# The Decision-Scope Approach to Specialization of Business Rules: Application in Business Process Modeling and Data Warehousing

**Michael Schrefl**[1] **Bernd Neumayr**[1] **Markus Stumptner**[2]

[1]Department of Business Informatics
Data & Knowledge Engineering
Johannes Kepler Univ. of Linz - Austria
Email: michael.schrefl@jku.at

[2]Advanced Computing Research Centre
Knowledge & Software Eng. Lab
University of South Australia
Email: mst@cs.unisa.edu.au

## Abstract

It is common in organizational contexts and in law to apply a decision-scope approach to decision making. Higher organization levels set a decision scope within which lower organization levels may operate. In case of conflict the regulations and rules of a higher level (e.g. European Union) take precedence over those of a lower level (e.g. a member state). This approach can also be beneficially applied to the specialization of the most important kind of business rules in information systems, action rules. Such rules define under which conditions certain actions may, must not, or need to be taken.

Applying the decision scope approach to business process modeling based on BPMN means that business rules should not be buried in decision tasks but be made explicit at the flow level. This requires a rethinking of the current BPMN modeling paradigm in that several aspects in conditional flow so far modeled jointly are separately captured: (a) potential ordering of tasks, (b) conditions under which a task may or may not be invoked, and (c) conditions under which a particular task needs to be invoked. Rather than re-defining BPMN as such, an appropriate extension may be provided on-top of BPMN and mapped to standard BPMN primitives.

Applying the decision scope approach to active data warehousing, where analysis rules express actionable knowledge, requires to consider two alternative hierarchies: (1) hierarchies of sets of points at the same granularity and (2) the roll-up hierarchy of points in multi-dimensional space.

This paper presents the decision scope approach, outlines how it complements inheritance and specialization approaches typically followed in object-oriented systems, and introduces consistency rules for business rule specialization as well as auto-correction rules, which rectify an inconsistent lower-level business rule such that it becomes consistent with higher-level business rules.

*Keywords:* Specialization, Inheritance, BPMN, Business Processes, Business Rules, Data Warehousing

This work is supported by the Austrian Ministry of Transport, Innovation, and Technology in program FIT-IT Semantic Systems and Services under grant FFG-829594 *Semantic Cockpit*: An Ontology-Driven, Interactive Business Intelligence Tool for Comparative Data Analysis.



## 1 Introduction

Specialization is one of the most prominent techniques in conceptual modeling to deal with complexity. It assists structuring design artifacts, making them more compact and better comprehensible. Artifacts are designed by specifying their features. With specialization, more specific artifacts inherit features from more general artifacts and may specialize inherited features through extension (i.e., adding orthogonal features) or refinement (i.e., adding details to inherited features); changing inherited features is termed "overriding". Specialization is used in a number of related areas, such as knowledge representation [11], programming languages [9], data modeling [36], and business process modeling [12].

While the idea of organizing artifacts (nodes in semantic networks or concepts in ontologies, types in programming languages, classes of objects in data modeling, and classes of business cases in business process modeling) in hierarchies is common across related areas, the particular notions and rule governing specializations differ, sometimes even within the same area. A "rule-free" (i.e., arbitrary and unconstrained) change of inherited features, which is possible in some programming languages by free code overriding, is in disaccord with the very ideas of conceptual modeling and of abstracting common features along specialization hierarchies of design artifacts.

### 1.1 Contravariance and Covariance

It is generally agreed that propositions that hold for members of a superclass[1] should also hold for members of a subclass, unless stated otherwise (in case of non-monotonic inheritance). The key differences between various approaches concern (1) what particular propositions over members of a class are made by a class specification and (2) the rules that govern specialization hierarchies (e.g., monotonic or non-monotonic inheritance).

In data modeling, classes usually specify constraints over properties of their members which are interpreted as invariant conditions that need to be satisfied by members of a class over their life time. In knowledge representation, classes (frequently called concepts) specify necessary and sufficient conditions for an object (frequently called individual) to be considered a member of a class; a change of an object's property may lead to a dynamic re-classification.

In programming languages, the notion of type substitutability is predominant [14, 28]: Any member

[1]Irrespective of the specific artifact, we will speak in the following of classes and objects, of members of classes, and of super- and subclasses for specialization relationships between classes.

(frequently called instance) of a class should be usable in every place in which a member of superclass is expected. Regarding the methods specified with a class, type substitutability comes with the following proposition: "Given any member of the class, if the pre-condition of a method is met, the method can be successfully performed on that member". Notice that the member of the class may also be a member of a subclass whose overridden method is not considered in this proposition. Type substitutability (in the presence of local[2] static type checking) requires contra-variant inheritance - the redefined type of a parameter of an inherited method needs to be the same as or a supertype of the inherited type and a redefined precondition of a method must be more general (i.e., be implied by the inherited precondition). This ensures that if the precondition of a method is met at a superclass it is also met at the subclass.

Semantic data and process models have since their earliest appearance promoted the idea of co-variance as the more natural approach to specialization [10]; the "real world" argument for co-variant specialization [16] has been adopted also by UML 2.0 [13]. Regarding the methods of a class (or an activity of a process), co-variant specialization comes with the following proposition: "Given any member of the class, if the pre-condition of a method is *not* met, the method *cannot* be sucessfully performed on that member". *Notice the difference in the propositions implied by class specifications adhering to contra- or co-variant specialization, it is seldom expressed ad verbis.*[3]

Both notions, contra- and co-variance, have their place; usually for different purposes, substitutability and incremental specification. It has been suggested to combine both notions by selecting at run time a more general method for an instance of a more specific class if dynamic types of input parameters do not match the static types of input parameters of an overridden method [15]. Such approaches may have their merit in special cases, but lead to inconsistencies if specific constraints of a more specific class need to be met (which a more general method would not be aware of).

Specialization in business process modeling has been addressed by consistency notions that are akin to co-variance and contra-variance, such as observation vs. invocation consistency of object life cycles (used for modeling business processes in an object-centric manner) [34], or projection vs. protocol inheritance [5, 40] of process nets. Invocation consistency and protocol inheritance support the idea of substitutability in that a process class may be seamlessly replaced by another process class without any effect of the embedding environment. Observation and projection inheritance support the idea of incremental specification through extension and refinement.

### 1.2 Specialization and Business Rules

In re-active systems, such as active databases, the concept of specialization has been extended to active rules (that initiate an activity based on events and conditions) and studied in depth with respect to type substitutability [6].

Constraints, pre- and postconditions of methods, and active rules capture business rules explicitly at the design level in a declarative form and do not bury them in hidden design artifacts (such as in internals of decision activities or in procedural method specifications). At the requirement level, business rules (which are defined as any "rule under business jurisdiction" by OMG) should, according to the Business Rule Manifesto [1], be specified in declarative from as provided by the SBVR-specification [2] for documenting the semantics of business vocabularies, business facts, and business rules. Scientific work has classified business rules into static business rules expressing invariant conditions over facts and their relationships and into dynamic business rules expressing rules that concern state changes [21].

Information system design needs to address the choice on how to best represent business rules in design artifacts. It is nowadays commonly accepted that business rules should not be hidden, but be explicitly captured in a declarative form[1]. We have argued in the past [25] that depending on the kind of business rule, different choices should be made on how to capture business rules in process modeling, such as by using constraints, by pre- or post-conditions of activities (defining enabling conditions and results, resp.), or by using activation conditions (defining under which conditions an activity is triggered). We have also introduced an object-centered design approach for jointly modeling object structure and business processes [24], an approach which currently receives considerable attention in conference panel discussions and industry promotion under the labels "artifact-centered business process modeling" [22] and "case base management process modeling" (CMPM). And we have separated several concerns in business process modeling that are frequently mixed: (1) potential ordering of activities, (2) passive pre- and post-conditions of activities (specified independently of the invocation of activities by pre- and post-states in a net representation or by expressions in some form of predicate logic over business cases and activity parameters, or by both), and (3) activation rules that define under which conditions (including events) an activity is invoked [27]. Such a separation is in our opinion a pre-requisite to appropriately handle specialization of business rules in the context of business processes modeling and data ware housing.

