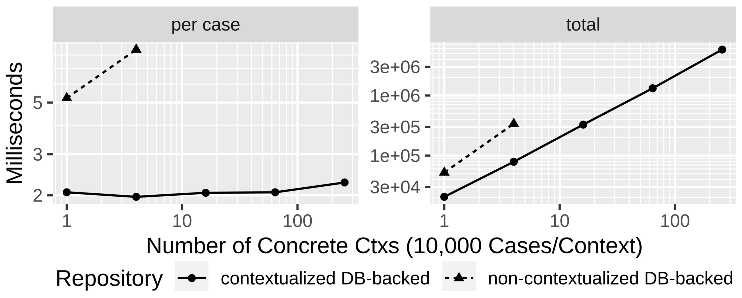


FIGURE 4.8: Varying the number of concrete contexts with a fixed number of 10,000 cases per concrete context in repositories with one data-related and one user-related parameter. Non-contextualized rule execution runs out of memory with 16 concrete contexts (160,000 cases), 64 concrete contexts (640,000 cases), and 256 concrete contexts (2,560,000 cases). With four concrete contexts (40,000 cases) contextualized rule execution outperforms non-contextualized rule execution by a factor of 4.28.

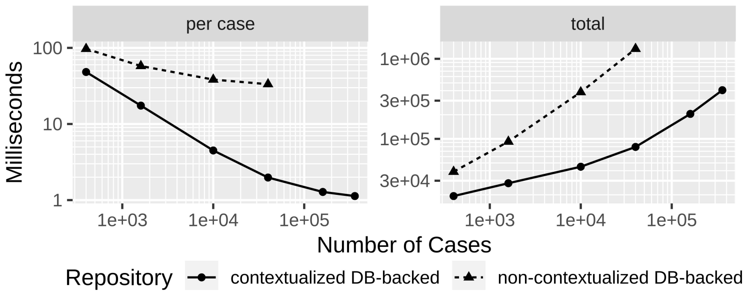


FIGURE 4.9: Varying the number of cases in repositories with a fixed number of four concrete contexts and three user- and four data-related parameters. Non-contextualized rule execution runs out of memory with 160,000 cases (40,000 cases per context) as well as with 360,000 cases (90,000 cases per context). With 40,000 cases (10,000 cases per context) contextualized rule execution outperforms non-contextualized rule execution by a factor of 16.81.

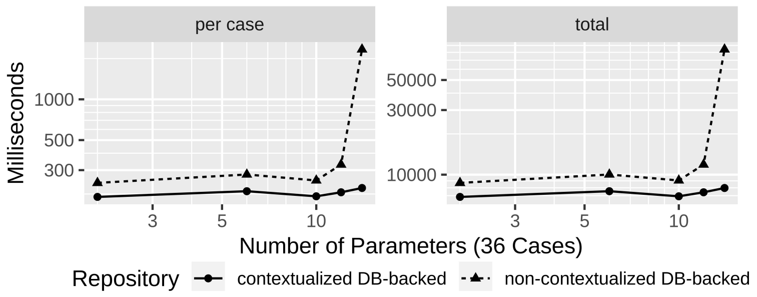


FIGURE 4.10: Varying the number of user- and data-related parameters in repositories with four concrete contexts and 36 cases (9 cases per context). Up to 10 parameters, the performance is not affected by an increasing number of parameters. With 14 parameters, however, non-contextualized rule execution slows down rapidly and is outperformed by contextualized rule execution by a factor of 10.55.

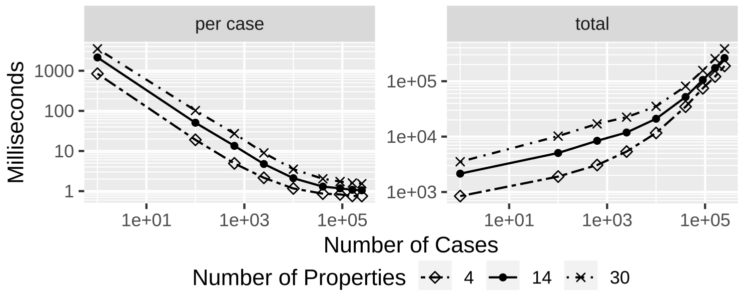


FIGURE 4.11: Varying the number of cases in contextualized rule repositories with a single context and two parameters for three different case sizes (in terms of the number of properties). Up to 250,000 cases and for all case sizes, performance measured in ms per case improves with an increase in the number of cases: with 100 cases and 250,000 cases, rule execution on cases with four properties takes 19.01 ms and 0.75 ms per case, respectively, and on cases with 30 properties takes 102.07 ms and 1.54 ms, respectively.

With a further series of experiments (see Figure 4.11 and Table A.3 in Appendix A) we have analyzed the impact of the size of cases, in terms of the number of properties of data items and user situations, on contextualized rule execution performance.

Discussion

Contextualized rule execution in database-backed CBRs scales well with the number of contexts, the number of cases, and the number of parameters. Database-backed CBRs outperform database-backed NCBRs regarding rule execution for all but very small repositories.

Large rule repositories (for instance, 100 concrete contexts with a total of about 5000 rules and 100 cases per concrete context) render contextualized rule execution the only of the two options since with non-contextualized rule execution ErgoAI runs out of memory. For medium-sized rule repositories (for instance, seven parameters, four concrete contexts and 40,000 cases), contextualized rule execution outperforms non-contextualized rule execution by an order of magnitude (for example, factor 16.81).

In the conducted experiments, in order to simplify the interpretation of experimental results, structure, rules, and cases of generated CBRs are highly homogeneous, for instance, every parameter value hierarchy is a perfect binary tree with 31 nodes.

CBRs in practice tend to be far more heterogeneous. Performance studies taking into account these heterogeneities are subject to ongoing work.

4.4 Conclusion

We presented three proof-of-concept prototypes realizing CBR models using different rule languages. We employed the rule languages $\mathcal{F}LORA$-2, Datalog$^{\pm}$/Vadalog, and SPIN for realization, enabling, when storing business rules in the respective rule language, direct rule execution. Feasibility studies investigating the performance of rule execution for the CBR realizations showed that the actualization of the expected speed-up compared to rule execution in NCBRs is far from trivial.

Therefore, we presented a database-backed CBR realization enabling efficient rule execution that scales well with the number of contexts, number of business cases, and number of parameters. Performance evaluations indicate that our database-backed CBR incurs, regarding rule execution, additional overhead for very small repositories but outperforms non-contextualized database-backed rule execution for larger ones.

Chapter 5

Modification Operations for CBR Maintenance

The increasing number, complexity, and variability of business rules in today's enterprises introduces the need for their effective and flexible maintenance and acquisition. To this end, an appropriate organization of business rules is needed. In several research fields, such as the semantic web, library science, and data tailoring, effective organization of knowledge is enabled by contexts. We previously proposed the generic CBR model modeling the organization of business rules along contexts. We complement the generic CBR model's instantiation mechanisms by atomic and composed modification operations. Each atomic modification operation is associated with one of four roles: rule repository administrator, rule developer, user, and domain expert. This enables effective separation of tasks and responsibilities promoting efficient business rule maintenance and business rule acquisition. Regarding business rule acquisition, we consider business rule entering only – other activities of business rule acquisition are mostly independent of CBRs and thus not considered. Composed modification operations describe combinations of atomic modification operations relevant in practice. In addition to operations, we describe a business rule management process for CBRs. Furthermore, we show CBRs' eased business rule maintenance and business rule entering compared to NCBRs. We apply the proposed approach to the real-world use case SemNOTAM.

5.1 Introduction

Businesses operate under thousands of business rules. These numbers of business rules cannot be elicited at once, an incremental approach is necessary [43, 201]. Once elicited, many factors influence existing business rules or even require new business rules, for instance, regulations, organizational changes, actualization of business policies, customization, or new partnerships [143, 147, 176] – frequent changes to rules are probable where their volatility may differ from rule to rule [59, 176, 208]. Consequently, effective, efficient, flexible, and continuous BRM [27, 43, 77, 81, 83, 156, 166], in particular business rule maintenance and business rule acquisition, is necessary, promoting flexible and adaptable businesses [58, 75, 81, 135, 143, 156, 167, 176]. Especially, in uncertain and turbulent business environments, business flexibility is vital to maintain competitive advantage [125].

Since the generic CBR model for business rule organization (presented in Chapter 3) enabling efficient business rule execution (demonstrated in Chapter 4) is insufficient for business rule maintenance and acquisition, especially regarding the

This chapter is mainly based on the publication F. Burgstaller, B. Neumayr, C. G. Schuetz, and M. Schrefl. "Modification Operations for Context-Aware Business Rule Management". In: *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. 2017, pp. 194–203.

required flexibility, we extend the generic CBR model's instantiation mechanisms by modification operations. These modification operations facilitate efficient business rule maintenance and business rule entering in CBRs. In particular, we introduce atomic low-level model operations and composed model operations ensuring model consistency, i.e., they ensure that constraints introduced in the static CBR model, such as uniform parameters, remain satisfied. Furthermore, we investigate the assignment of operations to CBR roles, namely, repository administrator, rule developer, user, and domain expert. This enables task and responsibility separation, promoting effective and efficient business rule maintenance and business rule entering. Employing the presented operations as well as their assigned CBR roles, we propose a systematic and controlled approach to manage business rules throughout their lifecycle in CBRs, as required in the literature [27, 43, 81, 88, 135, 146, 159, 161, 167, 186, 201, 208]. All three main activities of BRM (c.f. Section 2.4.2), acquisition, maintenance, and execution, are supported by CBRs. Comparing CBRs and NCBRs, CBRs provide, in addition to faster business rule execution, eased business rule maintenance and business rule entering. To demonstrate our approach, we apply it to SemNOTAM.

The remainder of this chapter is organized as follows: Section 5.2 discusses atomic modification operations for CBR models, delineates CBR roles necessary for business rule maintenance and business rule entering, and assigns each atomic operation a responsible CBR role. Section 5.3 introduces: a basic set of composed modification operations for CBR models maintaining the consistency of CBR models and the business rules organized within as well as a CBR-specific BRM process. Section 5.4 compares the introduced composed operations with their semantic equivalents in NCBRs. Section 5.5 discusses related work and Section 5.6 concludes the chapter.

5.2 Atomic CBR Model Operations and CBR Roles

Fully instantiated CBR models (see for example Figure 3.3 and 3.6) contain seven kinds of context model elements which can be modified by *atomic operations*: context classes (M1) and their contexts (M0) as well as referenced business rules (M0); parameters (M1) and their parameter values (M0); and business case classes (M1) and their business cases (M0). Each CBR model element can be instantiated (created), updated, and destructed (deleted). The concrete meaning of updates differs between CBR model elements:

- Updating *business rules* (M0) changes a specific rule's definition.
- Updates of *context classes* (M1) regard the contexts class' specification, i.e., changing attributes, associations, or method implementations. Updates of *contexts* (M0) regard parameter values and context relationships.
- Updates of *parameter* (M1) mainly regard the implementation of the method `detParamValue` but a parameters class' specification, like domain-specific attributes and associations, can be modified as well. Updates of *parameter values* (M0) regard parameter value relationships.
- Updates of *business case classes* (M1) consider the class' specification, i.e., its describing properties and relations. Updates of *business cases* (M0) regard changes in the concrete values for the describing properties and relations.

The introduced atomic modification operations do not consider consistency of CBR models, for instance, deleting a parameter does not include removal of the parameter from the context class. As such, atomic modification operations on CBR model elements do not propagate to affected model elements. This also holds for case-specific contexts. Thus, any manual modifications of a case-specific context, for example, manual conflict resolution, are not affected by modifications on the respective CBR model. Nevertheless, executing `detCaseSpecificCtx` with the same `SemNOTAMCase` returns a new case-specific context with all modifications taken into account. Operations considering consistency of CBR models are discussed in Section 5.3.

Effective, efficient, and flexible business rule maintenance requires documentation of modifications, for example, documenting the motivations for modifications like in BMM (described in Section 2.2.4). A viable documentation option is versioning. Depending on the concrete CBR implementation, an external or an internal versioning system may be used. Internal versioning requires the classes at M2 to be complemented by attributes `versionM1`[1] and `versionM0`[2]. A simple approach is to use a global version counter. Any `caseSpecificCtx` then has the highest version number of its contributing contexts. The implementations of the context class' methods need to be adapted, for example, to use the most recent version or a specific version for historic business cases.

We assign the introduced atomic modification operations to *CBR roles*. To this end, we shortly review related work: The ninth principle of the Business Rules Manifesto states that business rules should be formulated, validated, verified, and managed by business people and not IT-people [43, 75, 81]. Boyer and Mili [27] distinguish: business analysts, bridging IT and business; rule analysts, business analysts with deep knowledge about the employed BRMS and BRM; rule architects, defining and maintaining structure of rule-based applications; rule writers, writing the technical rules; subject matter experts, defining business policies and application requirements, leading rule acquisition, et cetera; and rule administrators, managing rule authoring and deployment. Common to both resources is the distinction of business and technical roles.

For CBRs we identify four basic roles, namely, repository administrators, rule developers, domain experts, and users. Depending on the specific domain or application, these CBR roles may be specialized as needed, for instance, domain experts for helicopter flights. To enable separation of tasks and responsibilities promoting effective and efficient business rule maintenance and business rule entering, we assign each CBR model element to a single CBR role as shown in Figure 5.1. *Repository administrators* administrate context classes, contexts, parameters, and parameter values, i.e., they are responsible for the organization of the repository; *rule developers* administrate business rule sets and the corresponding conflict resolutions of a set of contexts, i.e., they are responsible for the concrete implementation of business rules; *domain experts* administrate business case classes and identify relevant business rules as well as their applying context; and *users* supply business cases for which a CBR determines relevant business rules. Each CBR role has read access to the complete CBR model, for example, a domain expert can investigate business rules of specific contexts. Furthermore, communication between CBR roles is required, for instance, users and domain experts provide feedback to rule developers who then can modify business rule sets accordingly. Similarly, rule developers may ask repository administrators to delete one of the contexts they are responsible for.

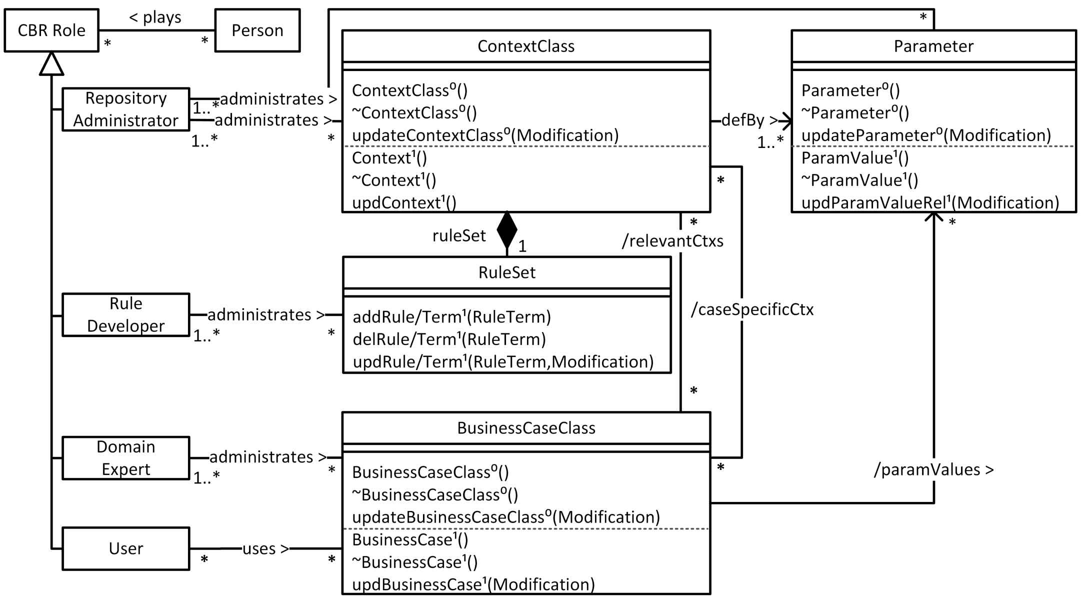


FIGURE 5.1: CBR Roles and their assigned CBR model elements and operations.

5.3 Composed CBR Model Operations

We combine atomic CBR model operations to more practicable composed CBR model modification operations (short: composed operations). The execution of composed operations can be (partially) automated. We distinguish composed operations for the purpose of (a) reorganization, i.e., the context model is modified but for any given business case executing the case-specific context retrieves the same results, and (b) modifying functionality, i.e., results for business cases may change (for instance, correcting misclassifications of NOTAMs).

The described composed operations ensure that CBR models move from one consistent state to another, i.e., all side-effects are taken care of. A side-effect is for instance that the removal of a rule from a context also removes it from all descendant contexts. Each composed operation defines a sequence of operations and the communication flow between CBR roles. Communicated are notifications containing the current composed operation, the concerned CBR model element, the executing person, the cause, and optionally the next operation to be executed. Repository administrators are notified about any modifications affecting context classes, contexts, parameters, or parameter values; rule developers are notified of any operations potentially affecting business rules associated with their contexts; and users are notified about modified business case classes. This enables the different CBR roles to react to modification operations affecting CBR model elements in their responsibility. An inappropriate reaction will be reflected in the retrieved results for given business cases and will cause users and domain experts to request composed operations correcting the inappropriate reaction.

Composed model operations need to fulfil certain properties similar to the ACID-properties [80] of database transactions. Composed as well as atomic modification operations are executed completely or not at all, i.e., they are atomic. Thus, any received notification must be acknowledged – if reactions (i.e. operations) are necessary, the notification is acknowledged after appropriate reactions have been performed.

To enable atomic execution of composed operations, each modification operation is executed on a copy of the corresponding CBR model. Once the composed operation has been successful, the old CBR model is exchanged for the modified CBR model. If the composed operation fails, this is the case if it takes too long to finish (user-defined timeout) or any of its atomic operations fails, for example, due to some runtime issues, the CBR model copy is deleted. This enables minimal downtime of business case evaluation. Nevertheless, until the current CBR model is replaced by the modified CBR model, business cases may retrieve outdated results. This approach can be considered a simple form of shadowing and could be enhanced by employing shadowing as proposed for recovery in databases [120, 162].

Regarding the remaining transactional properties we find: As described previously, composed operations ensure consistency of CBR models. Concerning isolation we limit our investigation in this chapter to one composed operation at a time; enabling concurrent composed operations, for instance, by drawing on locking mechanisms from databases like Gray's locking algorithm for hierarchical organized systems [78], is considered future work. Analogous to database transactions, composed operations need to be durable, i.e., finished CBR model operations have to persist even in case of failure. Therefore, we execute composed operations on CBR model copies and store CBR models in a non-volatile memory. An investigation of suitable recovery algorithms for CBRs from the domain of databases is considered future work.

In this chapter we consider basic composed operations for CBR models with only a single domain-specific context-class, i.e., operations instantiating or deleting context classes are not considered. We group composed model operations into CBR model maintenance operations, CBR model extension and refinement operations, and CBR model restriction and consolidation operations. For each composed operation we provide a textual description (where *cursive* operations refer to composed operations), a BPMN process model depicting the operation sequence and communication flow (where collapsed subprocesses refer to the respective composed operation), and an example utilizing our SemNOTAM use case as depicted Figure 3.5. To keep the process models compact, we model selected errors and timeouts only. Whenever an error or a timeout occurs the composed operation is assumed to have failed. Furthermore, we do not model compensation in case of failure, i.e., deleting the CBR model copy on which the composed operation has been performed.

5.3.1 CBR Model Maintenance Operations

Maintenance operations regard composed operations on rules, updating context hierarchies, context classes, and modifying business case classes.

Modify Rules

Modification of rules becomes necessary if: (a) Domain experts (or users) find that for a business case they submitted, additional rules hold, existing rules have become obsolete, or existing rules need adaption. As the relevant contexts and their parameter values are known, domain experts can inform the responsible rule developer. (b) A rule developer identifies optimization potentials by analysing his/her rules.

Once the necessity for modification of rules is known to the rule developer, he/she performs the corresponding atomic operation(s). Affected rule developers, i.e., rule developers for whom these operations incur changes in their contexts, are notified. These rule developers then may perform appropriate operations for their context(s),

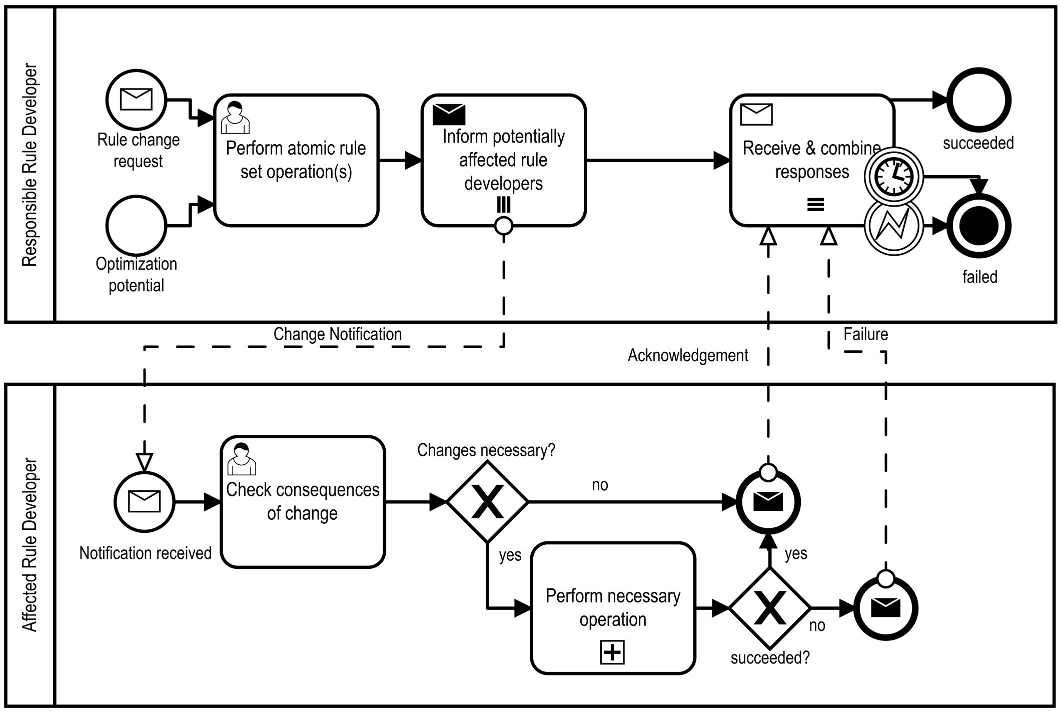

Responsible Rule Developer
Rule change request
Optimization potential
Perform atomic rule set operation(s)
Inform potentially affected rule developers
Receive & combine responses
succeeded
failed
Change Notification
Acknowledgement
Failure
Affected Rule Developer
Notification received
Check consequences of change
Changes necessary?
no
yes
Perform necessary operation
yes
no
succeeded?

FIGURE 5.2: BPMN model of the composed operation *modify rules*.

for instance, they may decide to maintain the original rule in their context or to specify necessary conflict resolutions. This takes care of rule dependencies as these can only exist within the same context or within a subtree of contexts. A BPMN representation of this process is given in Figure 5.2. The notification itself consists of the atomic operation, the concerned rule and its associated context, the ID of the rule developer, and the cause for modification. This composed operation can also be used to move rules to other contexts.

Example 5.1. A domain expert informs the rule developer responsible for context ⟨aircraft, allFlightPhases, obstruction⟩ (shown in Figure 3.5) that all NOTAMs concerning obstructions are highly important. Thus, the rule developer deletes rule `R2` and modifies `R1` to `highlyImportant`. Subsequently, the rule developer of context ⟨helicopter, allFlightPhases, obstruction⟩ is notified about the operation. Considering the operation, she determines that rule `R4` in her context has become superfluous and subsequently deletes it. She then asks the repository administrator to delete her now empty context. Once all necessary operations have successfully been performed, she acknowledges the original operation on rules `R1` and `R2`. Since no other child contexts exist, the composed operation succeeds.

Contextualize Rules

Contextualization of rules is required if: (a) a rule associated with a context is to be associated with a descendant context, or (b) a rule not associated with any context (either applying in all classes of situations or having its applying class of situations encoded in its rule body) is to be associated with a context. Contextualization of rules is employed within the composed operations *new context* and *split context*.

The responsible rule developer for the origin context sends the rule to the target context's rule developer and subsequently *deletes* it from the origin context. In this

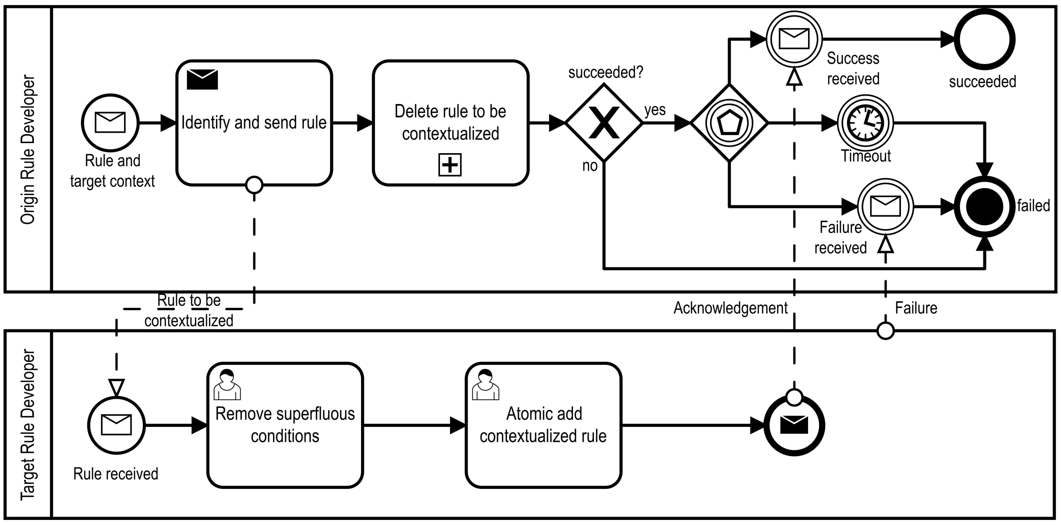


FIGURE 5.3: BPMN model of the composed operation *contextualize rules*.

case, potentially affected rule developers are rule developers for whom the rule is deleted from their contexts. Rule developers responsible for the target context or any of the target context's descendant contexts are not informed as the rule is still available to them after the operation succeeds. The target context's rule developer removes any conditions from the rule definition already encoded in her/his context (context knowledge) and adds the rule to her/his context. Subsequently the target context's rule developer acknowledges the operation. A graphical representation of this process is given in Figure 5.3.

Example 5.2. Domain experts informed us that rule R3 in ⟨aircraft, onground, closure⟩ (origin context) actually applies for landplanes and runway closures only, i.e, we need to contextualize rule R3 to context ⟨landplane, onground, runwayClosure⟩. The rule developer responsible for the origin context sends rule R3 to the target context's rule developer and subsequently *deletes* it. No changes to rule R3 are necessary and the rule developer of context ⟨landplane, onground, runwayClosure⟩ simply adds it. Once finished, the operation succeeds as no other affected contexts exist. Another example is given in Section 5.3.2.

Decontextualize Rules

Decontextualization of rules is necessary if a rule associated with a context is to be associated with an ancestor context preserving its semantics. This is the case within composed operation *merge context* and indirectly within composed operation *delete parameter*.

The origin context's parameter values need to be encoded in the rule's body where they are different from the target context's parameter values. The rule developer of the origin context sends the decontextualized rule to the responsible developer of the target context. Subsequently, the rule developer of the origin context deletes the rule and the rule developer of the target context adds the decontextualized rule. In this case, since decontextualization is semantics-preserving, no other rule developers are affected. If adding the decontextualized rule succeeds, the target context's rule developer acknowledges the operation. A BPMN representation of this process is depicted in Figure 5.4.

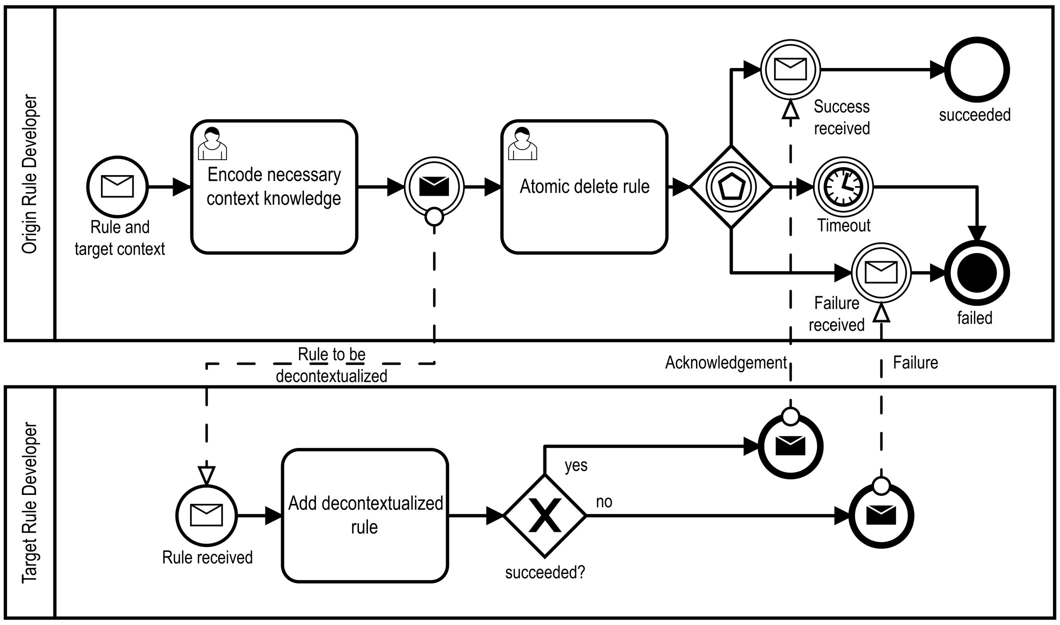


FIGURE 5.4: BPMN model of the composed operation *decontextualize rules*.

Example 5.3. Rule R7 is to be decontextualized to context ⟨allInterests, allFlightPhases, allEventScenarios⟩. This requires, for instance, adding the parameter value regarding the aircraft type to rule R7. Next, the rule developer of ⟨landplane, allFlightPhases, specialPort⟩ sends the decontextualized rule R7 to the rule developer of context ⟨allInterests, allFlightPhases, allEventScenarios⟩. Afterward she deletes it. The rule developer of the target context adds decontextualized rule R7 to his context and the decontextualization operation succeeds.

Update Context Hierarchy

Updates to the context hierarchy are necessary if domain experts (or users) find that invalid rules are applied for their business cases and an inspection of the context hierarchy by domain experts reveals invalid context relationships. If the context hierarchy is derived from parameter value hierarchies, the domain experts need to determine the cause in the parameter value hierarchies.

Once the cause is found, domain experts inform the repository administrator who then corrects the context or parameter value relationships respectively. Rule developers of affected contexts, i.e., contexts for which the super-context(s) change(s), are notified and can perform necessary operations, such as adding conflict resolutions. A BPMN model of the described process is given in Figure 5.5.

Example 5.4. A domain expert reports errors for business cases with flight phase `onground` due to missing rules. Since we derive context hierarchies from parameter values hierarchies, the domain experts inspect the parameter value hierarchies. Regarding the parameter `FlightPhase` they find that `onground` should be part of flight phase `departure`. The domain expert informs the repository administrator, who updates the `covers` relationships of `departure` and `flightPhase`. The rule developers of the contexts ⟨aircraft, onground, closure⟩, ⟨landplane, onground, runwayClosure⟩, and ⟨aircraft, onground, aerodromeEquipment⟩ are notified and acknowledge without performing any operations. Consequently, the operations succeeds.

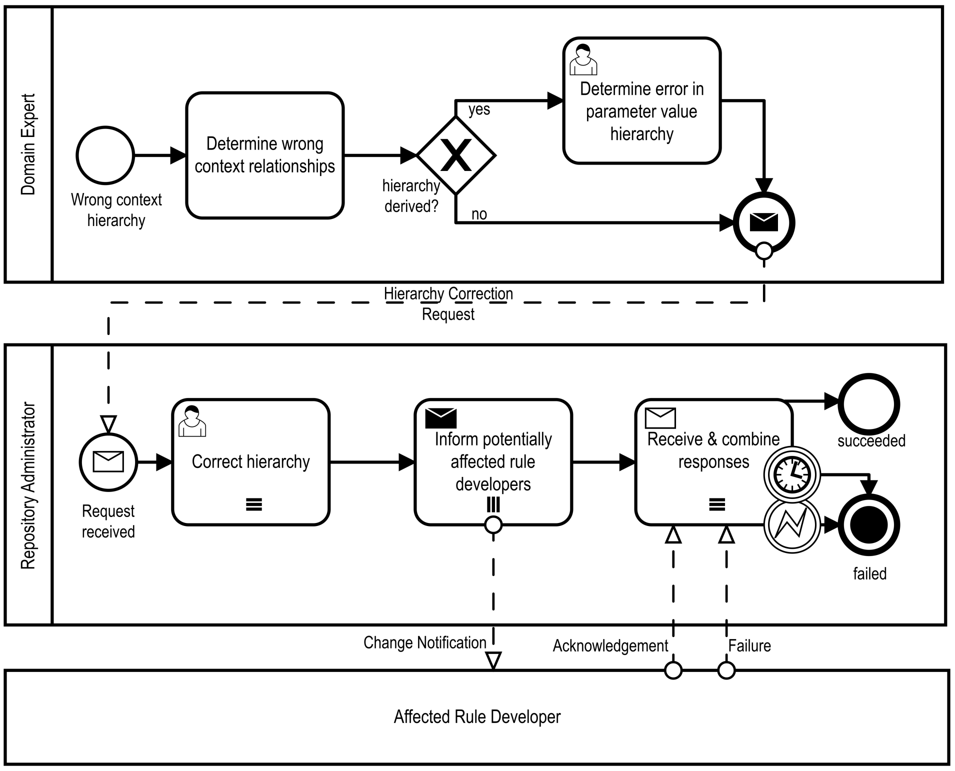


FIGURE 5.5: BPMN model of the composed operation *update context hierarchy* and the preceding process of determining the cause of the wrong context hierarchy by the domain expert. The process for the affected rule developers is the same as in Figure 5.2.

Update Context Classes

Updates to context classes are necessary if (a) domain experts or users determine functional issues, such as unresolved conflicts or parameters not modelled, or (b) repository administrators identify optimization potential, for instance, a more efficient implementation of inheritance resolution.

Updates to context classes regard their structure and methods. These updates are immediately propagated to contexts. If a repository administrator performs a context class update which influences the rules to be applied for specific business cases, for instance, he/she updates parameters or global conflict resolutions, the affected rule developers are notified. The operation is completed once all affected rule developers acknowledged the operation. A summary of this process is given in Figure 5.6.

Example 5.5. The repository administrator intends to add a global conflict resolution strategy for importance classification: if contradicting importance classes are derived, the stronger one, i.e., `highlyImportant`, prevails. For this purpose, she modifies `detCaseSpecificCtx` accordingly. Since this operation may affect rule developers, they are notified. The rule developer of context ⟨helicopter, allFlightPhases, obstruction⟩ *modifies rule* `R4` as the explicit overriding statement is now obsolete. All other rule developers acknowledge the operation in time. The composed operation succeeds.

Modify Business Case Classes

New business case classes, updates to existing business case classes, or deletion of existing business case classes are usually rendered necessary due to developments in

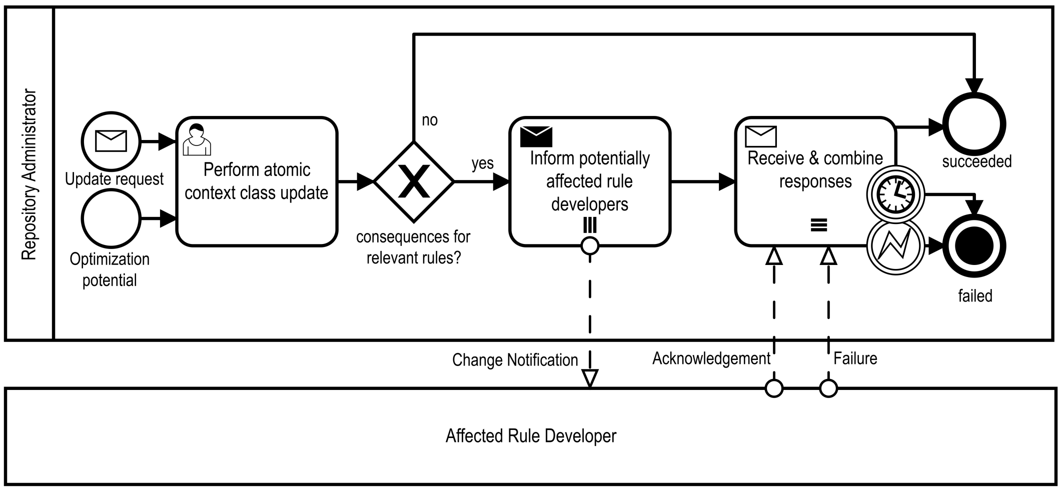


FIGURE 5.6: BPMN model of the composed operation *update context class*. The process for the affected rule developers is the same as in Figure 5.2.

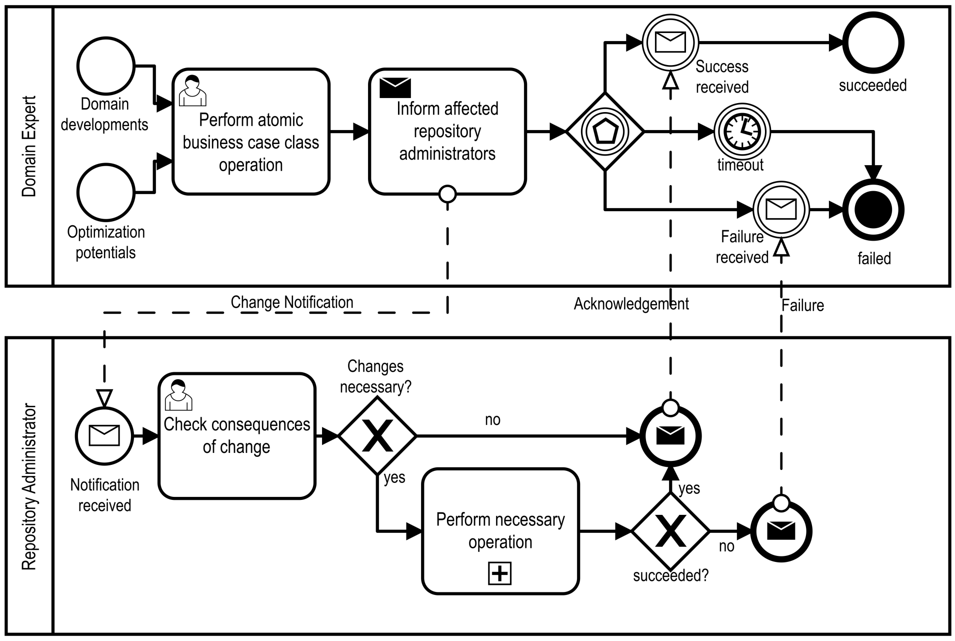


FIGURE 5.7: BPMN model of the composed operation *modify business case class*.

the domain like new data formats or optimizations.

The domain expert performs the necessary atomic operation on the business case class. Subsequently, he/she notifies the repository administrator about this operation which might require further operations, such as updating `detParamValue` methods, on the respective CBR model. The operation succeeds if a repository administrator acknowledges it. A BPMN representation of this process is shown in Figure 5.7.