The issue of specialization of business rules has been briefly discussed [8, 26, 37], but to the best of our knowledge not been extensively addressed so far in a comprehensive, uniform manner. At the requirement elicitation level, the SBVR-specification of several hundred pages comes without a comprehensive description of criteria for the specialization of business rules. At the design level, alternative approaches - as described above - have been promoted.

### 1.3 Business rules and decision scope

To move forward, we look for guidance by considering on how law is "specialized" along law hierarchies (e.g. constitutional law, simple law) or, similarly, on how rules are specialized along levels of jurisdictions (e.g., regulations at the level of the European Union, regulations at the level of a member state). In these contexts it is common to apply a *decision-scope approach* to decision making. Higher organization levels define what is permitted and what is forbidden. Different to deontic logic, permissions and prohibitions are not complementary but leave a decision-scope in which lower organization levels may operate. Regulations and rules of a higher level (e.g. European Union) take precedence over those of a lower level (e.g. a member state) in case of conflict. Such an approach can also be beneficially applied to the specialization of business rules. We describe this in the

[2] i.e., without global program code analysis

[3] The different semantics of pre- and postconditions of methods has been described from a different angle based on deontic logic, but without explicit reference to co- and contra-variance [41].

paper for the most important kind of dynamic business rules in information systems, action rules. Such rules define under which conditions certain actions may, must not, or need to be taken.

Applying the decision-scope approach to specialization reconciles the apparently conflicting specialization approaches of co- and contra-variance by following them separately in parallel. The propositions underpinning contra-variance inheritance represent permissions stating under which conditions a method may be invoked; the propositions underpinning co-variance represent prohibitions stating under which conditions (if a precondition is violated), a method cannot be invoked (i.e. the method invocation will fail).

The decision scope approach as described can be extended to activation conditions, which may trigger an activity if is permitted to do so. But not every activity that is permitted need to be actually invoked. There is a discretion to choose. This discretion is important in the context of conditional flow in business process modeling, where activation conditions may in effect chose any of several potentially applicable, i.e., permitted activities. In semi-formal decision making such a choice is deferred to the case officer. OMG promotes two opposite approaches to business process modeling, BPMN [35] where all flows and flow conditions are pre-defined and CMPM in which flow conditions and choices can be identified by a case manager on a case by case basis. The decision scope approach is applicable to both. For CMPM, it is required to consider different types of guards. For BPMN, multiple aspects in conditional flow so far modeled jointly need to be separately captured: (a) potential ordering of tasks, (b) conditions under which a task may or may not be invoked, and (c) conditions under which a particular task needs to be invoked. We will present such an extension to BPMN later in the paper and explain that rather than re-defining BPMN as such, an appropriate extension may be provided as well ontop of BPMN and mapped to standard BPMN primitives (based on event-based gateways and deferred decisions).

Another dimension of choice in inheritance is monotonicity. Specialization and the decision-scope approach are inherently monotonic as any potential conflict of rules is resolved by precedence of rules of higher organizational levels over those of lower organizational levels. To avoid repeated specification of the same rules at lower organization levels, apart from a few exceptions, non-monotonic rules are used in law systems as well. These are explicitly identified as such by stating that a rule is presumed for lower jurisdictions and may be overridden by them (e.g., world wide sales conditions state that warranty is for one year, unless local consumer law prescribes a longer warranty period). Approaches in which monotonic and non-monotonic rules coexist in a decision scope setting have been proposed in the context of authorization in database systems [7]: Authorization rules are classified into positive (corresponding to a permission) and negative (corresponding to a prohibition) rules and orthogonally thereto into strong (monotonic) and weak (non-monotonic) rules. Various rules govern the interplay of their combination in the context of inheritance along hierarchies of authorization objects, e.g., a weak positive authorization of a higher level may be canceled by a strong negative authorization rule at a lower level. This principle can be beneficially carried over to the specialization of business rules in general.

### 1.4 Decision Scope and Data Warehousing

The decision-scope approach is also relevant to active data warehousing [30, 38, 39], in which analysis rules express actionable knowledge, and to comparative data analysis in business intelligence [32], in which comparison results between two groups of facts (group of interest and group of comparison) may trigger actions providing guidance on how to proceed in analysis (guidance rules) or actions informing analysts about striking comparison results that may need further attention or action (judgment rules). We have applied some limited form of decision-scope approach in active data warehousing in the past and encountered the need to study specialization of analysis rules in the SemCockpit[32] project. In addition to business process modeling, two alternative hierarchies need to be considered: (1) hierarchies of sets of points at the same granularity and (2) the roll-up hierarchy of points in multi-dimensional space.

Specialization along subset relationships between sets of points at the same granularity is governed by the same rules as specialization between a superclass and a subclass of objects. Specialization along a roll-up hierarchy of points in multi-dimensional space actually concerns entities of different kinds, yet connected by some form of part-of relationship. This gives rise to two alternative rule evaluation strategies, both meaningful in practice, but with a different application space in mind.

In the *prerogative* strategy, an action triggered for a higher-level point implicitly implies the same action for each lower level point. E.g. if a company decides to abandon a product line (such as mobile phones) this decision is implied for every product (i.e., every phone model in our example) of this product line.

In the *presumed* strategy, an action (typically an informative one) triggered for a higher level point is presumed to cover also lower level points and is thus not carried out again for lower level points (to avoid unnecessary information overload, the basis of performing roll-up analysis in data warehousing), unless it is justified by a negative activation rule. E.g., if in comparative data analysis a judgment rule triggers an informative action that average treatment costs for patients with diabetes mellitus of type 2 patients in Austria are twice as high than in Germany last year, it is presumed that this will hold in general for each province in Austria. Minor deviations are generally not of interest in comparative data analysis, but major ones are. The knowledge what kind of exceptions should be reported (a kind of business rule) can be expressed by a negative activation rule according to the notion of negative rules described before. Such a negative activation rule may trigger an informative action reporting a notable exception if the difference in compared costs is less than 20% for comparing patients in a province of Austria with patients in Germany but a positive activation rule has led to report at a higher granularity a striking difference of average treatment cost per patient in Austria versus Germany. E.g., based on such deviating observation for comparing Tyrol (a province of Austria) with Germany this rule will report that contrary to Austria as a whole, average treatment costs per patient have been similar when comparing Tyrol with Germany.

We discuss the decision scope approach to specialization of business rules from a general perspective in the remainder of this paper, which is organized as follows. In Section 2, we quickly revisit co- and contravariant specialization approaches in object-oriented systems and describe how the decision scope approach applied in organizational contexts reconciles both. In

Section 3, we show by detailed examples how the decision-scope approach can be applied in the realm of business process modeling and propose an extension of BPMN to this purpose. We introduce a set of consistency rules that govern the decision scope approach and introduce a set of auto-correct-rules to modify inconsistent rules such that they become consistent. Continuing the law analogy, auto-correction amounts to repealing a simple law by a constitutional court if it contradicts constitutional law. In Section 4, we extend the decision-scope approach to data warehousing. Section 5 concludes the paper with a summary and outlook.