Example 5.6. A new XML-based representation of aircraft types has been introduced. Thus, the domain expert updates the attribute `interest` in class `UserSituation` accordingly. The repository administrator is notified and checks whether changes on his side are necessary. Since the `interest` attribute is used to derive parameter values for parameter `Interest`, the repository administrator updates `detParamValue` of `Interest` to work with the new aircraft representation. This update does not affect any rule

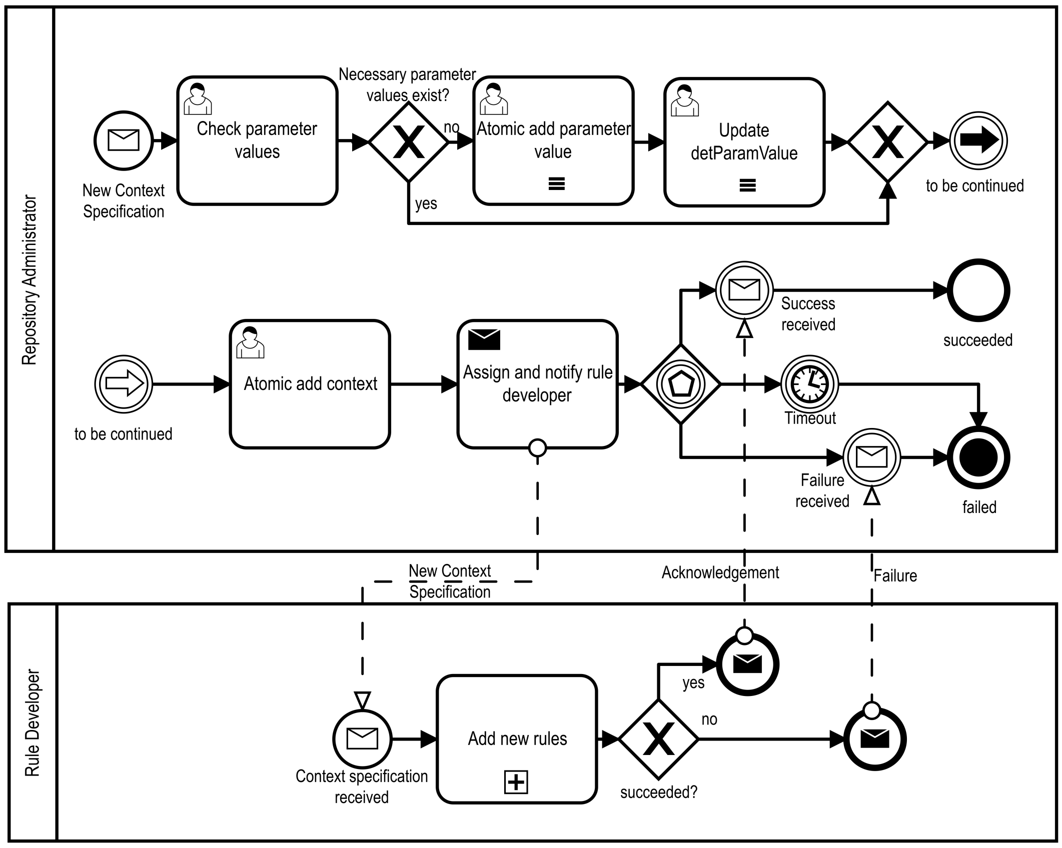


FIGURE 5.8: BPMN model of the composed operation *new context*.

developers, thus the repository administrator acknowledges the modification to the business case class and the operation succeeds.

5.3.2 CBR Model Extension and Refinement Operations

CBR model extension and refinement operations are used to make CBR models more fine-grained, i.e., rules are more finely separated into contexts.

New Context (Extension)

The incremental rule elicitation process reveals a set of new rules applying in a class of situations not covered yet. If this set justifies a new context, the respective CBR model is extended by a new context; otherwise, the rules are, depending on their implementation, *(de-) contextualized* to existing contexts.

Domain experts inform the repository administrator who checks whether the parameter values of the new context exist. If this is not the case, the new parameter values are added to the corresponding parameters and the `detParamValues` methods are updated accordingly. Subsequently, the repository administrator creates the new context, sets the corresponding parameter values, and assigns a responsible rule developer. This rule developer is notified and then *adds the new rules*. Once the rule developer successfully added the new rules, the operation has succeeded. A graphical representation of the process is given in Figure 5.8.

Example 5.7. A rule elicitation workshop identified rules applying in context ⟨aircraft, arrival, closure⟩. The repository administrator finds all necessary parameter values and thus simply creates the new context with the parameter values `aircraft`, `arrival`, and `closure`. Subsequently, she assigns a rule developer who is notified and *adds the identified rules* to context ⟨aircraft, arrival, closure⟩. Once all rules have been added, the rule developer acknowledges and thus the composed operation succeeds.

Split Contexts (Refinement)

Splitting a context becomes necessary if (a) the repository administrator identifies the need or (b) a rule developer asks the repository administrator to split a certain context. A context can be split if many rules refine the context's class of situations, i.e., check for more specific values. Values frequently checked become parameter values if they do not exist yet.

The repository administrator checks the parameter values, adds necessary parameter values if they do not exist yet, creates one context per more specific parameter value, and assigns rule developers. The composed operation *new context* could be reused but would require acknowledgements unnecessary for splitting contexts. Next, the rule developer of the origin context is notified of the created contexts and subsequently *contextualizes rules* whose conditions refine the context's class of situations to the respective newly created child-contexts. Once the rules have been contextualized, the origin context's rule developer acknowledges the split operation and the composed operation succeeds. A BPMN model of this process is shown in Figure 5.9.

Example 5.8. The rule developer assigned to context ⟨aircraft, allFlightPhases, obstruction⟩ identifies a group of rules applying only for tree obstacles and a group of rules only applying for non-tree obstacles. Thus, he asks the repository administrator to split the context regarding tree and non-tree obstacles. The repository administrator adds the parameter values `treeObstruction` and `nonTreeObstruction` under `obstruction` to the parameter `EventScenarios`. Subsequently, she *adds two new contexts* ⟨aircraft, allFlightPhases, treeObstruction⟩ and ⟨aircraft, allFlightPhases, nonTreeObstruction⟩ and assigns rule developers. Thereafter, the origin rule developer is notified and subsequently *contextualizes* rule `R1` to context ⟨aircraft, allFlightPhases, nonTreeObstruction⟩ and rule `R2` to context ⟨aircraft, allFlightPhases, treeObstruction⟩. For both rules we can remove their conditions as they are already encoded in the context knowledge, i.e., rule `R1` reduces to `Always of high importance` and rule `R2` to `Always of little importance`. The rule developer of ⟨helicopter, allFlightPhases, obstruction⟩ is informed that Rules `R1` and `R2` are removed from her context. She replaces rule `R4` with `Always of high importance` and acknowledges the contextualize operation. Thus, the composed operation succeeds.

New Parameter (Extension/Refinement)

Two cases exist in which the repository administrator adds a new parameter: (a) to extend the functionality covered by the respective CBR model and (b) to refine the respective CBR model.

In the former case (a), domain experts or users require an extension of the system's functionality, i.e., covering additional classes of situations by adding a new parameter. The repository administrator is informed who then adds the parameter, *updates the context class*, adds a root parameter value, and sets the root parameter value for all contexts. This process is depicted in Figure 5.10.

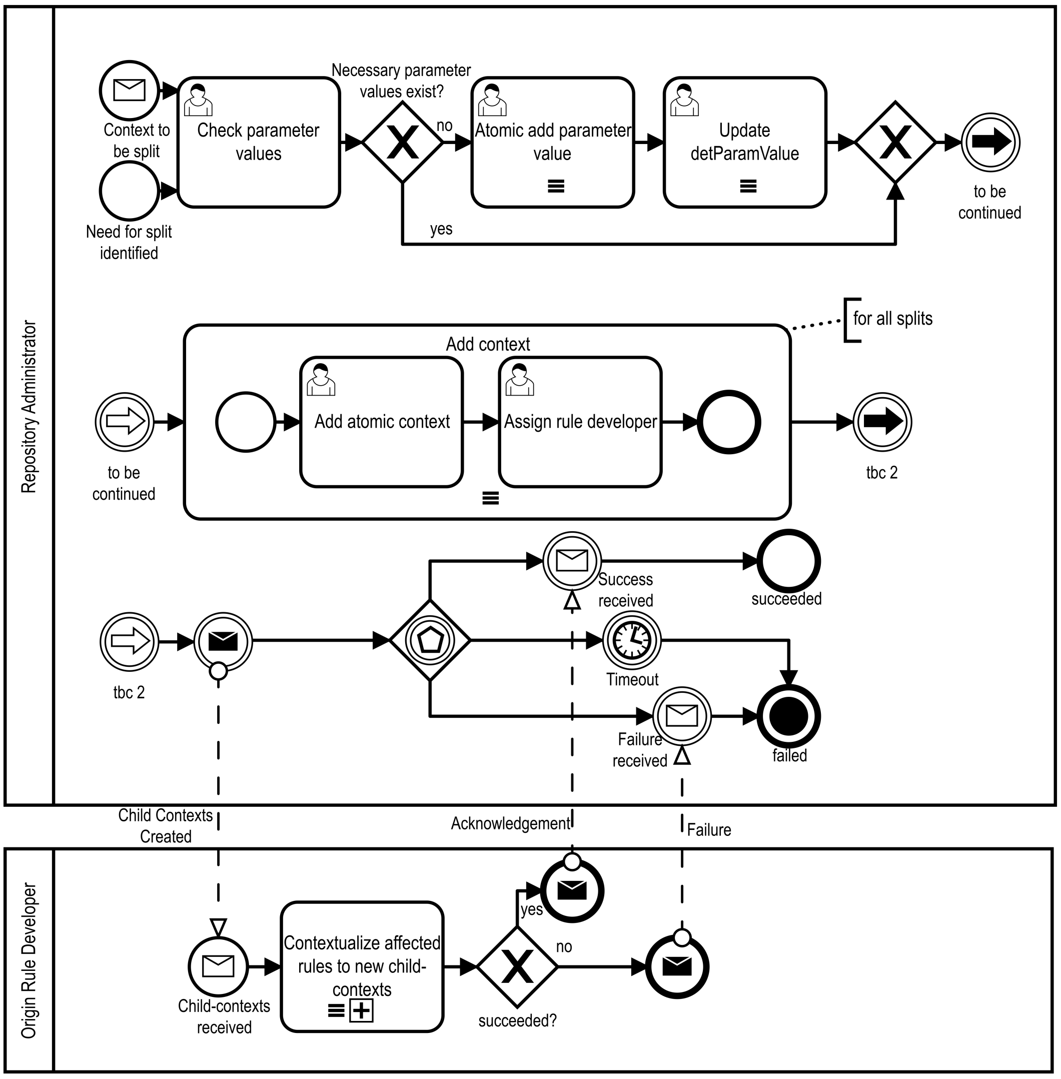


FIGURE 5.9: BPMN model of the composed operation *split contexts*.

Example 5.9. Domain experts inform the repository administrator to extend the functionality of the system to meteorological classes of situations. The repository administrator adds the parameter `MeteorologicalCondition` and the root parameter value `allMeteorologicalConditions`. Thereafter, `AIMCtx` is *updated* to contain the new parameter. Subsequently, the parameter `MeteorologicalCondition` of each context is set to `allMeteorologicalConditions`.

In the latter case (b), the repository administrator finds many rules querying the same business case attribute, checking for different values. The query becomes the `detParamValue` method of the new parameter; the checked values become the parameter values of the new parameter. The repository administrator adds the new parameter as described above and *splits* existing contexts to achieve contexts referring to the identified parameter values of the new parameter. The complete process is the process depicted in Figure 5.10 followed by as many *split* processes (c.f. Figure 5.9) as necessary.

Example 5.10. Domain experts inform the repository administrator to refine the CBR model by an additional parameter regarding the NOTAM status. The repository administrator adds the parameter `NOTAMStatus` and the root parameter value `allStati`.

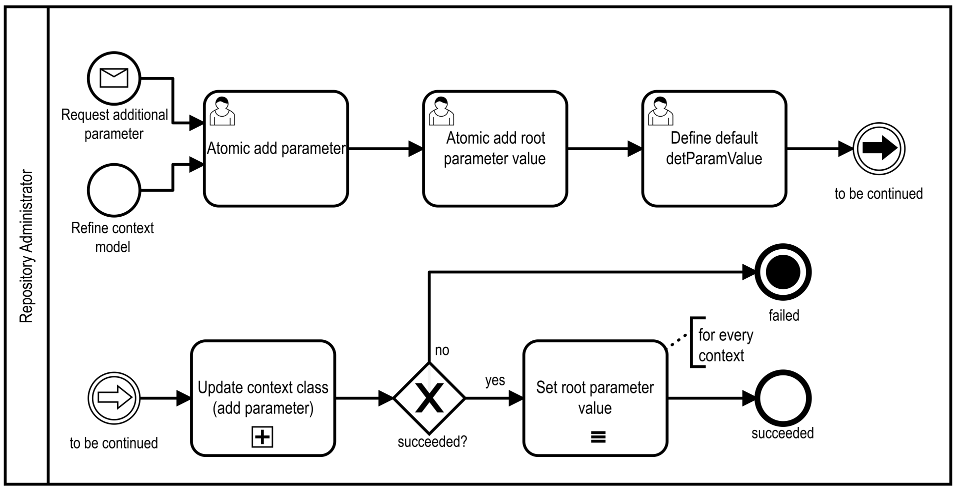


FIGURE 5.10: BPMN model of the composed operation *new parameter*.

Thereafter, `AIMCtx` is *updated* to contain `NOTAMStatus` and its value for each context is set to `allStati`. Subsequently, contexts containing rules employing the NOTAM status are *split*, for instance, context ⟨landplane, onground, runwayClosure, allStati⟩ is split into two contexts ⟨landplane, onground, runwayClosure, limited⟩ and ⟨landplane, onground, runwayClosure, closed⟩.

5.3.3 CBR Model Restriction and Consolidation Operations

CBR model restriction and consolidation operations are the inverse operations of CBR model extension and refinement operations. Thus, they reduce the granularity of a CBR model.

Delete Context (Restriction)

In the case where domain changes render a context and its rule set superfluous, the domain experts inform the repository administrator. The repository administrator notifies the responsible rule developer to *delete all associated rules*. Subsequently, the rule developer notifies the repository administrator who deletes the corresponding context. Thereafter, the repository administrator checks for unused parameter values and deletes them. A graphical representation of the process is presented in Figure 5.11.

Example 5.11. Domain experts inform the repository administrator of changes in aeronautical regulations making context ⟨helicopter, allFlightPhases, obstruction⟩ superfluous. The repository administrator determines the responsible rule developer and notifies her. The responsible rule developer then *deletes rule* `R4` and notifies the repository administrator who then deletes context ⟨helicopter, allFlightPhases, obstruction⟩ and the now unused parameter value `helicopter`.

Merge Contexts (Consolidation)

The repository administrator finds or a rule developer informs the repository administrator that a certain context contains only few rules. To keep the respective CBR model compact, this context can be merged with one of its ancestor contexts.

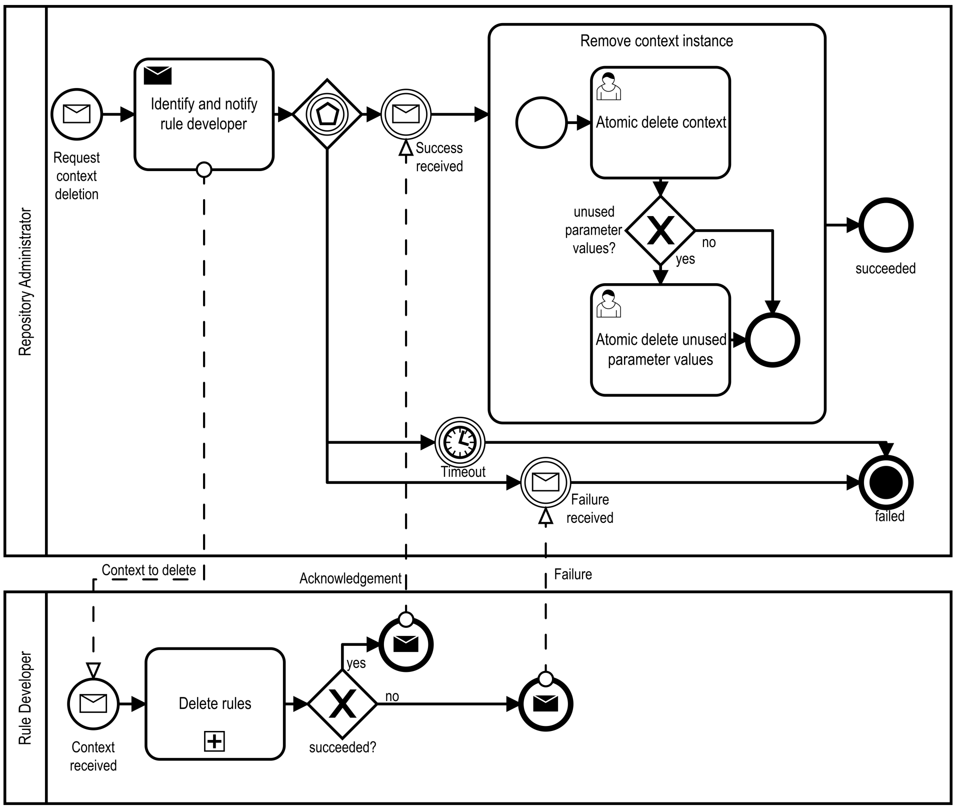


FIGURE 5.11: BPMN model of the composed operation *delete context*.

First, the repository administrator decides on a target ancestor context and notifies the rule developer of the origin context to *decontextualize each rule* to the selected ancestor context. This ensures that the semantics of the rule set is maintained. Once all rules have been decontextualized and the merge operation has been acknowledged, the repository administrator deletes the now empty context. For a BPMN representation of the process consider Figure 5.12.

Example 5.12. The repository administrator finds that context ⟨helicopter, allFlightPhases, obstruction⟩ contains only one rule and thus should be merged with context ⟨aircraft, allFlightPhases, obstruction⟩. He notifies the responsible rule developer to *decontextualize* all rules, i.e., rule `R4`, to context ⟨helicopter, allFlightPhases, obstruction⟩. Once done, the rule developer acknowledges the merge operation. Subsequently, the repository administrator deletes the context. The parameter value `helicopter` is not used by any other context in our use case and thus is deleted with the context.

Delete Parameter (Restriction/Consolidation)

Reasons to delete a parameter are (a) to remove obsolete functionality and (b) to compact the respective CBR model.

In the former case (a), the repository administrator is informed by domain experts that a parameter has become obsolete. Thus, the repository administrator intends to delete this old parameter, its parameter values, and all associated contexts. To this end, the repository administrator deletes all contexts having a parameter value different from the root parameter value for the old parameter. Next, the repository

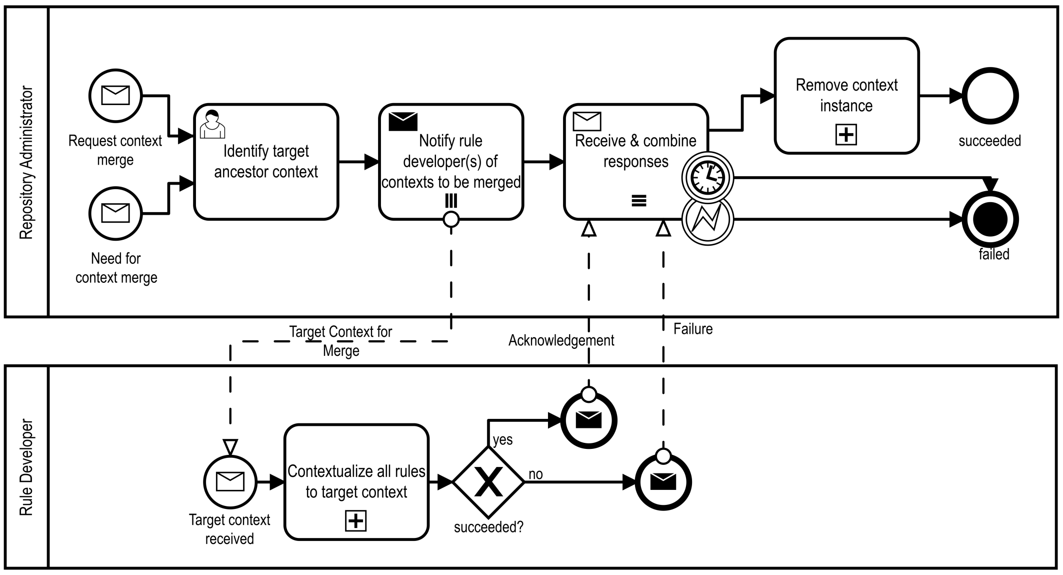


FIGURE 5.12: BPMN model of the composed operation *merge contexts*. `Remove context instance` refers to the subprocess in Figure 5.11.

administrator *updates the context class* to remove the old parameter from the context class. Finally, the repository administrator deletes the old parameter and its root parameter value. This process is visualized in Figure 5.13.

Example 5.13. Domain experts inform the repository administrator that parameter `FlightPhase` has become obsolete due to simplification of regulations. The repository administrator deletes all contexts having a parameter value for parameter `FlightPhase` different from `allFlightPhases`, i.e., ⟨aircraft, onground, closure⟩, ⟨landplane, onground, runwayClosure⟩, and ⟨aircraft, onground, aerodromeEquipment⟩, and their rules. Thereafter, he *updates the context class* `AIMCtx`, i.e., removes the association `flightPhase`. Thus, for instance, context ⟨aircraft, allFlightPhases, obstruction⟩ becomes context ⟨aircraft, obstruction⟩. Thereafter, the repository administrator deletes parameter `FlightPhase` and its root parameter value.

In the latter case (b), the repository administrator finds that the different parameter values of a parameter influence the relevant rule sets only marginally or not at all. Different to the former case, we want to maintain all rules. For this purpose, the repository administrator notifies affected rule developers to *merge their context(s)* to contexts independent of the old parameter, i.e., contexts referring to the root parameter value of the old parameter. Subsequently, the process as described for (a) is followed.

Example 5.14. The repository administrator found that parameter `FlightPhase` only marginally influences the relevant rule sets for business cases and thus will delete parameter `FlightPhase`. She notifies affected rule developers to merge their contexts to contexts independent of parameter `FlightPhase`. For instance, context ⟨aircraft, onground, closure⟩ is *merged* with context ⟨allInterests, allFlightPhases, allEventScenarios⟩. Next, the repository administrator *updates the context class* `AIMCtx`, i.e., removes the association `flightPhase`. Thus, for instance, context ⟨aircraft, allFlightPhases, specialPort⟩ becomes context ⟨aircraft, specialPort⟩. Subsequently, the repository administrator deletes parameter `FlightPhase` and its root parameter value.

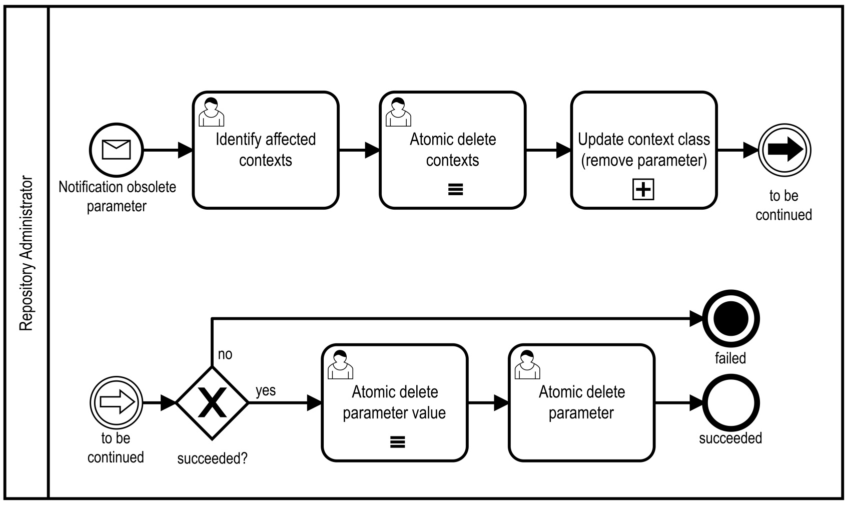


FIGURE 5.13: BPMN model of the composed operation *delete parameter*.

5.3.4 Business Rule Management in CBRs

In Section 2.4.2 we have discussed the activities to be supported by BRM in non-contextualized repositories and the process they form. On a coarse level, BRM supports acquisition, maintenance, and execution of business rules. In this section we shortly discuss the BRM process for: (1) CBRs where no business rules have been elicited yet and (2) situations where a set of existing business rules is to be migrated to a CBR. For both we focus on acquisition and maintenance; we do not discuss execution as execution depends on the concrete implementation of CBRs and the rule language employed (c.f. Chapter 4). Furthermore, we discuss how CBRs support orthogonal BRM activities. The CBR-specific BRM proposed here supports incremental elicitation of business rules.

Setting up a BRM with no business rules elicited yet (1), domain experts begin by identifying the most important parameters for business rule organization. Once found, domain experts ask repository administrators to add the identified parameters to the empty CBR model and to create an empty root context. In the next step, domain experts begin capturing/eliciting general business rules applying in any case. These will then be authored/specified by rule developers responsible for the root context and added to the root context. Subsequently, domain experts begin eliciting business rules for more specific contexts, one at a time. To this end, they inform repository administrators to create a new context and provide the business rules to be authored by the responsible rule developers. Once business rules have thus been elicited and authored for a few contexts, domain experts and users test the CBR. Any issues uncovered by domain experts and users, for example, invalid results, are reported to the repository administrators and rule developers who subsequently rectify them. From time to time, repository administrators and rule developers analyze the CBR model for optimizations, for instance, whether it can be improved by applying CBR model extension, refinement, restriction, or consolidation operations. Once all relevant business rules for the current parameters and parameter values have been found, functionality may be extended by adding further parameters starting the process anew. A summary of a generic BRM process regarding CBR is given in Figure 5.14.

In the case where business rules have already been elicited and should now be

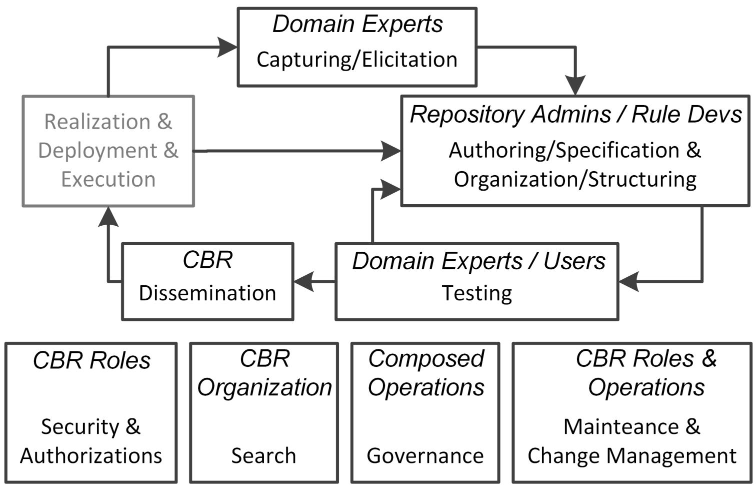

Domain Experts
Capturing/Elicitation
Realization & Deployment & Execution
Repository Admins / Rule Devs
Authoring/Specification & Organization/Structuring
CBR
Dissemination
Domain Experts / Users
Testing
CBR Roles
Security & Authorizations
CBR Organization
Search
Composed Operations
Governance
CBR Roles & Operations
Mainteance & Change Management

FIGURE 5.14: Generic process for BRM, based on Chapter 2, with responsible CBR roles assigned.

organized in a CBR (2), a similar process can be employed. In this case, the main activities are organization/structuring, testing, and dissemination (we do not consider execution here); capturing/elicitation are of less importance. Domain experts together with repository administrators and rule developers analyse the existing rule base regarding potential parameters and parameter values. Once found, the repository administrator creates them and adds a root context. This root context contains all business rules of the rule base. In subsequent steps, repository administrators and rule developers employ CBR model refinement operations to iteratively refine the CBR's organization. After every single or a set of refinement operations, domain experts and users will test the CBR and report any issues which will be rectified by repository administrators and rule developers.

Besides the main BRM process, CBRs and their CBR model operations support the orthogonal BRM activities discussed in Section 2.4.2, namely, security and authorization, maintenance, search, and governance (see Figure 5.14). The proposed CBR roles enable clear separation of responsibilities and tasks and also restrict access to CBR model elements, for instance, only assigned rule developers may change business rules in a context. Search is supported by the multidimensional organization, i.e., parameters and parameter values as well as their hierarchy may serve as index which can be used to narrow down the search space. Governance is supported by the composed operations introduced in this chapter, defining the communication and change management in CBRs. Maintenance is supported by CBR roles, for instance, domain experts and users will report issues or invalid/outdated results and repository administrators and rule developers will rectify them using the proposed composed operations. Furthermore, repository administrators and rule developers may employ extension, refinement, restriction, and consolidation operations to maintain and improve the CBR organization.

Example 5.15. We applied the BRM process as described above to SemNOTAM: we assumed the roles of repository administrator and rule developer while pilots acted as domain experts and users. We conducted expert interviews with the pilots to identify potential parameters and parameter values. We identified seven potential parameters and their parameter values, allowing approximately 19,000 different classes of situations, i.e., contexts, for which business rules can be elicited. Refer to Chapter 8 for details.

Subsequently, we added the three most important parameters, namely, interest, event scenario, and flight phase, to an empty CBR model. In workshops with pilots we then elicited business rules for each NOTAM event scenario separately – first for more general contexts, such as contexts regarding all flight phases, later refining them, for instance, contexts regarding flight phase `onground`. Each time we had added a few contexts, we asked the pilots to submit SemNOTAM cases to the system and to check whether the results are valid. For any issues, the cause could be easily identified due to explanations of the result (case-specific contexts containing only relevant business rules). After each workshop, we analyzed the business rule sets and performed CBR model refinement and consolidation operations where appropriate. So far, we have elicited approximately 700 rules applying in about 150 contexts. Once no further rules for the current parameters are found, we add the next parameter and continue eliciting rules as described.

5.4 Comparative Evaluation of CBRs and NCBRs

In this section we compare the composed CBR model modification operations from Section 5.3 with the modifications necessary within a Non-contextualized Business Rule Repository (NCBR) (c.f. Section 4.2.1) to achieve the same effect. To this end, we discuss semantic equivalents of composed operations in NCBRs. For each semantically equivalent operation we give a description, an example utilizing our SemNOTAM use case and the NCBR example given in Section 4.2.1, and a comparison to the respective composed operation in CBRs. In particular, we compare the search space, i.e., the number of business rules to consider in the respective repository type when looking for specific business rules, unintended side-effects resulting from operations, or obsolete business rules. The same roles we identified for CBRs are employed for NCBRs; the difference is that rule developers in NCBRs are assigned to rule definitions rather than to contexts. In this regard, we compare the search space for determining responsible rule developers, i.e., the number of potentially responsible rule developers. A summary of the semantically equivalent operations and the comparison is given in Table 5.1. Thereafter, we delineate the differences between CBRs and NCBRs from the viewpoint of CBR roles.

5.4.1 Maintenance Operations

Maintenance operations for NCBRs consider modification of rules, modifying the scope of rules, updating term hierarchies, modifying conflict resolution strategies, and modifying business case classes. For each NCBR operation we state the corresponding composed CBR operation. A semantic equivalent of *decontextualize* (Section 5.3.1) is not needed within NCBRs. Thus, *decontextualize* requires effort in CBRs while no effort is required in NCBRs. Furthermore, *update context class* (Section 5.3.1) does not have a fully equivalent operation.

Modify Rules

In NCBRs rule developers may (1) add, (2) delete, or (3) change rules. Addition (1) of rules is straightforward – nevertheless, the terms for defining the rule's scope-part and actual-rule-part need to be identified. Furthermore, the subsumption hierarchies of terms used in rule definitions need to be taken into account. For instance, using the term `AircraftTypeAircraft` also implies term landplane. Deletion (2) and change

TABLE 5.1: Comparison of CBR composed operations with their semantically equivalent NCBR operations.

CBR and NCBR Operations		CBR vs. NCBR	
CBR Composed Operation	**NCBR Operation**	**Search in Rule Set**	**Determine Responsible Rule Developers**
Maintenance			
Add rule	Add rule	+	-
Delete/change rule	Delete/change rule	+	○
Contextualize	Modify scope of rules	+	○
Decontextualize	N/A	(-)	(-)
Update context hierarchy	Update term hierarchy	+	+
Update context class	Modify conflict resolution	○	○
Modify business case class	Modify business case classes	+	○
Extension & Refinement			
New context	Add rules with same scope	+	○
Split context	N/A	(-)	(-)
New parameter	N/A	(-)	(-)
Restriction & Consolidation			
Delete context	Delete a scope	+	○
Merge context	N/A	(-)	(-)
Delete parameter (Restriction)	Delete Scoping term hierarchy	+	○
Delete parameter (Consolidation)	N/A	(-)	(-)

+	CBRs have a smaller search space	○	CBRs and NCBRs have an equal search space
-	CBRs have a larger search space	(-)	CBRs have a larger search space as no operation in NCBRs is necessary

(3) of rules is more complex. Domain experts or users report invalid results. First, the domain expert needs to search the complete rule set for the causing rule(s). Thereafter, the identified rule(s) are deleted or changed respectively. Once the atomic operation has been performed, the rule developer needs to subsequently check whether the modification has unintended side-effects on derivation or whether rules have become obsolete. This may be the case as dependencies between rules may exist.

Example 5.16. Considering Example 5.1, a domain expert informs the rule developer that all obstruction NOTAMs are highly important. Thus, the rule developer searches for all rules employing the term `Obstruction` in their definition (rules `R1`, `R2`, and `R3`). From these rules he identifies the ones which do not have `highlyImportant` in their head, i.e., rule `R2`. He modifies rule `R2` according to the domain expert's information. Subsequently, he checks the rule set for unintended side-effects and obsolete rules. He identifies the now obsolete rule `R4` and removes it.

Comparison to CBRs The major difference compared to *modify rules* in CBRs (Section 5.3.1) is the size of the rule search space to consider. While rule sets in CBRs are quite small due to the contextual organization, the complete rule set needs to be considered in NCBRs. Regarding addition of rules this is relevant to identifying the terms for the rule's definition. Regarding deletion and change of rules this is relevant to identifying the causing rule and the terms to use when changing the rule. Furthermore, the larger search space is challenging for checking side-effects and obsolete rules.

For larger rule sets, a NCBR will require several rule developers. Responsible rule developers for newly added rules are simply assigned by repository administrators. In a CBR the rule developer for the respective context needs to be identified first. Regarding deletion and change, identification of the responsible rule developer in CBRs and NCBRs is trivial once the causing rule has been found.

The complexity of rules in a CBR is less than in a NCBR (see rules `R6` and `R9` in Listing 4.5 and 4.6).

Modify the Scope of Rules

In NCBRs the scope-part of a rule may be narrowed or broadened. For these purposes, terms employed in the rule's definition will be exchanged for more specific or generic terms respectively. A rule developer needs to identify the terms representing the target scope and to modify the rule's definition accordingly. Once the modification has been performed, the rule developer needs to check whether the modification has unintended side-effects on the rule set or whether rules have become obsolete.

Example 5.17. Considering Example 5.2, a domain expert informs the rule developer that the scope of rule `R3` is to be narrowed to landplanes and runway closure NOTAMs. The rule developer searches the rule set for corresponding terms, i.e., `AircraftOfInterestTypeLandplane` and `RunwayClosure` respectively. Subsequently, she replaces the terms `AircraftOfInterestTypeAircraft` and `Closure` in rule `R3`. Finally, she checks for any unintended side-effects or rules rendered obsolete. Since she finds none the operation succeeds.

Comparison to CBRs The main difference to *contextualize rules* (narrowing scope; see Section 5.3.1) and *deletion* with subsequent *addition of the rule* to an ancestor context (broadening scope; both Section 5.3.1), is the size of the rule search space. In order to find the terms for a different scope in a CBR the context hierarchy is searched; in a NCBR all term hierarchies in the rule set need to be searched. The larger search space is also challenging when checking for unintended side-effects and obsolete rules. Identification of responsible rule developers is the same for CBR and NCBR since we know the rule for which we change the scope.

Update Term Hierarchies

In NCBRs, term hierarchies invalid from the viewpoint of users and domain experts may be corrected. In this case, domain experts find that issues uncovered in the derived results for business cases are caused by invalid term hierarchies, i.e., subsumption hierarchies of terms are incorrectly represented. Consequently, domain experts need to search all term hierarchy definitions in the rule set. Once the causing definition is found it needs to be updated and side-effects and obsolete rules need to be checked.

Example 5.18. Considering Example 5.4, a domain expert finds missing rules for business cases regarding flight phase `onground`. Subsequently, the domain expert analyses the terms employed in the rule set. This analysis reveals that flight phase `onground` should be a sub-phase of `departure`. The domain expert informs the rule developer about the error who then adds the corresponding statement to the rule set. This change may affect the scope of other rules. Thus, the rule developer checks for unintended side-effects and obsolete rules but finds none.

Comparison to CBRs Two cases when comparing to CBRs need to be distinguished: term hierarchies used for scoping and term hierarchies used in actual-rule-parts. For the former, the main difference to *udpate context hierarchy* in CBRs (Section 5.3.1) is the size of the rule search space for term hierarchy definitions, unintended side-effects, and obsolete rules. To find the terms causing the issue in a CBR either the context hierarchy or the parameter value hierarchies are considered. In a NCBR all term hierarchies have to be considered. While in CBRs the repository administrator is responsible, responsible rule developers need to be identified in NCBRs. For the latter, the differences to *modify rules* in CBRs are discussed in (Section 5.4.1).

Modify Conflict Resolution Strategies

In NCBRs conflict resolutions may be modified. A rule developer implements a requested conflict resolution using rules. Subsequently, unintended side-effects and obsolete rules have to be checked.

Example 5.19. Considering Example 5.5, the rule developer is asked by the repository administrator to add a conflict resolution strategy for importance classes: `highly-Important` prevails. The rule developer implements this strategy and subsequently checks for unintended side-effects and obsolete rules. In this case the rule developer simplifies rule `R2` by removing condition `\naf ?I:AircraftOfInterestTypeHelicopter`. This condition is not necessary anymore due to the conflict resolution strategy.

Comparison to CBRs Compared to *update context class* in CBRs (Section 5.3.1), aspects other than modification of conflict resolutions are covered by other composed operations (for instance modifying parameters) or not relevant (for example inheritance). The differences in modification of conflict resolutions are the same as for *modify rules* in NCBRs (Section 5.4.1).

Modify Business Case Classes

Modification to business case classes in NCBRs is almost identical to *modify business case classes* in CBRs (Section 5.3.1). The only difference is that instead of updating `detParamValue` definitions, term definitions are updated. Consequently, affected terms need to be identified in the complete rule set (i.e. larger rule search space). Identical to CBRs, any rules accessing modified parts of the business case class need to be adapted. Since all rules may be potentially affected, the size of the rule search space for unintended side-effects and obsolete rules for CBR and NCBR is the same. Regarding roles, repository administrators are responsible for updating the `detParam-Value` definitions in CBRs while in NCBRs rule developers are responsible.

5.4.2 Extension and Refinement Operations

NCBRs require only one extension operation, namely, adding a set of rules with the same scope to the rule set. Semantic equivalents of *split context* (Section 5.3.2) and *new parameter* (Section 5.3.2) in CBRs are not necessary for NCBRs. This is the case as these operations do not affect derived results for any business case. These operations require effort in CBRs while no effort is necessary in NCBRs.

In NCBRs, *addition of rules with the same scope* is set into motion by the rule elicitation process. The process may reveal a set of rules applying in a certain scope. A rule developer needs to identify the terms representing the given scope. If suitable terms cannot be found, the rule developer needs to add corresponding terms to the rule set. Afterwards, he/she employs *modify rules* to add the rules to the rule set.

Example 5.20. Considering Example 5.7, a set of rules applying during flight phase `arrival` for closure NOTAMs when using an aircraft has been elicited. The rule developer identifies in Listing 4.6 the terms representing this scope, i.e., `FlightPhase-Arrival`, `Closure`, and `AircraftTypeAircraft` respectively. Subsequently, she adds each rule employing *modify rules*. This ensures that side-effects are considered and obsolete rules are removed.

Comparison to CBRs The main difference to *new context* (Section 5.3.2) is the larger rule search space for scoping terms in NCBRs. To find the scope representation in a NCBR, all terms of the rule set have to be searched; in CBRs the parameter value hierarchies are searched. If no suitable terms exist, they need to be defined in CBRs as well as NCBRs. Responsible rule developer(s) are assigned by the repository administrator in both CBRs and NCBRs.

5.4.3 Restriction and Consolidation Operations

Restriction and consolidation operations for NCBRs regard the deletion of scopes and the deletion of term hierarchies. Composed operation *merge contexts* (Section 5.3.3) or *delete parameter (consolidation)* (Section 5.3.3) in CBRs are not necessary for NCBRs. Thus, these operations require efforts in CBRs not necessary in NCBRs.

Delete a Scope

In NCBRs, all rules belonging to a certain scope may be removed. This is the case if domain experts find that a certain scope is not relevant anymore. To this end, rule developers need to identify all rules specifying this scope and to employ *modify rules* in NCBRs (Section 5.4.1) to remove them from the rule set.

Example 5.21. Considering Example 5.11, domain experts inform the rule developer that rules applying to helicopters for NOTAMs concerning obstruction are not valid anymore due to changes in aeronautical regulations. The rule developer identifies rule `R4` as only rule within this scope and subsequently employs *modify rules* to delete it. Since term `AircraftOfInterestTypeHelicopter` is not employed anymore, it is deleted during the side-effect and obsolete rules check.

Comparison to CBRs The main difference to *delete context* in CBRs (Section 5.3.3) is that rules within the scope identified by domain experts need to be searched for in NCBRs; in CBRs the context to delete is already identified by domain experts. Since we employ *modify rules* in NCBRs these differences apply as well. Since each context is assigned a rule developer in a CBR, identification of the responsible rule developer is simple; in NCBRs identification of responsible rule developers is straightforward once all rules to delete are known.

Delete Scoping Term Hierarchy

In NCBRs, complete term hierarchies employed for scoping may be removed. In this case domain experts inform a rule developer that a certain term hierarchy used for scoping has become obsolete including all rules employing any of the terms in their scope-part. Thus, rule developers need to identify all rules employing any of the terms of the given hierarchy in their scope-part and to delete them from the rule set. Furthermore, the definitions of the terms and their hierarchy need to be removed. A check for unintended side-effects and obsolete rules is necessary due to potential rule dependencies.

Example 5.22. Considering Example 5.13, the rule developer is informed that the term hierarchy considering flight phases has become irrelevant. This term hierarchy is used for scoping. He identifies all rules employing any of the terms in their scope-part, namely rules `R3`, `R5`, `R6`, and `R8`, and deletes them. Subsequently, the rule developer deletes any term definitions considering flight phase and their hierarchy (lines 13–14 in Listing 4.6). No unintended side-effects are found.

Comparison to CBRs The main difference to *delete parameter* in CBRs (Section 5.3.3) is that in NCBRs the complete rule set needs to be searched for rules, i.e., the rule search space for NCBRs is larger. In a CBR the rules and contexts to delete are given by the context model and the parameter to delete. Identification of responsible rule developers is straightforward for both CBRs and NCBRs once the contexts or rules, respectively, have been identified.

5.4.4 Roles and Business Rule Management

In this section we compare roles and their support in CBRs and NCBRs. Regarding BRM we find no significant differences since we developed *CBR-specific BRM* based upon the activities we identified for BRM in conventional repositories (NCBRs).

CBR roles enable clear separation of responsibilities and tasks in CBRs, promoting effective and efficient business rule maintenance and business rule entering. Although NCBRs employ the same or similar roles, these roles are not supported to the same extent. Below we discuss the differences. Differences for roles stemming from the differences of composed operations in CBRs and their semantic equivalents in NCBRs, have been discussed in the previous sections.

For repository administrators, CBRs render large numbers of rules manageable. The classes of situations in which rules apply are explicitly defined and their relationships are known. This enables effective segregation of tasks and responsibilities: rule developers are responsible for rules, users for business cases, domain experts for business case classes, and repository administrators for the remaining CBR model elements. An overview of covered classes of situations is easily obtained by viewing the contexts and their hierarchies. Furthermore, if method implementations are assigned to rule developers, repository administrators need not be able to understand or author rules. Considering repository administrators for NCBRs, the classes of situations and contexts covered by the NCBR as well as their relationships are encoded in rules. Thus, an overview of covered classes of situations or scopes is not as easily obtained. Furthermore, since all rules reside in one rule set their maintainability is limited. Besides that, repository administrators need to design workarounds to enable systematic assignment of rule developers to rules as supported in CBRs.

For rule developers, CBRs reduce the number of rules to consider compared to NCBRs as only rules of assigned contexts and their super-contexts are relevant to them. For instance, the rule developer responsible for context ⟨helicopter, allFlightPhases, obstruction⟩ needs to consider the rules `R1`, `R2`, and `R4` whereas in context-unaware cases he/she needs to consider rules `R1` through `R9`. Moreover, the inheritance semantics employed in CBRs usually reduces rule redundancy. Consequently, it is easier for rule developers to keep an overview of relevant rules and their relationships, such as rule dependencies, and thus to find specific rules in a CBR than in a NCBR. Furthermore, due to the context relationships in CBRs, contexts affected by a modification operation can be immediately identified and their responsible rule developers notified. Notified rule developers can react to the respective modification operation.

For users and domain experts, CBRs provide an explanation of the returned result, i.e., the relevant rules applied. In data tailoring use cases like SemNOTAM this enables application of the returned rules to other data of the same type, for instance, the same user situation but different NOTAM of the same event scenario. In NCBRs, an explanation includes all rules in the rule set; an explanation containing only rules contributing to the result would have to be constructed first. In CBRs the relevant contexts determined for a specific business case are associated with the business case. This allows users and domain experts to determine responsible rule developers if needed. Due to the separation of tasks, users and domain experts need not be able to author rules. They communicate with the rule developers who then author the rules. In NCBRs, since the rules relevant to a specific business case are not filtered, all rule developers may be responsible. In order to determine the rule developers responsible for the result, users or domain experts need to be able to understand rules.

5.5 Related Work

Most related work has already been discussed in Section 3.5. For CARVE [25, 160], Virgilio and Torlone's [199] general framework, CKRs [183], approaches regarding business rule organization (like [187]), and commercial BRMSs, modification operations of some form have been discussed.

Regarding *CARVE*, we note that any modification of the CDT by evolution operators [160] is logged. This log is used to map outdated contexts provided by users to current contexts. Compared to CBRs, CARVE defines one role for all operations, whereas we assign modification operations to four different roles allowing for specialization (for instance rule developers work only with rules). The mapping of outdated contexts can be achieved in CBRs by adapting the `detParamValue` methods.

Virgilio and Torlone's framework [199] comprises a general data model as well as basic primitives for manipulation. Our CBR model's modification operations are more extensive than the reported primitives. Furthermore, CBR models are presented in more detail and allow automatic determination of relevant contexts for a given business case.

According to Bozzato, Eiter, and Serafini [28] *CKRs* [183] comprise two levels: global context, containing meta-knowledge about contexts as well as context-independent knowledge; and local context, which adheres to the meta-knowledge defined in the global context, inherits context-independent knowledge, and additionally contains context-dependent knowledge. The global context may contain axioms which are defeasible and thus can be overridden. These axioms hold in the general case but allow exceptions in local contexts. Exceptions occur if including the axiom in the local context would lead to conflicts. Compared to CBRs, CKRs lack modification operations and roles as well as a mechanism for the automatic determination of relevant contexts for a given business case. Furthermore, CKRs are designed for ontological concepts and not for rules.

Business rule organization approaches propose to collect business rules into rule sets regarding a single attribute (for example commercial BRMSs, [82, 140, 161]) and multiple attributes (for example [33, 44, 87, 102, 140, 187, 188]. Sordo, Tokachichu, Vitale, Maviglia, and Rocha [187] supports the context operations presented in Section 2.5.4. Other approaches [33, 44, 82, 87, 102, 140, 161, 188] do not report on modification operations. Consequently, most approaches to business rule organization lack support for business rule maintenance.

Current *BRMSs* often provide tools or methods to modify the business rules in the repository, some also provide corresponding roles. Notification of users affected by a modification may be implemented, for instance, an external BPM tool may be used for change management in JRules. Nevertheless, most investigated BRMSs do not report on automatic notification of users affected by a modification operation. Compared to CBRs, BRMSs do not provide modification operations to the same extent.

5.6 Conclusion

In order to promote effective and efficient business rule maintenance and business rule entering vital to today's competitive businesses, we introduced atomic CBR model modification operations. We assigned the different atomic modification operations to four CBR roles: repository administrator, rule developer, domain expert, and user. Without these CBR roles, CBRs would not be practicable as a clear separation of responsibilities and tasks would be missing. Furthermore, since the four CBR roles usually already exist in businesses, migration from NCBRs to CBRs is eased.

In addition to atomic operations and CBR roles, we introduced composed modification operations relevant in practice. These composed operations maintain the consistency of CBR models and the organized business rules. We compared each composed operation to its semantic equivalent in NCBRs. Furthermore, we discussed the application of composed operations and CBR roles in CBR-specific BRM and described business rule acquisition and maintenance for SemNOTAM.

Regarding our claims, the atomic and composed modification operations as well as CBR roles introduced in this chapter further ease business rule maintenance.

Chapter 6

Rule Module Inheritance with Modification Restrictions

Adapting rule sets to different settings, yet avoiding uncontrolled proliferation of variations, is a key challenge of rule management. One fundamental concept to foster reuse and simplify adaptation is inheritance. Building on rule modules, i.e., rule sets with input and output schema, we precisely define inheritance of rule modules by incremental modification in single-inheritance and multi-inheritance hierarchies. To avoid uncontrolled proliferation of modifications, we introduce modification restrictions which flexibly regulate the degree to which a child module may be modified in comparison to its parent. As concrete rule language family, we employ Datalog$^{\pm}$ which can be regarded a common logical core of many rule languages. We evaluate the approach by a proof-of-concept prototype employing Vadalog, an implementation of Warded Datalog$^{\pm}$. An integration of this approach with the generic CBR model further improves business rule maintenance in CBRs by having a precise definition of inheritance and by using rule modules instead of simple rule sets .

6.1 Introduction

In data- and knowledge-intensive systems it is good practice to separate explicit knowledge, elicited from domain experts and translated into declarative expressions by rule developers, from application code developed by application developers. A vital form of declarative expressions are rules enabling complex reasoning tasks. Rule-based knowledge representation and reasoning build the core of systems for business rule engines [146], web data extraction [72], data wrangling [109], knowledge graph management [16], and information tailoring [41]. With the increasing number and complexity of rules, their maintenance and their adaptation to different settings become key challenges of rule developers. In this chapter we present an approach utilizing rule modules and inheritance to cope with these challenges.

Similar challenges are addressed by CBRs for business rules: one of the claims of CBRs, stated in Section 1.5 and demonstrated in Chapter 5, is the improvement of business rule maintenance. The integration of the generic CBR model and Rule Module Inheritance with Modification Restrictions (RMI), promises further benefits for business rule maintenance. For such an integration, the rule sets employed in the generic CBR model are replaced by rule modules. Thus, for the business rules of a context we explicitly define output and input schema and may furthermore define

This chapter extends the publication: F. Burgstaller, B. Neumayr, E. Sallinger, and M. Schrefl. “Rule Module Inheritance with Modification Restrictions”. In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. Ed. by H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman. 2018, pp. 404–422 modification restrictions. The inheritance hierarchy of rule modules is derived from the inheritance hierarchy of contexts; the inheritance mechanism itself is precisely defined. The integration of RMI and the generic CBR model is discussed in Chapter 7.

In the following, we first sketch challenges and our RMI approach along a self-contained business rule example. Subsequently, we zoom out to give the big picture of potential application areas where the presented approach may serve as a central building block. Finally, we give an outline of the remaining chapter.

6.1.1 Challenges and Approach

In this chapter, we present an approach which builds on *rule modules* (short: modules), i.e., rule sets with interfaces describing the schema of input and output data, to provide a clear separation and interfaces between rule sets and data-intensive applications. These interfaces shield application developers from the intricacies of rule sets as well as rule developers from application code and clarify which data schemata and knowledge are internal to the module and which interface either as input or output. — For example, a bank clerk is responsible for processing a set of mortgage applications, i.e., assessing the applications' credit worthiness in order of their priority. Domain knowledge regarding assessment and prioritization of mortgage applications is encoded in a rule module `MortgageApps`. This rule module is employed by a data-intensive application collecting and managing mortgage applications and their assessment. Rule module `MortgageApps` takes as input a set of mortgage applications each described by the mortgage value and the estimated values of real estate securities. As output it produces a preliminary assessment of the credit worthiness, either `good` or `bad`, of each application together with a prioritization of the applications for detailed assessment. Any issues with applications, such as having a mortgage value below the specified minimum loan value, are also output.

Organizing rules into rule modules entails the danger of *redundancy*: One and the same rule may be relevant in different settings and thus introduced and maintained separately in different rule modules. This duplicates human effort in developing and maintaining rule modules and makes it difficult to keep rules synchronized across modules. — For example, our bank decides to offer private loans as well, creating a rule module `PrivateLoanApps`. Some rules, such as the minimum loan value, apply in module `MortgageApps` as well as module `PrivateLoanApps`. Thus, any changes to this rule need to be performed in both rule modules.

In this chapter we introduce *inheritance* of parent modules to child modules by incremental modification as one way to mitigate these problems without sacrificing flexibility. In child modules, rule developers may introduce additional rules, eliminate inherited rules, as well as extend and/or reduce the input and output interfaces. This reduces redundancy and thus should ease maintenance of rule modules adapted to different business settings. — For example, extracting common rules, such as the minimum loan value rule, into a parent module `LoanApps` we can remove redundancy. Any changes to a common rule are made in the parent module and by inheritance propagated to all child modules. Moreover, since we allow modifications to inherited rules, we can define default rules for loan application assessment and ranking in module `LoanApps`.

Allowing arbitrary modifications in child modules, however, would undermine the benefits of inheritance and would pave the way for *uncontrolled proliferation of variations*: a rule or application developer trying to get an overall picture of the interfaces and the behavior, i.e., the output knowledge, encoded in a hierarchy of rule modules would still have to inspect all rule modules. Furthermore, undesired changes to inherited rules and interfaces would be possible. — For example, consider our minimum loan value rule in module `LoanApps`. Since arbitrary modifications are allowed we can simply eliminate this rule in child modules; any problematic loan applications would not be output anymore.

In this chapter, we introduce a set of *modification restrictions* to flexibly regulate the degree to which a module's interfaces and behavior may be modified in comparison to its parent module's interfaces and behavior. Treating rule modules as black boxes, modification restrictions set boundaries within which a module may be modified. Thus, it should be sufficient to inspect the root module and its modification restrictions to get an abstract overview of the behavior implemented in a hierarchy of modules.

Structural restrictions restrain allowed modifications to the input and output schema of a module. — For example, module `LoanApps` specifies the basic input for any loan application assessment. Thus, we define `LoanApps`' inputs as non-omittable, i.e., they must not be eliminated in child modules. Child module `MortgageApps` requires further inputs like real estates provided as securities. Therefore, no structural restrictions are necessary. Module `MortgageApps` fixes the output for all mortgage application modules. Thus, we define its output as non-extensible, i.e. further outputs must not be added in child modules.

Behavioral restrictions constrain modifications changing the output at instance level (for a particular output schema element a behavioral modification may lead to different instances in the output). — For example, we want to prohibit child modules from deriving a proper subset of minimum loan value issues compared to module `loanApps`. Thus, we employ restriction non-shrinkable. Similarly, child modules must not have weaker requirements for good credit worthiness. Thus, we employ restriction non-growable defining that child modules must not derive a proper superset of loan applications with credit worthiness good compared to module `loanApps`.

6.1.2 Potential Application Areas

Potential application areas, besides business rules as shown in the examples, are rule-based systems for information tailoring [41], web data extraction [72], and knowledge graph management [16] where rule modules encode knowledge for tasks such as data extraction, transformation, cleansing, and filtering and need to be adapted to different settings.

For example, in the DIADEM system for web data extraction [72], multiple rule modules, each responsible for a particular task such as web form understanding or form filling, are dynamically orchestrated in networks where one module's output is another module's input. Some of the encoded knowledge necessary for web form understanding is generic to 'all' web sites, while other encoded knowledge is specific to domains such as 'real estate' websites. Inheritance from the rule module for generic web form understanding to a rule module for web form understanding for 'real estate' websites should help to reduce redundancy and thus ease maintenance. Structural modification restrictions can be used to ensure that child modules remain orchestrable, similar to co- and contravariance in object-orientation which can be used to ensure type-safety and substitutability. Behavioral modification restrictions should help to keep the overall rule base understandable and thus maintainable, for instance, looking at a parent module and its behavioral restrictions gives an overview of possible behavior in child modules.

Regarding web developments, such as Internet of Things, semantic web, or Smart Things, rules become a vital and integral part [204]. Due to the number of rules and their context-dependent applicability, efficient rule management is essential. Rule modules and module inheritance are means to manage such large rule sets and can be extended to manage contexts of application (c.f. [37]).

We expect our approach to be applicable and beneficial in these areas – an evaluation is yet to be performed. In this chapter we focus on rule modules, their inheritance, and modification restrictions independent of specific applications.

6.1.3 Contributions and Overview

Currently, work on rule inheritance and related work on contextualized knowledge representation is fragmented and there exists no approach for inheritance of rule modules where modifications can be flexibly restricted. — In this chapter we introduce an overall approach to rule module inheritance and specifically make the following contributions:

- precise definition of (1) downward rule and interface inheritance of modules, and (2) modification restrictions and inheritance of modification restrictions
- discussion of conformance checks for detecting violations of defined structural and behavioral modification restrictions
- proof-of-concept prototype implementing definitions in Vadalog

The remaining chapter is structured as follows: Section 6.2 presents our Datalog$^\pm$-based rule language and rule modules. In Section 6.3 we present our inheritance mechanism. Section 6.4 introduces modification restrictions and delineates inheritance of modification restrictions. In Section 6.5 we present a proof-of-concept prototype. Section 6.6 discusses related work. Section 6.7 concludes the chapter.

6.2 Rule Modules

First, we discuss expressing rules with Datalog$^\pm$ where a rule is constructed from predicates and operators like logical conjunction. Subsequently, we introduce rule modules and discuss their structure and behavior.

6.2.1 Underlying Rule Language

This section delineates our notion of rules based on a formal language. In order to focus on rule module inheritance and modification restrictions, the underlying rule language and data model should be simple, i.e., have few constructs and operators, to avoid unnecessary complexity. A fitting family of formal languages is Datalog$^\pm$ [45] which has clearly defined formal semantics and employs a relational data model. For a general description of Datalog$^\pm$and its implementation Vadalog refer to Section 4.2.2.

A plain Datalog *rule* comprises a body (premise) and a head (conclusion) where the conclusion is derived if the premise holds. Both conclusion and premise are conjunctions of atoms (i.e., predicates with arguments). Datalog$^\pm$ allows use of existentially quantified variables in conclusions enabling value creation, truth value `false` in conclusions (negative constraints), and equality-generating dependencies like `Y=Z:- r1(X,Y), r2(X,Z).`

Definition 6.1 (Rule Structure). Predicates (a.k.a. relations) are taken from a universe of predicates P and are identified by their name. Rules are taken from a universe of rules R. A rule $r \in R$ has body predicates $B_r \subseteq P$ and head predicates $H_r \subseteq P$.

Example 6.1. Our bank deems mortgage loans of less than 10,000 Euro as not worth the organizational costs (rule `R0`). We translate this natural language rule to Vadalog: `lowLValue(X,V) :- lValue(X,V), V < 10000.` The mortgage value applied for is represented by `lValue/2` (predicate `lValue` with arity two) relating loan applications with loan values while `lowLValue/2` annotates mortgage applications below the defined mortgage value threshold. Rule `R0` has body predicates $B_{\texttt{R0}} = \{\texttt{lValue/2}\}$ and head predicates $H_{\texttt{R0}} = \{\texttt{lowLValue/2}\}$.

6.2.2 Rule Module Structure and Behavior

A collection of rules and facts (i.e., rules without variables and with premise true) is often called a program, for instance, a set of Datalog rules and facts is called a Datalog program. Such programs are often subject to global restrictions imposed by the respective rule language, i.e., restrictions that cannot be violated by a single rule but by a set of rules. For example, many Datalog$^\pm$ languages support a form of stratified negation, hence, programs in these concrete Datalog$^\pm$ languages do not allow for cycles with negation in the dependency graph. In the following, we extend valid programs in concrete languages of the Datalog$^\pm$ family, i.e., conforming to global restrictions of the respective concrete Datalog$^\pm$ language, by input schema and output schema to rule modules where the programs themselves are the implementation of the rule module. With rule modules being a generic concept (i.e., a rule-language-independent concept), it remains the task of the rule-language-specific rule engine to check observance of global restrictions.

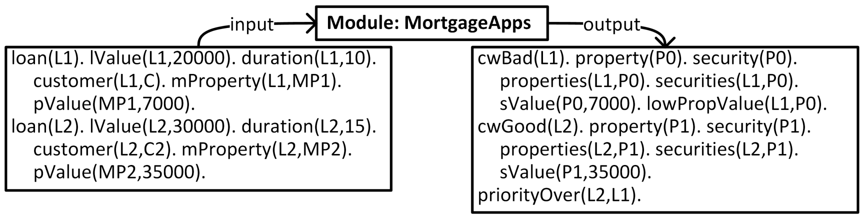
We derive *structural* aspects of rule modules from Datalog and Vadalog. Datalog splits the predicates of a program into extensional database (EDB) and intensional database (IDB). EDB contains predicates asserted in the knowledge base, IDB predicates defined by rules. A similar distinction is made in DMN's rule representation [145]. Vadalog [17] extends this idea to *inputs* provided by external sources (input predicates) and derived predicates *output* to external sinks (output predicates). These two sets are disjoint. Predicates derived and potentially used in rule bodies but not exported are auxiliary predicates.

Definition 6.2 (Rule Modules). Rule modules are taken from a universe of rule modules M. A rule module $m \in M$ is defined by a set of rules R_m, a set of input predicates $I_m \subseteq P$, and a set of output predicates $O_m \subseteq P$. The set of rules R_m must satisfy all global restrictions imposed by the employed rule language. The sets of input and output predicates are disjoint. The predicates of a module m, P_m, are the union of its rule head, rule body, input predicates, and output predicates.

We now discuss the development of a rule module from an organizational perspective: Domain experts elicit and determine rules, often in natural language, and organize them into rule sets which may cover a specific business case. Furthermore, domain experts determine the necessary data input and derived output for the elicited set of rules. Once the domain experts consider a rule set and its inputs and outputs complete, rule developers translate the elicited rules and interfaces into a rule module with a Datalog$^\pm$ program as implementation.

```
Module: MortgageApps
Input: loan/1, lValue/2, duration/2, customer/2, mProperty/2, pValue/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), properties(X,S), #sum{SV: sValue(S,SV)} = A, A > 0.8 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[R4_1] ∃N property(N), properties(X,N), sValue(N,PV) :- loan(X), mProperty(X,P), pValue(P,PV).
[R4_2] mProperty(X,Y) :- loan(X), mProperty(X,Z), hasPart(Z,Y).
[R5_1] securities(X,P) :- properties(X,P).
[R5_2] security(X) :- property(X).
[R6] lowPropValue(X,P) :- properties(X,P), sValue(P,V), V < 30000.
Output: cwGood/1, cwBad/1, priorityOver/2, security/1, securities/2, properity/1, properties/2, sValue/2,
  lowLValue/2, lowPropValue/2
```

FIGURE 6.1: Visual summary of module MortgageApps as described in Example 6.1 and 6.2 comprising three compartments: the *Input* and *Output* compartments containing input and output declarations of predicate names and their arity respectively and the *rules* compartment containing Vadalog rules with rule identifiers in brackets.



```
loan(L1). lValue(L1,20000). duration(L1,10).
  customer(L1,C). mProperty(L1,MP1).
  pValue(MP1,7000).
loan(L2). lValue(L2,30000). duration(L2,15).
  customer(L2,C2). mProperty(L2,MP2).
  pValue(MP2,35000).
```

```
cwBad(L1). property(P0). security(P0).
  properties(L1,P0). securities(L1,P0).
  sValue(P0,7000). lowPropValue(L1,P0).
cwGood(L2). property(P1). security(P1).
  properties(L2,P1). securities(L2,P1).
  sValue(P1,35000).
priorityOver(L2,L1).
```

FIGURE 6.2: Visualization of the execution of module MortgageApps on a dataset containing facts describing the two mortgage applications L1 and L2 and its output.

Example 6.2. We want to elicit all relevant rules for the process of assessing mortgage loan applications. In addition to rule `R0` (see Example 6.1) we identified: Credit worthiness of a mortgage application is deemed good (predicate `cwGood/1`) if the value of provided properties exceeds 80 % of the mortgage's value (`R1` – output `cwGood/1`). In all other cases credit worthiness is deemed bad (`R2` – `cwBad/1`). Loan applications of higher loan values have priority over those with lower loan values (`R3` – `priorityOver/2`). Each provided property and its value are associated with the corresponding loan application (`R4` – `property/1, properties/2, sValue/2`). Associated properties are securities (`R5` – `securities/2, security/1`). Properties of a value below 30,000 Euro are reported as problematic securities (`R6` – `lowPropValue/2`). The predicates derived by rules `R0`-`R6` form module `MortgageApps`'s output predicates $O_{\texttt{MortgageApps}}$.

From these natural language rules our domain experts derive the necessary input. The rules employ information given with each mortgage application (`loan/1`). Such applications need to state the intended mortgage value (`lValue/2`), the intended duration (`duration/2`), the applying customer (`customer/2`), and any real estate properties which may be used as securities (`mProperty/2`) as well as their value (`pValue/2`). These predicates form the input predicates $I_{\texttt{MortgageApps}}$. Furthermore, properties may be hierarchically organized (`hasPart/2`), for instance, a property may consist of an area containing buildings. A visual summary including the Vadalog representation is given in Figure 6.1.

Besides structure, a rule module exhibits behavior when executed. For this purpose, the comprised rule set must satisfy the global restrictions imposed by the employed rule language. We regard rule module *behavior* as observable effects, i.e., derived facts, when applying the set of rules in a module to a dataset containing facts for its input predicates. Multiple facts for each input predicate may be provided. A data set providing facts for all input predicates of a module is called applicable.

Definition 6.3 (Module Execution). Data sets are taken from a universe of data sets D. A data set $d \in D$ with schema $P_d \subseteq P$ contains extensions, i.e., sets of facts, for all $p \in P_d$. A data set d is applicable input for module m if $I_m \subseteq P_d$. The execution of a module m on applicable input data d results, for each output predicate $p \in O_m$, in a set of derived facts, denoted as p^d_m (p^d_m is the set of derived p-facts resulting from executing m on d).

Example 6.3. Executing module `MortgageApps` on a dataset of two mortgage applications `L1` and `L2`, its rules are evaluated and derived facts of output predicates (behavior) are returned (see Figure 6.2).

6.3 Inheritance

In the following, we shortly summarize different aspects of inheritance in general. We describe the scope of our approach regarding the found aspects and thereafter define for single-inheritance and multi-inheritance our notion of inheritance hierarchy for rule modules, inheritance of rules and sets of interface predicates (input and output predicates) therein, and abstract predicates and modules.

A common notion of inheritance is inheritance as incremental modification [194], that is, "reusing a conceptual or physical *entity* in constructing an incrementally similar one" [203, p. 55]. From related work we extracted various aspects and their options (Table 6.1): All found inheritance mechanisms are transitive. Furthermore, inheritance is often discussed regarding certain *foci*, i.e., signatures (schema of inputs and outputs), behavior, and implementation [20, 104, 113, 131, 150, 181, 203]. Besides these aspects, inheritance is usually distinguished by the *supported parent-cardinality* into single- and multi-inheritance [46, 104, 113, 150, 181, 194, 203, 205], i.e., child entities inherit from a single or multiple parent entities respectively. *Direction* of inheritance is distinguished into downward [46, 104, 194], like inheritance in Java, upward [46, 104, 194], such as an array inheriting properties from its entries, and lateral [46], like a motorized trike is a motorcycle except that is has three wheels. The latter direction can also be achieved by downward inheritance with elimination of features. *Granularity* [104, 194, 203] regards whether an inheritance mechanism is specified for groups of entities or single entities. Considering inheritance as incremental modification we found different *modifications*: extension, elimination, and redefinition. Extension adds features to child entities [22, 104, 113, 150, 180, 181, 194, 203, 205]. The contrary modification is elimination (also called reduction) removing features [180, 181, 194, 203]. Redefinition redefines but does not eliminate inherited features [104, 113, 150, 194, 203, 205]. Special cases of redefinition are refinement, specialization of inherited features (for example, to a more specific type) [22, 104, 113, 150, 180, 181, 203], and generalization, abstraction of inherited features (for example, to a more general type) [118, 181].

Regarding multi-inheritance, a discussion of potential conflicts and proposed resolution strategies is in order. Multi-inheritance produces conflicts if identically named features (semantically related or distinct) are inherited from different parents with different usages or implementations [194]. To resolve conflicts of fields, replication and unification can be used [56]. To resolve behavioral conflicts, various options have been proposed [56]: rejection [113, 181], not allowing; linearization, definition of a lookup method; prioritization, manual ranking [20, 192, 205]; renaming conflicting features; operation combination, combining conflicting operations, for instance,

TABLE 6.1: Summary of inheritance aspects and our focus for rule module inheritance. Cursive ones are considered in our approach.

Aspect	**Options**
Focus	*Signature* / *behavior* / implementation
Parent cardinality	*Multi* / single
Direction	*Downward* / upward / lateral
Granularity	*Group* / single entity
Modification	*Extension* / *elimination* / redefinition

averaging results; most specific arguments, most specific operation signature is used; explicit inheritance, using fully qualified names; credulous reasoning, overview of possible results [192]; skeptical reasoning, conflicting results are not returned [192].

We restrict our investigation of rule module inheritance to the following inheritance options: Focus is on *signature* and *behavior* inheritance. *Multi-inheritance* is investigated. Direction of inheritance is *downward*, from parent module to child module. Granularity is rule modules and thus *group*. Regarding modification we support *extension* and *elimination*.

In the following, we define a single-inheritance and a multi-inheritance mechanism for rules and sets of interface predicates of rule modules. These inheritance mechanisms do not provide guarantees regarding global restrictions on rule sets made by rule languages. More specifically, when a child rule module inherits rule sets from parent rule modules, which, each one by itself, fulfil all global restrictions of the rule language, the inheritance mechanism does not guarantee that the resolved rule set of the child module also fulfils these global restrictions. For example, when a child module inherits from two modules with only stratified negation, it is not guaranteed that the child module is also stratified. With RMI being a generic approach (i.e., a rule-language-independent approach), it remains the task of the rule-language-specific rule engine to check observance of global restrictions and it remains the task of the rule developer of the child module to remove violations of global restrictions.

6.3.1 Single-inheritance Mechanism

Rules sets are arranged in a single-inheritance hierarchy forming a forest, i.e., a child module may inherit from one parent only.

Definition 6.4 (Single-Inheritance Hierarchy). Rule modules are arranged in an single-inheritance hierarchy $H \subset M \times M$ which forms a forest (i.e., a set of trees). We say module m' directly inherits from m if $(m', m) \in H$, with m' playing the role of *child* and m playing the role of *parent*.

In this single-inheritance hierarchy, rules and sets of interface predicates are propagated from parent modules to their child modules. A child module may be modified by introducing additional rules or interface predicates and/or by eliminating inherited rules or interface predicates. Redefinition, not discussed here, can be achieved by a combination of extension and elimination. Modifying rules usually implies modification of a module's behavior.

Elimination requires a mechanism enabling identification of rules and interface predicates. We employ the combination of module name and local rule label as unique identifier, similar to package and class name in object-oriented languages. If a rule label is already uniquely identifying, the module name may be omitted. If it is not uniquely identifying and the module name is not stated, all matching rules are eliminated. Regarding interface predicates, we use their name as identifier as they are unique in a rule module.

Definition 6.5 (Single-inheritance by Incremental Modification). When a module m' directly inherits from module m, $(m', m) \in H$, the child module m' inherits the parent's rule set R_m from which it may eliminate a set of rules $R^-_{m'} \subseteq R_m$. Furthermore, a child module may add a set of rules $R^+_{m'} \subseteq R$. A child module's rule set $R_{m'}$ is then determined as $R_m \cup R^+_{m'} \setminus R^-_{m'}$. Single-inheritance and incremental modification of the sets of input and output predicates are defined analogously, i.e., $I_{m'} \overset{\text{def}}{=} I_m \cup I^+_{m'} \setminus I^-_{m'}$ and $O_{m'} \overset{\text{def}}{=} O_m \cup O^+_{m'} \setminus O^-_{m'}$.

Often discussed, in particular with object-orientation, are *abstract entities*. For instance, an abstract method is a method for which the signature is defined but no implementation is available [131, 181, 194, 203]. Methods fully defined and implemented are usually called *concrete*. Similarly, Wimmer et al. [205] define abstract rules as rules not executable per se but providing reusable behavior. Analogously, we distinguish predicates in a module into concrete and abstract predicates. We define concrete predicates as predicates which are either in the input interface, the truth value `true` (used to define facts and represented as a nullary predicate), or predicates which contain only concrete predicates in the body of any rule having them in the rule head. Abstract predicates are defined as predicates in a module which are not concrete. In object-oriented design, a class is abstract if it contains abstract elements. Analogously, we call a rule module abstract if it contains abstract predicates and concrete otherwise. Similar to abstract classes which cannot be instantiated, abstract modules should not be applied as their behavior is incomplete. Consequently, abstract predicates and modules should always be concreted in descendant modules. Leaf modules in the module hierarchy should always be concrete.

Definition 6.6 (Abstract Predicates and Modules). A predicate p depends on a predicate p' in module m, denoted as $dep_m(p, p')$, if there is a rule r in R_m which has p in the head and p' in the body, i.e., $dep_m(p, p') \overset{\text{def}}{=} \exists r \in R_m : p \in H_r \wedge p' \in B_r$. A predicate p is *concrete* for a module m, if it is nullary predicate `true` or an input predicate or depends on some and only concrete predicates, i.e., $concrete_m(p) \overset{\text{def}}{=} (p = \texttt{true}) \vee (p \in I_m) \vee ((\forall p' : dep_m(p, p') \rightarrow concrete_m(p')) \wedge (\exists p' : dep_m(p, p')))$. A predicate $p \in P_m$ is *abstract* if it is not concrete. A module is abstract if it has an abstract predicate.

From an organizational viewpoint, inheritance of rule modules can be employed for rule elicitation, authoring, and organization: (a) An existing module can be adapted to a more specific setting by constructing a child module. For this purpose, rule developers have to know which rules and interface predicates are contained in a module. To this end, they look at resolved modules, i.e., modules for which all inheritance relations have been resolved. (b) A parent module can be constructed by extracting common or similar rules and interface predicates from child modules. Abstract modules are of importance to the latter approach, allowing to extract common interface predicates although the rules defining those predicates are different in child modules. Constructing new rule modules, rule developers need to ensure that the resolved module's rule set satisfies global restrictions of the employed rule language. Hierarchical organization of modules eases management, in particular maintenance, as rule redundancy can be reduced and rule reuse is promoted.

Example 6.4. Recently, our bank decided to extend its services to private loan applications. Therefore, we elicited relevant rules for private loan applications. We uncovered overlaps with module `MortgageApps`, in particular rules `R0-R3`. Rule `R0` overlaps with 'Private loan values must exceed 12,000' (for now `RX`), `R1` with 'Credit worthiness of a private loan application is deemed good (predicate `cwGood/1`) if the value of provided securities exceeds 60 % of the loan's value' (for now `RY`), and rules `R2` and `R3` are identical. Non-overlapping are: Regarding income, private loan applications are reported if the customer earns less than 600 Euro per month (`R7`). The attachable income (`attachableIncome/1`) is calculated as 30 % of the income earned over the loan's duration and associated with the loan application (`R8` – `incomes/2`) as security (`R9` – `securities/2, security/1`), allowing to reuse rule `R1` for deriving credit worthiness. $O_{\texttt{PrivateLoanApps}}$ contains `cwGood/1`, `cwBad/1`, `attachableIncome/1`, `incomes/2`, `priorityOver/2`, `lowLValue/2`, `sValue/2`, and `lowIncome/2`.

For these rules we identified the required input predicates: private loan application (`loan/1`), the intended loan value (`lValue/2`), the duration (`duration/2`), the applying customer (`customer/2`), and any incomes which may be used as securities (`income/2`). These predicates form the input predicates $I_{\texttt{PrivateLoanApps}}$ which considerably overlap with $I_{\texttt{MortgageApps}}$.

Rules `R0`, `R2`, `R3`, and `RY` are actually default rules applying to all existing and future loan types. We extract the default rules and `MortgageApps`'s and `PrivateLoanApps`'s common interface predicates into a parent module `LoanApps`. Rule `RX` is renamed to `R0.1`, `R1` to `R1.1`, and `RY` to `R1`. Regarding interfaces we have $I_{\texttt{LoanApps}} = \{$`loan/1`, `lValue/2`, `duration/2`, `customer/2`$\}$ and $O_{\texttt{LoanApps}} = \{$`cwGood/1`, `cwBad/1`, `priority-Over/2`, `lowLValue/1`, `sValue/2`, `securities/2`, `security/1`$\}$. Since predicates `securi-ties/2`, `security/1`, and `sValue/2` are not derived in `LoanApps` but in its child modules, module `LoanApps` is abstract. A visual summary of our use case including modification restrictions and Vadalog representations is given in Figure 6.3. There, rule `R0.1` is added instead of rule `R0` and rule `R1.1` instead of rule `R1`. Concerning module `PrivateLoanApps` we have regarding rules: $R^{-}_{\texttt{PrivateLoanApps}} = \{\texttt{R0}\}$ and $R^{+}_{\texttt{PrivateLoanApps}} = \{\texttt{R0.1, R7, R8, R9\_1, R9\_2}\}$. Figure 6.4 depicts module `Private-LoanApps` with inheritance resolved.