## 2 Inheritance: A Quick Review and an Introduction to the Decision Scope Approach

We quickly review the notions of co- and contravariant inheritance in conceptual modeling and object-oriented systems, introduce the decision scope approach to specialization as observed in law and organizational context, and show how co- and contravariant inheritance can be reconciled in the decision-scope approach by following both notions independently in parallel.

### 2.1 Inheritance in Conceptual Modeling and Object-Oriented Programming

The notion of specialization according to co-variance has been promoted by semantic data and process models as a natural approach to reflect how sets and subsets of objects of the real world relate. It supports information systems design by specializing a more general class through extension, i.e. by adding new properties (attributes or relationships), and refinement, i.e., by providing more detail for inherited properties or by restricting them. Usually, classes specify invariant conditions over properties of its members. These are treated as constraints that need to be satisfied by its members over their life time. Inherited invariant conditions may be strengthened at subclasses. Any member of a subclass is considered to be also a member of the superclass such that if inherited constraints (invariant conditions) are strengthened at subclasses a member of a subclass will also satisfy the more general constraints of its superclasses. In case of relationships between members of classes, a subclass of one side of the relationship may restrict the inherited relationship to be to a subclass of the original other side of the inherited relationship. Considering relationships does not add to the nature of the problem itself, we restrict our discussion to attributes of object classes with simple types (such as Integer, String or restrictions of these) from here on.

UML 2.0 [18] has also adopted co-variant specialization and we will use it to present our examples.

**Example 1.** We consider a very abbreviated description of selected parts of a loan application process in this paper. Figure 1 depicts a class hierarchy of loan applications in UML notation. Loans are described by an amount (a), a credit worthiness (w)[4], and a monthly rate (r). Private loan applications are further described by the attachable income (i) that is of relevance for salary seizures should the private loan not be repaid. Mortgage loans are further described by a mortgage lending limit (l) of the mortgaged property that is of relevance if the mortgage is to be redeemed should the loan not be repaid.

[4] Actually, the credit worthiness relative to the loan application is a function of available disposable income and requested loan amount; we abstract from such details here.

Invariants specify that amounts of loans are between 10 and 900k $, amounts of private loans between 10 and 70k $, and amounts of mortgage loans between 30 and 900k $.

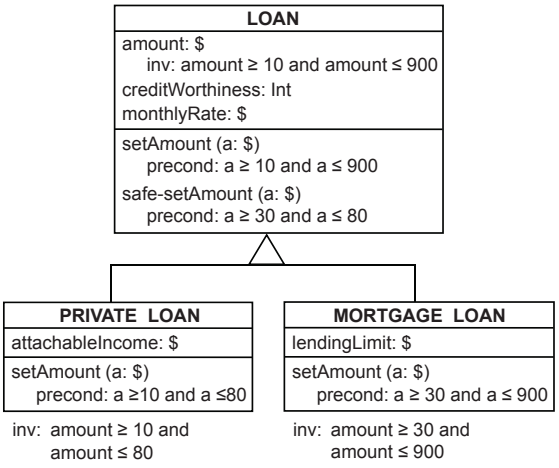


Figure 1: Class Hierarchy of Loans

Strengthening of constraints over inherited object properties, either by restricting their type to a subtype (subclass) or by strengthening explicitly stated invariant conditions, require co-variant inheritance for "mutators", i.e., methods that update the values of attributes. The domain of input parameters of inherited methods must be accordingly restricted to a subtype of the updated inherited property and preconditions of inherited methods must be accordingly strengthened to reflect to stronger invariant condition at the subclass. This applies also to types of return values of methods and their postconditions. We repeat what we already mentioned in the introduction that a precondition of a method is to be interpreted as a proposition that "Given a member of the class, if the precondition of a method is not met, the methods cannot be performed on that member".

**Example 2.** Method setAmount(a:$) of class LOAN, which sets the amount of a loan application to the value supplied as parameter, has a precondition coinciding with the corresponding invariant. Notice that in general preconditions of methods may be different to re-stating relevant invariants. In fact it should be unnecessary to include relevant invariant conditions in pre-conditions as these kinds of preconditions should ideally be inferred (at least for setter- and getter-methods). The precondition of method setAmount is overridden at subclasses PRIVATE_LOAN and MORTGAGE_LOAN to reflect the invariants of these classes accordingly. The specialization applied complies with co-variance as explained earlier.

Class hierarchies designed according to co-variance in conceptual data modeling match class hierarchies that would be built by ontology reasoners based on class descriptions that contain no reference to superclasses but contain explicit descriptions of otherwise inherited attributes.[5] The main difference is that in conceptual modeling the class hierarchy is the primary design artifact along which attribute descriptions are inherited by subclasses from superclasses,

[5] We assume here that concepts in the ontology refer to a common set of objects and that class descriptions are interpreted as concept expression stating that the concept comprises all objects of the given object class that possess the attributes of the class and satisfy the invariant conditions of the class.

whereas in current ontology engineering practice, the relationships of classes (concepts) to their attributes, but without reference to a superclass, are the primary design artifact and subclass relationships are inferred through subsumption reasoning.

There are - depending on the ontology and conceptual model used - other differences between ontologies and conceptual data modeling, but these are not relevant for the point we wish to make here that class hierarchies and corresponding concept hierarchies align one-to-one. While originally, ontologies were envisioned as an explicit formal specification of a conceptual model [19, 20], especially since the inception of Semantic Web research, ontology languages frequently take on the open world assumption, while conceptual modeling mostly adheres to the closed world assumption. Also, ontology languages treat conditions over attributes as conditions for classifying objects whereas data models typically treat conditions over attributes as constraints that may not be violated in updating a member of a class. But this is not a universal assumption, as since the early days of conceptual modeling primitive classes have been complemented by derived classes whose members are dynamically re-classified if they no longer meet the derivation condition of the derived class. When one takes into account the full history of conceptual modeling, many claimed differences between conceptual models and ontology languages become blurred, especially since within ontology research basic distinctions of approach such as the ability to infer a subsumption (inheritance) hierarchy by reasoning or to define it by specification depend on the language used (e.g., [4, 3]).

While conceptual modeling and ontology engineering would be expected[6] to come up with generally the same hierarchy, but use different paths to achieve the same result, this "common" class hierarchy will usually not constitute a class hierarchy that satisfies type substitutability as demanded by the contra-variant approach to inheritance promoted in object-oriented programming. Contra-variant inheritance requires to generalize the types of input parameters of inherited methods to super types and to generalize inherited preconditions of methods in overriding. This approach ensures that any method that can successfully be applied on some instance of a class can also be successfully applied on any instance of subclass of the former. We repeat what we have already stated in the introduction: a precondition of a method is to be interpreted as proposition "Given a member of the class, if the precondition of the method is met, the method can be successfully performed on that member."

**Example 3.** The class hierarchy of loans in Figure 1 violates substitutability. However, method safe-SetAmount(a:$) of class LOAN with precondition $a \geq 30$ and $a \leq 90$ is "type-safe". It can be applied on any member of class LOAN, be it a mortgage loan or a private loan.

### 2.2 The Decision-Scope Approach in Law: Constitutional Law before Simple Law

Law hierarchies and organizations use a decision-scope approach for decision making along hierarchies of law or organizational levels. Regulations and rules come (among others) in two typical forms, permission that allow actions and prohibitions that forbid actions. Permissions and prohibitions are not complementary. There are actions that are on the one hand not explicitly permitted and on the other hand not explicitly forbidden. This leaves room for maneuver in decision making at lower organizational levels. We also observe that legal cases usually are related to a most specific jurisdiction and in case of potential multiple jurisdictions there are tie-brakers to decide upon in which concrete jurisdiction a legal case is to be handled. Similarly, we assume the abstract superclass rule, which has been identified as good design practice, is applied in designing class hierarchies [23]: Each object belongs to a single, most-specific class that is concrete, i.e., has no subclasses.