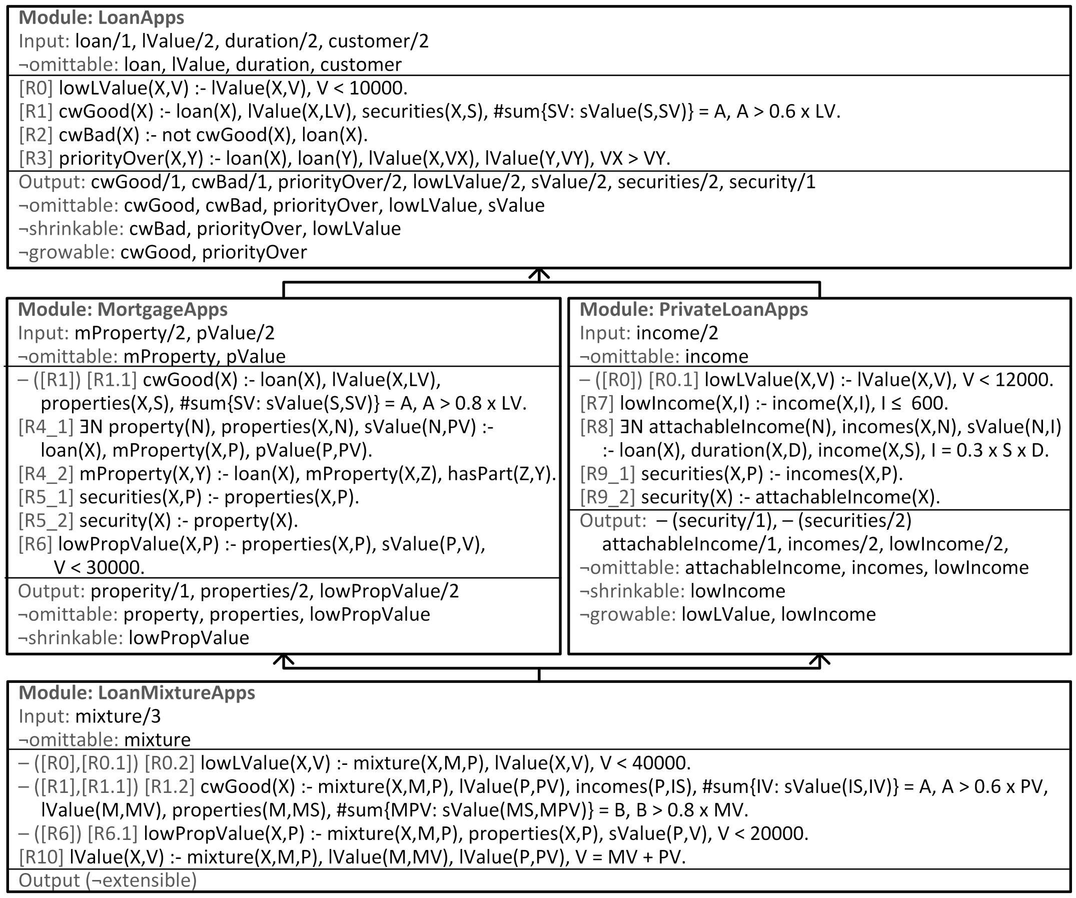
6.3.2 Multi-inheritance Mechanism

Multi-inheritance extends the single-inheritance mechanism introduced in the previous section. Rule sets are arranged in multi-inheritance hierarchies, that is, directed acyclic graphs.

Definition 6.7 (Multi-Inheritance Hierarchy). Rule modules are arranged in an inheritance hierarchy $H \subset M \times M$ which forms a directed acyclic graph. We say module m' directly inherits from m if $(m', m) \in H$, with m' playing the role of *child* and m playing the role of *parent*. We write H^+ for the transitive closure of H. We say module m' inherits from m if $(m', m) \in H^+$, with m' playing the role of *descendant* and m playing the role of *ancestor*.

Consequently, a child module may inherit from multiple parents, i.e., multi-inheritance merges several parent modules into one child module which may modify inherited rules and interface predicates. Abstract predicates and rule modules in multi-inheritance hierarchies are defined as for single inheritance (c.f. Definition 6.6).

Definition 6.8 (Multi-inheritance by Incremental Modification). A module m' directly inherits from modules m:$(m', m) \in H$. The child module m' inherits the union of its



```
Module: LoanApps
Input: loan/1, lValue/2, duration/2, customer/2
¬omittable: loan, lValue, duration, customer
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), securities(X,S), #sum{SV: sValue(S,SV)} = A, A > 0.6 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
Output: cwGood/1, cwBad/1, priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1
¬omittable: cwGood, cwBad, priorityOver, lowLValue, sValue
¬shrinkable: cwBad, priorityOver, lowLValue
¬growable: cwGood, priorityOver

Module: MortgageApps
Input: mProperty/2, pValue/2
¬omittable: mProperty, pValue
– ([R1]) [R1.1] cwGood(X) :- loan(X), lValue(X,LV),
   properties(X,S), #sum{SV: sValue(S,SV)} = A, A > 0.8 x LV.
[R4_1] ∃N property(N), properties(X,N), sValue(N,PV) :-
   loan(X), mProperty(X,P), pValue(P,PV).
[R4_2] mProperty(X,Y) :- loan(X), mProperty(X,Z), hasPart(Z,Y).
[R5_1] securities(X,P) :- properties(X,P).
[R5_2] security(X) :- property(X).
[R6] lowPropValue(X,P) :- properties(X,P), sValue(P,V),
    V < 30000.
Output: properity/1, properties/2, lowPropValue/2
¬omittable: property, properties, lowPropValue
¬shrinkable: lowPropValue

Module: PrivateLoanApps
Input: income/2
¬omittable: income
– ([R0]) [R0.1] lowLValue(X,V) :- lValue(X,V), V < 12000.
[R7] lowIncome(X,I) :- income(X,I), I ≤  600.
[R8] ∃N attachableIncome(N), incomes(X,N), sValue(N,I)
   :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
[R9_1] securities(X,P) :- incomes(X,P).
[R9_2] security(X) :- attachableIncome(X).
Output:  – (security/1), – (securities/2)
   attachableIncome/1, incomes/2, lowIncome/2,
¬omittable: attachableIncome, incomes, lowIncome
¬shrinkable: lowIncome
¬growable: lowLValue, lowIncome

Module: LoanMixtureApps
Input: mixture/3
¬omittable: mixture
– ([R0],[R0.1]) [R0.2] lowLValue(X,V) :- mixture(X,M,P), lValue(X,V), V < 40000.
– ([R1],[R1.1]) [R1.2] cwGood(X) :- mixture(X,M,P), lValue(P,PV), incomes(P,IS), #sum{IV: sValue(IS,IV)} = A, A > 0.6 x PV,
   lValue(M,MV), properties(M,MS), #sum{MPV: sValue(MS,MPV)} = B, B > 0.8 x MV.
– ([R6]) [R6.1] lowPropValue(X,P) :- mixture(X,M,P), properties(X,P), sValue(P,V), V < 20000.
[R10] lValue(X,V) :- mixture(X,M,P), lValue(M,MV), lValue(P,PV), V = MV + PV.
Output (¬extensible)
```

FIGURE 6.3: Visual summary of inheritance relations and rule modules as described in Examples 6.4–6.8. Child modules `MortgageApps`, `PrivateLoanApps`, and `MixtureLoanApps` incrementally modify their respective parent module(s) by adding and removing (denoted by '–()') rules, input declarations, and output declarations. Input and output compartments contain modification restrictions (denoted by ¬`omittable`, ¬`extensible`, ¬`shrinkable`, and ¬`growable`) with child modules adding additional modification restrictions.

parents' rule sets, i.e., $R^i_{m'}$ where $R^i_{m'} \stackrel{\text{def}}{=} \bigcup_{m:(m',m)\in H} R_m$, from which it may eliminate a set of rules $R^-_{m'} \subseteq R^i_{m'}$. Furthermore, a child module may add a set of rules $R^+_{m'} \subseteq R$. A child module's rule set $R_{m'}$ is then determined as $R^i_{m'} \cup R^+_{m'} \setminus R^-_{m'}$. Multi-inheritance and incremental modification of the sets of input and output predicates are defined analogously, i.e., $I_{m'} \stackrel{\text{def}}{=} I^i_{m'} \cup I^+_{m'} \setminus I^-_{m'}$ and $O_{m'} \stackrel{\text{def}}{=} O^i_{m'} \cup O^+_{m'} \setminus O^-_{m'}$.

The merging of parent modules can give rise to conflicts (known as diamond problem [194]). Since predicates are identified by their name, naming conflicts where two of a rule module's ancestors introduce an (interface) predicate with the same name but different semantics, cannot arise. Conflicts arise if a rule module's parents modify a rule or interface predicate inherited from a common ancestor differently. How a specific conflict is resolved, depends on the concrete use case. Thus, we decided for manual conflict resolution where we inform users about conflicts and they need to determine and apply the correct resolution strategy for each conflict.

Definition 6.9 (Multi-inheritance Conflicts). A module m' directly inherits from modules m_1 $(m', m_1) \in H$ and m_2 $(m', m_2) \in H$ with $m_1 \neq m_2$. Conflicts may arise if rules inherited by parent modules m_1 and m_2 from a common ancestor module m,

```
Module: PrivateLoanApps
Input: loan/1, lValue/2, duration/2, customer/2, income/2
¬omittable: loan, lValue, duration, customer, income
[R0.1] lowLValue(X,V) :- lValue(X,V), V < 5000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), securities(X,S), #sum{SV: sValue(S,SV)} = A, A > 0.6 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[R7] lowIncome(X,I) :- income(X,I), I ≤ 600.
[R8] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
[R9_1] securities(X,P) :- incomes(X,P).
[R9_2] security(X) :- attachableIncome(X).
Output: cwGood/1, cwBad/1, priorityOver/2, lowLValue/2, sValue/2, attachableIncome/1, incomes/2, lowIncome/2
¬omittable: cwGood, cwBad, priorityOver, lowLValue, sValue, attachableIncome, incomes, lowIncome
¬shrinkable: cwBad, priorityOver, lowLValue, lowIncome
¬growable: cwGood, priorityOver, lowLValue, lowIncome
```

FIGURE 6.4: Visual summary of module `PrivateLoanApps` with inheritance of rules, input declarations, and output declarations resolved according to Definition 6.5 and inheritance of modification restrictions resolved according to Definition 6.13.

$(m_1, m) \in H^+ \wedge (m_2, m) \in H^+$, are eliminated in one parent module or its ancestor modules but not the other, i.e., for any module m the set $(R_m \setminus R_{m_1}) \triangle (R_m \setminus R_{m_2})$ contains rules which are eliminated in one parent module or its ancestors modules only ($\triangle$ denotes symmetric difference). The conflict set is defined analogously for input predicates and output predicates.

Taking an organizational viewpoint, multi-inheritance of rule modules can be employed for rule elicitation, definition, and organization: Existing rule modules can be combined to construct a new child module applying in a more specific or different setting. To this end, rule developers consider resolved rule modules again. Any conflicts, violated global restrictions of the rule language, or unexpected behavior resulting from inheriting from several modules and modifying inherited rules and interface predicates need to be handled by the rule developers – they need to apply the right conflict resolution strategy or modification for the specific setting.

Example 6.5. Our bank decides to offer in addition to mortgage and private loans a mixture of both. Therefore, the new module `MixtureLoanApps` inherits from the modules `MortgageApps` and `PrivateLoanApps`. Subsequently, we discuss the resulting module `MixtureLoanApps` with domain experts and elicit missing and superfluous rules and interface predicates: We found that mixture loan values below 40,000 euro are considered problematic (`R0.2`), replacing rules `R0` and `R0.1`. A loan mixture's loan value is calculated by summing up the individual mortgage and private loan value (`R10`). Good creditworthiness for a mixture loan is given if the contained mortgage and private loan application fulfil their respective conditions for good creditworthiness (`R1.2`), rule `R1.2` thus replacing rules `R1` and `R1.1`. Since two loan types are combined, we lower the limit for properties acceptable as securities to 20,000 (`R6.1`) replacing rule `R6`.

These rules require the input of a loan mixture application (`mixture/3`). This loan mixture applications has an ID and references the mortgage application and the private loan application to be combined. The referenced mortgage and private loan applications need to be input as well. These predicates form the input predicates $I_{\texttt{MixtureLoanApps}}$. Compared to its parent modules, `MixtureLoanApps` does not output any additional predicates.

Once module `MixtureLoanApps` is specified, we take a look at potential conflicts resulting from multi-inheritance. We detect the following conflicts: (1) Module `Private-LoanApps` eliminates the predicates `securities/2` and `security/1` whereas module

```
Module: LoanMixtureApps
Input: loan/1, lValue/2, duration/2, customer/2, mProperty/2, pValue/2, income/2, mixture/3
¬omittable: loan, lValue, duration, customer, mProperty, pValue, income, mixture
[R0.2] lowLValue(X,V) :- mixture(X,M,P), lValue(X,V), V < 40000.
[R1.2] cwGood(X) :- mixture(X,M,P), lValue(P,PV), incomes(P,S), #sum{SV: sValue(S,SV)} = A, A > 0.6 x PV,
   lValue(M,MV), properties(M,SP), #sum{SV: sValue(SP,SV)} = A, A > 0.8 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[R4_1] ∃N property(N), properties(X,N), sValue(N,PV) :- loan(X), mProperty(X,P), pValue(P,PV).
[R4_2] mProperty(X,Y) :- loan(X), mProperty(X,Z), hasPart(Z,Y).
[R5_1] securities(X,P) :- properties(X,P).
[R5_2] security(X) :- property(X).
[R6.1] lowPropValue(X,P) :- mixture(X,M,P), properties(X,P), sValue(P,V), V < 20000.
[R7] lowIncome(X,I) :- income(X,I), I ≤  600.
[R8] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
[R9_1] securities(X,P) :- incomes(X,P).
[R9_2] security(X) :- attachableIncome(X).
[R10] lValue(X,V) :- mixture(X,M,P), lValue(M,MV), lValue(P,PV), V = MV + PV.
Output (¬extensible): cwGood/1, cwBad/1, priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1,
   property/1, properties/2, lowPropValue/, attachableIncome/1, incomes/2, lowIncome/2
¬omittable: cwGood, cwBad, priorityOver, lowLValue, sValue, property, properties, lowPropValue,
   attachableIncome, incomes, lowIncome
¬shrinkable: cwBad, priorityOver, lowLValue, lowPropValue, lowIncome
¬growable: cwGood, priorityOver, lowLValue, lowIncome
```

FIGURE 6.5: Visual summary of module `MixtureLoanApps` with inheritance of rules, input declarations, and output declarations resolved according to Definition 6.5 and inheritance of modification restrictions resolved according to Definition 6.13.

`MortgageApps` does not. (2) Rule `R0` is replaced in module `PrivateLoanApps` but not in module `MortgageApps`. Furthermore, (3) rule `R1` is replaced in module `MortgageApps` whereas module `PrivateLoanApps` does not modify it. Regarding the first conflict, we consider the eliminated predicates as useful for mixed loans. Therefore, we do not perform any action. For the latter two conflicts, conflict resolution is addressed by our new rules regarding low loan value and creditworthiness in module `MixtureLoanApps` replacing rules `R0` and `R0.1` as well as `R1` and `R1.1` respectively. Figure 6.5 depicts module `MixtureLoanApps` with inheritance and conflicts resolved.

6.4 Modification Restrictions

Of particular interest to this chapter are modification restrictions constraining the allowed modifications in child modules. We introduce restrictions for module structure prohibiting (a) to extend interfaces and (b) to eliminate specific predicates inherited from ancestor module interfaces. Regarding module behavior we introduce restrictions prohibiting: (c) to extend a module's behavior and (d) to reduce a module's behavior.

Definition 6.10 (Modification Restrictions). A module m may define a set of modification restrictions S_m of the following forms: *no_additional_input*, *no_additional_output*, *non_omittable_input(p)* with $p \in I_m$, *non_omittable_output(p)* with $p \in O_m$, as well as *non_growable(p)* and *non_shrinkable(p)* with $p \in O_m$.

6.4.1 Structural Modification Restrictions

In order to regulate modification operations on module interfaces, we introduce four restrictions. The restrictions *no_additional_input* and *no_additional_output* prohibit the addition of predicates to the input and output interface respectively. The restrictions *non_omittable_input* and *non_omittable_output* prohibit elimination of the specified predicate from the input and output interface respectively.

Definition 6.11 (Consistent Structural Modification). Let module m' directly inherit from module m, $(m', m) \in H$. Structural modifications in child module m' conform with modification restrictions in parent module m if the following conditions hold:

1. if $no\_additional\_input \in S_m$ then $I^+_{m'} = \emptyset$,
2. if $no\_additional\_output \in S_m$ then $O^+_{m'} = \emptyset$,
3. if $non\_omittable\_input(p) \in S_m$ then $p \notin I^-_{m'}$,
4. if $non\_omittable\_output(p) \in S_m$ then $p \notin O^-_{m'}$.

Example 6.6. During the rule elicitation process our domain experts determined several modification restrictions regarding module `LoanApps`: Any application for a specific loan type must contain at least the same information as an application for a generic loan. Consequently, we define *non_omittable_input* for predicates `loan/1`, `lValue/2`, `duration/2`, and `customer/2`. Furthermore, any loan application module must output at least the predicates for credit worthiness, priority, low loan values, and security values. Output securities may be replaced by more specific forms of outputs. Thus, we define `cwGood/2`, `cwBad/2`, `priorityOver/2`, `lowLValue/2`, and `sValue/2` as $non\_omittable\_output$. Predicates `security/1` and `securities/2` may be eliminated if necessary. In Figure 6.3 these restrictions are listed under ¬*omittable* in the respective interface.

Module `MixtureLoanApps` finalizes the output for any rule module inheriting from it, i.e., such a module may employ more input predicates but may not output additional predicates. Therefore, we define *no_additional_output* for module `MixtureLoanApps`. In Figure 6.3 we denoted this as ¬*extensible* written next to the output of module `MixtureLoanApps`.

From an organizational perspective, determining structural restrictions is part of rule elicitation, authoring, and organization. Conformance checking of structural restrictions is necessary whenever an interface of a rule module is changed, a new rule module is added to the module hierarchy, or any structural restrictions of an existing module are modified. In the latter case, conformance of all descendant modules of the modified module with the modified module's restrictions needs to be checked. A prerequisite for structural conformance checks is the identification of performed modifications. This is achieved by simple interface comparisons, for example, comparing a parent's input schema with its child's input schema.

6.4.2 Behavioral Modification Restrictions

Modifying rules contained in a rule module influences the module's behavior with respect to derived facts of output predicates. A child module, when applied on a dataset (see Def. 6.3), may return for a specific predicate the same, a proper superset, a proper subset, or a proper subset of a proper superset of derived facts compared to its parent module. To regulate behavioral modifications, we introduce two restrictions for output predicates: the restriction *non_growable* prohibits the derivation of a proper superset of facts for a predicate in child modules whereas *non_shrinkable* prohibits child modules from deriving a proper subset of the facts derived by the parent module for a predicate.

Definition 6.12 (Consistent Behavioral Modification). Let module m' directly inherit from module m, $(m', m) \in H$. Behavioral modifications in child module m' are consistent with modification restrictions in parent module m if the following conditions hold for every data set $d \in D$ which is applicable to both m and m', as well as for every predicate p which is in the output of m and m':

1. if $non\_growable(p) \in S_m$ then $p^d_{m'} \subseteq p^d_m$ and
2. if $non\_shrinkable(p) \in S_m$ then $p^d_{m'} \supseteq p^d_m$.

Example 6.7. Regarding behavior, domain experts reported several restrictions for module `LoanApps`: the basic rule for good credit worthiness (`R1`) is the minimum requirement, i.e., its condition may only be stricter. Hence, we specify $non\_growable(cwGood)$ defining that a loan application not deemed credit worthy according to the rules in module `LoanApps` must not be derived as credit worthy by descendant modules. Since every loan application is classified either `cwGood/1` or `cwBad/1` we state $non\_shrinkable(cwBad)$. Furthermore, the loan value in `R0` is the minimum threshold for loan values. Consequently, the value may only be increased when specializing module `LoanApps`, represented as $non\_shrinkable(lowLValue)$. As every loan application must be prioritized, behavior regarding `priorityOver/2` must not change represented as $non\_growable(priorityOver)$, $non\_shrinkable(priorityOver)$.

From an organizational viewpoint, behavioral restrictions are determined during rule elicitation, authoring, and organization. To perform *behavioral conformance checks*, performed behavioral modifications are compared to behavioral modification restrictions. This check can be performed (a) during rule module testing or (b) during execution: (a) Before modifications to a module are disseminated and deployed, they need to be thoroughly tested, i.e., the module, any parent module, and any child modules are tested with various input data sets. In addition to traditional testing, we then employ modification detection (see below) to check conformance to defined modification restrictions. (b) We can also compare the behavior of parent and child module at runtime when the child module is executed. Any violated behavioral restrictions are reported.

6.4.3 Detection of Behavioral Modifications

To determine performed behavioral modifications we propose: (a) asking responsible rule developer(s) to state his/her performed behavioral modifications manually, and (b) to automatically detect (most likely) performed behavioral modification operations employing static or dynamic detection.

We expect complete *static detection* of behavioral modification operations (by automatic reasoning over programs in concrete Datalog$^{\pm}$ languages such as Vadalog) to be undecidable due to rule dependencies as well as references and predicates within input data (a formal proof is beyond this thesis). Nevertheless, determination of likely behavioral modification operations is feasible by comparing a child module's with its parent module's rule dependency graph: For a parent module m and child module m' we determine for each predicate $p \in P_m$ whether the set of facts derived in module m is likely to remain the same, grow, shrink, or change compared to the set of facts derived in module m'. If added rules $r \in R^+_{m'}$ have p in their rule head, the set of derived output facts most likely grows ($p^d_{m'} \supset p^d_m$). Analogously, if two rules from different parents, $r_1 \in R_{m_1}$ and $r_2 \in R_{m_2}$ where $(m', m_1) \in H$, $(m', m_2) \in H$, and $m_1 \neq m_2$, have p in their rule head. If inherited rules are eliminated $r \in R^-_{m'}$

which have p in their rule head, the set of derived output facts most likely shrinks ($p^d_{m'} \subset p^d_m$). We regard any other changes, such as modifications to conditions, as arbitrary modifications for which we cannot determine potential effects on the derived output facts. Thus, we report the set of output facts for predicates whose derivation rules are arbitrarily modified to potentially grow and shrink (change).

The determined likely behavioral modifications need to be propagated along rule dependencies, for instance, in our use case deriving less facts for predicate `cwGood` implies more facts for predicate `cwBad`. To this end, we assume: (a) If the set of output facts for a predicate employed non-negated in a rule's body most likely grows or shrinks, this implies potential growth respectively potential shrinkage of the set of output facts for each predicate employed in the rule's head. (b) If the set of output facts for a predicate employed negated in a rule's body most likely grows or shrinks, this implies potential shrinkage respectively potential growth of the set of output facts for each predicate employed in the rule's head. In the end, we have determined for each predicate p in the parent module m its likely behavioral modification(s) in child module m'.

We now introduce *dynamic detection* of behavioral modifications which can be used as part of testing or during rule execution. For this purpose, a child module and its parent module are executed on the same input data and their output facts compared. For a specific parent module m and child module m' we select a set of data sets from D applicable to both the child and the parent module. For each data set d, we execute both modules and compare the derived facts for concrete predicates; abstract predicates are not considered in conformance checks as their behavior is incomplete. For each predicate p, concrete in the parent and child module, we compare $p^d_{m'}$ with p^d_m. If $p^d_{m'} \subset p^d_m$ the set of output facts has shrunk; if $p^d_{m'} \supset p^d_m$ the set of output facts has grown; if neither $p^d_{m'} \subset p^d_m$ nor $p^d_{m'} \supset p^d_m$ but $p^d_{m'} \neq p^d_m$ the set of output facts has changed (reported as both shrunk and grown); and lastly $p^d_{m'} = p^d_m$ implies no changes in the set of output facts. The more data sets from D are employed, the more reliable the detected modification(s) in behavior are. The overall modification in behavior for a specific predicate is the union of all detected modifications in behavior.

Detected behavioral modifications are then compared to specified behavioral modification restrictions.

6.4.4 Inheritance of Modification Restrictions

In order to achieve transitive conformance to modification restrictions we introduce inheritance of modification restrictions. Basically, child modules must not eliminate any modification restrictions imposed on their ancestors.

Definition 6.13 (Inheritance of Modification Restrictions). Let module m' inherit directly from parent modules $m : (m', m) \in H$. The child module m' inherits the union of its parents' modification restrictions, i.e., $S^i_{m'}$ where $S^i_{m'} \overset{\text{def}}{=} \bigcup_{m:(m',m)\in H} S_m$. A child module may add a set of modification restrictions $S^+_{m'}$. The child module's modification restrictions are then determined as $S_{m'} \overset{\text{def}}{=} S^i_{m'} \cup S^+_{m'}$.

Example 6.8. Module `PrivateLoanApps` inherits from module `LoanApps`. Besides rules and interface predicates, the defined modification restrictions are inherited. Consequently, any modification restriction defined in module `LoanApps` must hold in module `PrivateLoanApps` as well. This is depicted in Figure 6.4 where inheritance has been resolved for module `PrivateLoanApps`. For example, $non\_omittable(cwGood)$ defined in module `LoanApps` must also hold in module `PrivateLoanApps`.

Since modifications restrictions are inherited and cannot be revoked, only one kind of multi-inheritance issue can arise: two parents of a child module differ in the restrictions they specify. Nevertheless, this is not a critical issue as restrictions remove rights and consequently no conflicts can arise by adding restrictions.

Example 6.9. Module `PrivateLoanApps` prohibits child modules from deriving different loan applications with a low loan value by defining $non\_growable(lowLValue)$. Module `MortgageApps` does not so. Since module `MixtureLoanApps` inherits from both modules, $non\_growable(lowLValue)$ must hold although not defined in module `MortgageApps`.

6.5 Proof-of-Concept Prototype

Our proof-of-concept prototype implements the presented definitions, structural and behavioral conformance checks, and proposed modification restrictions using Vadalog. Consequently, our prototype must provide the following functionalities:

(1) Given an inheritance hierarchy of rule modules and a target rule module, return a new Vadalog program for the target module with all inheritance relations resolved.

(2) Given an inheritance hierarchy of rule modules, warn about potential issues with inheritance and modification operations such as multi-inheritance conflicts or referencing an unknown rule identifier.

(3) Given a resolved rule module, detect abstract predicates and determine whether the rule module is abstract.

(4) Given an inheritance hierarchy of rule modules, detect violations of structural modification restrictions.

(5) Given an inheritance hierarchy of rule modules, detect likely violations of behavioral modification restrictions.

To this end, we utilize meta-representations of rule modules including their interfaces, inheritance relations, modification operations, and modification restrictions. We embed our prototype in a Vadalog environment able to generate meta-representations of Vadalog programs and able to map meta-representations back to Vadalog programs. Before we discuss the concrete implementation of our prototype we discuss Vadalog annotations and introduce the employed meta-representation of Vadalog programs.

6.5.1 Vadalog Annotations

As described in Section 4.2.2, Vadalog is an implementation of Warded Datalog$^{\pm}$. To mark predicates which are input by external sources or output to external sinks, Vadalog introduces the concept of *annotations*. Annotations can be defined for a Vadalog program or for specific statements in a Vadalog program (currently rules and facts). Several predefined annotations with specific semantics exist; user-defined annotations may be introduced. Important predefined annotations are given in

TABLE 6.2: Important predefined program- and statement-level annotations in Vadalog.

Annotation	Semantics
`@label`	Labels a Vadalog statement, i.e., a rule or fact
`@input`	Marks a predicate as input predicate
`@output`	Marks a predicate as output predicate
`@bind`	Binds a predicate to an external source/sink such as a database table
`@post`	Specifies post processing steps such as ordering on facts of an input/output predicate
`@implement`	Specifies an external realization for a predicate

LISTING 6.1: Sample Vadalog program with input and outputs.

```
1       @input("loan").
2       @implement("val","java","~/valueEstimate.jar","estimate").
3
4       @label("R0") lowLValue(X,V) :- lValue(X,V), V < 10000.
5
6       @output("cwGood"). @bind("cwGood","postgres","apps","cw").
7       @output("cwBad").
```

Table 6.2. All annotations except `@label`, used to label statements, are program-level annotations. Annotations `@input` and `@output` specify that facts of the specified predicate are input to or output by the program. Annotations `@bind`, `@post`, and `@implement` may be used to specify external sources, post-processing operations, and external Java or Python functions respectively.

Example 6.10. Listing 6.1 displays an excerpt of the Vadalog program for module `LoanApps`. Line 1 specifies predicates `loan/1` as input. Line 2 defines that predicate `val/1` refers to a function `estimate` in the Jar-file `valueEstimate.jar`. Line 4 displays a rule definition labeled `R0`. Line 6 specifies predicate `cwGood/2` as output predicate which is bound to the PostgreSQL database schema `apps` and table `cw`. Line 7 specifies `cwBad` as output predicate.

6.5.2 Vadalog Meta-Representations

Vadalog provides a tool to generate meta-representations of Vadalog programs and to transform meta-representations back into Vadalog programs. A meta-representation vocabulary has been defined (shown in Figure 6.6), enabling each kind of statement (rule, fact, or annotation) to be represented by a specific set of meta-predicates. An example utilizing this meta-representation vocabulary is given below.

In order to identify the different statement parts in a program, an ID-generating function is used: each `rule`, `annotation`, `relationalAtom`, `nonRelationalAtom`, and `term` is assigned an ID unique within the Vadalog program for which a meta-representation is to be generated. To enable global identification, the ID-generating function can be provided a prefix identifying the original Vadalog program.

Example 6.11. Listing 6.2 presents the meta-representation of rule `R0` in our use case. We employed the prefix `loan` for generating the meta-representations resulting in

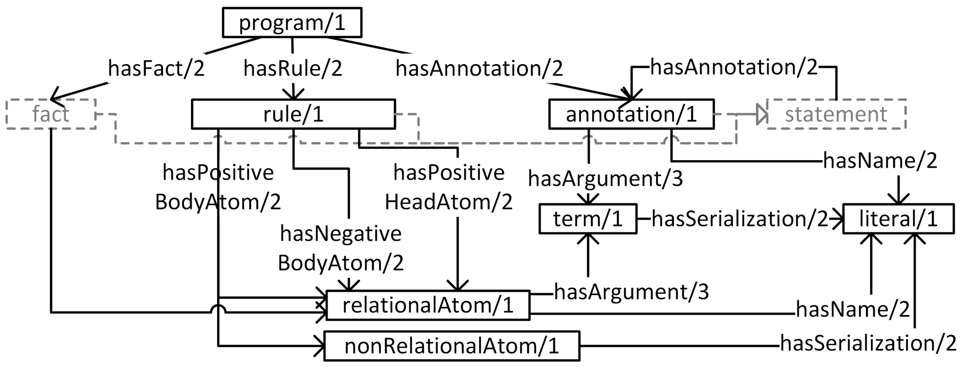


FIGURE 6.6: Visual summary of the meta-representation vocabulary used in Vadalog. For each kind of statement the predicates used to represent the statement are shown. Rectangles represent unary predicates while associations represent binary predicates except for `hasArgument` which is a ternary predicate also stating the position of the argument. Elements in gray with dotted lines are only shown for homogeneity but are not included in the vocabulary. Associations with the same name are represented by the same predicate.

`program("loan")` in Line 1. Line 2 states that the program contains a rule with the generated ID `loan:0`. Lines 5–7 express the rule's head atom `lowLValue(X,V)`, Lines 10–12 its body atom `lValue(X,V)` using the meta-vocabulary. These two atoms are relational atoms. Line 13 expresses the rule's condition `V<10000`, a non-relational atom. Lines 16–17 express the label annotation `@label("R0")` of the rule.

6.5.3 Prototype Design

To implement our approach, we introduce several user-defined annotations to Vadalog enabling us to specify module names, module parents, modification operations, and modification restrictions (c.f. Table 6.3). General annotations enable naming of rule modules (`@module`), specification of the rule module which generated a fact set (`@resultset`), as well as specification of parent modules (`@inherits`). Annotations

LISTING 6.2: The meta-representation of rule `R0` presented in Listing 6.1 (Line 4) employing the meta-representation vocabulary shown in Figure 6.6 with prefix `loan`.

```
1   program("loan").
2   hasRule("loan","loan:0"). rule("loan:0").
3
4   % Rule head
5   hasPositiveHeadAtom("loan:0","loan:1"). relationalAtom("loan:1").
      ↪ hasName("loan:1","lowLValue").
6   hasArgument("loan:1","loan:2",0). term("loan:2"). hasSerialization("loan:2","X").
7   hasArgument("loan:1","loan:3",1). term("loan:3"). hasSerialization("loan:3","V").
8
9   % Rule body
10  hasPositiveBodyAtom("loan:0","loan:4"). relationalAtom("loan:4").
      ↪ hasName("loan:4","lValue").
11  hasArgument("loan:4","loan:5",0). term("loan:5"). hasSerialization("loan:5","X").
12  hasArgument("loan:4","loan:6",1). term("loan:6"). hasSerialization("loan:6","V").
13  hasPositiveBodyAtom("loan:0","loan:7"). nonRelationalAtom("loan:7").
      ↪ hasSerialization("loan:7","V<10000").
14
15  % Rule annotation
16  hasAnnotation("loan:0","loan:8"). annotation("loan:8"). hasName("loan:8","label").
17  hasArgument("loan:8","loan:9",0). term("loan:9"). hasSerialization("loan:9","""R0""").
```

TABLE 6.3: User-defined Vadalog program- and statement-level annotations utilized for RMI.

Annotation	**Semantics**
`@module`	Defines a rule module's name
`@resultset`	Used with dynamic behavioral detection to annotate output fact sets with the rule module's name whose execution lead to the output fact set
`@inherits`	Specifies a parent rule module
`@eliminateQ`	Eliminates the inherited statement with the specified label introduced in the given ancestor rule module
`@eliminate`	Eliminates inherited statement(s) with the specified label
`@no_additional_input`	Additional input predicates are prohibited
`@no_additional_output`	Additional output predicates are prohibited
`@no_additional_annotation`	Additional annotations for the specified predicate and annotation type are prohibited
`@non_omittable`	The specified predicate must not be eliminated
`@non_shrinkable`	Deriving a proper subset of facts for the specified predicate in a child module is prohibited
`@non_growable`	Deriving a proper superset of facts for the specified predicate in a child module is prohibited

for elimination are available using the label of the rule/fact and the rule module (`@eliminateQ`) and using the label only (`@eliminate`). For each modification restriction, except `non_omittable_input` and `non_omittable_output`, we provide one annotation. A single annotation for `non_omittable` is sufficient as input and output predicates must not overlap in Vadalog. Additionally, we provide `@no_additional_annotation` prohibiting addition of specific annotations.

Example 6.12. Listing 6.3 displays an example Vadalog program for a slightly modified module `PrivateLoanApps`. Line 1 defines the rule module's name to be `PrivateLoan-Apps` which inherits from module `LoanApps`. Line 2 specifies predicate `income/2` as input predicate and furthermore defines that it must not be omitted in child modules. Line 4 displays a rule definition labeled `R0.1`. Line 5 defines the elimination of rule `R0` introduced in rule module `LoanApps`. Line 6 exemplifies that modification operation *addition* does not require any annotation. Line 8 defines that descendant modules must not add output predicates. Line 9 defines `lowIncome/2` as output predicate which must not be omitted in child modules and where child modules must derive exactly the same facts as `PrivateLoanApps` does. Elimination of predicates `security/1` and `securities/2` introduced in module `PrivateLoanApps`' output interface is not shown as not supported by the current Vadalog implementation (annotations cannot be labeled and thus cannot be referenced).

We implemented inheritance resolution, abstractness checking, modification detection, and conformance checking in Vadalog programs[1].

[1]The full Vadalog programs as well as Jupyter notebooks containing tests are available at `http://files.dke.uni-linz.ac.at/publications/burgstaller/Inheritance.zip`

LISTING 6.3: Slightly modified excerpt of the Vadalog program for `PrivateLoanApps` with annotations for rule module name, module parents, modification operations, and modification restrictions.

```
@module("PrivateLoanApps"). @inherits("LoanApps").
@input("income"). @non_omittable("income").

@label("R0.1") lowLValue(X,V) :- lValue(X,V), V < 12000.
@eliminateQ("R0","LoanApps").
@label("R7") lowIncome(X,I) :- income(X,I), I <= 600.

@no_additional_output.
@output("lowIncome"). @non_omittable("lowIncome"). @non_shrinkable("lowIncome").
↪  @non_growable("lowIncome").
```

Inheritance resolution is implemented in `inheritanceOnly.vada`; Jupyter notebook files starting with `Inheritance` contain tests thereof. The program needs the meta-representations of the relevant rule modules forming an inheritance hierarchy as well as the target rule module, i.e., the rule module which should be resolved. The program outputs the meta-representation of the resolved rule module. This meta-representation can subsequently be mapped to a Vadalog rule module. Furthermore, the program warns about issues in the provided rule modules, such as referencing labels not associated with any statements as well as multi-inheritance issues. This program implements functionalities (1), inheritance resolution, and (2) warnings about potential issues.

The check for abstract predicates is implemented in `abstractsOnly.vada` and tested in `AbstracsOnly.ipynb`. The program needs the meta-representation of a resolved rule module as input. For this resolved rule module the program outputs abstract predicates and whether the rule module is abstract or not. Thus, functionality (3), detection of abstract predicates and rule modules, is supported.

Conformance checks are implemented in `conformanceOnly.vada` and tested in `ConformanceOnly.ipynb`. The program needs the detected or reported modification operations (both structural and behavioral) for a rule module as well as the modification restrictions holding for the rule module. The program outputs any violations of modification restrictions by the given modification operations.

In order to perform conformance checks, the performed modification operations need to be reported or detected. Regarding detection of modification operations we provide three programs: Structural modification operations can be detected employing `structuralDetectionOnly.vada` while behavioral modification operations can be detected utilizing either `staticBehavioralDetectionOnly.vada` or `dynamicBehavioralDetectionOnly.vada`. Tests for these programs are given in the respective Jupyter Notebooks. Programs `structuralDetectionOnly.vada` and `staticBehavioralDetectionOnly.vada` need the meta-representations of the relevant rule modules forming an inheritance hierarchy. The former program outputs added and omitted interface predicates as well as added and omitted predicate annotations. The latter outputs for each output predicate whether the output fact set is more likely to remain the same, to grow, to shrink, or to change. Program `dynamicBehavioralDetectionOnly.vada` requires meta-representations of two result sets which were generated by a module and one of its ancestors when applied to the same dataset as well as the inheritance relationship of their producing rule modules. Outputs are the same as for `staticBehavioralDetectionOnly.vada`. These programs together with `conformance-Only.vada` provide functionalities (4) and (5), detecting violations of modification restrictions.

6.6 Related Work

In the following we discuss related work on: inheritance of single rules; customization rules for meta-model extension; rule sets, rule modules, and their inheritance; as well as contextual knowledge and its inheritance. Several researchers proposed inheritance of pre- and postconditions of operations (rule premises) [131, 142, 181] where modifications to conditions are restricted to strengthening or weakening. Other research regards inheritance of triggers (which can be considered event-condition-action rules) in object-oriented databases [20], where a trigger's premises may become weaker and a trigger's actions may be extended. $\mathcal{FLORA}$-2 [104] combines rules and object-oriented features where methods are specified as rules and can be inherited.

Rule inheritance is an important reuse mechanism in model-to-model transformation languages. Wimmer et al. [205] propose a comparison framework for rule inheritance in model-to-model transformation languages and evaluate a number of languages employing their framework. The framework comprises criteria covering static aspects such as syntax and static semantics, indicating whether an inheritance hierarchy of transformation rules is well-formed, as well as dynamic semantics, indicating how such a hierarchy behaves at run time. Their understanding of rules is that each rule has input and output elements as well as conditions which are inherited by child rules. Abstract rules, which are not executable per se, must be refined to executable rules. Several modes of refinement are described: override, inherit, merge, and extend. Furthermore, several static checks are provided, testing, for instance, for multi-inheritance conflicts, missing concrete rules for abstract ones, or co-variance of input and output elements. Regarding dynamic semantics various dispatch and execution semantics are discussed. The evaluation revealed that the compared languages have similar syntax but different inheritance semantics, are limited in their support for static semantics checks, often have fixed dynamic semantics, and lack access modifiers.

Compared to our RMI approach, rule inheritance in model-to-model transformation languages [205] regards single rules compared to rule sets in our approach. The concept of rules with input and output elements as well as conditions is quite similar to rule modules at rule set level. Furthermore, abstract rules are comparable to our abstract predicates. The discussed refinement modes cover a larger range than our extension and elimination modification operations. Our modification restrictions can be seen as kind of access modifiers in rule modules.

Jácome and De Lara [98] utilize rules to define allowed customizations and extensions to meta-models. This is important as some meta-models are designed to be extended by other developers. However, current approaches use natural language to express the ways in which a meta-model is to be extended, conflicting with the preciseness of the instantiation mechanisms of meta-models. Jácome and De Lara propose a customization meta-model which can be used to annotate elements of a meta-model which is to be extended (base meta-model). This customization meta-model comprises rules allowing extension, deletion, update, addition, and redefinition of elements in the base meta-model, for instance, a concrete rule may define that a specific class may be extended at most once. Due to the strict semantics of these rules, violations can be easily detected. Furthermore, the customization meta-model is non-intrusive and generic, i.e., it can be used with any meta-model.

The basic idea of this approach is similar to modification restrictions in our RMI approach. A difference is that Jácome and De Lara [98] use permits while RMI employs restrictions. Their rules regard meta-model elements while our modification restrictions regard rule modules. Jácome and De Lara's permits are more fine-grained than the modification restrictions we employ in RMI. Compared to modification restrictions, their rules regard static aspects of meta-models only. Analogously to RMI, violations of defined permits can be detected.

6.6.1 Rule Sets, Modules, and Inheritance

Modularization is employed in many research fields, such as software engineering or ontologies, enabling controlled and structured development of large systems [151]. Regarding knowledge, a knowledge base may either be partitioned into non-overlapping modules or relevant portions extracted into possibly overlapping modules. The concrete modularization strategy depends on the use case.

Regarding ontologies, a triple containing an ontology, a query language, and a vocabulary can be considered a module, where the query language and the vocabulary serve as interface allowing interaction by querying [108]. A simple extension mechanism enabling addition of ontological statements is described [108].

Brown and Pomykalski [33] propose two approaches to rule base partitioning. One partitions rule bases into sets of possibly overlapping rule chains leading from inputs to outputs. The other partitions the rule base into sequences of rule groups where the output of one group is the input for the next group.

Rule modules with relational schemas as interfaces are a simplified variant of relational transducers [1]. In the original proposal, relational transducers serve as "declarative specifications of business models, using an approach in the spirit of active databases" [1, p. 1]. Relational transducers transform sequences of input into sequences of output relations. In addition to input and output relations, a relational transducer specifies database, state, and log relations, where the log relations are the semantically relevant subset of input and output relations. Regarding inheritance, they discuss customization of relational transducers and with regard to restricted modification they discuss "containment and equivalence of relational transducers relative to a specified log" [1, p. 7].

Modular web rule bases [4] separate interfaces, i.e., predicates used and predicates defined, from the logic program, i.e., rules. Each predicate in a module defines its reasoning mode, its scope, and its origin rule bases and rule bases it is visible to. Modular rule base extension is supported, allowing to add new rules to existing rule bases and to add new rule bases to the set of rule bases.

Inheritance of (business) rules is touched in [37, 39] (see contextualized knowledge below) and discussed specific to situation-condition-actions rules in [113]. Inheritance of rule sets is discussed in [150]. Lang [113] identifies the rule of origin as prerequisite, i.e., a feature may only be defined in a single point. Moreover, he utilizes the abstract parent class rule. Situations, conditions, and actions may be specialized if the occurrence of a situation implies occurrence of its child situations and rule conditions are only weakened. Based on these conditions modification operations are proposed [113]: extension denotes addition of rules or redefining rules to fire more often, refinement denotes concreting abstract rules, i.e., concreting rule-triggering interval classes to event classes, and redefinition is constrained to specialized actions and events as well as weakened conditions. Pachet [150] describes rule set inheritance as inheriting all rules from parent rule sets and allows for unconstrained redefinition of rules.

A related field are business rule management systems (BRMSs) like IBM's JRules, JBoss Drools, FICO Blaze Advisor, or Oracle Business Rules, which organize business rules into rule sets. JRules supports inheritance of rules and rule sets where rules may be overridden. Drools and Blaze both support inheritance of rule conditions. Oracle does not report support for rule (set) inheritance.

Many of the above approaches support rule sets ([33, 37, 39, 104, 108, 113, 150, 151], BRMSs) but only Abiteboul et al. [1], Analyti et al. [4], and Konev et al. [108] describe modules with explicit definition of input and output interfaces as supported by our approach. Rule set inheritance is supported by quite a few approaches, where some allow no or only predefined modification types ([1], [108], [113], [37], Drools, Blaze) and others allow arbitrary modifications ([39], [150], JRules). Nevertheless, none of them provides any means to assert fine-grained control over the allowed types of modifications to inherited rule sets as our approach does with the presented modification restrictions. By these modification restrictions, our RMI approach allows, unlike the ones discussed above, to flexibly adapt the inheritance mechanism to the specific needs at hand.

6.6.2 Contextualized Knowledge Representation and Inheritance

A related research field are contextual knowledge and its inheritance. Detailed descriptions of contextual approaches are given in Sections 2.5 and 3.5. A notable approach are CKRs [28, 183] for the semantic web which organize ontological concepts employing hierarchically ordered contexts. Concepts propagate along this hierarchy from general contexts to more specific contexts. A similar idea was proposed for knowledge organization in Cyc [115] where it is explicitly allowed to contradict or override inherited knowledge. McCarthy [127] views contexts as generalizing collections of assumptions where a child context must have at least the same assumptions as its parent context. Analyti, Theodorakis, Spyratos, and Constantopoulos describe context refinement where a child context inherits all objects, names, and relationships from a parent context while references of objects in the parent context are either undefined or refined by references in the child context.

Building on these approaches, we introduced CBRs (Chapter 3) organizing (business) rule sets into multi-dimensional hierarchies along their context of application. Regarding inheritance, we differentiate additive and most-specific inheritance of rule sets. With the former inheritance mechanism all rules of a parent rule set apply in its child rule sets as well. The latter approach allows redefinition of inherited rules. Previous work on contextualized rule repositories for the semantic web [37] employed an additive inheritance semantics only. For these inheritance mechanisms we only gave the basic idea, i.e., we did not precisely define them. Integrating RMI into the generic CBR model utilizes RMI's precise definition of inheritance as well as RMI's rule modules to further improve business rule maintenance in CBRs. This integration is discussed in Chapter 7.

Since contexts can be considered modules, all of the above approaches support modules. Nevertheless, none of them explicitly defines clear input and output interfaces for modules as our RMI approach does. Regarding module inheritance, all of the above approaches support inheritance of knowledge, some [28, 39, 115] allow redefinition of knowledge whereas others do not [5, 37, 127, 183]. Compared to

RMI, none of the above approaches supports modification restrictions, i.e., none can control modifications to the knowledge in a module.

6.7 Conclusion

We investigated inheritance of rule modules to foster reuse of rules, simplify adaptation, and ease maintenance. We introduced the Rule Module Inheritance with Modification Restrictions (RMI) approach, comprising rule modules, precisely defined inheritance mechanisms for rule modules, and modification restrictions regulating modifications to inherited rules and interfaces with corresponding conformance checks as mechanisms for keeping child modules aligned with parent modules. The inheritance mechanisms presented in this thesis do not provide guarantees regarding global restrictions made on rule sets by the specific rule language employed. Our Vadalog-based prototype implementation allows easy integration into the Vadalog environment.

Chapter 7

The Generic ECBR Model: Extending the Generic CBR Model with Rule Module Inheritance

The generic CBR model does only touch upon inheritance of rule sets. To further improve business rule maintenance, a precise definition for inheritance of rule sets is needed. The RMI approach precisely defines inheritance of rule modules, i.e., of rules sets with input and output schemata. Consequently, an integration of these two approaches is beneficial. For this purpose, we identify challenges arising from such an integration and discuss how we address these challenges in the generic ECBR model. Furthermore, we touch upon realization considerations for ECBRs.

7.1 Introduction

Chapter 3 presented the generic CBR model and its instantiation levels; Chapter 5 the atomic and composed CBR model operations enabling business rule maintenance and business rule entering. Although a fully instantiated CBR model creates a multi-dimensional space where a context may inherit or specialize another one, we did not discuss such relationships and their semantics in detail. Chapter 6 on the other hand presents the RMI approach which focuses solely on inheritance of rule sets, in particular, rule modules, and imposing restrictions on allowed modifications to inherited rules and interface predicates. Consequently, a natural step is to integrate these two approaches into a single generic ECBR model. This generic ECBR model satisfies all claims and requirements identified in Chapters 1 and 3.

Integrating the generic CBR model and RMI, we face several challenges: (1) An interesting aspect of integration is that the generic CBR model is object-based, i.e., it accepts objects like the domain-specific SemNOTAM cases, while RMI is relation-based, i.e., it accepts facts for predicates as inputs. (2) A challenge is that domain-specific CBR models may model the schema for rule execution outputs, for instance, regard `littleImportant` in Figure 3.3. As a consequence, application-specific CBR models may contain outputs of rule execution, for an example, regard the value `true` for the derived attribute `littleImportant` of object `bc1` in Figure 3.6. The model for RMI presented in Figure 7.1 specifies the schemata of its inputs and outputs but may not contain the actual output for a rule module. In brief, CBR models may comprise rule execution outputs while the RMI model does not consider execution outputs. (3) Another issue arises from the fact that the RMI model has two levels whereas the CBR model has three. (4) An important challenge is to satisfy the same claims and requirements as well as to maintain the same advantages as the generic CBR model.

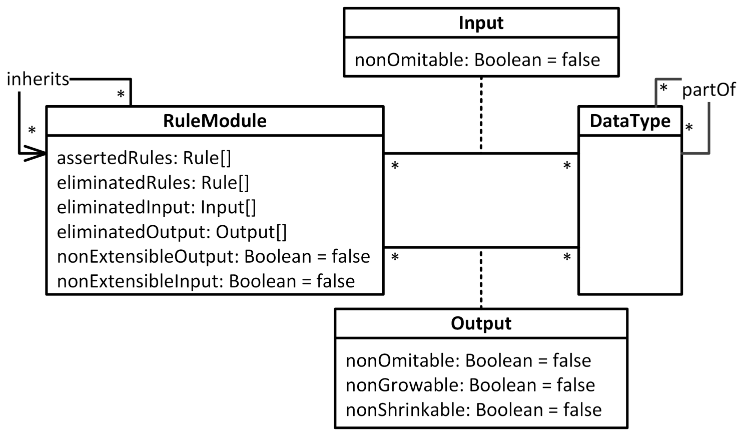

Input
nonOmitable: Boolean = false
inherits
*
*
RuleModule
assertedRules: Rule[]
eliminatedRules: Rule[]
eliminatedInput: Input[]
eliminatedOutput: Output[]
nonExtensibleOutput: Boolean = false
nonExtensibleInput: Boolean = false
*
*
*
*
DataType
*
partOf
*
Output
nonOmitable: Boolean = false
nonGrowable: Boolean = false
nonShrinkable: Boolean = false

FIGURE 7.1: UML class diagram for RMI independent of Datalog$^{\pm}$.

Having named some of the challenges, integration of the generic CBR model with RMI is not as trivial as it may seem at first glance.

The remaining chapter is organized as follows: Section 7.2 discusses how we address the challenges (1) to (4). Section 7.3 describes the generic ECBR model and its instantiation levels and discusses how we address challenge (4) in detail. Section 7.4 describes the creation of executable rule modules and their execution in ECBRs. Section 7.5 discusses considerations for realizing ECBR models. Section 7.6 concludes the chapter.

7.2 Addressing Challenges

In this section we delineate how we address the challenges discussed in the introduction: (1) object-based versus relation-based models, (2) rule execution outputs may be comprised in CBR models but not in RMI models, (3) different number of levels required for modeling, and (4) satisfying the same requirements and maintaining the same advantage as the generic CBR model.

In order to integrate the generic CBR model and RMI we need to generalize RMI to a level where it is independent of any specific input form. The basic concepts behind RMI can be easily mapped to a generic variant like the one presented in Figure 7.1: A `RuleModule` defines its behavior by a set of asserted rules and facts (`assertedRules`). Furthermore, a `RuleModule` may inherit from an arbitrary number of `RuleModules` and may eliminate rules and facts (`eliminatedRules`) as well as input predicates (`eliminatedInput`) and output predicates (`eliminatedOutput`). Each `RuleModule` specifies the `DataType` of its `inputs` and `outputs`. A `DataType` may be part of another `DataType`. For each `Input` and `Output`, as well as for the complete `RuleModule`, restrictions may be defined – by default no restrictions apply. The restrictions are modelled as attributes of the association classes `Input` and `Output` and class `RuleModule` respectively. By this generalization we partly address challenge (1) – object-based versus relation-based.

Addressing challenges (1), object-based versus relation-based, and (2), execution modeling, we decide to strictly separate the determination of relevant contexts and rule execution. As a result, rule execution effects must not be modeled in ECBR models – static model and dynamic rule execution aspects are to be clearly separated. As a consequence, context determination may stay object-based while the underlying

TABLE 7.1: The three levels of RMI models.

Level	Model
M2	Generic RMI model
M1	Domain-specific RMI model
M0	Application-specific rule sets, input schemata, output schemata, eliminations, and modification restrictions

data format for rule execution may differ depending on the concrete rule language employed.

In order to be able to integrate the RMI model in Figure 7.1 with the generic CBR model, they need to have the same number of levels. Therefore, we introduce a meta-level to the RMI model presented in Figure 7.1. A summary of the resulting three levels of RMI models is given in Table 7.1; the resulting three-level model is depicted in Figure 7.2: Meta-level M2 contains classes `RuleModuleClass` and `DataTypeClass` which define all relationships as well as attributes for asserted rules, inputs, and outputs as well as performed eliminations and allowed modification restrictions. The reflective relationships `inherits` and `partOf` have a potency of two, i.e., they are instantiated at level M0. Relationships `Input` and `Output` (association classes) need to be configured on level M1 to domain-specific relationships with domain-specific classes. Level M2 is instantiated to a domain-specific model at level M1 where rule modules and data types as well as relationships may be enriched with domain-specific attributes, for example, domain-specific rule modules may be described by tags or contain information about the last modification. Furthermore, domain-specific inputs and outputs need to be configured. The application-specific level M0 contains concrete rule modules with their asserted rules, input and output schemata, eliminations of rules and of interface predicates, as well as defined modification restrictions. With this multi-level model for RMI we address challenge (3) – different number of levels.

Example 7.1. In Figure 7.2 we depict the generic RMI model (upper part) as well a sample instantiation of the generic RMI model for SemNOTAM (lower part). At the domain-specific level M1, we define that instances of `AIMRuleModule` need to specify the date of their `lastModification`. Furthermore, we configure the input `NOTAM` to be of type `NOTAMType` and cardinality one and the input `SemCase` to be of type `SemNOTAMCaseType` and cardinality one as well. We configure output `NOTAMClasses` to be of type `NOTAMClassificationType` and arbitrary cardinality (since several classifications may be output per NOTAM). The `partOf` relationship is not needed for SemNOTAM.

Challenge (4), satisfying the same claims and requirements as the generic CBR model and maintaining the advantages of the generic CBR model, is addressed by employing the generic CBR model as basis and carefully integrating the three-level RMI model. For each modification on the generic CBR model we check whether the model still satisfies the claims and requirements identified in Chapter 1 and Chapter 3 and whether its advantages are maintained.

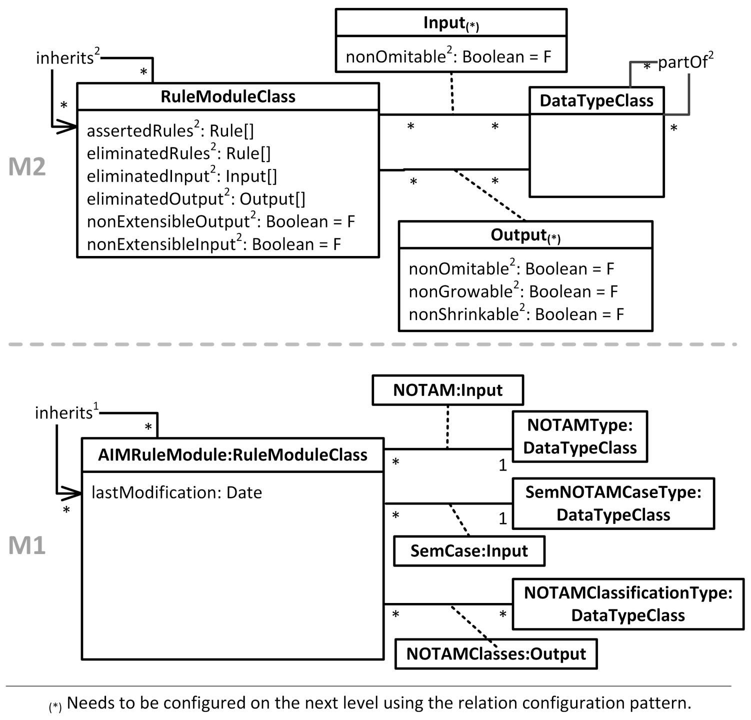

Input(*)
nonOmitable²: Boolean = F
inherits²
partOf²
RuleModuleClass
assertedRules²: Rule[]
eliminatedRules²: Rule[]
eliminatedInput²: Input[]
eliminatedOutput²: Output[]
nonExtensibleOutput²: Boolean = F
nonExtensibleInput²: Boolean = F
DataTypeClass
M2
Output(*)
nonOmitable²: Boolean = F
nonGrowable²: Boolean = F
nonShrinkable²: Boolean = F
NOTAM:Input
inherits¹
NOTAMType: DataTypeClass
AIMRuleModule:RuleModuleClass
lastModification: Date
SemNOTAMCaseType: DataTypeClass
M1
SemCase:Input
NOTAMClassificationType: DataTypeClass
NOTAMClasses:Output
(*) Needs to be configured on the next level using the relation configuration pattern.

FIGURE 7.2: Multi-level UML class diagram for RMI independent of Datalog$^{\pm}$ and its instantiation to the domain of SemNOTAM. Attributes are only shown on the level they are defined, intermediate attributes having a potency greater than zero are hidden [54].

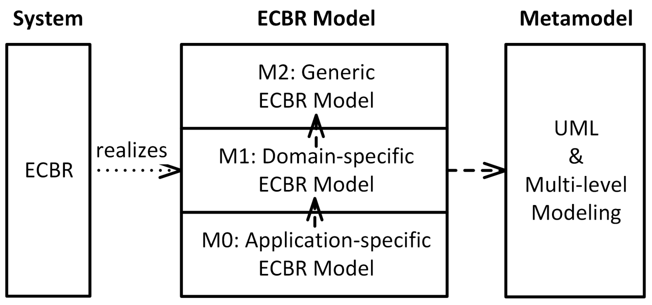
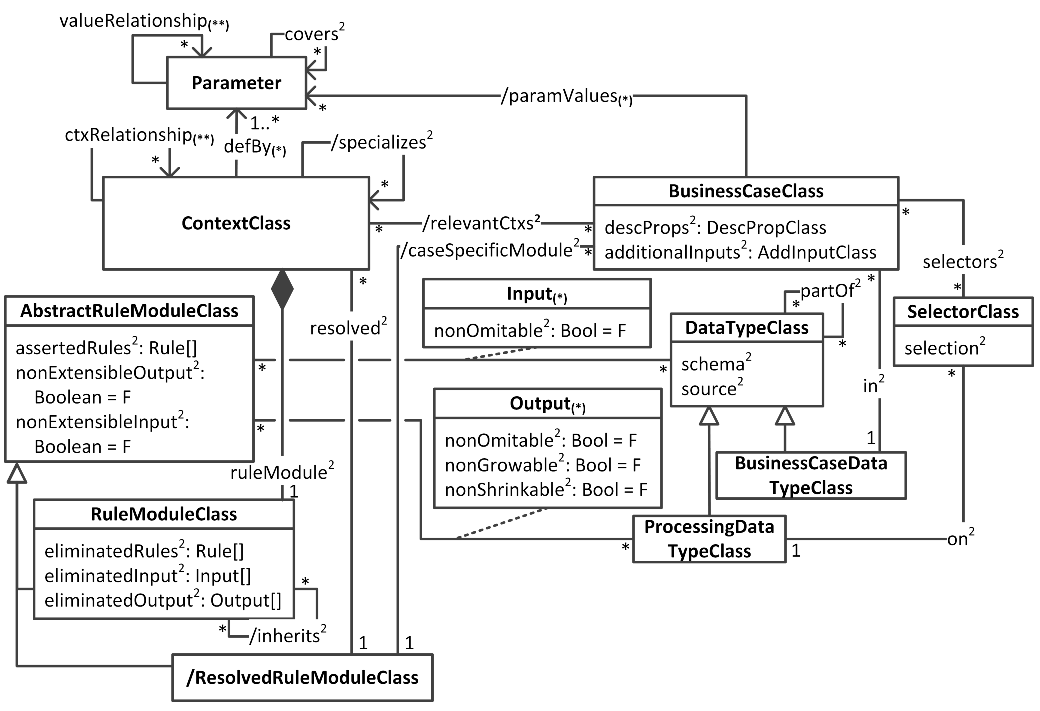

System
ECBR Model
Metamodel
ECBR
realizes
M2: Generic ECBR Model
M1: Domain-specific ECBR Model
M0: Application-specific ECBR Model
UML & Multi-level Modeling

FIGURE 7.3: The three levels of our integrated conceptual ECBR model and an overview of the orthogonal classification architecture for ECBR models drawing on Atkinson and Kühne [11]. The realization of an ECBR model in an ECBR enables the execution of rule modules on provided data.

7.3 Modeling Context and Rule Module Inheritance for Business Rule Organization

In this section we present the generic ECBR model and its instantiation levels, addressing the identified challenges as described above. We base the generic ECBR model mainly on the generic CBR model presented in Chapter 3. Thereby, and by careful integration, the generic ECBR model will satisfy the same claims and requirements as well as maintain the same advantages as the generic CBR model does.

To represent the generic ECBR model, we employ the same object-oriented and graphical formalism as we do in Chapter 3. ECBR models comprise the three levels depicted in Figure 7.3; the realization of an ECBR model in an ECBR enables the execution of rule modules (discussed in Section 7.4).



(*) Needs to be configured on the next level using the relation configuration pattern.
(**) Needs to be reflexively configured on the next level using the relation configuration pattern.

FIGURE 7.4: The generic ECBR model resulting from integrating the generic CBR model with the generic RMI model. Attributes and methods already presented in Chapter 3 have been omitted.

Level M2 defines a generic context model integrating the generic models for CBR and RMI; it is a meta-model for business rule organization along contexts employing rule modules. As such it describes contexts, parameters, business cases, rule modules, and data types. Models of level M1 instantiate the generic ECBR model from level M2 to domain-specific models, i.e., they refine contexts, parameters, business cases, rule modules, and data types to domain-specific ones and furthermore specify allowed inputs and outputs for domain-specific rule modules. Models of level M0 instantiate domain-specific ECBR models to application-specific models. Consequently, they contain concrete contexts, their associated rule modules with their input and output interfaces, concrete data types, as well as concrete business cases.

In the following we detail the different levels, employing SemNOTAM to depict example instantiations.

7.3.1 Generic Model (M2)

The generic ECBR model is mainly based on the generic CBR model. In order to integrate the generic RMI model we have to replace `RuleSetClass`, to add classes for data types, and to adapt `ContextClass` and `BusinessCaseClass`; the remaining class `Parameter` and its relationships are not modified. In the following we focus on necessary adaptions rather than discussing unaltered model parts. The integrated generic ECBR model is depicted in Figure 7.4.

We replace RuleSetClass in the generic CBR model by class RuleModuleClass from Figure 7.2. In the generic CBR model we construct for each business case a case-specific context. Moving from rule sets to rule modules, we construct case-specific rule modules for which all relationships have been resolved. Consequently, we need to distinguish rule modules RuleModuleClass, with inheritance relationships and eliminations, and resolved rule modules ResolvedRuleModuleClass. Common attributes of these two classes have been extracted to the common super class Abstract-RuleModuleClass. Each ContextClass comprises one RuleModuleClass, where the inheritance relationships are derived from the context hierarchy, and references one ResolvedRuleModuleClass (replacing attribute resolved in the generic CBR model). The resolved rule module for a context is derived from its RuleModuleClass and all ancestor rule modules as described in Chapter 6. The derived relationship caseSpecificCtx is replaced by a derived relationship caseSpecificModule which relates business cases with its resolved case-specific rule module. The generation of a case-specific context is not necessary anymore – we replace method detCaseSpecific-Ctx in the generic CBR model with method detCaseSpecificModule. For simplicity we assume that for each business case a most-specific context exists.

Regarding data, we assume that most data processed by rules in rule modules is stored in external data sources such as databases or files. For instance, all NOTAMs in a SemNOTAM system will most likely reside in an XML-database. Therefore, DataTypeClass contains information about the source of data of the data type, for instance, a relation in a database, and the schema of the data type, for example, a relational table with three columns. The mapping from sources to rule module input or from rule module output to sinks depends on the concrete data type schema as well as the language employed to define rule module behavior. Therefore, the association class modelling input (Input) defines a mapping method (not shown) mapping from the source schema to the internal schema employed in the rules. Similarly, association class Output defines a mapping method (not shown) mapping from the internal schema to the schema specified for the respective sink.

Regarding business cases we derive from the strict separation of context determination and rule execution that business case attributes used for context determination and attributes used as input for rule execution have to be separated. This is modeled by BusinessCaseClass having an attribute descProps containing information used for context determination and an attribute additionalInputs containing information which may be used as input for rule execution. Attribute additionalInputs does not need to be modeled explicitly in class BusinessCaseClass, it could also be modeled solely using the schema of the respective BusinessCaseDataTypeClass. For any relationship relatesTo in the generic CBR model, it has to be decided whether it is descProps or additionalInputs.

The separation of descProps and additionalInputs renders specializations of DataTypeClass necessary – we distinguish data types for business cases (BusinessCase-DataTypeClass) and data types relevant to rule execution (ProcessingDataTypeClass). The former may only be input data types for rule modules whereas the latter may be input data types as well as output data types of rule modules. To relate business cases and business case data types we introduce the relationship in defining that a business case instance is contained in the respective source in the specified schema (defined by instances of BusinessCaseDataTypeClass).

Usually, only a subset of the stored data is actually used as input to rule module execution, for example, in SemNOTAM the caseSpecificModule is executed on a subset of available NOTAMs, namely, a subset regarding a specific event scenario.

Therefore, we introduce `selectors` of class `SelectorClass` to `BusinessCaseClass`. A `SelectorClass` defines a selection upon a `ProcessingDataTypeClass`.

Challenge (4), satisfying the same claims and requirements as well as maintaining the same advantages as the generic CBR model, is addressed by the modifications to the generic CBR model described above. The only semantic change to the generic CBR model is that the construction of case-specific contexts has been replaced by resolved rule modules associated with contexts (`ResolvedRuleModuleClass`). Nevertheless, ECBR models determine, as do CBR models, for a given business case a rule set containing all relevant business rules with all relationships and conflicts resolved. All other changes extend the generic CBR model by elements necessary for rule modules and explicit modeling of data sources and sinks.

As a result, the same atomic and composed operations and CBR roles introduced in Chapter 5 for CBR models can be employed for ECBR models. The `Input` and `Output` are considered an integral part of rule modules, i.e., modifications of input or output interfaces require the same composed operation as modifying business rules. As a result, `Input` and `Output` of rule modules including the specification of the mapping from data types to rule modules are managed by rule developers. Data types, i.e., instances of `DataTypeClass` are maintained by domain experts as these know the domain-specific data formats as well as where to obtain the data. Domain experts also maintain the available instances of `SelectorClass` while users instantiate them for specific business cases. If a data type is changed by a domain expert, affected rule developers and domain experts have to be informed. Once all rule developers and domain experts reacted and acknowledged the change the operation has been successful. If selectors are modified, affected users must be informed.

Since the generic CBR model is only extended and its semantics preserved, the generic ECBR model satisfies the same claims and requirements as well as provides the same advantages as the generic CBR model. Moreover, context inheritance is now precisely defined by integrating the RMI approach.

7.3.2 Domain-specific Model (M1)

Any domain-specific ECBR model is an instantiation of the generic ECBR model, i.e., it restricts and refines the generic ECBR model to a domain-specific ECBR model. We focus on these parts of domain-specific ECBR models which differ from domain-specific CBR models, namely, rule modules, business cases, and data types.

Rule modules are instantiated to domain-specific rule modules. These domain-specific rule modules may specify additional attributes such as responsible developer or modification date. In particular, `AbstractRuleModuleClass`, `RuleModuleClass`, `ResolvedRuleModuleClass`, `BusinessCaseDataTypeClass`, and `ProcessingDataTypeClass` are instantiated to domain-specific classes. Instances of `AbstractRuleModuleClass` are abstract, i.e., cannot be instantiated. Class `BusinessCaseDataTypeClass` is instantiated to a singleton class, i.e., all application-specific business cases of the domain-specific model refer to the same application-specific business case data type. The relationships `Input` and `Output` are configured to domain-specific rule module and data type classes.

`BusinessCaseClass` is instantiated to domain-specific business cases. To this end, the respective class of their `descProps` and `additionalInputs` attribute is refined. The `SelectorClass` is instantiated to domain-specific selectors and their relationships `on` are refined to instances of `ProcessingDataTypeClass`, i.e., to domain-specific data types not provided by business cases. Furthermore, relationship `in` is instantiated

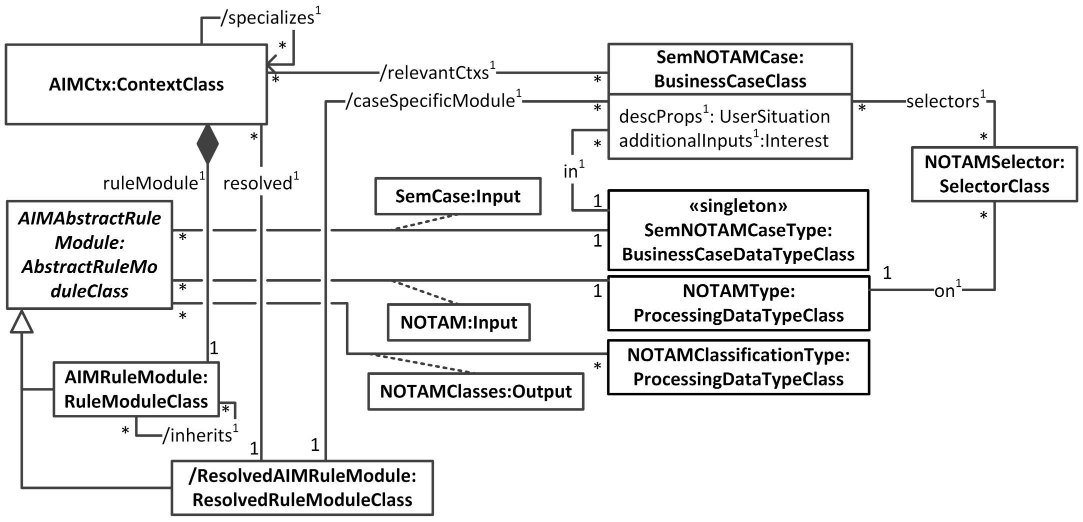

/specializes1
AIMCtx:ContextClass
/relevantCtxs1
/caseSpecificModule1
SemNOTAMCase: BusinessCaseClass
descProps1: UserSituation
additionalInputs1:Interest
selectors1
NOTAMSelector: SelectorClass
in1
ruleModule1
resolved1
SemCase:Input
«singleton» SemNOTAMCaseType: BusinessCaseDataTypeClass
AIMAbstractRule Module: AbstractRuleMo duleClass
NOTAMType: ProcessingDataTypeClass
on1
NOTAM:Input
NOTAMClassificationType: ProcessingDataTypeClass
NOTAMClasses:Output
AIMRuleModule: RuleModuleClass
/inherits1
/ResolvedAIMRuleModule: ResolvedRuleModuleClass

FIGURE 7.5: Excerpt of a domain-specific integrated ECBR model instantiated for SemNOTAM. Instantiation of class `Parameter` is not shown for readability. Furthermore, attributes with a potency greater than zero are hidden if not refined.

from a domain-specific `BusinessCaseClass` instance to a domain-specific instance of `BusinessCaseDataTypeClass` which is singleton.

Example 7.2. We instantiate the generic ECBR model for our use case SemNOTAM. Figure 7.5 depicts an excerpt of this instantiation. The parameters and the domain-specific `ContextClass` are, except for attribute `resolved`, identical to the ones presented in Figure 3.3. Since we do not need additional information to be specified for rule modules, instantiation of `AbstractRuleModuleClass`, `RuleModuleClass`, and `ResolvedRuleModuleClass` to `AIMAbstractRuleModule`, `AIMRuleModule`, and `ResolvedAIMRuleModule` respectively, is trivial.

We instantiate `BusinessCaseClass` to the SemNOTAM-specific business case `SemNOTAMCase`. For this purpose, we need to identify describing properties as well as additional inputs. In the case of SemNOTAM, a user situation containing the flight phase, the interest to be considered, as well as the type of event scenario to consider forms the domain-specific `descProps` class `UserSituation`. Additional inputs (`additionalInputs`) to rule execution are detailed information about the `Interest` to consider. The schema and source of application-specific instances of `SemNOTAMCase` are specified using the application-specific instance of `SemNOTAMCaseType`, thus relation `in` relates `SemNOTAMCase` and `SemNOTAMCaseType`.

Regarding data types, we identify NOTAM data types, NOTAM classification data types, and business case data types as relevant. Consequently, we instantiate `ProcessingDataTypeClass` to `NOTAMType` and `NOTAMClassificationType` and `BusinessCaseDataTypeClass` to `SemNOTAMCaseType`. Each rule module accepts one `NOTAMType` and one `SemNOTAMCaseType` as input. Furthermore, each rule module may output several `NOTAMClassificationTypes`. This is modeled by the relationship configuration shown in Figure 7.5.

In SemNOTAM, NOTAMs reside in a database. Since a NOTAM's event scenario influences the relevant rule modules, the NOTAMs input to a rule execution need to be of the same event scenario. Thus, we instantiate a NOTAM-specific selector `NOTAMSelector` for `SemNOTAMCase`. This selector applies to `NOTAMType`.

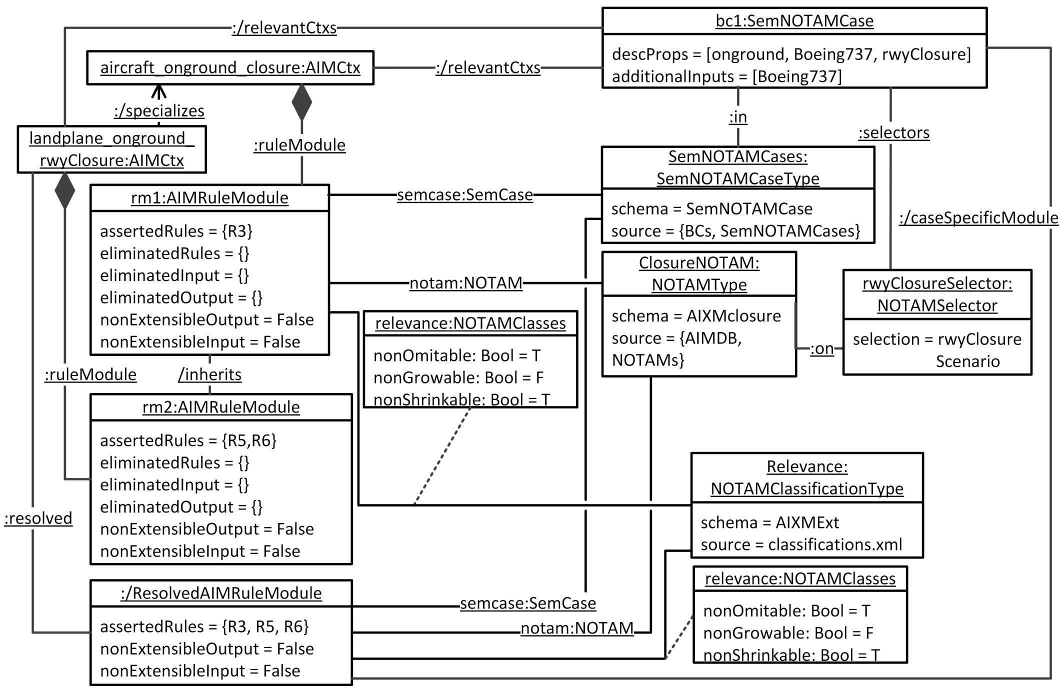


FIGURE 7.6: Excerpt of an application-specific integrated ECBR model instantiated for SemNOTAM. Data types for importance and instantiation of class `Parameter` are not shown for readability.

7.3.3 Application-specific Model (M0)

Models of level M0 instantiate domain-specific ECBR models to concrete contexts, rule modules, data types, and business cases. This requires business rule elicitation and organization, i.e., performing CASA-feature tagging of business rules. CASA-feature presentation is realized by `relevantCtxs`, as in domain-specific CBR models, and `caseSpecificModule`. CASA-feature execution may not be modeled compared to application-specific CBR models – it is realized in ECBRs. Regarding data types, the concrete data sources and input and output schemata need to be defined. Mappings from source schemata to rule engine or mappings from rule engine outputs to target schemata may have to be specified using the respective `mapping` methods, depending on the concrete implementation and rule language employed.

Example 7.3. The parameter values and parameters determined for our use case SemNOTAM are presented in Figure 3.4; the elicited rules and contexts are depicted in Figure 3.5. Parameter values and instances of `AIMContext` are, except for attribute `resolved`, instantiated as in the application-specific CBR model for SemNOTAM. For the excerpt depicted in Figure 7.6 we employ the same contexts and business case as in Figure 3.6.

Regarding data types, we identified three types as essential: `ClosureNOTAM`, `Relevance`, and the business case's type `SemNOTAMCases`. We omit importance data types for readability. The `ClosureNOTAM` data type specifies the parts of Aeronautical Information Exchange Model (AIXM) (c.f. Section 8.2) relevant to closure NOTAMs as schema. The NOTAMs are stored in a database named AIMDB with tablespace NOTAMs specified in the `source` attribute. The `Relevance` data type specifies an extended AIXM as schema (extended by relevance for NOTAMs) and specifies `classifications.xml` as output sink via the attribute `source`. The business case data type `SemNOTAMCases` specifies the schema (class `SemNOTAMCase`) as well as the source for application-specific business cases (instances of `SemNOTAMCase`). Depending on the implementation, mappings for all three types may be necessary.

We have two context instances: `aircraft_onground_closure` and `landplane_onground_rwyClosure` and their respective rule modules `rm1` and `rm2`. Rule module `rm1` specifies the input schemata `SemNOTAMCases` and `ClosureNOTAM`. For these inputs we do not modify the default restrictions and thus do not show the instances of association classes. The output schema is specified by `Relevance`. For the output we define that `Relevance` is the minimum output and that specializing rule modules need to produce at least the same output as this rule module does for the same input data. This is shown by the instantiated association class `relevance:NOTAMClasses`. Other restrictions are not defined. Rule module `rm2` inherits these specifications and does not eliminate any of them. Thus, the resolved rule module `:/ResolvedAIMRuleModule` references the same input and output schemata.

Our example instance `bc1` of `SemNOTAMCase` is shown in Figure 7.6. Semantically, it is the same business case as we employed in Figure 3.6; structurally, we now distinguish describing properties and additional inputs. Regarding describing properties, the instance of `UserSituation` associated with business case `bc1` comprises flight phase `onground`, interest `Boeing737`, and event scenario `rwyClosure`. From these describing properties the relevant contexts (`relevantCtxs`) `aircraft_onground_closure` and `landplane_onground_rwyClosure` as well as the most specific relevant context `landplane_onground_rwyClosure` are determined. Additional inputs of business case `bc1` comprise details on the `Boeing737`.