The decision scope approach is governed by a simple, intuitive set of consistency rules:

1. Permission and prohibitions are always disjoint and they are complementary at a concrete bottom-level organizational entity / law.
2. What is permitted at a higher level is also permitted at lower levels.
3. What is forbidden at a higher level is also forbidden at lower levels.

If we take this perspective and revisit the notions of co- and contra-variant inheritance, we can observe that the propositions made by contra-variant specialization correspond to permissions and the propositions made by co-variant specialization stretch over permissions and the decision scope, or if negated, correspond to prohibitions (i.e., to what is forbidden).

This observation allows us to superimpose both kinds of propositions into a single, dual-faceted class description. Behavior attributed to the permissive part of the class description is type safe in the traditional sense of type substitutability. Behavior attributed to the prohibitive part of the class descriptions can be identified as "illegal", i.e., such action cannot be performed on any member of the class. To decide whether behavior attributed to the decision scope is permitted or forbidden, one needs to consult lower level classes along the class hierarchy to the concrete class of an object. For concrete classes, the decision scope is closed, i.e. what is forbidden is not permitted and what is permitted is not forbidden. The dual facets of class descriptions of abstract classes become a single facet for concrete classes.

**Example 4.** The class hierarchy of loans in Figure 1 with methods setAmount and safe-SetAmount of class LOAN already represents such a dual-faceted class description. However, an integrated description would only represent a single method with a positive (reflecting permission) and a negative (reflecting prohibition) precondition. Similarly, in an integrated description the current invariant shown for class LOAN would be converted into a negative invariant condition $a < 10$ or $a > 900$ (which if met when updating a member will constitute an invalid update, irrespective to which specific class the member belongs to) and complemented by a positive invariant condition $a \geq 30$ and $\leq 90$ (which if satisfied when updating a member will constitute a valid update, irrespective to which specific class a member belongs to).

Figure 2 illustrates the decision scope approach and its relation to co- and contra-variant inheritance.

## 3 The Decision Scope Approach in Business Process Modeling

In this section we discuss how the decision scope approach can be applied to business process modeling.

[6] Notice that we do not claim a one-to-one correspondence but consider the global picture.

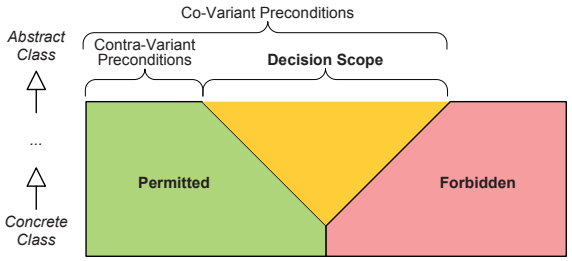


Figure 2: The basic Decision-Scope Approach

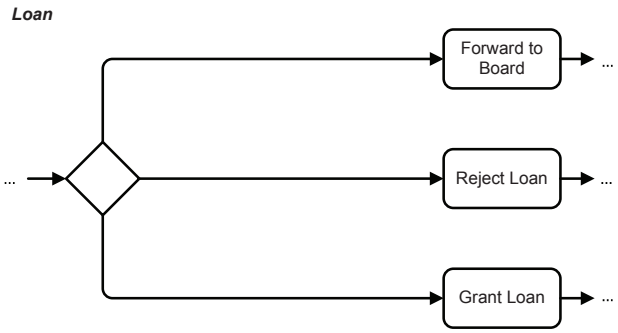


Figure 3: Fragment of Loan Application Process in BPMN

In particular we consider the, in our opinion, most important kind of dynamic business rules: action rules. They concern under which conditions certain actions (activities, tasks) may, must not, or need to be taken in the process flow.

### 3.1 Business Rules in Business Process Modeling

Business rules in business process modeling express constraints over properties of business cases, the potential order of activities, pre- and postconditions of activities, and activation conditions (which express under which conditions a particular activity is invoked). At the design level, constraints over properties of business cases are typically captured by the modeling primitives of conceptual data models and associated constraint language (e.g., by UML cardinality constraints or OCL); the potential order of activities is typically captured by various forms of process diagrams based on Petri nets, state charts, or similar formalisms. Conditions under which activties may, must not, or need to be taken are frequently not considered independently from process flow. Applying a decision scope approach to modeling business rules that govern the execution of business processes requires to consider these kinds of conditions independently of each other and of process flow. But before we look into the specialization of business rules, we revisit the modeling of business processes from a general perspective.

The order of activities of a business process is typically expressed by some form of process diagram, e.g., by BPMN. Of particular interest are decision points at which process flow splits into alternatives.

**Example 5.** Figure 3 depicts a fragment of a loan application process. Once the credit worthiness for a loan application has been determined (not shown), the loan application is granted ( g), rejected ( j), or forwarded for decision to the board ( f). Note: This is a simplified picture, the loan application process modeled may have other alterative paths at this decision point such as requesting additional information from the applicant if the case officer is not satisfied with the information at hand. Note: The loan application process modeled in full (in Petri-net notation) can be found in [26].

Activities are governed by business rules that define under which conditions an activity $a$ of a process class $O$ may be invoked ($P^a_o$), must not be invoked ($F^a_o$), or need to be invoked ($O^a_o$). To keep the explanation of the decision scope approach simple, we assume herein for simplicity that conditions are expressed over properties of a business case. However, the presented approach can be extended to conditions over parameters of activities and to events. Note that we use the terms permission ($P$), prohibition / forbidden ($F$), and obligation ($O$) from deontic logic; but different to deontic logic "permitted is not equivalent to not forbidden" and, conversely, "forbidden is not equivalent to not permitted". Likewise, we will later use disobliged ($D$) as "counterpart" to obliged ($O$), where both conditions are again not complementary.

**Example 6.** Figure 4 visualizes the business rules governing activity forward depending on the amount of the loan and the credit worthiness of the applicant with respect to the loan application. (Note that different to Figure 1 we assume here and in the following that all kinds of loans are between 0 and 80k $. This is to simplify the presentation and to be able to focus on key points). A loan application with a low credit rating must not be forwarded to the board (as it should be immediately rejected). A loan application for a low amount and with a very high credit rating must not be forwarded to the board either (as it should be immediately granted). A loan application for a high amount and a medium credit rating may (dependent on the discretion of the case officer) be forwarded to the board.

The white "open space" in Figure 4 constitutes the decision scope for activity forward for subclasses of class LOAN, which will be introduced later.

Note that we have used positive ($+$), negative ($-$), and activation ($\lightning$) conditions to specify permitted, forbidden, and obliged invocation of an activity. We intentionally use here a different terminology and notation ($+, -, \lightning$), for conditions indicated with activities as we will later auto-complete and auto-correct indicated conditions to permissions ($P$), prohibitions ($F$), and obligations ($O$).

Similarly, Figure 5 visualizes the business rules governing activity grant of class LOAN and Figure 6 the business rules for activity reject of class LOAN.

The business rules governing a single activity must be conflict-free (see: Figure 7), i.e., for the state of a business case (given by values of its properties): (1) an activity may not be permitted and forbidden as well, (2) an activity may not be obliged and disobliged as well, (3) if an activity is obliged it is permitted, and (4) if an activity is forbidden it is disobliged.