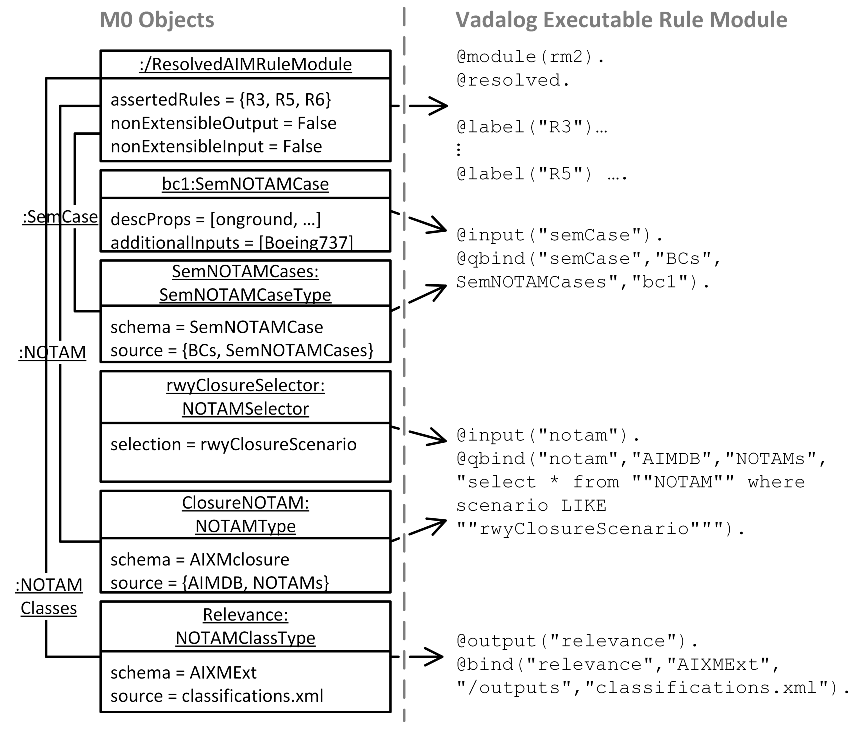
As mentioned in the example for the domain-specific ECBR model, the event scenario included in the describing properties has a selecting function on the NOTAMs input to rule modules at execution. Consequently, the concrete selector `rwyClosureSelector` for business case `bc1` selects NOTAMs of event scenario runway closure.

7.4 Rule Module Execution in an ECBR

Since ECBR models are conceptual models and rule module execution depends on the concrete rule language and rule execution engine employed, we do not consider execution to be part of ECBR models; it is part of ECBRs realizing ECBRs models.

The basic idea of rule module execution in ECBRs is that concrete instances of `ResolvedRuleModuleClass` are executed on concrete input data. To this end, executable forms of rule modules as well as concrete data items need to be created: Executable rule modules are created from application-specific instances of `ResolvedRuleModuleClass`. These application-specific rule modules need to be concrete in the rule module sense. Depending on the concrete rule language and rule execution engine employed, executable rule modules may look different, for instance, the sources and schema mapping may be defined within the rule module. Actual input data is taken from the sources specified in concrete instances of `DataTypeClass` employed as input data type. To this end, any given selectors are applied on the specified source and the selected data is mapped. Actual output data is created by mapping the data derived by executing a rule module on input data to the given source using the specified schema.

Example 7.4. Assuming we employ a Vadalog rule execution engine for our use case SemNOTAM, the executable rule module for the application-specific ECBR model given in Figure 7.6 may look as depicted on the right in Figure 7.7.


M0 Objects
Vadalog Executable Rule Module
:/ResolvedAIMRuleModule
assertedRules = {R3, R5, R6}
nonExtensibleOutput = False
nonExtensibleInput = False
bc1:SemNOTAMCase
descProps = [onground, …]
additionalInputs = [Boeing737]
SemNOTAMCases: SemNOTAMCaseType
schema = SemNOTAMCase
source = {BCs, SemNOTAMCases}
rwyClosureSelector: NOTAMSelector
selection = rwyClosureScenario
ClosureNOTAM: NOTAMType
schema = AIXMclosure
source = {AIMDB, NOTAMs}
Relevance: NOTAMClassType
schema = AIXMExt
source = classifications.xml
:SemCase
:NOTAM
:NOTAM Classes

```
@module(rm2).
@resolved.

@label("R3")…
⋮
@label("R5") ….

@input("semCase").
@qbind("semCase","BCs",
SemNOTAMCases","bc1").

@input("notam").
@qbind("notam","AIMDB","NOTAMs",
"select * from ""NOTAM"" where
scenario LIKE
""rwyClosureScenario""").

@output("relevance").
@bind("relevance","AIXMExt",
"/outputs","classifications.xml").
```

FIGURE 7.7: Exemplary creation of an executable rule module for the application-specific excerpt presented in Figure 7.6. On the left side we depict the objects from Figure 7.6; the arrows to the right side depict the creation steps for the executable rule module.

Employing Vadalog as rule language and rule execution engine, we utilize the annotations discussed in Section 6.5.1 for creating executable rule modules and concrete data items. In particular, `@bind` and `@qbind` allow to bind predicates to various sources, such as relations in databases or comma-separated values files, within executable rule module code and `@mapping` allows to integrate data mapping into executable rule module code.

The rule behavior specified in a concrete and resolved rule module `ResolvedAIMRuleModule` is simply copied to the executable rule module. Input and output predicates are added to the executable rule module using `@input` and `@output` annotations. Input and output data are associated with input and output predicates using `@bind` and `@qbind` annotations. The `@bind` annotation specifies to input all data contained in the source as facts of the specified predicate into rule module execution or to output all derived facts of the specified predicate to the defined data source. The `@qbind` annotation allows to specify that only data satisfying the given query are to be input as facts of the specified predicate into rule module execution. Thus, `@qbind` can be used to represent selectors within executable rule modules. `@mapping` can be used to specify the mapping from data sources to predicate positions. Any mappings exceeding the capabilities of `@mapping` need to be performed externally.

Considering our concrete example in Section 7.3.3, in particular, our derived concrete rule module `:/ResolvedAIMRuleModule`, a corresponding executable Vadalog program is created as follows (c.f. Figure 7.7): The schemata of the corresponding data sources are too complex for the `@mapping` annotation, thus we will need external mappings for `SemNOTAMCases`, `ClosureNOTAM`, and `Relevance`. Input predicates for `SemNOTAMCases` and `ClosureNOTAM` are specified using `@input`; output predicate `Relevance` is specified using `@output`. The mapped `bc1` is loaded from the set of concrete `SemNOTAMCases` using `@qbind`. For mapped NOTAMs residing in tablespace `NOTAMs` in a database named `AIMDB`, a `@qbind` is necessary as a selector on event scenario `rwyClosure` is defined. The target file `classification.xml` of output `Relevance` is specified using `@bind`.

7.5 Realization Considerations

To realize an ECBR model in an ECBR, two components are necessary. One component determines for a given business case the relevant contexts and the most specific relevant context(s). The other component accepts the relevant contexts as well as a most specific relevant context as input and resolves inheritance and any conflicts for the rule module associated with the most specific relevant context. If several most specific relevant contexts are reported, inheritance and conflicts must be resolved for each most specific relevant context separately and the resulting rule modules subsequently merged into one. Depending on the volatility of the associated rule modules, rule module resolution may be materialized (low volatility) or computed dynamically (high volatility).

Example 7.5. An ECBR employing dynamic computation can be constructed by chaining the Vadalog prototypes presented in Section 4.2.2 and Section 6.5. The former prototype is employed to determine relevant contexts (using `detRelevantCtxs`) as well as the most specific relevant context (`detMostSpecificCtx`) for a given business case. For each relevant context, we retrieve its associated instance of `RuleModuleClass` and add the corresponding `@inherits` statements to it. Meta-representations of the adapted instances of `RuleModuleClass` as well as the most specific rule module (determined from the most specific context) are passed to the inheritance resolution program. The result is a meta-representation of the resolved most specific rule module. Transforming this rule module meta-representation into an actual rule module, and performing the steps described in Section 7.4, the rule module is made executable.

ECBRs necessitate additional checks, such as structural or behavioral conformance. We discuss the integration of these checks into the atomic and composed modification operations introduced for CBRs in Chapter 5. Any atomic or composed modification operation modifying the behavior of rule modules needs to be followed by a structural and a behavioral conformance check of affected rule modules. Affected rule modules are determined using the context hierarchy in the respective application-specific ECBR model. Should any of the conformance checks fail, the modification operation fails as well. Once structural and behavioral conformance is ensured, rule module resolution for all affected rule modules is necessary if resolved modules are to be materialized. This ensures that materialized rule modules associated with contexts are up to date. Before executing any rule module, it needs to be checked whether the rule module is concrete - if the rule module is abstract it should not be executed. Abstract predicates, structural modifications, and behavioral modifications can also be reported during a modification operation to support rule developers in their work.

Example 7.6. Considering the application-specific excerpt presented in Figure 7.6, we employ the composed operation *modify rules/terms* to modify rule `R3` in such a way that only specific closure NOTAMs are relevant. For this purpose, we modify rule module `rm1` of context `aircraft_onground_closure`. Employing the context hierarchy we determine that rule module `rm2` associated with context `landplane_onground_rwy-Closure` is also affected by this composed modification operation. Assuming that the rule module associated with the root context does not define any modification restrictions, structural and behavioral conformance is given for rule modules `rm1` and `rm2`. Next the resolved rule modules for contexts `aircraft_onground_closure` and `landplane_onground_rwyClosure` need to be computed and associated with the corresponding contexts. If business case `bc1` is submitted, the resolved rule module associated with context `landplane_onground_rwyClosure` is returned. Since this rule module is concrete, i.e., does not contain abstract predicates, it can be executed.

7.6 Conclusion

In this chapter we presented the generic ECBR model integrating the generic RMI approach into the generic CBR model. This integration is beneficial as the generic CBR model does not precisely define inheritance of rule sets. Nevertheless, a precisely defined inheritance mechanism for business rules is important to further ease business rule maintenance (claim 1 in Section 1.5).

In order to perform the integration we had to address four challenges: object-based (CBR) models versus relation-based (RMI) models, rule execution outputs may be modeled in CBR models but not RMI models, different number of levels required for modeling, and maintaining claim and requirement satisfaction as well as advantages of the generic CBR model. The integrated ECBR model maintains, by careful integration, all the claims and requirements satisfied by the generic CBR model.

In addition to the generic ECBR model and its instantiations, we touched upon realization considerations for ECBRs. Furthermore, we discussed how the presented Vadalog prototypes, introduced in Section 4.2.2 and 6.5, can be employed to realize an ECBR.

Chapter 8

Use Cases of ECBRs

To demonstrate the usefulness of the generic ECBR model we instantiate it for two use cases, namely, for SemNOTAM and for financial services in a large US tech-company. For this purpose, we detail the use cases, describe the methods employed to determine parameters and parameter values for them, instantiate the generic ECBR model for them, and discuss the instantiated ECBR models.

8.1 Introduction

In Chapter 7 we presented the generic ECBR model. In this chapter we detail two use cases: SemNOTAM and financial services of a large US Tech-Company (USTC). A prerequisite required for any ECBR instantiation is that the use case has contextualized knowledge. In the case of SemNOTAM, contextualized knowledge are business rules applying in specific classes of situations while context knowledge is the class of situations in which these apply. In the case of USTC, contextualized knowledge are the business rules defining specific services; context knowledge specifies the service(s) and its/their configuration for which the contextualized knowledge applies.

The remaining chapter is organized as follows: In Section 8.2 we detail the requirements regarding SemNOTAM, the architecture chosen for the SemNOTAM prototype, the method employed to determine a suitable ECBR model, and finally we instantiate and discuss a suitable ECBR model. In Section 8.3 we detail USTC, describe the method utilized to determine a suitable ECBR model, and instantiate and discuss a suitable ECBR model for USTC. Section 8.4 concludes the chapter.

8.2 SemNOTAM[1]

In Section 1.4 we argued for the necessity of an effective intelligent filter and query system for NOTAMs and how SemNOTAM addresses this need in general. In particular, SemNOTAM utilizes, unlike currently available systems, possibilities introduced by Digital NOTAMs to provide intelligent filtering, classification, and ranking exceeding the capabilities of current systems. Currently, NOTAMs are mostly provided in a text-based and loosely structured form. This requires significant cognitive effort to comprehend a NOTAM's meaning and significance, and furthermore to filter and classify NOTAMs for a given situation [190]. With the increasing number of NOTAMs [63, 65], such a manual process is not sustainable any more, i.e., automatic processing is vital. Digital NOTAMs enable automatic and sophisticated processes by being encoded in a machine-readable (XML-based) standard for exchanging aeronautical information, namely, AIXM [62]. Utilizing Digital NOTAMs, SemNOTAM presents

[1]This chapter is based on the publications [34, 35, 39–41, 189, 190].

fewer irrelevant NOTAMs than current systems while still maintaining 100 % recall; hence the risk of information overload is reduced. Consequently, the flight operations personnel's stress level is lowered and contributions to air safety are made. Furthermore, SemNOTAM allows for more effective airspace utilization and improvements in trajectory management.

Digital NOTAMs (henceforth simply called NOTAMs) are structurally and semantically heterogeneous as they regard a wide range of different types of aeronautical objects, each described by different properties. Common to all NOTAMs is the specification of their valid time and valid space, expressed using the temporality and spatiality model of AIXM. Which properties and which values to use for a certain purpose, like an aerodrome closure, is defined in event scenarios. For instance, event scenario `aerodrome closure` precisely defines the properties of NOTAMs concerning aerodrome closures and the allowed values for these properties. Different aeronautical authorities describe different event scenarios, for example, EUROCONTROL and FAA identify 18 [66] whereas FAA alone identifies over 80 preliminary event scenarios [68]. Commonalities of event scenarios, for instance, common properties with the same value range, may be extracted to parent event scenarios, for example, common properties of event scenarios `aerodrome closure` and `runway closure` may be extracted to an event scenario `closure`.

This section is structured as follows: Section 8.2.1 delineates the SemNOTAM use case and its requirements in detail. Section 8.2.2 discusses the architecture we employed for the SemNOTAM prototype. Section 8.2.3 describes the research design, data collection method, and method of analysis we employed to instantiate an ECBR model for SemNOTAM. Section 8.2.4 builds on Section 8.2.3 to derive an ECBR model for SemNOTAM. Finally, Section 8.2.5 discusses the suitability of applying ECBR to SemNOTAM.

8.2.1 Problem Description

Automatic processing of NOTAMs requires that the necessary business rules are defined in an unambiguous and machine-processable way. At the same time, defined business rules should be intuitively understandable by business/domain people working with them. Besides this challenge, we already mentioned a few challenges in Section 1.4, such as situation-dependent application of business rules, the legal requirement of 100 % recall, the number of business rules, or the continuous evolution of the domain.

The problem complexity is aggravated by the different scenarios which have to be supported. These are delineated by Porosnicu, Hughes, and Standar [158]: (a) Pilot Briefing supports pilots by providing Flight Planning Briefing, Departure Briefing, and Debriefing. Flight Planning Briefing is used to decide whether an intended flight is feasible. Departure Briefing determines the current NOTAMs relevant to a specific flight. Finally, Debriefing is used to collect feedback from pilots on any possible discrepancies between provided information and observed reality. (b) Dispatcher Briefing supports dispatchers in Flight Preparation, i.e., preparation of departure briefing packages for pilots. Furthermore, Flight Update needs to be provided which determines any changes in NOTAMs. As for Pilot Briefing a Debriefing is available. (c) On-board Briefing provides Pilot Briefing capabilities before departure and during flight. To this end, data for Pilot Briefing along the planned route as well as for possible re-routings need to be available. Furthermore, any relevant updates need to be provided to the on-board briefing device. As for the briefing scenarios described

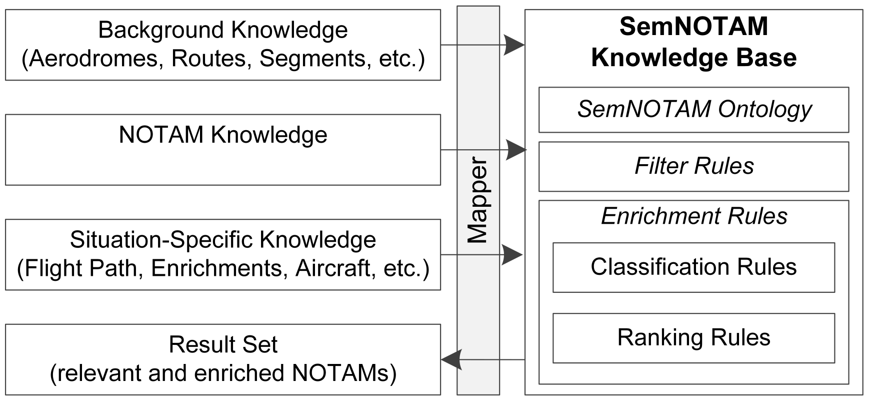


FIGURE 8.1: The SemNOTAM knowledge base, its inputs, and its output.

before, a function for reporting discrepancies is part of On-board Briefing. (d) Controller Briefing supports air traffic controllers by providing functions to determine relevant NOTAMs in their working area. Additionally, Controller Briefing allows reporting discrepancies between observations and reality.

From these scenarios, experiences in the SemNOTAM project, and the NOTAM Search and Filter Options report [169], requirements for intelligent filtering have been derived: (1) accepting flight plans including vertical as well as horizontal corridors regarding the flight path and optionally additional aerodromes along the flight path, (2) prioritizing NOTAMs according to importance, (3) grouping NOTAMs according to logical criteria, (4) determining changes to previous query results (so called delta queries), (5) customizing without redevelopment, redesign, or redeployment, and (6) migrating system profiles. Other requirements are filtering regarding (7) runways, all NOTAMs regarding runways, and runways by characteristics, (8) geographic regions, (9) flight information regions, (10) procedures, (11) NOTAMs effective during a given time span, and (12) keywords. Furthermore, (13) personalization without redevelopment, redesign or redeployment, (14) user profiles, (15) accountability of results, and (16) NOTAM ordering along temporal and spatial proximity have also been identified as requirements.

8.2.2 Prototype Architecture

To satisfy the above scenarios and requirements, we decide for a knowledge-based system facilitating maintenance, extensibility, and adaptability through the declarative representation of data, vocabulary, and rules. An overview of the SemNOTAM knowledge base is given in Figure 8.1.

SemNOTAM requires background knowledge, such as information about aerodromes, routes, or segments, knowledge about currently active NOTAMs, and situation-specific knowledge, such as flight path, role, aircraft type, or desired classifications, in order to filter, classify, and rank NOTAMs. Therefore, the required knowledge, usually available in standardized formats such as AIXM, needs to be mapped into a form processable by the SemNOTAM knowledge base. Similarly, the result set must be mapped to a form machine-readable by systems other than SemNOTAM. Thus, we introduce a mapper component enabling these mappings.

The actual knowledge about filtering, classification, and ranking is stored in the SemNOTAM knowledge base in the form of business rules (c.f. Figure 8.1). These business rules rely on the vocabulary defined in the SemNOTAM ontology.

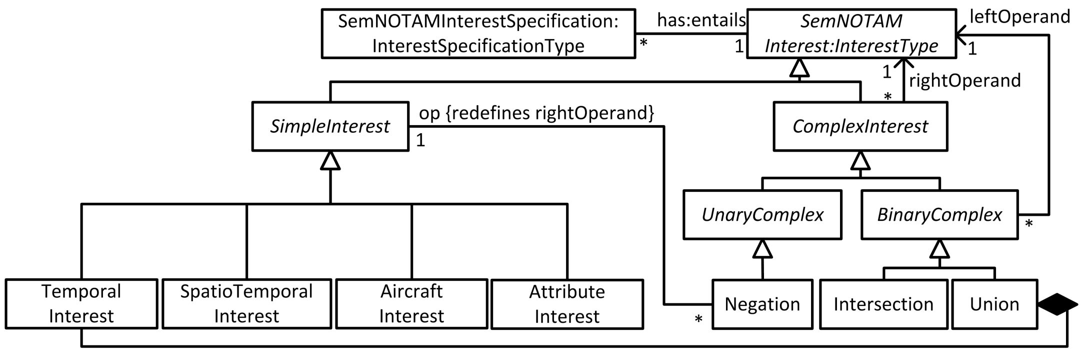


FIGURE 8.2: The SemNOTAM interest specification used for querying NOTAMs.

In order to satisfy the requirements regarding customizing and personalization, we employ a two-model knowledge base architecture. The generic model is application-independent and contains business rules and ontology vocabulary applicable in general. This generic model can be customized to an application-specific model by adding application-specific knowledge to the SemNOTAM knowledge base.

Of particular interest, since there is no standard available yet, is the specification of situation-specific knowledge. In SemNOTAM, situation-specific knowledge is provided in the form of interest specifications which allow users to define their situation. The structure of interest specifications is depicted in Figure 8.2. A user specifies for defined aspects, so called simple interests, in which qualities he/she is interested or in which situation he/she currently finds him/herself. Four simple interests are provided, namely, spatial interest, temporal interest, attribute interest (for instance, attributes of NOTAMs or AIXM features), and aircraft interest. Employing these, a user can specify, for instance, that he/she is interested in NOTAMs from Washington Dulles using a spatial interest. Each of these simple interests is evaluated separately.

To evaluate a simple interest, the corresponding SemNOTAM filter and enrichment rules need to be applied on the NOTAM and background knowledge. This application produces a result set for the simple interest. To enable more complex situation descriptions, simple interests can be negated, and furthermore, simple interests can be combined via complex interests using the set operators union and intersection on the result sets. Besides the simple and complex interests, the user states the enrichments, i.e., ranking and classifications he/she wants to be applied.

We implemented a SemNOTAM prototype using ObjectLogic [182]. To test customization to an application-specific SemNOTAM system, we determined the most prevalent NOTAM event scenarios. For each event scenario we elicited filter and enrichment rules as well as their corresponding vocabulary in workshops with pilots flying passenger aircrafts. The elicited business rules and business terms represent only a fraction of those relevant to a fully-fledged SemNOTAM. Nevertheless, about 300 business rules and 400 business terms have been determined from the viewpoint of pilots and encoded in ObjectLogic. The business rules are organized in ObjectLogic files according to the event scenario they consider. To execute SemNOTAM for a specific interest specification, all files need to be loaded.

8.2.3 Method for Determining an ECBR Model for SemNOTAM

In order to instantiate a suitable ECBR model for SemNOTAM, we conducted a small qualitative study. In the following, we discuss and derive our research design, data collection method, and method of analysis.

Due to the complexity of the aeronautical domain [39], formulating a comprehensive yet compact ECBR model is challenging. Therefore, we choose a *case-by-case study* design allowing an in-depth analysis of the context parameters and their parameter values without loosing sight of the overall picture. The main research question is the instantiation of an adequate ECBR model. Therefore, we want to analyse standard cases, i.e., flight operations personnel and their main situations. Regarding the data collection method, we decided for *problem-centred interviews with experts*. A group discussion would have been an equally suitable choice of method but knowing about the personal differences between the experts to be interviewed, the problem-centred interview was favoured. Furthermore, the open and semi-structured interview enables the interviewer to identify previously unknown aspects. Moreover, the interviewer may examine aspects previously not planned in the interview guide.

The *interview guide* consists, besides the prelude like consent, of several questions to estimate the background knowledge and experience as well as the role and viewpoint of the pilots, several questions to demonstrate the concrete objective, and questions to determine further context parameters and parameter values. Once the interview guide had been developed, it was discussed and tested with SemNOTAM project members. The results were incorporated into the final interview guide.

The *method of analysis* applied is grounded theory as described in Mayring [126]. We decided for grounded theory as its integrated data collection and analysis approach is most suitable. It allows to verify parameters and parameter values unravelled in interviews or documents in subsequent interviews and document analysis. The final context model has been constructed by comparing the parameters, their parameter values, and their relationships uncovered in the different interviews, workshop documents, and aeronautical authorities' documents. To this end, we also applied qualitative content analysis.

Using the interview guide described above, the interviews were conducted in German in January and February 2016 with three pilots. These three pilots were all previously known to the interviewer from preceding workshops. All three pilots interviewed were male and were scheduled flight pilots flying mid-range flights; two of them distances from one to almost six hours using an Airbus A320 and one distances up to three hours using a Dash 8-400. Two of the pilots had about 17 years of experience in scheduled flights with additional three to four years of experience as hobby pilot; the other pilot had two and a half years of experience in scheduled flights. An obvious limitation is the diversity of the sample. The interviewees were all male short- and mid-range pilots flying for the same airline. Consequently, the context model derived here is valid only from the viewpoint of a pilot from this airline. We refrained from interviewing other flight operations personnel or integrating a feedback loop into the research design due to limited resources. Furthermore, it must be noted that the pilots were paid for their contributions to SemNOTAM.

8.2.4 ECBR Model for SemNOTAM

Applying grounded theory and qualitative content analysis, we derived the contextualized knowledge, parameters, and parameter values displayed in Figure 8.3. Regarding contextualized knowledge, i.e., business rules, the interviews revealed only few differences in filtering rules between different contexts; nevertheless, enrichment rules may differ quite much. In the following, we delineate the identified

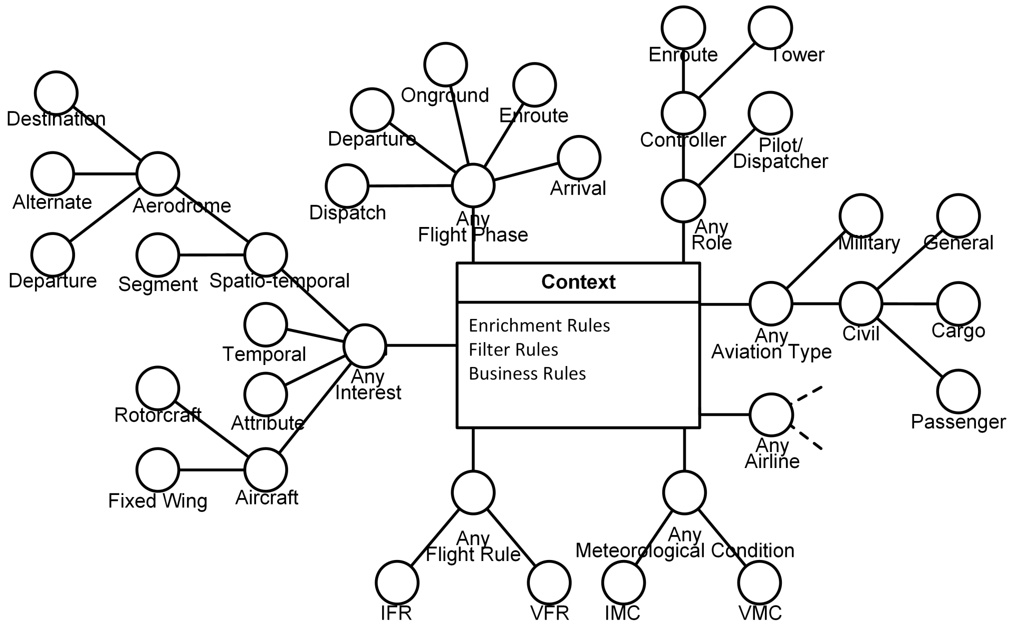


FIGURE 8.3: The context model for SemNOTAM resulting from applying grounded theory and qualitative content analysis to the available resources. Each branch represents a parameter; circles represent parameter values. Business rules encompass terms, i.e., they can be used to express the SemNOTAM ontology.

parameters and parameter values. Example instantiations of the generic ECBR model for SemNOTAM are depicted in Chapter 7.

The first parameter to be discussed is *Airline*. Two of three interviewees stated that different airlines define different business rules. One of them moderated the statement by adding that presumably about 80 % of the business rules will be the same. Workshop discussions showed a similar picture; they revealed, for example, that different airlines also diverge in the safety buffers applied to filtering. The dotted lines in Figure 8.3 indicate that the parameter values specializing `Any Airline` depend on the specific airlines to be supported.

The parameter *Aviation Type* was mentioned in all three interviews but with a flat hierarchy, i.e., no hierarchical relationships between the parameter values were mentioned. The final hierarchical arrangement depicted in Figure 8.3 was derived from workshops and other sources. A first distinction regarding parameter Aviation Type is into `Military` and `Civil`. Besides different business rules applying for these aviation types, military and civil NOTAMs [69] are distinguished. Since the interviewed pilots work in the civil domain and we did not have access to military documentations, further detailing the aviation type `Military` was not possible. Within the civil domain a distinction into `General Aviation` (such as private flights, or business aviation), `Cargo`, and `Passenger` is made. This distinction mostly derives from the different areas used at aerodromes. Consequently, different business rules apply for specific civil aviation types, for example, a pilot conducting a passenger flight needs information about gates and passenger boarding bridges whereas a pilot conducting a cargo or business flight is not interested in them.

The relevance of parameter *Role* regarding filter and enrichment rules was already determined in previously conducted workshops. Thus, the existence was known and the interviews were used to verify and clarify the parameter values and their relationships. Three basic roles were mentioned in all interviews, namely, controller, dispatcher, and pilot. `Controllers` are responsible for the safe and quick flow of air traffic, whether it is at an airport or in air. All three interviewees concurred that a further separation of controller into `Tower Controller` and `Enroute Controller` exists, one spoke of a even more fine-grained distinction into tower, ground, approach, flight information region, and enroute controller. This fine-grained distinction is not modelled as the differences in business rules are reportedly rather small. Nevertheless, if it should become necessary in the future, the proposed model can be easily extended in this way. Dispatchers support pilots by preparing a briefing package for them containing all relevant NOTAMs for their specific flight. Since dispatchers and pilots have similar or identical interests, the roles pilot and dispatcher have been combined into a single parameter value `Pilot/Dispatcher`. One interviewee mentioned that the ranking of a few NOTAMs may vary between pilots and dispatchers. Another interviewee argued that any differences between pilots and dispatchers regarding the relevance and ranking of NOTAMs are not a question of their role, it rather is a question of the flight phase considered (see below). Since the interviews and workshops confirmed the existence of a flight phase `Dispatch` the latter argument was adopted.

The previous paragraph already touched on parameter `Flight Phase`. Two of three interviewees stated that for different flight phases different business rules apply, especially regarding ranking. The actual phases defined here were found in aeronautical authorities' documents, discussed in one of the interviews, and reviewed in workshops. Any flight is separated into several flight phases. Depending on the interviewee, `Dispatch` may be considered a flight phase that occurs before the actual flight, as it is done in Figure 8.3. The subsequent flight phase `Onground` covers everything from aircraft checks to taxiing. Once the aircraft is rolling on the runway, flight phase `Departure` starts and ends once the aircraft has left the assigned Standard Instrument Departure. Flight phase `Enroute` comprises everything from the end of the Departure phase until the start of the Arrival phase. Flight phase `Arrival` covers the Standard Terminal Approach Procedure and ends once the aircraft leaves the runway. From then on, until disembarkation is considered part of flight phase `Onground`.

The parameter *Interest* is mainly derived from the requirements and the interest specification described in Section 8.2.2. Each simple interest forms a separate parameter value as different business rules for their evaluation apply. The parameter values temporal and attribute are not further distinguished. The Spatio-temporal value is further discriminated into `Aerodrome` and `Segment`. Aerodrome is further specialized by `Destination`, `Alternate`, and `Departure` aerodrome. Aircraft is further specialized into `Rotorcraft` and `Fixed Wing` which use different types of aerodromes, for instance, rotorcraft use heliports whereas fixed wing aircraft use airports. Consequently different filter and enrichment rules hold for each aircraft type.

The parameter *Flight Rule* was discussed in all interviews as a criterion of distinction. Two specialisations, namely, `Instrumental Flight Rules (IFR)` and `Visual Flight Rules (VFR)` exist for which different business rules hold. IFR comprises a set of rules for flying in instrumental meteorological conditions; VFR a set of rules for flying in visual meteorological conditions.

The parameter *Meteorological Condition* occurred in the workshops only; in the interviews visibility, a consequence of meteorological conditions, was considered an important factor but not a parameter. We consider two parameter values: weather conditions requiring a pilot to fly by IFR, `Instrumental Meteorological Conditions (IMC)`, and by VFR, `Visual Meteorological Conditions (VMC)`. Parameter Meteorological Condition in addition to parameter Flight Rule is necessary as, for example, the equipment for IFR could be offline with current weather being IMC.

The graphic presented in Figure 8.3 summarizes the parameters and parameter values derived from our qualitative study. Due to the limitations of this study, the generalizability of the model is limited – On the one hand, an application-specific instance of SemNOTAM may not make use of all parameters or parameter values depending on the concrete application scenario, for instance, a SemNOTAM system may only be used by controllers rendering the parameter value Pilot/Dispatcher superfluous. On the other hand, an application-specific ECBR model may require additional parameters and parameter values.

During our qualitative study we identified potential candidates for parameters and parameter values which we did not include in the context model for SemNOTAM. Most of them regarded orthogonal classifications of aircraft like wake turbulence class (which influences the start time intervals between two aircrafts), weight, design group, instrument landing system category, or wingspan. These classifications were not used in the model as the difference in business rules was determined to be quite small in the interviews. Other potential parameters discussed were aerodrome equipment and available navigational equipment onground. They were omitted as they are not decisive for the business rules to be applied but rather information accessed in business rules. Suggested by one pilot was to employ areas in the world, like continents, as parameter. Since only mentioned by one pilot and not discussed in any workshop, this parameter was omitted.

Domain- and application-specific instantiations of the generic ECBR model are shown in Chapter 7.

8.2.5 Discussion

In this section we discuss the suitability of instantiating the generic ECBR model for SemNOTAM. General advantages of the generic ECBR model are discussed in Chapter 3 to 7 – here we discuss functionalities and advantages specific to SemNOTAM.

A vital part of the generic ECBR model regarding SemNOTAM are business cases and the automatic determination of relevant contexts. For SemNOTAM, interest specifications represent business cases. For each interest specification the relevant contexts are determined and returned. This feature is vital to support the different scenarios described in Section 8.2.1. In particular, the scenarios are supported by including the parameter `Role` in the ECBR model for SemNOTAM. The different parameter values (specific roles) enable the definition of different business rules for each specific scenario. Moreover, the parameter Role enables personalization and user profiles by detailing the parameter values.

One of the main advantages of employing the generic ECBR model is the simplified maintenance by explicit definition of the classes of situations in which business rules hold. These classes of situations may form a multi-dimensional hierarchy. Considering the SemNOTAM prototype described in Section 8.2.2, its 300 business rules are organized into 22 files – one per NOTAM event scenario; the 400 business terms are split into two files. To organize these business rules in an ECBR, we employ three parameters from the ones identified in Section 8.2.4, namely, flight phase, flight rule, and meteorological condition. In addition, we employ parameter `Time of Day` which we identified as useful for organizing the business rules in the SemNOTAM prototype. Each of the parameters defines three parameter values resulting in $3 \times 3 \times 3 \times 3 = 81$ potential contexts for which we can define specific business rules. Using these parameters, we organize the prototype's rules and vocabulary into 17 of these 81 contexts.

A more fine-grained organization would be enabled by introducing a parameter for NOTAM event scenarios as in the running example (c.f. Figure 3.3).

The introduced composed modification operations, their associated CBR roles, and their employment for BRM in an ECBR further simplify maintenance. In particular, they realize the required debriefing functionality [158] which collects feedback and improves the system. Furthermore, they support customizing and personalization without downtime. A description how we employ the composed modification operations, the proposed roles, and the proposed BRM process for CBR in SemNOTAM is given in Chapter 5.

Employing rule modules for SemNOTAM contexts provides clear interfaces for each context, i.e., the aeronautical data to be provided and the filter and enrichment classifications output are clearly defined. Furthermore, precisely defined inheritance of rule modules reduces redundancy of rules, further simplifying maintenance. For instance, employing NOTAM event scenarios as parameter values, business rules common to several event scenarios, for example, to runway and aerodrome closures, may be extracted to a common parent context, for example, a context containing business rules regarding closures in general.

The modification restrictions introduced in Chapter 6 enable the definition of default behavior as well as fixed behavior for rule modules (and thus contexts). Default behavior may be changed in child contexts, restrictions constrain the way in which the behavior may be changed, for example, a defined safety buffer may only be enlarged. Fixed behavior must not be changed in child rule modules, for example, NOTAMs of event scenario aerodrome closure are always highly important. This feature can be used to ensure legal requirements in the aeronautical domain, such as relevant NOTAMs must not be omitted.

Further important features of the ECBR model for SemNOTAM are conflict resolution and the generation of case-specific contexts. Conflict resolution in SemNOTAM is necessary when mutually exclusive filter or enrichment classifications are derived, for example, a NOTAM is both relevant and irrelevant. Such cases may arise due to incoherent business rule definitions or multi-inheritance. The generation of case-specific contexts provides an explanation for derived classifications. Furthermore, compared to the original prototype without contextualization, only a fraction of the available business rules needs to be evaluated when organizing the business rules in an ECBR.

8.3 Large US Tech-Company

The second use case we consider for this thesis is one from a large US tech-company providing services, such as similarity ranking of companies, for people in the financial industry. This use case has been analyzed and developed during an Erasmus+ stay at the University of Oxford. We denote the use case as USTC.

In order to instantiate a suitable ECBR model for USTC we performed a qualitative study quite similar to the one performed for SemNOTAM. Regarding study design we decided for *case-by-case study* in order to analyse USTC in detail. Regarding the data collection method we employed document analysis and group discussions. As method of analysis we employed, as for SemNOTAM, grounded theory due to its integrated data collection and analysis approach. Both, document analysis as well as group discussion with groups of various sizes, were performed in December 2018 during an Erasmus+ stay at the University of Oxford. Group participants were members of the University of Oxford implementing services for the US tech-company. Consequently, a limitation is that the participants are not employees of the US tech-company but contractors and therefore may not have the same overview and knowledge of details.

This section is structured as follows: Section 8.3.1 delineates USTC in detail. Section 8.3.2 describes the ECBR model we instantiate for USTC and depicts excerpts of the domain-specific as well as the application-specific ECBR model. Finally, Section 8.3.3 discusses the suitability of employing RMI and the ECBR model for the use case USTC.

8.3.1 Problem Description

The large US tech-company employs data available to them to derive business signals, i.e., measures describing a business. Examples for business signals are an organization's size or its current investment stage. More complex business signals exist but are under non-disclosure. Any set of business signals may be combined to higher-level business signals.

The US tech-company offers services in the financial industry. These services employ the above described business signals in order to fulfil their tasks. An exemplary service implemented is the similarity ranking of organizations for various investor types: A user provides his/her investor type (venture capitalist, institutional investor, angel investor, or private investor) and an organization he/she already invested in or is interested in. The service returns a list of organizations similar to the given organization and reports their similarity score. For this purpose, a set of business signals, including organization size and investment stage, are employed. Depending on the investor type different business signals are employed and the calculation of the final similarity score varies. In future, user-specific calculations should be supported.

Alongside the service offering similarity ranking of organizations, further services like reporting, valuation, credit rating, or development predication are to be implemented for organizations. Each of these services should be offered at different levels of detail from low, for getting an overview, to high, for detailed analysis. Furthermore, the different services are not only to be provided at company granularity but for geographical region, industry, or conglomerate granularity as well.

Executing the correct service at an appropriate level of detail and for the expected granularity requires the user to provide further information in addition to the investor types described above. To this end, the currently supported investor types may have to be detailed or extended. Furthermore, additional information about the user, such as age, risk aversion, knowledge of markets and operations, portfolio goals (diversification or consolidation), capital goals (protection or growth), or his/her current portfolio, may be necessary in order to perform the services satisfactorily.

The various services will be bundled into a service portal. Users may sign up for this portal. They need to provide some personal information such as age or risk aversion. Furthermore, users may detail their profile or portfolio if they wish to use specific services. Once signed up and logged in users may consume services.

At the time the qualitative study was performed, the service offering similarity ranking of organizations as well as the required business signals had been implemented. Further services and business signals will follow. To implement the service and the required business signals, Vadalog has been used. The Vadalog code is organized in Vadalog modules. A Vadalog module is identified by a module name and may include other Vadalog modules. This mechanism is heavily used for the

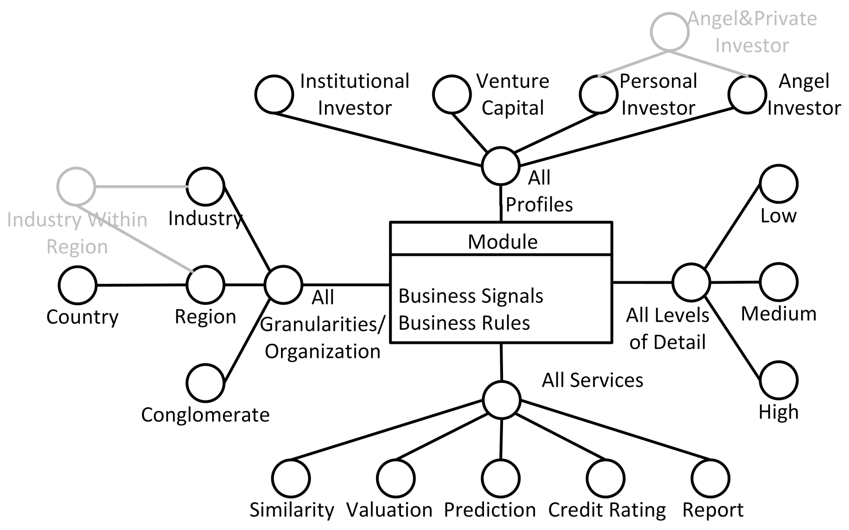


FIGURE 8.4: The context model for USTC resulting from applying grounded theory and qualitative content analysis to available resources.

implementation of similarity ranking: Each business signal is realized as Vadalog module and may include Vadalog modules it depends on, for example, the Vadalog module for investment stage includes a Vadalog module containing information about funding stages. Furthermore, each investor type is represented by a Vadalog module. The main Vadalog module deriving the organization similarity includes all Vadalog modules described above.

8.3.2 ECBR Model for USTC

We investigate ECBRs for the service portal. To this end, we identify relevant parameters, parameter values, and domain-specific business cases and subsequently instantiate the generic ECBR model for the intended service portal.

Our qualitative study revealed several factors influencing the applying business rules and the employed business signals, namely, level of detail, profiles, granularity, and service. These factors form the parameters of the ECBR model for USTC summarized in Figure 8.4. Initial parameters and their parameter values were identified by analysing various resources such as financial web sites and products. Subsequently, group discussions were employed to verify the found parameters and their parameter values as well as to determine further parameters and parameter values.

For the parameter *Level of detail* we determined the three different levels `Low`, `Medium`, and `High`. Thus, services of varying granularity may be consumed with different user profiles at different levels of detail. Regarding parameter *Profile* we employ the four investor types identified for similarity ranking of organizations as parameter values. Each of these profiles may employ different business rules for specific services, levels of detail, and granularity of service. The profiles may be combined, for instance, angel and private investor profile. Considering parameter *Granularity* of services, we have parameter value `All Granularities/Organization` allowing to define business rules for different services regarding organizations, at different levels of detail, and for different profiles. The granularity levels `Industry`, `Region`, `Country`, and `Conglomerate` are descendant parameter values of granularity level `All Granularities/Organization`. Consequently, all business rules employed at granularity level `All Granularities/Organization` are available to them and can be reused. This is helpful as industries, regions, and conglomerates comprise sets of organizations. Country is a specific region granularity level restricting regions to countries. As for profiles, granularity levels may be combined, for example, industry within a region. Services identified and verified are `Similarity` ranking, `Valuation`, `Prediction` of future development, `Credit Rating`, and `Report` forming the parameter *Service*.

In order to instantiate the generic ECBR model for USTC, we need to define the business case and its schema. The current implementation of the service offering similarity ranking of organizations requires the investor profile and the organization of interest. Therefore, these two inputs are part of the business case schema. Furthermore, the requested service and its granularity have to be part of the business case schema. Further information, as described in Section 8.3.1, may be useful to provide the described services.

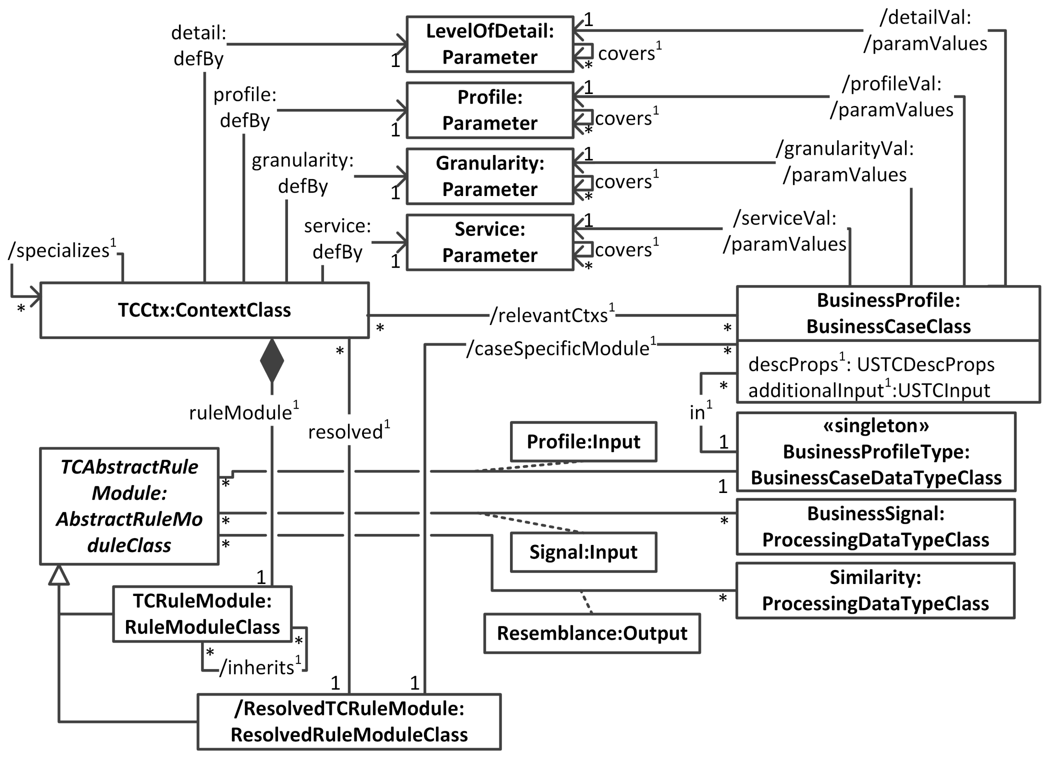
The described parameters, parameter values, and business cases can be instantiated to a domain-specific ECBR model for USTC. This model can be further instantiated to an application-specific one for USTC. An excerpt of these instantiations is described in the following examples.

Example 8.1. Figure 8.5 depicts a detail of the domain-specific ECBR model for USTC. Our domain-specific `ContextClass` is called USTC context, short `TCCtx`. It is described by the domain-specific parameters listed above, i.e., `LevelOfDetail`, `Profile`, `Granularity`, and `Service`, instantiating `Parameter`. The generic relationship `defBy` is configured accordingly. We employ hierarchical relationships only (`specializes` and `covers`) where the context hierarchy is derived from the parameter value hierarchy. Since we do not need any specific information regarding rule modules relevant to a `TCCtx` we simply instantiate `AbstractRuleModuleClass` to `TCAbstractRuleModule`, `RuleModuleClass` to `TCRuleModule`, and the derived module `ResolvedRuleModuleClass` to `ResolvedTCRuleModule`.

The domain-specific `BusinessCaseClass` for USTC is `BusinessProfile`. This class comprises information about the user such as the desired service, the granularity, the level of detail (based on user's experience), and the user's investor profile. These information form the domain-specific `descProps` class `USTCdescProps`. Other information such as age, risk aversion, portfolio goals, and capital goals form the domain-specific `additionalInput` class `USTCInput`.

Regarding data types, we identify business signals and business profiles as inputs to rule modules. Thus, we instantiate `ProcessingDataTypeClass` to `BusinessSignal` and `BusinessCaseDataTypeClass` to `BusinessProfileType`, respectively. Each USTC rule module accepts arbitrary many `BusinessSignal` and one `BusinessProfileType` as input. Class `BusinessProfileType` is singleton, i.e., all application-specific business cases refer to the same application-specific business case data type. Figure 8.5 only depicts the output of the similarity ranking service, i.e., `Similarity`.

Example 8.2. Figure 8.6 depicts an excerpt of an application-specific ECBR model for USTC. The concrete data types employed in the detail are: `tcbc` for application-specific instances of `BusinessProfile` such as `bc`, `orgSize` as specific business signal, and `orgSimiliarity` as specific similarity type. The application-specific `BusinessProfileType` `tcbc` describes the source of the set of `BusinessProfile` instances of schema `BusinessProfile`. Concrete business case `bc` is an instance in this set. Business signal `orgSize` references table `signals` in a database named `SignalDB` containing business signals in a specific signal schema. Similarity measure `orgSimilarity` refers to table `res` in a database named `ResultsDB` where organizational similarity measures



detail: defBy
profile: defBy
granularity: defBy
service: defBy
LevelOfDetail: Parameter
Profile: Parameter
Granularity: Parameter
Service: Parameter
covers1
/detailVal: /paramValues
/profileVal: /paramValues
/granularityVal: /paramValues
/serviceVal: /paramValues
/specializes1
TCCtx:ContextClass
/relevantCtxs1
/caseSpecificModule1
BusinessProfile: BusinessCaseClass
descProps1: USTCDescProps
additionalInput1:USTCInput
in1
ruleModule1
resolved1
Profile:Input
«singleton» BusinessProfileType: BusinessCaseDataTypeClass
TCAbstractRule Module: AbstractRuleModuleClass
BusinessSignal: ProcessingDataTypeClass
Signal:Input
Similarity: ProcessingDataTypeClass
TCRuleModule: RuleModuleClass
Resemblance:Output
/inherits1
/ResolvedTCRuleModule: ResolvedRuleModuleClass

FIGURE 8.5: An excerpt of the domain-specific integrated ECBR context model instantiated for USTC. Methods are not shown for readability.

are stored in relational form. Mappings for the different data types may be necessary depending on the implementation.

The excerpt shows two concrete contexts: one context with business rules applying for classes of situations regarding any level of detail, angel investors, any granularity of service, and similarity service (`all_angel_all_sim`) and a specializing context regarding low level of detail, angel investors, industry level of granularity, and similarity service (`low_angel_industry_sim`). Each of these contexts has a concrete rule module (respectively `rm1` or `rm2`) associated. Context `low_angel_industry_sim` also has a resolved rule module (`:/ResolvedTCRuleModule`) associated. The concrete and derived rule modules have data types `orgSize` and `tcbc` as inputs and data type `orgSimilarity` as non-omitable output.

Figure 8.6 depicts a concrete `BusinessProfile` called `bc`. It contains describing properties `low`, `angel`, `industry`, and `similarity`. From these describing properties we determine that contexts `all_angel_all_sim` and `low_angel_industry_sim` are relevant to `bc`. Thus, the `caseSpecificModule` is `:/ResolvedTCRuleModule`. The business rules in this module employ the `additionalInput` `Boeing` as the organization of interest for which to derive similar organizations.

In addition to the organization of business rules in the presented ECBR model, the business rules for business signals may be organized in an orthogonal RMI hierarchy. To this end, each business signal is represented by a rule module. Common rules and facts employed for business signals are extracted to a super rule module. There exists one root rule module from which all rule modules representing business signals need to inherit. This rule module defines interface predicates (for instance, the form in which business signals need to be output), rules, facts, and modification restrictions applying to all business signals. New business signals can be introduced by creating a

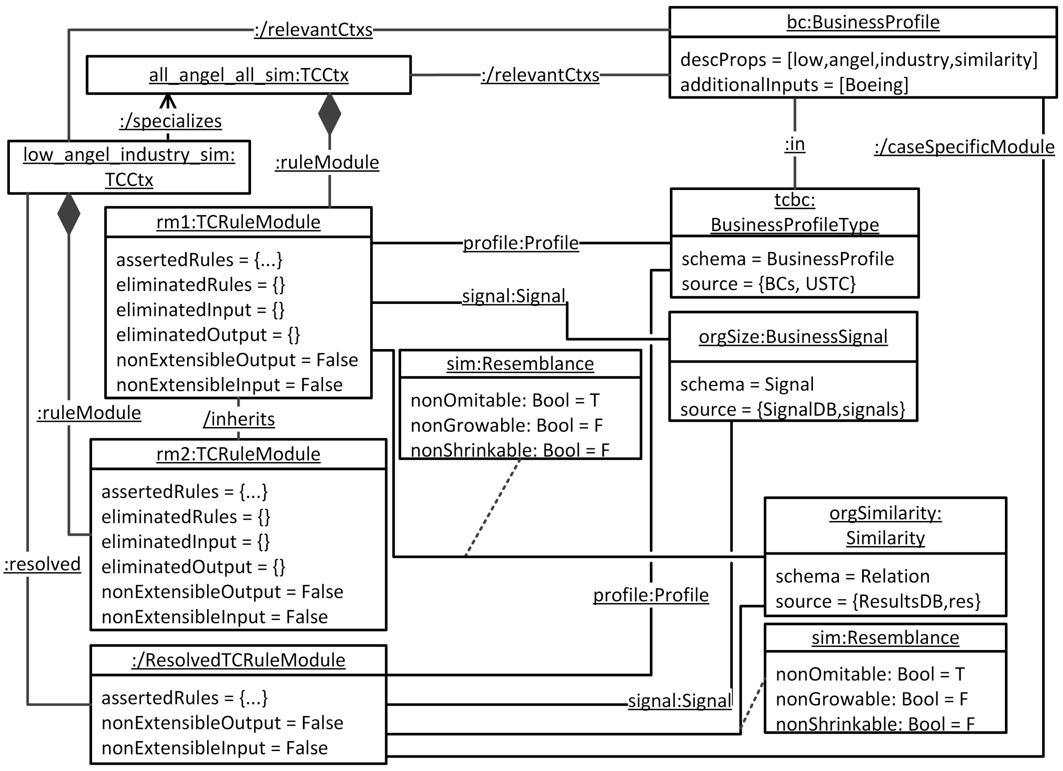


FIGURE 8.6: Detail of an application-specific ECBR model instantiated from the model in Figure 8.5. Instantiations of parameters and further data types have been omitted for readability.

new rule module inheriting from the root rule module; variations of existing business signals can be achieved by creating child modules and modifying the behavior and structure accordingly. Any modifications performed must not violate modification restrictions defined for ancestors. Business signals can be combined by creating a new rule module inheriting from all rule modules representing the business signals to be combined.

8.3.3 Discussion

In this section we discuss the suitability of employing the generic ECBR model and RMI for USTC. General advantages are discussed in Chapter 3 to 7 – here we discuss functionalities and advantages specific to USTC.

An essential feature of the generic ECBR model for USTC is the automatic determination of relevant contexts and thus rule modules. Considering the USTC service portal, business profiles are constructed from user information and user interaction like the chosen service. The presented ECBR models for USTC determine for a given business profile the relevant contexts and rule modules. User-specific services can be enabled by refining the `Profile` parameter. The explicit definition of classes of situations along the parameters and parameter values presented in Figure 8.4 eases maintenance of business rules for USTC.

Maintenance is further simplified by the composed modification operations introduced in Chapter 5: The feedback loop enables continuous improvement of user experience as well customization/personalization of services. Furthermore, additional services can be easily introduced employing the extension operations. Similarly, outdated (personalized) services may be removed using the restriction operations.

Utilizing rule modules for USTC contexts enables clear interfaces for services, i.e., each service defines the information required and the information output. Inheritance of rule modules reduces redundancies and promotes reuse of existing rule modules. Consequently, maintenance is eased. For instance, a user-specific service can be created by inheriting from a generic service and performing necessary adaptions. Modification restrictions allow to constrain the allowed modifications to services when inherited, for instance, they may specify a mandatory output schema.

Other features provided by the ECBR model for USTC are conflict-resolution and case-specific rule modules. The former is essential if multi-inheritance is employed for service construction, for example, introducing a profile combining angel and private investors; in all other cases it depends on the concrete implementation. Case-specific rule modules are interesting for rule developers when users report errors. Whether the case-specific rule modules are made available to users as well, depends on whether the company considers the implementation of services a competitive advantage.

A vital part of RMI, regarding the organization of business rules implementing business signals, are rule modules. Rule modules specify clear interfaces for business signals easing their embedding in an application landscape and improving their reusability. Furthermore, rule module interfaces can be utilized to form rule module chains where the output of one rule module is input to the next rule module. This is quite useful to business signals when constructing combined business signals from other business signals. Combinations of business signals can also be modeled using multi-inheritance as defined in RMI. RMI inheritance in general is useful to reduce redundancy and to control the minimum inputs and outputs to any business signal module. Furthermore, addition of business signals to the existing set of business signals is simplified.

8.4 Conclusion

In this chapter we have detailed two use cases, namely SemNOTAM and USTC. For both we instantiated the generic ECBR model and discussed its suitability. These discussions show that the generic ECBR model is suitable for both use cases, i.e., the problems and requirements can be satisfied with the generic ECBR model presented in Chapter 7.

Chapter 9

Conclusion

We have identified business rule organizations enabling effective and efficient maintenance, search, and execution of business rules as vital for today's organizations. We have investigated a promising organization form – the organization of business rules along their context of application. As a result we introduced the generic and conceptual Contextualized Business Rule Repository (CBR) model and presented CBR prototypes realizing CBR models. Furthermore, we investigated Rule Module Inheritance with Modification Restrictions (RMI) and integrated it into the generic CBR model, resulting in the generic Extended CBR with Rule Modules (ECBR) model. In the following, we detail our three main contributions.

9.1 The Generic CBR Model and CBR Prototypes

We reviewed background literature identifying aspects and attributes relevant to business rules, organization of business rules, management of business rules, contexts, context modeling, and development and deployment of CASAs. Employing these aspects and attributes, we have derived requirements for context-aware organization of business rules. Subsequently, we have utilized design-science to develop the generic and conceptual multi-level CBR model. The generic CBR model enables multi-dimensional hierarchical organization of business rule sets along their applying classes of situations. Furthermore, the generic CBR model can be instantiated to domain-specific CBR models, for example, SemNOTAM or USTC, and domain-specific CBR models to application-specific CBR models within the domain. We have demonstrated the usability of the generic CBR model by instantiating it for SemNOTAM and by realizing CBR models in several prototypes.

In order to enable effective and efficient business rule maintenance, we have introduced atomic and composed modification operations enabling changes to domain- and application-specific CBR models. Atomic modification operations comprise creation (instantiation), deletion, and modification of CBR model elements. These atomic modification operations are assigned to CBR roles to enable clear separation of responsibilities and tasks. Composed modification operations ensure that CBR models move from one consistent state to another (i.e. all side-effects are taken care of). To this end, composed modification operations define a sequence of atomic modification operations along with the necessary communication between CBR roles. Any composed operation must be executed completely or not at all.

The designed generic CBR model, its modification operations, and its CBR roles have a number of advantages such as explicit definition of the classes of situations in which business rules apply, automatic detection of relevant business rules for a given business cases, increased business rule readability, reduced redundancy of business rules, ensuring consistency of CBR models, and enabling clear separation of responsibilities and tasks. For example, an application-specific CBR model may organize business rule sets along responsible business units aligning business rule set organization and the organization of the business. Furthermore, the proposed generic CBR model can organize generic rules or knowledge as well. These benefits come at a cost – CBR models introduce additional effort to managing business rule repositories as the context model needs to be managed as well.

Regarding the claims made in Section 1.5, the generic CBR model satisfies all four claims. Claim (1), eased business rule maintenance, is mainly supported by the introduced atomic and composed modification operations, the proposed CBR roles, and the CBR-specific BRM described. Claim (1) is further supported by the multi-dimensional organization of business rules reflecting their applying classes of situations as well as the explicit and structured description of classes of situations: A context in a CBR model is described by a set of parameters (or dimensions). A specific context defines a value for each parameter. Depending on the use-case, the parameters and their parameter values may be used to determine the hierarchy of contexts and thus the hierarchy of classes of situations or the hierarchy of contexts may be explicitly defined. Claim (2), improved business rule execution performance, and claim (3) faster search, are supported by the organization of business rules into contexts as well as the explicit and structured description of contexts. Claim (4), support of executable as well as non-executable business rules, is fulfilled since no restriction regarding the representation of business rules or the class of business rules is made.

9.2 Rule Module Inheritance with Modification Restrictions

An important aspect in CBRs is inheritance, promoting reuse and adaption of business rule sets to different classes of situations. This is also relevant to generic rules and terms. Therefore, we have investigated rule modules, rule module inheritance, and modification restrictions on inherited rules and terms resulting in a precise definition of rule module inheritance with modification restrictions.

We have discussed and defined relation-based (Datalog$^{\pm}$) rule modules, i.e., rule sets defining input and output interfaces. A rule module comprises rule module structure, i.e., its name and interfaces (inputs and outputs), and rule module behavior defined by rules and facts. A rule module exhibits behavior if its rules and facts are applied to input data. To this end, the rule set comprised in the rule module must satisfy the global restrictions made on rule sets by the employed rule language, for instance, Vadalog allows only safe, stratified, and grounded negation [16, 17, 76]. The exhibited behavior is data derived for the output interface. For such rule modules we have investigated and precisely defined downward single-inheritance as well as downward multi-inheritance of rule module structure and behavior where extension and elimination of inherited rules, facts, input interfaces, and output interfaces are allowed. The defined inheritance mechanisms do not provide guarantees regarding global restrictions of the employed rule language – the rule developers are responsible that rule modules satisfy all global restrictions.

To avoid uncontrolled proliferation of modifications, we have introduced modification restrictions. These allow to restrict extension and elimination regarding rule module structure and rule module behavior. Modification restrictions defined for a parent rule module apply to all child modules. Child modules must not modify inherited modification restrictions. In order to check for violations of specified modification restrictions, we provide conformance checks. Furthermore, we have presented a prototypical implementation of rule modules and rule module inheritance employing Vadalog.

Rule modules enable clear separation of rule-based knowledge from application code by providing interfaces. Thus, rule developers can be shielded from application code and application developers from the intricacies of rule sets. Inheritance of rule modules enables the reuse of existing rule modules and hence reduces rule and fact redundancy. Consequently, maintenance of rules is eased. Modification restrictions allow to constrain structural and behavioral modifications in descendant rule modules. As such, they can be used to represent organizational constraints, such as specifying the minimum requirements for a good credit worthiness which may be strengthened, or ensuring a minimum overlap in behavior. Therefore, RMI mainly supports claim (1), eased business rule maintenance, stated in Section 1.5.

9.3 The Generic ECBR Model

In order to further ease maintenance of business rules in CBRs, a precise definition of inheritance is necessary. To this end, we have integrated the generic CBR model with RMI. This integration had to be performed carefully in order to maintain the claims and requirements satisfied by the generic CBR model. Further challenges arose from the two approaches being developed independently: the generic CBR model is object-based while RMI is relation-based, application-specific CBR models may contain business rule execution outputs while RMI may not, and the generic CBR model is multi-level while RMI is not. Addressing these challenges, we have developed the generic multi-level ECBR model. Similar to the generic CBR model, the generic ECBR model can be instantiated to domain-specific ECBR models and these can again be instantiated to application-specific ECBR models. Different to CBR models, ECBR models may not contain business rule execution outputs. Nevertheless, we have discussed the execution of rule modules in ECBRs and how to create executable rule modules from application-specific ECBR models. Furthermore, we have discussed realization considerations and how the prototypes presented for CBR and RMI can be combined to realize ECBRs.

We have demonstrated the designed generic ECBR model by applying it to two use cases: SemNOTAM and USTC. We have delineated details of each use case and presented domain-specific ECBR models for them. Furthermore, we have shown excerpts of their application-specific ECBR models. For each use case we have discussed the suitability of employing and instantiating the generic ECBRs model.

Regarding the claims stated in Section 1.5, the generic ECBR model maintains the satisfaction of all claims satisfied by the generic CBR model. Moreover, it further eases business rule maintenance by integrating RMI into the generic CBR model, improving on claim (1) eased business rule maintenance.

9.4 Outlook

Our investigations resulted in conceptual models for CBRs and ECBRs as well as several proof-of-concept prototypes. We limited our investigations to cases where model elements are not shared across domain-specific or application-specific models.

Future work will consider tool-support for CBRs and ECBRs, including support for CBR roles and modification operations. We also intend to investigate contextualized repositories for contextualized knowledge other than (business) rules. Furthermore, regarding CBR and ECBR models, investigations of shared model elements, such as shared parameters, between domain-specific models and application-specific models respectively, are of interest. Particularly interesting are the effects on the presented CBR roles and composed modification operations. Regarding rule module inheritance, future work will consider rule modules as part of derivation chains or module networks in big data wrangling [109] and knowledge graph management [122].

Bibliography

[1] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. "Relational Transducers for Electronic Commerce". In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 1998, pp. 179–187.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. "Towards a Better Understanding of Context and Context-Awareness". In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. London, UK, UK, 1999, pp. 304–307.

[3] U. Alegre, J. C. Augusto, and T. Clark. "Engineering context-aware systems and applications: A survey". In: *Journal of Systems and Software* 117 (2016), pp. 55–83.

[4] A. Analyti, G. Antoniou, and C. V. Damasio. "MWeb: A Principled Framework for Modular Web Rule Bases and Its Semantics". In: *ACM Trans. Comput. Logic* 12.2 (2011), 17:1–17:46.

[5] A. Analyti, M. Theodorakis, N. Spyratos, and P. Constantopoulos. "Contextualization as an independent abstraction mechanism for conceptual modeling". In: *Information Systems* 32.1 (2007), pp. 24–60.

[6] T. Athan, H. Boley, and A. Paschke. "RuleML 1.02: Deliberation, Reaction and Consumer Families." In: *Challenge+ DC@ RuleML*. 2015.

[7] C. Atkinson. "Meta-Modeling for Distributed Object Environments". In: *1st International Enterprise Distributed Object Computing Conference (EDOC '97), 24-26 October 1997, Gold Coast, Australia, Proceedings*. 1997, p. 90.

[8] C. Atkinson, B. Kennel, and B. Goß. "The Level-Agnostic Modeling Language". In: *Software Language Engineering*. Springer Berlin Heidelberg, 2011, pp. 266–275.

[9] C. Atkinson and T. Kühne. "The Essence of Multilevel Metamodeling". In: *≪UML≫ 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*. Springer Berlin Heidelberg, 2001, pp. 19–33.

[10] C. Atkinson and T. Kühne. "Rearchitecting the UML infrastructure". In: *ACM Transactions on Modeling and Computer Simulation* 12.4 (2002), pp. 290–321.

[11] C. Atkinson and T. Kühne. "Model-driven development: a metamodeling foundation". In: *Software, IEEE* 20.5 (Sept. 2003), pp. 36–41.

[12] M. Bajec and M. Krisper. "A methodology and tool support for managing business rules in organisations". In: *Information Systems* 30.6 (2005), pp. 423–443.

[13] M. Baldauf, S. Dustdar, and F. Rosenberg. "A survey on context-aware systems". In: *International Journal of Ad Hoc and Ubiquitous Computing* 2.4 (2007), p. 263.

[14] C. Bauer. “A comparison and validation of 13 context meta-models”. In: *ECIS 2012 Proceedings*. 17. 2012.

[15] M. Bazire and P. Brézillon. “Understanding Context Before Using It”. In: *Modeling and Using Context*. Ed. by A. Dey, B. Kokinov, D. Leake, and R. Turner. Vol. 3554. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 29–40.

[16] L. Bellomarini, G. Gottlob, A. Pieris, and E. Sallinger. “Swift Logic for Big Data and Knowledge Graphs”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*. 2017, pp. 2–10.

[17] L. Bellomarini, E. Sallinger, and G. Gottlob. “The Vadalog System: Datalog-based Reasoning for Knowledge Graphs”. In: *PVLDB* 11.9 (2018), pp. 975–987.

[18] M. Benerecetti, P. Bouquet, and C. Ghidini. “Contextual reasoning distilled”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 12.3 (2000), pp. 279–305.

[19] M. Benerecetti, P. Bouquet, and C. Ghidini. “On the Dimensions of Context Dependence: Partiality, Approximation, and Perspective”. In: *Modeling and Using Context*. Springer Berlin Heidelberg, 2001, pp. 59–72.

[20] E. Bertino, G. Guerrini, and I. Merlo. “Trigger inheritance and overriding in an active object database system”. In: *IEEE Transactions on Knowledge and Data Engineering* 12.4 (2000), pp. 588–608.

[21] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. “A survey of context modelling and reasoning techniques”. In: *Pervasive and Mobile Computing* 6.2 (2010), pp. 161–180.

[22] P. Bichler and M. Schrefl. “Inheritance of Business Rules”. In: *Hildesheimer Informatik-Berichte, 7. Workhop “Grundlagen von Datenbanken”*. 1995.

[23] C. Bolchini, C. A. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca. “And what can context do for data?” In: *Communications of the ACM* 52.11 (2009), p. 136.

[24] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. “A data-oriented survey of context models”. In: *ACM SIGMOD Record* 36.4 (2007), p. 19.

[25] C. Bolchini, E. Quintarelli, and L. Tanca. “CARVE: Context-aware automatic view definition over relational databases”. In: *Information Systems* 38.1 (2013), pp. 45–67.

[26] P. Bouquet, C. Ghidini, F. Giunchiglia, and E. Blanzieri. “Theories and uses of context in knowledge representation and reasoning”. In: *Journal of Pragmatics* 35.3 (2003). Context Context, pp. 455–484.

[27] J. Boyer and H. Mili. *Agile Business Rule Development*. Springer Berlin Heidelberg, 2011, pp. 49–71.

[28] L. Bozzato, T. Eiter, and L. Serafini. “Enhancing context knowledge repositories with justifiable exceptions”. In: *Artificial Intelligence* 257 (2018), pp. 72–126.

[29] N. Bradley and M. Dunlop. “Toward a Multidisciplinary Model of Context to Support Context-Aware Computing”. In: *Human-Computer Interaction* 20.4 (2005), pp. 403–446.

[30] BRCommunity.com. *Business Rules Community*. 2016. URL: `BRCommunity.com`.

[31] P. Brézillon. *Context in Artificial Intelligence: I. A survey of literature*. 1999.

[32] P. Brézillon. *Context in Artificial Intelligence: II. Key elements of contexts*. 1999.

[33] D. E. Brown and J. J. Pomykalski. "Reliability estimation during prototyping of knowledge-based systems". In: *IEEE Transactions on Knowledge and Data Engineering* 7.3 (1995), pp. 378–390.

[34] F. Burgstaller, B. Neumayr, C. G. Schuetz, and M. Schrefl. "Modification Operations for Context-Aware Business Rule Management". In: *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. 2017, pp. 194–203.

[35] F. Burgstaller. "Employing Aviation-Specific Contexts for Business Rules and Business Vocabularies Management in SemNOTAM". In: *On the Move to Meaningful Internet Systems: OTM 2016 Workshops*. 2017, pp. 315–325.

[36] F. Burgstaller, B. Neumayr, E. Sallinger, and M. Schrefl. "Rule Module Inheritance with Modification Restrictions". In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. Ed. by H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman. 2018, pp. 404–422.

[37] F. Burgstaller, C. Schütz, B. Neumayr, and M. Schrefl. "Towards Contextualized Rule Repositories for the Semantic Web". In: *Joint Proceedings of the Web Stream Processing workshop (WSP 2017) and the 2nd International Workshop on Ontology Modularity, Contextuality, and Evolution (WOMoCoE 2017) co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 22nd, 2017*. Oct. 2017, pp. 98–109. URL: `http://ceur-ws.org/Vol-1936/paper-09.pdf`.

[38] F. Burgstaller, M. Stabauer, R. Morgan, and G. Grossmann. "Towards Customised Visualisation of Ontologies: State of the Art and Future Applications for Online Polls Analysis". In: *Proceedings of the Australasian Computer Science Week Multiconference ACSW '17*. 2017, 26:1–26:10.

[39] F. Burgstaller, D. Steiner, B. Neumayr, M. Schrefl, and E. Gringinger. "Using a model-driven, knowledge-based approach to cope with complexity in filtering of notices to airmen". In: *Proceedings of the Australasian Computer Science Week Multiconference, Canberra, Australia, February 2-5, 2016*. 2016, 46:1–46:10.

[40] F. Burgstaller, D. Steiner, and M. Schrefl. "Modeling Context for Business Rule Management". In: *2016 IEEE 18th Conference on Business Informatics (CBI)*. 2016, pp. 262–271.

[41] F. Burgstaller, D. Steiner, M. Schrefl, E. Gringinger, S. Wilson, and S. van der Stricht. "AIRM-based, fine-grained semantic filtering of notices to airmen". In: *2015 Integrated Communication, Navigation and Surveillance Conference (ICNS)*. 2015, pp. D3–1—-D3–13.

[42] L. Burgueño et al., eds. *Proceedings of the 4th International Workshop on Multi-Level Modelling (MULTI 2017)*. Vol. 2019. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 213–217. URL: `http://ceur-ws.org/Vol-2019/multi%5C_1.pdf`.

[43] Business Rules Group. *Business Rules Manifesto: The Principles of Rule Independence*. 2003. URL: `http://www.businessrulesgroup.org/brmanifesto.htm`.

[44] R. Butleris and K. Kapocius. "The business rules repository for information systems design". In: *In Research Communications of 6th East European Conference ADBIS 2002, Vydavatel'stvo STU*. 2002, pp. 64–77.

[45] A. Calì, G. Gottlob, and T. Lukasiewicz. "A general Datalog-based framework for tractable query answering over ontologies". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 14.Supplement C (2012), pp. 57–83.

[46] J. G. Carbonell. "Default Reasoning and Inheritance Mechanisms on Type Hierarchies". In: *SIGMOD Rec.* 11.2 (1980), pp. 107–109.

[47] M. Chen. "A Model-Driven Approach to Accessing Managerial Information: The Development of a Repository-Based Executive Information System". In: *Journal of Management Information Systems* 11.4 (1995), pp. 33–63.

[48] Context. In: *Cambridge Dictionary*.

[49] Context. In: *Dictionary by Merriam-Webster*.

[50] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. "Context is key". In: *Communications of the ACM* 48.3 (2005), p. 49.

[51] C. Curino, E. Quintarelli, and L. Tanca. "Ontology-Based Information Tailoring". In: *22nd International Conference on Data Engineering Workshops*. 2006.

[52] M. D'Aquin, M. Sabou, and E. Motta. "Modularization: A Key for the Dynamic Selection of Relevant Knowledge Components". In: *Proceedings of the 1st International Conference on Modular Ontologies - Volume 232*. WoMO'06. 2006, pp. 15–28.

[53] M. D'Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou. "Ontology Modularization for Knowledge Selection: Experiments and Evaluations". In: *Database and Expert Systems Applications*. Ed. by R. Wagner, N. Revell, and G. Pernul. 2007, pp. 874–883.

[54] J. De Lara, E. Guerra, and J. S. Cuadrado. "When and How to Use Multilevel Modelling". In: *ACM Transactions on Software Engineering and Methodology* 24.2 (2014), pp. 1–46.

[55] G. De Simoni and R. Edjlali. *Magic Quadrant for Metadata Management Solutions*. Tech. rep. Gartner, 2016.

[56] M. De Wael, J. Swalens, and W. De Meuter. "Just-in-time Inheritance: A Dynamic and Implicit Multiple Inheritance Mechanism". In: *SIGPLAN Not.* 52.2 (2016), pp. 37–47.

[57] A. K. Dey. "Understanding and Using Context". In: *Personal Ubiquitous Comput.* 5.1 (2001), pp. 4–7.

[58] J. L. G. Dietz. "On the Nature of Business Rules". In: *Advances in Enterprise Engineering I*. Ed. by J. L. G. Dietz, A. Albani, and J. Barjis. 2008, pp. 1–15.

[59] A. Dormer. "A Framework for Optimising Business Rules". In: *Business Information Systems Workshops*. Ed. by W. Abramowicz. 2017, pp. 5–17.

[60] P. Dourish. "What we talk about when we talk about context". In: *Personal and Ubiquitous Computing* 8.1 (2004), pp. 19–30.

[61] R. Ducournau and J. Privat. "Metamodeling semantics of multiple inheritance". In: *Science of Computer Programming* 76.7 (2011), pp. 555–586.

[62] EUROCONTROL. *Aeronautical Information Exchange Model (AIXM)*. 2016. URL: `aixm.aero`.

[63] EUROCONTROL. *Performance Statistics Year over Year Evolution*. 2017. URL: https://www.ead.eurocontrol.int/cms-eadbasic/opencms/en/ead-operations/performance-statistics/yoy-evolution/.

[64] EUROCONTROL. *Digital NOTAM (Phase 3 P-21)*. Oct. 15, 2018. URL: https://www.eurocontrol.int/articles/digital-notam-phase-3-p-21.

[65] EUROCONTROL. *Performance Statistics Current Year*. Oct. 15, 2018. URL: https://www.ead.eurocontrol.int/cms-eadbasic/opencms/en/ead-operations/performance-statistics/current-year/.

[66] EUROCONTROL and FAA. *Digital NOTAM Event Specification 1.0*. Tech. rep. 2011. URL: http://aixm.aero/page/digital-notam.

[67] M. Evans, K. Dalkir, and C. Bidian. "A Holistic View of the Knowledge Life Cycle: The Knowledge Management Cycle (KMC) Model." In: *Electronic Journal of Knowledge Management* 12.2 (2014), pp. 148–160.

[68] FAA. *Federal NOTAM System NOTAM Scenarios*. 2010. URL: https://notams.aim.faa.gov#Documentation.

[69] FAA. *FAI FSS - NOTAM Overview*. 2016. URL: http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/systemops/fs/alaskan/alaska/fai/notam/ntm_overview/.

[70] FICO. *Blaze Advisor and Decision Modeler*. 2017. URL: https://community.fico.com/community/fico-blaze-advisor-and-decision-modeler-community/content?filterID=contentstatus%5Bpublished%5D~category%5Bdocumentation%5D.

[71] M. Frické. In: *Logic and the Organization of Information*. Springer New York, 2012. Chap. Logic and the Organization of Information, pp. 277–282.

[72] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. "DIADEM: Thousands of Websites to a Single Database". In: *PVLDB* 7.14 (2014), pp. 1845–1856.

[73] F. Giunchiglia and P. Bouquet. "Introduction to contextual reasoning. An Artificial Intelligence perspective". In: *Perspectives on Cognitive Science*. NBU Press, 1997, pp. 138–159.

[74] R. J. Glushko. *The discipline of organizing*. Vol. 40. 1. Wiley Subscription Services, Inc., A Wiley Company, 2013, pp. 21–27.

[75] E. Gottesdiener. "Business Rules Show Power, Promise". In: *Application Development Trends* 4.3 (1997), pp. 36–42.

[76] G. Gottlob and A. Pieris. "Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. 2015, pp. 2999–3007.

[77] I. Graham. *Business Rules Management and Service Oriented Architecture: A Pattern Language*. John Wiley & Sons, 2006.

[78] J. N. Gray. "Notes on data base operating systems". In: *Operating Systems: An Advanced Course*. 1978, pp. 393–481.