The business rules governing the set of alternative activities $A$ at a given decision point in a business process must be decision-consistent (Figure 8), i.e., for a state $\sigma$ of a business case at the decision point: (1) at least one activity must be permitted, and (2) at most one activity is obliged (which means it is permitted). - Note that for simplicity we consider only alternative (i.e., exclusive) choice here. Notice also that it is possible that several activities are permitted at the

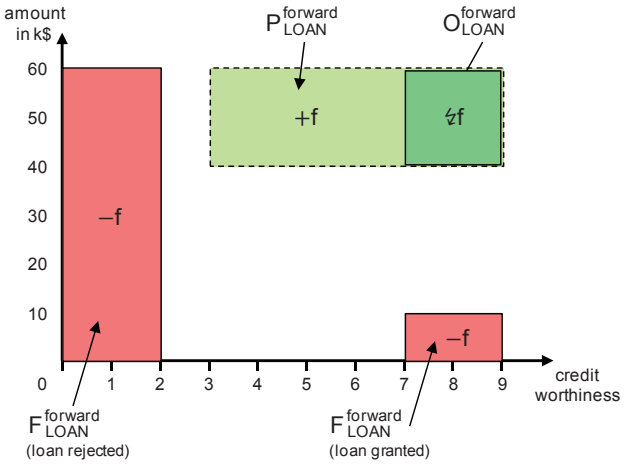


Figure 4: Business process LOAN: Business Rules Regarding forward

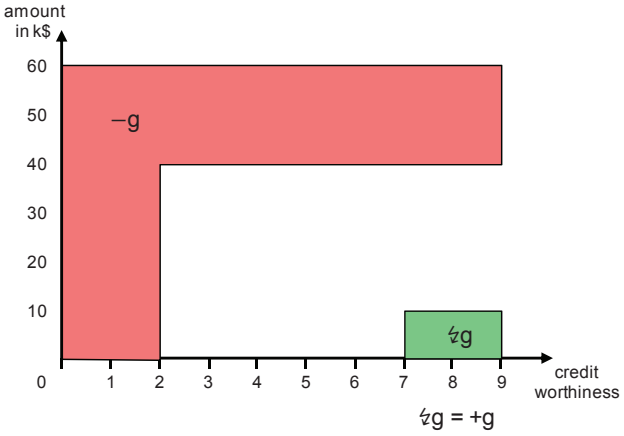


Figure 5: Business process LOAN: Business Rules Regarding grant

same time and it is possible that no activity is obliged for a given state of a particular instance of a business process (as determined by the values of its properties). Then it is at the discretion of a human process agent to decide which one of the permitted alternative activities to choose for a given process instance.

Considering the propositions we attached to permission and prohibition of an activity, i.e., that an activity may be executed for a business case if its permission is met and that an activity cannot be executed for a business case if its prohibition is met, we add two other consistency criteria such that these two propositions are correctly reflected in case of alternative activities: If an activity is obliged, alternative activities are forbidden (criterium no. 3 in Figure 8) and if an activity is permitted, alternative activities are disobliged (criterium no. 4 in Figure 8), which means they cannot be obliged at the class or some subclass. We note that criterium no. 2 is redundant. It follows from no. 4 in Figure 8 and no. 3 in Figure 7.

During business process design, adding an obligation to an activity may be restricted by the permissions defined with alternative activities and may require to extend the prohibitions specified with alternative activities such that criteria no. 3 and 4 of inter-activity consistency are met. To avoid such redundant specification, we introduce appropriate auto-completion and auto-correction rule in subsection 3.3.

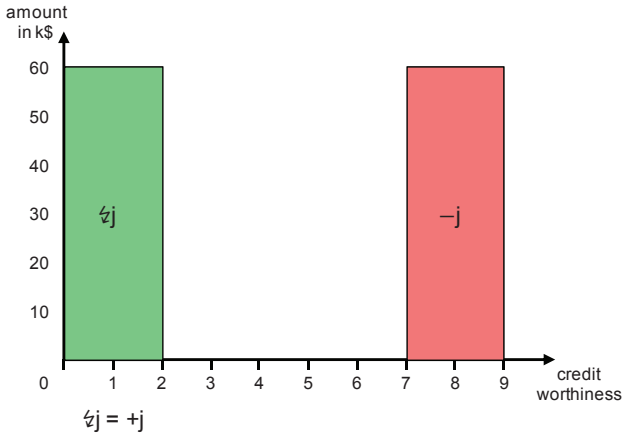


Figure 6: Business process LOAN: Business Rules Regarding reject

1. Permitted is not forbidden, and vice versa: $P_o^a(\sigma) \rightarrow \neg F_o^a(\sigma)$, $F_o^a(\sigma) \rightarrow \neg P_o^a(\sigma)$
2. Obliged is not disobliged, and vice versa: $O_o^a(\sigma) \rightarrow \neg D_o^a(\sigma)$, $D_o^a(\sigma) \rightarrow \neg O_o^a(\sigma)$
3. What is obliged is permitted: $O_o^a(\sigma) \rightarrow P_o^a(\sigma)$
4. What is forbidden is disobliged: $F_o^a(\sigma) \rightarrow D_o^a(\sigma)$

Figure 7: Criteria for Intra-Activity Consistency

**Example 7.** The business rules for activities forward (Figure 4), grant (Figure 5), and reject (Figure 6) of process class LOAN are intra-activity and inter-activity consistent.

Figure 9 shows the business rules for activities forward, grant, and reject for loans superimposed. Note that situations that can be inferred according to the consistency criteria are not specifically indicated: If an activity is obliged (shown) all other activities are forbidden (not shown), and if an activity is obliged (shown) it is permitted as well (not shown).

Business rules governing the permitted, forbidden, and obliged invocation of an activity in a business process should be explicitly represented at the design level rather than buried within activities. The latter approach would contravene the objective of conceptual design to make business rules explicit. Just as cardinality constraints, an important class of static business rules, are captured in UML graphically by

1. At least one activity permitted: $\exists a \in A : P_o^a(\sigma)$
2. At most one activity is obliged: $a \in A, b \in A, a \neq b : O_o^a(\sigma) \rightarrow D_o^b(\sigma)$
3. If an activity is obliged, alternative activities are forbidden: $a \in A, b \in A, a \neq b : O_o^a(\sigma) \rightarrow F_o^b(\sigma)$
4. If an activity is permitted, alternative activities are disobliged: $a \in A, b \in A, a \neq b : P_o^a(\sigma) \rightarrow D_o^b(\sigma)$

Figure 8: Criteria for Inter-Activity Consistency

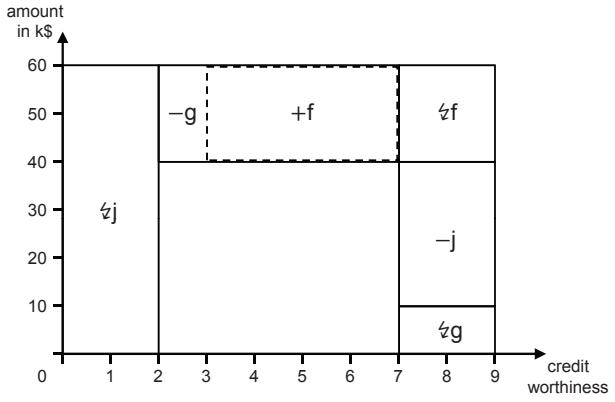


Figure 9: Business process LOAN. Business Rules Regarding grant, forward, and reject (superimposed)

cardinality indications associated with associations, action rules should likewise be explicitly captured next to the process flow in business process modeling notations such as BPMN or captured by different kinds of guards in case-based management process modeling (CMPM).

**Example 8.** Figure 10 indicates, as extension to BPMN, permitted, forbidden, and obliged invocation of an activity in process flow as positive (+), negative (−), and activation (↯) condition.

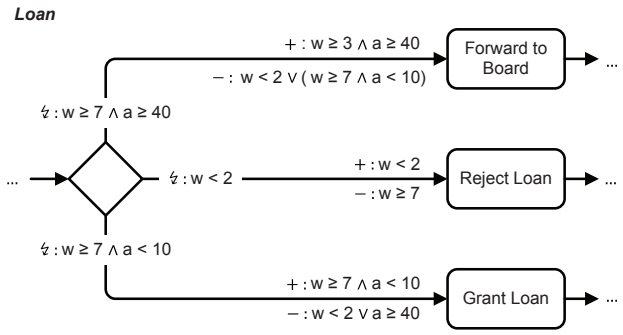


Figure 10: Business process LOAN in BPMN, extended with positive (+), negative (−), and activation conditions (↯)

Rather than requiring a change in BPMN, this proposed extension can be provided on top of BPMN and mapped to standard BPMN modeling primitives. E.g., an exclusive OR-gateway with positive, negative, and activation conditions on its outflow may be mapped to an event-based, deferred decision gateway, with a send-event at its inflow and different catch-events for each activity at its outflow. The decision procedure itself may be mapped to a decision process (e.g., following the "decision logic abstraction pattern" in rBPMN [29]) that is initiated by catching the send event associated with the inflow of the OR-gateway and completed by sending one of the events caught in the ouflows of the OR-gateway. For semi-automatic decision making, these events may be explicitly raised in all or some occasions by a human process agent.

### 3.2 Specialization of Business Rules

Following the decision scope approach, action rules of a process class $o$ may be specialized at a subclass $\hat{o}$ in that more specific rules at the subclass must act within the decision scope set by the superclass and providing room for manoeuvre at the subclass (see: Figure 11).

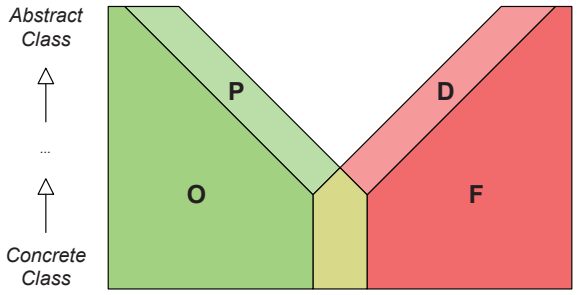


Figure 11: The Decision-Scope Approach

In particular, see Figure 12, (1) an action rule at the subclass must be more general than the same kind of rule at the superclass (generalization), (2) action rules at the subclass must not contradict those given at the superclass (super-subclass inter-activity conflict-freeness), and (3) (a) permissions and prohibitions for concrete classes are complementary, (b) an activity is obliged if all alternative activities are forbidden, (c) an activity is disobliged if some other activity is permitted, and (d) obligations and disobligations are complementary (closed decision scope for concrete classes). Note that for checking overall consistency it would be sufficient to check first for generalization and then for activity-consistency (see above); the criterium "super-subclass intra-activity conflict-freeness" additionally captures a core principle of the decision-scope approach that superclasses should dominate subclasses. This will be of particular relevance when we later introduce auto-correction rules. We also note that one of the three criteria (3b-3d) is redundant.

We now apply this perspective to our use case.

**Example 9.** Figures 13 and 14 visually indicate the business rules governing activities forward, grant, and reject for private loans and mortgage loans respectively. As before with Figure 9, if an activity is shown obliged, all other activities are forbidden (not shown), and if an activity is shown obliged it is permitted as well (not shown). Two diagrams are shown for private loans, one covering the case where the attachable income $i$ is less than the monthly rate $r$, the other one where it is higher (in which case its is much riskier to grant a loan as rates cannot be fully recovered by salary seizures). Similarly, two diagrams are shown for mortgage loans, one covering the case where the amount $a$ of the loan is at less than 70% of the mortgage lending limit $l$ of the property, the other one where it is higher (in which case its is riskier to grant the loan as the loan amount may not be fully redeemable by sale of the property). Note: The application process for loans and the application process for private loans specialize the more general application process for loans by adding activities and states. They have been modeled in a Petri-net notation in [26] and are governed by the consistency criteria for process specialization presented in [34].

One can easily verify by visual inspection that the rules for mortgage loans and the rules for private loans

1. Generalization
   (a) $P^a_{\hat{o}}(\sigma) \rightarrow P^a_o(\sigma)$
   (b) $F^a_{\hat{o}}(\sigma) \rightarrow F^a_o(\sigma)$
   (c) $O^a_{\hat{o}}(\sigma) \rightarrow O^a_o(\sigma)$
   (d) $D^a_{\hat{o}}(\sigma) \rightarrow D^a_o(\sigma)$
2. Intra-Activity Conflict-Freeness
   (a) $P^a_o(\sigma) \rightarrow \neg F^a_{\hat{o}}(\sigma)$
   (b) $F^a_o(\sigma) \rightarrow \neg P^a_{\hat{o}}(\sigma)$
   (c) $O^a_o(\sigma) \rightarrow \neg D^a_{\hat{o}}(\sigma)$
   (d) $D^a_o(\sigma) \rightarrow \neg O^a_{\hat{o}}(\sigma)$
3. Closed decision scope for concrete class $o$
   (a) $P^a_o(\sigma) = \neg F^a_o(\sigma)$
   (b) $O^a_o(\sigma) = \bigwedge_{b \in A, b \neq a} F^a_o(\sigma)$
   (c) $D^a_o(\sigma) = \bigvee_{b \in A, b \neq a} P^a_o(\sigma)$
   (d) $O^a_o(\sigma) = \neg D^a_o(\sigma)$

Figure 12: Criteria for Decision Scope Consistency: Subclass $o$, superclass $\hat{o}$, state of business case $\sigma$

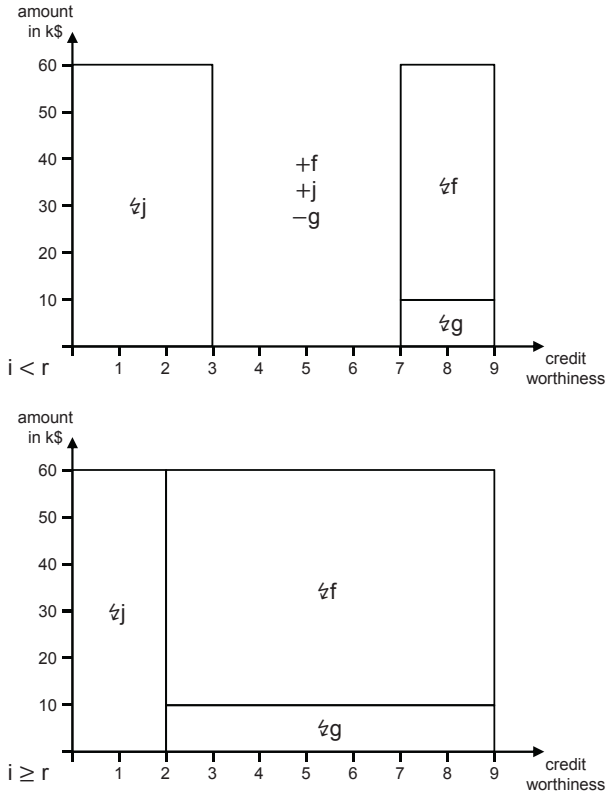


Figure 13: Business process PRIVATE_LOAN. Business rules regarding grant, forward, and reject (superimposed)

are "decision-scope consistent" with those for loans in general (see: Figure 9). Moreover, note that no decision scope is left with mortgage loans and for private loans as regard to permissions and prohibitions. It is known for each activity whether it is permitted or forbidden.