[79] T. Gu, H. K. Pung, and D. Q. Zhang. "A service-oriented middleware for building context-aware services". In: *Journal of Network and Computer Applications* 28.1 (2005), pp. 1–18.

[80] T. Haerder and A. Reuter. “Principles of Transaction-oriented Database Recovery”. In: *ACM Comput. Surv.* 15.4 (1983), pp. 287–317.

[81] B. von Halle. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. 1st. Wiley Publishing, 2001.

[82] H. Hamza and M. Fayad. “A novel approach for managing and reusing business rules in business architectures”. In: *The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.* 2005, pp. 973–978.

[83] D. Hay, K. A. Healy, J. Hall, et al. *Defining business rules - what are they really*. 2000. URL: `www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf`.

[84] K. Henricksen. “A Framework for Context-aware Pervasive Computing Applications”. PhD thesis. University of Queensland, 2003.

[85] K. Henricksen, J. Indulska, and A. Rakotonirainy. “Modeling Context Information in Pervasive Computing Systems”. English. In: *Lecture Notes in Computer Science*. Ed. by F. Mattern and M. Naghshineh. Vol. 2414. 2002, pp. 167–180.

[86] H. Herbst. “Business rules in systems analysis: A meta-model and repository system”. In: *Information Systems* 21.2 (1996), pp. 147–166.

[87] H. Herbst and T. Myrach. “A Repository System for Business Rules”. In: *Database Applications Semantics: Proceedings of the IFIP WG 2.6 Working Conference on Database Applications Semantics (DS-6) Stone Mountain, Atlanta, Georgia U.S.A., May 30–June 2, 1995*. Ed. by R. Meersman and L. Mark. Almost identical to "A meta-model for business rules in systems analysis". 1997, pp. 119–139.

[88] H. Herbst, G. Knolmayer, T. Myrach, and M. Schlesinger. “The Specification of Business Rules: A Comparison of Selected Methodologies”. In: *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle*. 1994, pp. 29–46.

[89] A. R. Hevner, S. T. March, J. Park, and S. Ram. “Design Science in Information Systems Research”. In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[90] A. Hinton. *Understanding Context: Environment, Language, and Information Architecture*. O’Reilly Media, 2014.

[91] B. Hnatkowska and J. M. Alvarez-Rodriguez. “Business Rule Patterns Catalog for Structural Business Rules”. In: *Software Engineering: Challenges and Solutions*. Ed. by L. Madeyski, M. Śmiałek, B. Hnatkowska, and Z. Huzar. 2017, pp. 3–16.

[92] R. Hoeft, F. Jentsch, and J. Kochan. “Freeing NOTAMs From Teletype Technology”. In: *Flight Safety Digest* 23.4 (2004), pp. 1–35. URL: `http://trid.trb.org/view.aspx?id=610796`.

[93] M. Homola, L. Serafini, and A. Tamilin. “Modeling contextualized knowledge”. In: *Procs. of the 2nd Workshop on Context, Information and Ontologies (CIAO 2010)*. Vol. 626. 2010.

[94] M. Hussein, J. Han, and A. Colman. “An Approach to Model-Based Development of Context-Aware Adaptive Systems”. In: *2011 IEEE 35th Annual Computer Software and Applications Conference*. 2011.

[95] IBM. *IBM Websphere ILOG JRules BRMS V7.1.x documentation*. 2018. URL: `https://www.ibm.com/support/knowledgecenter/SS6MTS_7.1.1/welcome.html`.

[96] N. Ibrahim and F. le Mouel. “Context-aware Specialization of Semantic Rules for choosing Services in Pervasive Environments”. In: *IEEE International Conference on Pervasive Services*. 2007, pp. 391–396.

[97] F. Imgrund, M. Malorny, and C. Janiesch. “Eine Literaturanalyse zur Integration von Business Rules und Business Process Management”. In: *Wirtschaftsinformatik 2017 Proceedings*. 2017, pp. 211–225.

[98] S. Jácome and J. De Lara. “Controlling Meta-Model Extensibility in Model-Driven Engineering”. In: *IEEE Access* 6 (2018), pp. 19923–19939.

[99] L. Jokste and J. Grabis. “Context-aware Adaption of Software Entities using Rules”. In: *Proceedings of the 19th International Conference on Enterprise Information Systems*. 2017, pp. 166–171.

[100] B. H. Kang, P. Compton, and P. Preston. “Multiple Classification Ripple Down Rules: Evaluation and Possibilitie”. In: *Possibilities Proceedings 9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop Banff. Feb 26 - March 3*. 1995, pp. 11–17.

[101] G. M. Kapitsaki and I. S. Venieris. “Model-Driven Development of Context-Aware Web Applications Based on a Web Service Context Management Architecture”. In: *Models in Software Engineering*. 2009, pp. 343–355.

[102] P. Kardasis and P. Loucopoulos. “Expressing and organising business rules”. In: *Information and Software Technology* 46.11 (2004), pp. 701–718.

[103] M. Kifer and H. Boley. *Rule Interchange Format (RIF) Overview*. Tech. rep. 2. 2013. URL: `http://www.%20w3.%20org/TR/rif-overview`.

[104] M. Kifer, G. Yang, H. Wan, and C. Zhao. *$\mathcal{ERGO}^{L}$ite (a.k.a. $\mathcal{FLORA}$-2): User Manual*. 1.2. 2017.

[105] M. Kirsch-Pinheiro, R. Mazo, C. Souveyet, and D. Sprovieri. “Requirements Analysis for Context-oriented Systems”. In: *Procedia Computer Science* 83 (2016), pp. 253–261.

[106] H. Knublauch, J. A. Hendler, and K. Idehen. *SPIN - Overview and Motivation*. Tech. rep. W3C Member Submission, 2011. URL: `https://www.w3.org/Submission/spin-overview/`.

[107] H. Koç, E. Hennig, S. Jastram, and C. Starke. “State of the Art in Context Modelling – A Systematic Literature Review”. In: *Lecture Notes in Business Information Processing*. 2014, pp. 53–64.

[108] B. Konev, C. Lutz, D. Walther, and F. Wolter. “Modular Ontologies”. In: *Lecture Notes in Computer Science*. 2009. Chap. Formal Properties of Modularisation, pp. 25–66.

[109] N. Konstantinou et al. “The VADA Architecture for Cost-Effective Data Wrangling”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1599–1602.

[110] A. Kovacic. “Business renovation: business rules (still) the missing link”. In: *Business Process Management Journal* 10.2 (2004), pp. 158–170.

[111] I. Kovacic, D. Steiner, C. Schuetz, B. Neumayr, F. Burgstaller, M. Schrefl, and S. Wilson. “Ontology-based data description and discovery in a SWIM environment”. In: *2017 Integrated Communications, Navigation and Surveillance Conference (ICNS)*. 2017, 5A4-1–5A4-13.

[112] M. M. Kwan and P. Balasubramanian. “KnowledgeScope: managing knowledge in context”. In: *Decision Support Systems* 35.4 (2003), pp. 467–486.

[113] P. Lang, W. Obermair, and M. Schrefl. “Modeling business rules with situation/activation diagrams”. In: *Proceedings 13th International Conference on Data Engineering*. 1997, pp. 455–464.

[114] A. Lapouchnian and J. Mylopoulos. “Modeling Domain Variability in Requirements Engineering with Contexts”. In: *Conceptual Modeling - ER 2009*. Ed. by A. H. F. Laender, S. Castano, U. Dayal, F. Casati, and J. P. M. de Oliveira. 2009, pp. 115–130.

[115] D. Lenat. *The Dimensions of Context-Space*. Tech. rep. CYCORP, 1998. URL: `http://web.media.mit.edu/~lieber/Teaching/Common-Sense-Course/Dimensions-Context-Space.pdf`.

[116] S.-H. Liao. “Expert system methodologies and applications—a decade review from 1995 to 2004”. In: *Expert Systems with Applications* 28.1 (2005), pp. 93–103.

[117] B. P. Lientz. *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980.

[118] B. H. Liskov and J. M. Wing. “A Behavioral Notion of Subtyping”. In: *ACM Trans. Program. Lang. Syst.* 16.6 (1994), pp. 1811–1841.

[119] B. Liu, M. Hu, and W. Hsu. “Multi-level organization and summarization of the discovered rules”. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*. 2000.

[120] R. A. Lorie. “Physical Integrity in a Large Segmented Database”. In: *ACM Trans. Database Syst.* 2.1 (1977), pp. 91–104.

[121] J. Luftman. “Assessing Business-IT Allignment Maturity”. In: *Strategies for Information Technology Governance*. 2004, pp. 99–128.

[122] E. S. Luigi Bellomarini. *VadalogEngine QuickGuide*. 2017.

[123] A. Manzoor, H.-L. Truong, and S. Dustdar. “Quality of Context: models and applications for context-aware systems in pervasive environments”. In: *The Knowledge Engineering Review* 29.02 (2014), pp. 154–170.

[124] S. T. March and G. F. Smith. “Design and natural science research on information technology”. In: *Decision Support Systems* 15.4 (1995), pp. 251–266.

[125] L. Mathiassen and J. Pries-Heje. “Business agility and diffusion of information technology”. In: *European Journal of Information Systems* 15.2 (2006), pp. 116–119.

[126] P. Mayring. *Einführung in die qualitative Sozialforschung: Eine Anleitung zu qualitativem Denken*. Weinheim: Beltz Verlag, 2002.

[127] J. McCarthy. “Notes on Formalizing Context”. In: *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*. 1993, pp. 555–560.

[128] J. F. Mejia Bernal, P. Falcarin, M. Morisio, and J. Dai. “Dynamic Context-aware Business Process: A Rule-based Approach Supported by Pattern Identification”. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. 2010, pp. 470–474.

[129] Z. Meng and J. Lu. "A Rule-based Service Customization Strategy for Smart Home Context-Aware Automation". In: *IEEE Transactions on Mobile Computing* 15.3 (2016), pp. 558–571.

[130] Z. Meng. "Investigation of a Hierarchical Context-aware Architecture for Rule-based Customisation of Mobile Computing Service". PhD thesis. University of Huddersfield, 2014.

[131] B. Meyer. "Applying 'design by contract'". In: *IEEE Computer* 25.10 (1992), pp. 40–51.

[132] M. Miraoui, C. Tadj, and C. B. Amar. "Architectural Survey of context-aware systems in pervasive computing environment". In: *Ubiquitous Computing and Communication Journal* 3.3 (2008), pp. 1–9.

[133] P. T. Moore and H. V. Pham. "Personalization and rule strategies in data-intensive intelligent context-aware systems". In: *The Knowledge Engineering Review* 30.2 (2015), pp. 140–156.

[134] P. Moore, B. Hu, M. Jackson, and J. Wan. "'Intelligent Context' for Personalized Mobile Learning". In: *Architectures for Distributed and Complex M-Learning Systems*. IGI Global, 2010, pp. 236–270.

[135] T. Morgan. *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley Professional, 2002.

[136] L. Morgenstern, C. Welty, H. Boley, and G. Hallmark. *Rule Interchange Format (RIF) Primer*. Tech. rep. 2. 2013. URL: `https://www.w3.org/TR/rif-primer/`.

[137] M. zur Muehlen and M. Indulska. "Modeling languages for business processes and business rules: A representational analysis". In: *Information Systems* 35.4 (2010). Vocabularies, Ontologies and Rules for Enterprise and Business Process Modeling and Management, pp. 379–390.

[138] G. Nalepa. *Modeling with Rules Using Semantic Knowledge Engineering*. Springer International Publishing, 2018.

[139] M. L. Nelson, R. L. Rariden, and R. Sen. "A Lifecycle Approach towards Business Rules Management". In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. 2008, pp. 113–113.

[140] A. Nowak and A. Wakulicz-Deja. "Knowledge reprensentation for composited knowledge bases". In: *Intelligent Information Systems 2008*. 2008, pp. 405–414.

[141] OMG. *Prodcution Rule Representation (PRR)*. Tech. rep. 1. Object Management Group, 2009. URL: `http://www.omg.org/spec/DMN/`.

[142] OMG. *Object Constraint Language*. Tech. rep. 1.3. Feb. 2, 2014. URL: `http://www.omg.org/spec/OCL`.

[143] OMG. *Business Motivation Model (BMM)*. Tech. rep. 1.3. Object Management Group, 2015. URL: `http://www.omg.org/spec/SBVR/1.3/PDF`.

[144] OMG. *Case Management Model and Notation (CMMN)*. Tech. rep. 1.1. Object Management Group, 2016. URL: `https://www.omg.org/spec/CMMN/1.1/PDF`.

[145] OMG. *Decision Model and Notation (DMN)*. Tech. rep. 1.1. Object Management Group, 2016. URL: `http://www.omg.org/spec/DMN/`.

[146] OMG. *Semantics of Business Vocabulary and Business Rules (SBVR)*. Tech. rep. 1.4. Object Management Group, 2017. URL: `http://www.omg.org/spec/SBVR/1.4/PDF`.

[147] M. van Oosterhout, E. Waarts, E. van Heck, and J. van Hillegersberg. "Business agility: need, readiness and alignment with IT-strategies". In: *Agile Information Systems: Conceptualization, Construction and Management*. 2006, pp. 52–69.

[148] Oracle Corporation. *Oracle Fusion Middleware - Designing Business Rules with Oracle Business Process Management*. 2016. URL: `https://docs.oracle.com/middleware/1213/bpm/rules-user/`.

[149] G. Orsi and L. Tanca. "Context Modelling and Context-Aware Querying". In: *Datalog Reloaded*. Ed. by O. de Moor, G. Gottlob, T. Furche, and A. Sellers. 2011, pp. 225–244.

[150] F. Pachet. "Rule Base Inheritance". In: *Représentations par objets*. 1992.

[151] C. Parent and S. Spaccapietra. *Modular Ontologies*. Springer Berlin Heidelberg, 2009.

[152] J. Pascoe. "Adding generic contextual capabilities to wearable computers". In: *Digest of Papers. Second International Symposium on Wearable Computers (Cat. No.98EX215)*. 1998.

[153] J. Pascoe, N. Ryan, and D. Morse. "Issues in Developing Context-Aware Computing". In: *Handheld and Ubiquitous Computing*. 1999, pp. 208–221.

[154] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. "A Design Science Research Methodology for Information Systems Research". In: *J. Manage. Inf. Syst.* 24.3 (Dec. 2007), pp. 45–77.

[155] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. "Context Aware Computing for The Internet of Things: A Survey". In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 414–454.

[156] A. Perkins. "Business rules=meta-data". In: *Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS 34*. 2000.

[157] J. C. Pomerol and P. Brézillon. "About Some Relationships between Knowledge and Context". In: *Modeling and Using Context*. Ed. by V. Akman, P. Bouquet, R. Thomason, and R. Young. Vol. 2116. Lecture Notes in Computer Science. 2001, pp. 461–464.

[158] E. Porosnicu, D. Hughes, and A. Standar. *Operational Service and Environment Definition (OSED) Step 2*. Tech. rep. Single European Sky ATM Research Program, 2013.

[159] N. Prakash, D. K. Sharma, D. Prakash, and D. Singh. "A Framework for Business Rules". In: *Advances in Conceptual Modeling: ER 2013 Workshops, LSAWM, MoBiD, RIGiM, SeCoGIS, WISM, DaSeM, SCME, and PhD Symposium, Hong Kong, China, November 11-13, 2013, Revised Selected Papers*. Ed. by J. Parsons and D. Chiu. 2014, pp. 68–73.

[160] E. Quintarelli, E. Rabosio, and L. Tanca. "A principled approach to context schema evolution in a data management perspective". In: *Information Systems* 49 (2015), pp. 65–101.

[161] V. K. Rai and C. Anantaram. "Structuring business rules interactions". In: *Electronic Commerce Research and Applications* 3.1 (2004), pp. 54–73.

[162] A. Reuter. "A Fast Transaction-Oriented Logging Scheme for Undo Ro overy". In: *IEEE Transactions on Software Engineering* SE-6.4 (1980), pp. 348–356.

[163] D. Rosca and C. Wild. "Towards a flexible deployment of business rules". In: *Expert Systems with Applications* 23.4 (2002), pp. 385–394.

[164] F. Rosenberg and S. Dustdar. "Business rules integration in BPEL - a service-oriented approach". In: *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*. 2005, pp. 476–479.

[165] F. Rosenberg, C. Nagl, and S. Dustdar. "Applying Distributed Business Rules - The VIDRE Approach". In: *2006 IEEE International Conference on Services Computing (SCC'06)*. 2006, pp. 471–478.

[166] R. G. Ross. *The Business Rule Book: Classifying, Defining and Modeling Rules: Ross Method*. Business Rule Solutions Incorporated, 1997.

[167] R. G. Ross. *Business Rule Concepts: Getting to the Point of Knowledge*. 4th. Business Rule Solutions, 2013.

[168] R. G. Ross. *Principles of the Business Rule Approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[169] RTCA. *NOTAM Search and Filter Options*. 2014. URL: `http://www.rtca.org/Files/Miscellaneous%20Files/NOTAM%20Search%20and%20Filter%20Options%20drft.pdf`.

[170] Rule. In: *Cambridge Dictionary*.

[171] Rule. In: *Collins Dictionary*.

[172] Rule. In: *Oxford Dictionary*.

[173] Rule. In: *Dictionary by Merriam-Webster*.

[174] RuleML. *RuleML Wiki*. RuleML. 2017. URL: `http://wiki.ruleml.org`.

[175] K. Sandkuhl. "Context Modeling for Knowledge Management Systems". In: *WM2017 - 9te Konferenz Professionelles Wissensmanagement*. 2017.

[176] M. Schacher and G. Patrick. *Agile Unternehmen durch Business Rules*. Springer-Verlag, 2006.

[177] A. Schäfer and M. Kreher. "Wie strukturiere ich meine Business Rules für eine effektive Pflege und Steuerung von kritischen, komplexen und dynamischen Geschäftsprozessen". In: *GI Jahrestagung (1)*. 2010, pp. 213–218.

[178] B. Schilit, N. Adams, and R. Want. "Context-Aware Computing Applications". In: *1994 First Workshop on Mobile Computing Systems and Applications*. 1994.

[179] A. Schmidt, M. Beigl, and H.-W. Gellersen. "There is more to context than location". In: *Computers & Graphics* 23.6 (1999), pp. 893–901.

[180] M. Schrefl, B. Neumayr, and M. Stumptner. "The Decision-Scope Approach to Specialization of Business Rules: Application in Business Process Modeling and Data Warehousing". In: *Ninth Asia-Pacific Conference on Conceptual Modelling, APCCM 2013*. 2013, pp. 3–18.

[181] M. Schrefl and M. Stumptner. "Behavior-consistent Specialization of Object Life Cycles". In: *ACM Trans. Softw. Eng. Methodol.* 11.1 (2002), pp. 92–148.

[182] Semafora Systems GmbH. *ObjectLogic Tutorial*. Semafora Systems GmbH. 2012. URL: `http://www.semafora-systems.com/fileadmin/user_upload/Publications_EN/ObjectLogic_Tutorial.pdf`.

[183] L. Serafini and M. Homola. "Contextualized knowledge repositories for the Semantic Web". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 12-13.0 (2012). Reasoning with context in the Semantic Web, pp. 64–87.

[184] H. A. Simon. *The Sciences of the Artificial (3rd Ed.)* MIT Press, 1996.

[185] A. V. Smirnov and K. Sandkuhl. "Context-Oriented Knowledge Management for Decision Support in Business Networks : Modern Requirements and Challenges". In: *Joint Proceedings of the BIR 2015 Workshops and Doctoral Consortium co-located with 14th International Conference on Perspectives in Business Informatics Research (BIR 2015), Tartu, Estonia, August 26-28, 2015. :* vol. 1420. CEUR Workshop Proceedings 1420. 2015, pp. 9–23.

[186] K. Smit, M. Zoet, and B. Matthijs. "Functional Requirements for Business Rules Management Systems". In: *Proceedings of the 23rd Americas Conference on Information Systems (AMCIS)*. 2017.

[187] M. Sordo, P. Tokachichu, C. J. Vitale, S. M. Maviglia, and R. A. Rocha. "Modeling Contextual Knowledge for Clinical Decision Support". In: *AMIA ... Annual Symposium proceedings. AMIA Symposium*. 2017, pp. 1617–1624.

[188] D. Steiner, B. Neumayr, and M. Schrefl. "Judgement and Analysis Rules for Ontology-driven Comparative Data Analysis in Data Warehouses". In: *11th Asia-Pacific Conference on Conceptual Modelling (APCCM 2015)*. 2015, pp. 71–80.

[189] D. Steiner, F. Burgstaller, E. Gringinger, M. Schrefl, and I. Kovacic. "In-flight Provisioning and Distribution of ATM Information". In: *Proceedings of the 30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016)*. 2016.

[190] D. Steiner, I. Kovacic, F. Burgstaller, M. Schrefl, T. Friesacher, and E. Gringinger. "Semantic enrichment of DNOTAMs to reduce information overload in pilot briefings". In: *2016 Integrated Communications Navigation and Surveillance (ICNS)*. 2016, 6B2–1–6B2–13.

[191] T. Strang and C. L. Popien. "A Context Modeling Survey". In: *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*. 2004.

[192] C. Strasser and A. G. Antonelli. *Non-monotonic Logic*. Center for the Study of Language and Information (CSLI), Stanford University. 2016.

[193] G. Witt. *Writing Effective Business Rules*. Ed. by G. Witt. Boston: Morgan Kaufmann, 2012, pp. 305–322.

[194] A. Taivalsaari. "On the Notion of Inheritance". In: *ACM Comput. Surv.* 28.3 (1996), pp. 438–479.

[195] H. Takatsuka, S. Saiki, S. Matsumoto, and M. Namamura. "RuCAS". In: *International Journal of Software Innovation* 3.3 (2015), pp. 57–68.

[196] The JBoss Drools Team. *Drools Documentation Version 7.6.0. Final*. jboss.org. 2018. URL: `https://docs.jboss.org/drools/release/7.6.0.Final/drools-docs/html_single/`.

[197] V. Vaishnavi and B. Kuechler. *Design Science Research in Information Systems*. DESRIST. 2015.

[198] V. Vieira, P. Brézillon, A. C. Salgado, and P. Tedesco. "Towards a generic contextual elements model to support context management". In: *Proceedings of the 4th International Workshop on Modeling and Reasoning Context (MRC2007)*. 2007, p. 49.

[199] R. D. Virgilio and R. Torlone. “A Framework for the Management of Context Data in Adaptive Web Information Systems”. In: *2008 Eighth International Conference on Web Engineering*. 2008.

[200] Visual Paradigm. *What is Case Managment Model and Notation (CMMN)*. Visual Paradigm. 2018. URL: `https://www.visual-paradigm.com/guide/cmmn/what-is-cmmn/`.

[201] B. Von Halle and L. Goldberg, eds. *The Business Rule Revolution: Running Business the Right Way*. Happy About, 2006.

[202] H. Wang, R. Mehta, L. Chung, S. Supakkul, and L. Huang. “Rule-based context-aware adaptation: a goal-oriented approach”. In: *International Journal of Pervasive Computing and Communications* 8.3 (2012), pp. 279–299.

[203] P. Wegner and S. B. Zdonik. “Inheritance as an Incremental Modification Mechanism or What Like Is and Isn’t Like”. In: *ECOOP ’88 European Conference on Object-Oriented Programming*. 1988, pp. 55–77.

[204] H. Weigand and A. Paschke. “The Pragmatic Web: Putting Rules in Context”. In: *Rules on the Web: Research and Applications*. 2012, pp. 182–192.

[205] M. Wimmer et al. “Surveying Rule Inheritance in Model-to-Model Transformation Languages”. In: *Journal of Object Technology* 11.2 (2012), 3: 1–46.

[206] W. Xue, H. K. Pung, and S. Sen. “Managing context data for diverse operating spaces”. In: *Pervasive and Mobile Computing* 9.1 (2013), pp. 57–75.

[207] Z. Zainol and K. Nakata. “Generic context ontology modelling: A review and framework”. In: *2010 2nd International Conference on Computer Technology and Development*. 2010.

[208] M. M. Zoet. “Methods and Concepts for Business Rules Management”. PhD thesis. Utrecht Unisersity, 2014.

[209] M. Zoet, J. Versendaal, P. Ravesteyn, and R. Welke. “Alignment of Business Process Management and Business Rules”. In: *ECIS 2011 Proceedings* (2011), pp. 1–12.

Glossary

AI	Artificial Intelligence.
AIM	Aeronautical Information Management.
AIXM	Aeronautical Information Exchange Model.
BMM	Business Motivation Model.
BPM	Business Process Management.
BPMN	Business Process Model and Notation.
BRE	Business Rule Engine.
BRG	Business Rule Group.
BRM	Business Rule Management.
BRMS	Business Rule Management System.
CASA	Context-Aware System or Application.
CBR	Contextualized Business Rule Repository.
CDT	Context Dimension Tree.
CKR	Contextualized Knowledge Repository.
CMMN	Case Management Model and Notation.
DMN	Decision Model and Notation.
ECA	Event-Condition-Action.
ECBR	Extended CBR with Rule Modules.
IFR	Instrumental Flight Rules.
IMC	Instrumental Meteorological Conditions.
MDD	Model-Driven Development.
NCBR	Non-contextualized Business Rule Repository.
NOTAM	Notice to Airmen.
OMG	Object Management Group.
OWL	Web Ontology Language.
PRR	Production Rule Representation.
QoC	Quality of Context.
RDF	Resource Description Framework.
RIF	Rule Interchange Format.
RMI	Rule Module Inheritance with Modification Restrictions.
RMM	Rule Maturity Model.

RuleML	Rule Markup Language.
SBVR	Semantics of Business Vocabulary and Business Rules.
SemNOTAM	SemanticNOTAMs: Ontology-based Representation and Semantic Querying of Digital Notices to Airmen.
SPARQL	SPARQL Protocol and RDF Query Language.
SPIN	SPARQL Inferencing Notation.
SQL	Structured Query Language.
SWRL	Semantic Web Rule Language.
UI	User Interface.
UML	Unified Modeling Language.
USTC	US Tech-Company.
VFR	Visual Flight Rules.
VMC	Visual Meteorological Conditions.
XML	Extensible Markup Language.

List of Figures

List of Tables

List of Listings

Appendix A

Experiments for Contextualized Database-backed Rule Execution

TABLE A.1: Performance comparison between contextualized and non-contextualized database-backed rule execution (part I).

Figure	Data-related Parameters	Situation-related Parameters	Concrete Contexts	Cases per Concrete Context	Cases	Properties	Non-contextualized Total Time (ms)	Contextualized Total Time (ms)	Non-contextualized Time per Case (ms)	Contextualized Time per Case (ms)	Speed-up Factor
Figure 4.10	1	1	4	9	36	14	8,699	6,850	241.64	190.28	1.27
	3	3	4	9	36	14	10,028	7,549	278.56	209.69	1.33
	5	5	4	9	36	14	9,062	6,923	251.72	192.31	1.31
	6	6	4	9	36	14	11,875	7,411	329.86	205.86	1.60
	7	7	4	9	36	14	84,126	7,973	2,336.83	221.47	10.55
Figure 4.9	4	3	4	100	400	14	38,873	19,287	97.18	48.22	2.02
	4	3	4	400	1600	14	92,457	27,910	57.79	17.44	3.31
	4	3	4	2,500	10,000	14	383,060	44,857	38.31	4.49	8.54
	4	3	4	10,000	40,000	14	1,332,883	79,269	33.32	1.98	16.81
	4	3	4	40,000	160,000	14	∞	204,932	∞	1.28	∞
	4	3	4	90,000	360,000	14	∞	405,636	∞	1.13	∞
Figure 4.7	1	1	1	100	100	14	6,171	5,267	61.71	52.67	1.17
	1	1	4	100	400	14	2,3295	18,464	58.24	46.16	1.26
	1	1	16	100	1,600	14	112,587	73,470	70.37	45.92	1.53
	1	1	64	100	6,400	14	570,108	303,113	89.08	47.36	1.88
	1	1	256	100	25,600	14	∞	1,299,813	∞	50.77	∞

TABLE A.2: Performance comparison between contextualized and non-contextualized database-backed rule execution (part II).

Figure	**Data-related Parameters**	**Situation-related Parameters**	**Concrete Contexts**	**Cases per Concrete Context**	**Cases**	**Properties**	**Non-contextualized Total Time (ms)**	**Contextualized Total Time (ms)**	**Non-contextualized Time per Case (ms)**	**Contextualized Time per Case (ms)**	**Speed-up Factor**
Figure 4.8	1	1	1	10,000	10,000	14	52,279	20,596	5.23	2.06	2.54
	1	1	4	10,000	40,000	14	337,781	78,912	8.44	1.97	4.28
	1	1	16	10,000	160,000	14	∞	327,682	∞	2.05	∞
	1	1	64	10,000	640,000	14	∞	1,319,027	∞	2.06	∞
	1	1	256	10,000	2,560,000	14	∞	5,805,779	∞	2.27	∞
Figure 4.6	4	3	4	100	400	14	38,072	19,571	95.18	48.93	1.95
	4	3	16	100	1,600	14	182,281	77,822	113.93	48.64	2.34
	4	3	49	100	4,900	14	883,411	245,779	180.29	50.16	3.59
	4	3	100	100	10,000	14	∞	510,849	∞	51.08	∞
	4	3	400	100	40,000	14	∞	2,195,417	∞	54.89	∞

TABLE A.3: Experiments for analyzing the effect of number of properties on the performance of contextualized database-backed rule execution.

Data-related Parameters	Situation-related Parameters	Contexts	Cases per Context	Cases	Properties	Contextualized Total Time (ms)	Contextualized Time per Case (ms)
1	1	1	1	1	4	845	845.00
1	1	1	100	100	4	1901	19.01
1	1	1	625	625	4	3067	4.91
1	1	1	2500	2500	4	5367	2.15
1	1	1	10000	10000	4	11640	1.16
1	1	1	40000	40000	4	34629	0.87
1	1	1	90000	90000	4	74484	0.83
1	1	1	160000	160000	4	121215	0.76
1	1	1	250000	250000	4	188213	0.75
1	1	1	1	1	14	2147	2147.00
1	1	1	100	100	14	5065	50.65
1	1	1	625	625	14	8400	13.44
1	1	1	2500	2500	14	11867	4.75
1	1	1	10000	10000	14	20999	2.10
1	1	1	40000	40000	14	51885	1.30
1	1	1	90000	90000	14	105542	1.17
1	1	1	160000	160000	14	172545	1.08
1	1	1	250000	250000	14	261991	1.05
1	1	1	1	1	30	3522	3522.00
1	1	1	100	100	30	10207	102.07
1	1	1	625	625	30	16977	27.16
1	1	1	2500	2500	30	22439	8.98
1	1	1	10000	10000	30	35171	3.52
1	1	1	40000	40000	30	81254	2.03
1	1	1	90000	90000	30	156034	1.73
1	1	1	160000	160000	30	255500	1.60
1	1	1	250000	250000	30	384061	1.54

Appendix B

Curriculum Vitae

Felix Burgstaller, MSc
Museumstrasse 30
4020 Linz

+43 680 21 80 435
felix.burgstaller@gmx.net

Personal Information

Birthday	27th April 1989, in Garmisch-Partenkirchen, Germany
Nationality	Austria

Work Experience

University Assistant

Aug. 2014 - present

Johannes Kepler University Linz (JKU) - Data & Knowledge Engineering Inst.

- Research Areas: (Business) rules; (business) rule organization; rule (set) inheritance; information filtering, classification, provisioning within the aeronautical domain; logic programming; visualization of ontologies.
- Projects: SemNOTAM – semantic filtering and classification of Notices to Airmen; Pinpoll – online market research; and further projects.
- Teaching: VL and UE Data Mining (English), PR Data & Knowledge Engineering, Business Intelligence (Data Mining part) for university course "Angewandtes Wissensmanagement", UE Data & Knowledge Engineering.
- Advising: bachelor and master thesis.
- Member of the Study Commsion.

Lecturer for Master Study e-Commerce

Sep. 2018 - Jan. 2019

Fachhochschule Wiener Neustadt

Taught class Databases & Information Management.

Project Member

Mar. 2014 - Aug. 2014

JKU - Data & Knowledge Engineering Institute

Participation in the SemNOTAM project. Assignments for UE Data & Knowledge Engineering have been designed and bachelor theses advised.

Project Management Office / IT Engineer

Oct. 2010 - Aug. 2012

Atos

Tasks included project management office, Sharepoint administration, change request management, implementing SAP ABAP reports, developing SAP workflows, and contributing to the SAP basis team.

Tutor

Oct. 2010 - Jan. 2011

JKU

Grading exercises and providing help to students in class Software Development.

Gefreiter

Oct. 2008 - Apr. 2009

Bundesheer

National Service in the Austrian forces.

Several internships at Siemens IT Solutions and Services, Siemens Business Services GmbH, and Applied International Informatics AG.

2005 - 2010

Internships

Education

	DOCTORAL PROGRAM IN SOCIAL AND ECONOMIC SCIENCES.
Feb. 2015 - Jun. 2019	Johannes Kepler University Doctoral thesis at the Data & Knowledge Engineering Institute. Thesis Topic: Contextualized Business Rule Repositories
	BUSINESS INFORMATICS MSC (WITH DISTINCTION).
Mar. 2012 - Jul. 2014	Johannes Kepler University Focus on complex event processing, business intelligence, semantic technologies, machine learning, service engineering, and business engineering. Thesis: Implementation of a Bitemporal Complex Event Processor (BICEP) with H2 and a Compiler for its Language (BICEPL)
	BUSINESS INFORMATICS BSC (WITH DISTINCTION).
Oct. 2010 - Feb. 2012	Johannes Kepler University Linz Courses covered (data-)base management, logics and algorithms, system and deployment planning, project management, business management, accounting, software engineering, communications engineering, operating systems, and more. Thesis: Implementation of an Ebay Agent Using OXPath and a Lightweight Rule Engine Based on SQLite.
	MATURA (A-LEVELS, PASS WITH MERIT).
Sep. 2003 - Jun. 2008	Higher Technical and Vocational College for Organization and Software Engineering (Leonding) Various subjects regarding business management, databases, software development, accounting, and project management.

International Experience

	TALKS AT INTERNATIONAL CONFERENCES
2015 - 2019	Johannes Kepler University Linz Presenting peer-reviewed publications at scientific and aeronautical conferences (see section Publications & Talks).
	ERASMUS+ STAFF MOBILITY
Dec. 2018	University of Oxford Invited visit to discuss further collaborations, especially regarding my dissertation.
	ERASMUS+ STAFF MOBILITY
Jun. 2018	University of Oxford Invited visit to discuss integration of research efforts, in particular my dissertation, and deepening research collaboration.
	GRADUATE EXCHANGE STUDENT
Aug. 2012 - Mar. 2013	University of Nebraska Omaha Classes taken regarded business intelligence, eCommerce, research methods, communication technologies/architectures, SAP configuration, and bioinformatics.

Skills

Basic	C++, operating systems, Python
Moderate	Java, Relational DBs, Active DBs, semantic technologies, business rule engines, business process modeling, SQL, PL/SQL, XQuery, SAP, office products
Advanced	Logic programming (Vadalog, Datalog, F-Logic), LaTeX, R, machine learning, data mining, business rules

Soft

- Structured, goal-oriented working: scientific publications, project participation
- Logical, connected, rational, and critical thinking: discussions, scientific publications, research
- Presentation written and oral, public speaking: international conference, teaching, scientific publications, course material
- Research, collaboration, discussion: scientific publications, research exchange
- Time management: organizing projects, publications, dissertation

Publications & Talks

- Main author of six peer-reviewed conference papers (five presented)
- Main author of two peer-reviewed workshop papers (two presented)
- Contributing author of three peer-reviewed conference papers

Certificates

Oct. 2013	IBM Infosphere Warehouse Technical Professional v1
May 2007	Business English Certificate Vantage

Languages

German	Native language
English	CEFR C1/C2

Awards & Competitions

2017	ICNS Best Paper in Track 5 – Special Topics/Other
2016	OTM Academy Best Contribution Award
2015	ICNS Best Student Paper
2013	Peter Kiewit Student Entrepreneurial Award, Maverick Business Plan Competition - 2nd place, JKU Study Abroad Excellence Award

Interests

Family, sport climbing, running, race biking, swimming, natural sciences (physics, biology, chemistry), reading

Publication Details

F. Burgstaller, B. Neumayr, E. Sallinger, and M. Schrefl. “Rule Module Inheritance with Modification Restrictions”. International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE). In: On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Proceedings, Part II. Vol. 11230. Lecture Notes in Computer Science. Springer International, 2018, pp. 404–422.

F. Burgstaller, B. Neumayr, C. Schuetz, and M. Schrefl. “Modification Operations for Context-Aware Business Rule Management”. In: 21st IEEE International Enterprise Distributed Object Computing Conference, EDOC 2017. IEEE Computer Society, 2017, pp. 194–203.

F. Burgstaller, C. Schütz, B. Neumayr, and M. Schrefl. “Towards Contextualized Rule Repositories for the Semantic Web”. In: Joint Proceedings of the Web Stream Processing workshop (WSP 2017) and the 2nd International Workshop on Ontology Modularity, Contextuality, and Evolution (WOMoCoE 2017) colocated with 16th International Semantic Web Conference (ISWC 2017). CEUR-WS.org, 2017, pp. 98–109.

F. Burgstaller, M. Stabauer, R. Morgan, and G. Grossmann. “Towards Customised Visualisation of Ontologies: State of the Art and Future Applications for Online Polls Analysis”. In: Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017. ACM, 2017, 26:1–26:10.

I. Kovacic, D. Steiner, C. Schuetz, B. Neumayr, F. Burgstaller, M. Schrefl, and S. Wilson. “Ontology-based data description and discovery in a SWIM environment”. In: 2017 Integrated Communications, Navigation and Surveillance Conference (ICNS). Best Paper in Track 5 – Special Topics/Other. IEEE, 2017, 5A4–1—5A4–13.

F. Burgstaller. “Employing Aviation-Specific Contexts for Business Rules and Business Vocabularies Management in SemNOTAM”. OnTheMove Academy (OTMA). In: On the Move to Meaningful Internet Systems: OTM 2016 Workshops - Confederated International Workshops: EI2N, FBM, ICSP, Meta4eS, and OTMA 2016, Revised Selected Papers. Vol. 10034. Lecture Notes in Computer Science. Best Contribution Award. Springer International, 2016, pp. 315–325.

F. Burgstaller, D. Steiner, B. Neumayr, M. Schrefl, and E. Gringinger. “Using a model-driven, knowledge-based approach to cope with complexity in filtering of notices to airmen”. In: Proceedings of the Australasian Computer Science Week Multiconference. ACM, 2016, 46:1–46:10.

F. Burgstaller, D. Steiner, and M. Schrefl. “Modeling Context for Business Rule Management”. In: 18th IEEE Conference on Business Informatics, CBI 2016, Volume 1 - Conference Papers. IEEE, 2016, pp. 262–271.

D. Steiner, F. Burgstaller, E. Gringinger, M. Schrefl, and I. Kovacic. “In-flight Provisioning and Distribution of ATM Information”. In: Proceedings of the 30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016). 2016.

D. Steiner, I. Kovacic, F. Burgstaller, M. Schrefl, T. Friesacher, and E. Gringinger. “Semantic enrichment of DNOTAMs to reduce information overload in pilot briefings”. In: 2016 Integrated Communications Navigation and Surveillance (ICNS). IEEE, 2016, 6B2–1—6B2–13.

F. Burgstaller, D. Steiner, M. Schrefl, E. Gringinger, S. Wilson, and S. van der Stricht. “AIRM-based, fine-grained semantic filtering of notices to airmen”. In: 2015 Integrated Communication, Navigation and Surveillance Conference (ICNS). Best Student Paper. IEEE, 2015, pp. D3–1—D3–13.