Figures 15 and 16 show the representation of these business rules at the design level using the proposed extended BPMN notation. Notice that the diagrams shown provide a full picture for each activity, including inherited and redundant conditions (e.g., if an activity is obliged, the activity is permitted and alternative activities are forbidden). Auto-completion rules introduced below enable designers to specify much simpler diagrams (e.g., Figure 18).

### 3.3 Auto-Completion and Auto-Correction of Business Rules

Decision consistency can be ensured in several ways: (1) a priori by having reasoners checking for subsumption or disjointness of rule conditions according to the identified consistency criteria, (2) a priori by auto-correction and auto-completion of specified rule conditions in a way such that the consistency criteria are met, and (3) a posteriori by checking for conflicts during process execution. All three options are sensible and may also be used together, each for a different kind of consistency criterium.

A priori checking by reasoners can immediately identify decision conflicts but will usually limit the expressiveness of the language for writing rule conditions (a situation encountered frequently with ontology languages where languages supported efficiently by reasoners often do not meet the desired expressiveness). Moreover, it will require repeating rule conditions at subclasses, while in conceptual modeling subclasses usually only specify the "delta" (i.e., additional or changed features with respect to the superclass).

Auto-correction and auto-completion of specified activity conditions according to the intent of the decision-scope approach is more appropriate in general. Auto-correction and auto-completion rules determine based on given (but possibly incomplete and inconsistent) conditions, whether an activity is permitted, forbidden, or obliged for a given process state of a business case. Similar approaches are employed in database systems or programming languages. In database systems, the violation of an integrity constraint need not lead to a transaction abort but may corrected in that other data are inserted, deleted, or modified as well such that all integrity constraints are finally met (e.g. the SQL-clause, ON DELETE CASCADE). In programming languages, pre-conditions of inherited methods are implicitly "or"-ed such that the criteria of contra-variant contract specialization are met (e.g., Eiffel).

Auto-correction and auto-completion take the role of remedying inconsistencies similar to the way in which simple law is repealed by a constitutional court if it is in conflict with constitutional law. The consistency criteria can be enforced as follows by auto-correction and auto-completion.

*Generalization:* Auto-completion in that an activity condition at the subclass is extended (using logical "or") by the activity condition of the same kind at the superclass.

*Super-sub class intra-activity conflict-freeness*: Auto-correction in that the activity condition at the subclass is restricted (using logical "and") to the decision scope set by the superclass.

*Closed decision scope for concrete classes:* There are two choices for auto-completion to close the decision scope for permission and prohibition, either permitting everything that is not forbidden or forbidding everything that is not permitted. Auto-completion for obligation is to set obligation of an activity to the

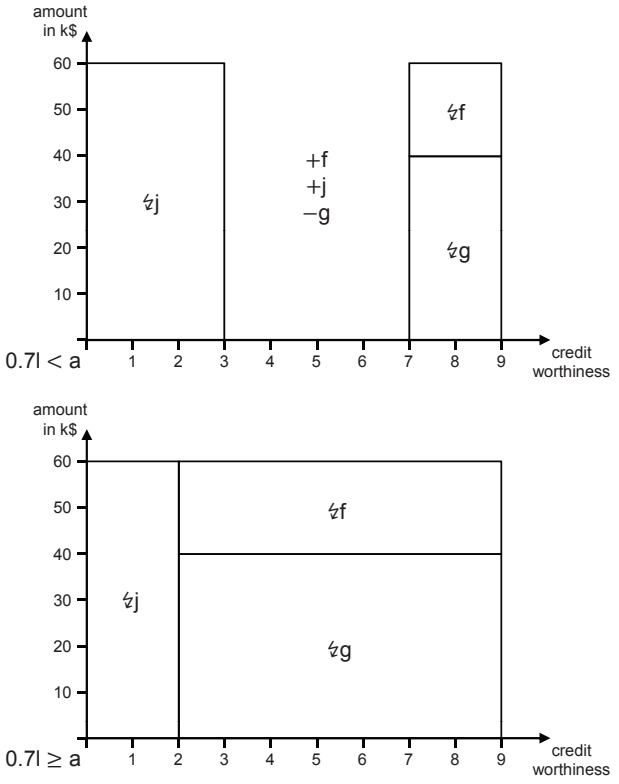


Figure 14: Business process MORTGAGE_LOAN. Business rules regarding grant, forward, and reject (superimposed)

conjunction of prohibitions of all alternative activities. And the decision scope between obligation and disobligation can be closed by setting disobligation to the complement of obligation.

*Intra-activity consistency:* There are two choices for auto-correction of conflicts between permission and prohibition depending on whether prohibitions should take dominance over permissions or vice versa. Auto-correction is that the subordinate condition is restricted to the complement of the dominant condition, e.g., permission to negated prohibition. The same situation applies for obligation and disobligation. Inconsistencies between obligation and permission and between prohibition and disobligation can be rectified by auto-completion in that permission is extended by obligation and disobligation by prohibition. Alternatively, one could use auto-correction by restricting obligation to permission and prohibition to disobligation.

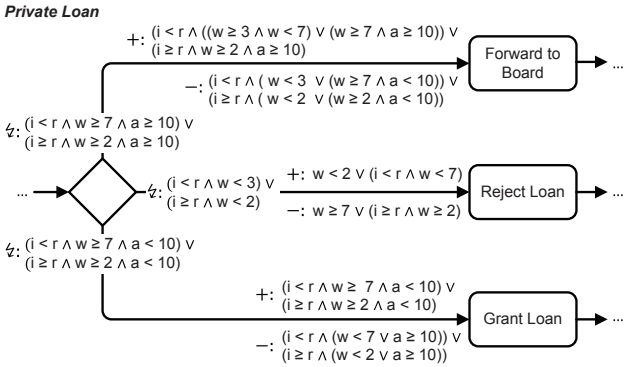


Figure 15: Business process PRIVATE_LOAN in extended BPMN

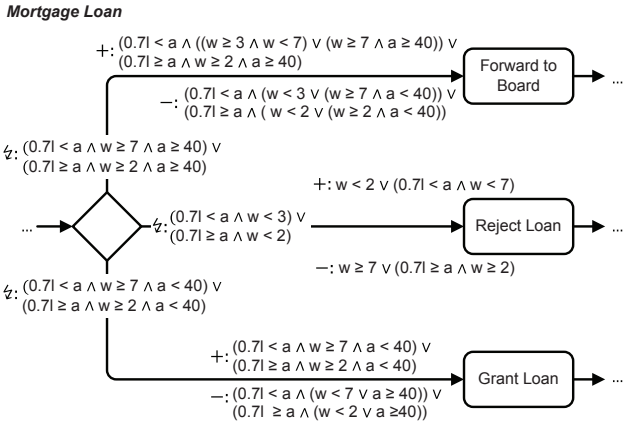


Figure 16: Business process MORTGAGE_LOAN in extended BPMN

*Inter-activity consistency:* The criterium that at least one activity is to be permitted for each possible state of a business case amounts to checking for deadlock and cannot be covered easily in a meaningful way by auto-correction; such a decision needs to be managed manually a priori (at design time) or at business process execution time. The criterium that alternative activities are forbidden if an activity is obliged can be enforced by restricting obligations or by extending prohibitions, the criterium that if an activity is disobliged if an alternative activity is permitted can be met by restricting permissions or by extending disobligations.

*Missing activity conditions:* Missing activity conditions are set to "false".

We note that the presented options for auto-correction and completion cannot be arbitrarily combined since also the interplay of the various consistency criteria has to be taken into account.

Figure 17 summarizes auto-correction and auto-completion rules for a particular choice of the possible options identified above. They determine under which conditions an activity $a$ of an abstract subclass $o$ with superclass $\hat{o}$ is permitted ($P^a_o$), forbidden ($F^a_o$), obliged ($O^a_o$), and disobliged ($D^a_o$) based on positive preconditions ($+a_o$), negative preconditions ($-a_o$), and activation conditions ($\lightning a_o$) specified for activity $a$ at subclass $o$ and based on the permission ($P^a_{\hat{o}}$), prohibition ($F^a_{\hat{o}}$), obligation ($O^a_{\hat{o}}$), and disobligation ($D^a_{\hat{o}}$) determined previously for the activity at superclass ($\hat{o}$). For simplicity, the shown auto-correction and completion rules assume single inheritance: Each class has at most one superclass; if a class has a superclass it inherits all activities from the superclass and may introduce additional activities. The case of multiple inheritance can be handled by considering the union of the conditions inherited from superclasses. Further, we assume that each activity $a$ belongs to a single set of alternative activities $A$.

The first rule in Figure 17 extends the notions of permission, prohibition, obligation, and disobligation for an activity $a$ that is introduced at subclass $o$, i.e., does not belong to superclass $\hat{o}$, to $\hat{o}$ such that for every activity of a class these conditions are defined also at the superclass, which simplifies the notation of auto-correction and auto-completion rules. $F^a_{\hat{o}}$ and $D^a_{\hat{o}}$ are set according to the criteria no. 3 and 4 of

inter-activity consistency: Activity $a$ is forbidden for the case that an alternative activity is obliged and activity $a$ is disobliged for the case that an alternative activity is permitted.

The auto-correction rules fix several kinds of conflicts in one step: Interactivity conflicts between prohibition and permission (criterium no. 1 of Figure 7), whereby prohibition takes dominance; super-subclass intra-activity conflicts (criterium no. 2 of Figure 12), whereby according to the decision-scope approach conditions at the superclass are dominant; and inter-activity conflicts with regard to obligations, whereby obligation is restricted to cases in which no alternative activity is permitted or obliged. Note: The auto-correction rules are defined in a situation in which inter-activity conflicts between obligations and permissions have not been yet resolved (which will be done thereafter by auto-completion) such that the rules need to refer to prohibitions and positive activation-conditions.

The rationale behind the auto-completion rules is that designers can specify business rules in the same simple way as we presented them in Figures 9, 13, and 14. Thereby we assumed that (1) an activity is permitted if it is obliged, (2) an activity is forbidden if an alternative activity is obliged, and (3) an activity is disobliged if it is forbidden or if an alternative activity is permitted. Further, according to the decision-scope approach activity conditions at the subclass are extended by the corresponding activity conditions at the superclass (criterium no. 1 of Figure 12).

For classes without super classes, auto-completion and auto-correction is subject to the same rules whereby all conditions that refer to the superclass are to be substituted by "false". For concrete classes, the decision scope for permissions and prohibition needs to be closed by extending for each activity its permission to the complement of its prohibition, its obligation to the conjunction of the prohibitions of alternative activities, and its disobligations to the complement of its obligation.

1. For activity $a \in A$, $a$ introduced at $o$:
   (a) $P^a_{\hat{o}} = \mathit{false}$
   (b) $O^a_{\hat{o}} = \mathit{false}$
   (c) $F^a_{\hat{o}} = \bigvee_{b \in A: b \neq a} O^b_{\hat{o}}$
   (d) $D^a_{\hat{o}} = \bigvee_{b \in A: b \neq a} P^b_{\hat{o}}$
2. Auto-Correction
   (a) $\breve{F}^a_o = -a_o \wedge \neg P^a_{\hat{o}}$
   (b) $\breve{P}^a_o = (+a_o \wedge \neg \breve{F}^a_o) \wedge \neg F^a_{\hat{o}}$
   (c) $\lightning \tilde{a}_o = (\lightning a_o \wedge \neg \breve{F}^a_o) \wedge \neg D^a_{\hat{o}}$
   $\breve{O}^a_o = \lightning \tilde{a}_o \wedge \neg \bigvee_{b \in A: b \neq a} (\lightning \tilde{b}_o \vee \breve{P}^b_o)$
3. Auto-Completion
   (a) $O^a_o = \breve{O}^a_o \vee O^a_{\hat{o}}$
   (b) $P^a_o = \breve{P}^a_o \vee P^a_{\hat{o}} \vee O^a_o$
   (c) $F^a_o = \breve{F}^a_o \vee F^a_{\hat{o}} \vee \bigvee_{b \in A: b \neq a} O^b_o$
   (d) $D^a_o = F^a_o \vee \bigvee_{b \in A: b \neq a} P^b_o$

Figure 17: Auto-Completion and Auto-Correction: Abstract subclass $o$, superclass $\hat{o}$, set of alternative activities $A$ of $o$ that includes $a$

**Example 10.** Figure 18 shows business process MORTGAGE_LOAN, a concrete business process specializing business process LOAN of Figure 10, but incompletely and incorrectly defined. The introduced auto-completion and auto-correction rules produce the permissions, prohibitions, and obligations as expressed by the the activity conditions $+$, $-$, and $\lightning$ shown for business process MORTGAGE_LOAN in Figure 16. Disobligations (not shown) are the complement of obligations.

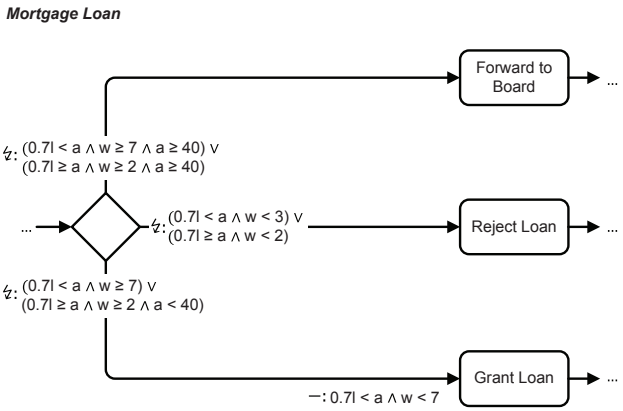


Figure 18: Business process MORTGAGE_LOAN, partially specified in extended BPMN

### 3.4 Discretionary Choice

We have treated activation conditions of an activity as obligations in the previous subsection and reasoned that if an activity is obliged all alternative activities are forbidden. This is a meaningful assumption for modeling under which conditions an activity of a business process may, must not be, or needs to be invoked. In the case of service use, the situation is different. The user of a service may by discretion choose to select automatically (i.e., without human agent involvement) one of several permitted alternative activities. Such a discretionary choice trigger, however, does not imply that alternative activities in the used business process are forbidden in general. For service compatibility[17] it is only required that (1) choice implies permission, (2) a particular choice at a superclass implies the same choice at a subclass, and (3) choices are exclusive (in case of decision points between exclusive activities, which we consider herein), see: Figure 19. Discretionary choice conditions may be annotated to sequence flow in BPMN as another kind of condition.

1. Choice implies permission:
   $C^a_o \rightarrow P^a_o$
2. Generalization:
   $\hat{o}$ is superclass of $o$: $C^a_{\hat{o}} \rightarrow C^a_o$
3. Exclusive choice:
   $a \in A, b \in A, a \neq b : C^a_o \rightarrow \neg C^b_o$

Figure 19: Criteria for Discretionary Choice

**Example 11.** A discretionary choice condition may define that a loan application is automatically forwarded to the board if the loan amount is high and the credit rating is average ($(w \geq 3 \wedge w < 5) \wedge a \geq 40$),

which is permitted (cf. Figure 4). This choice condition may be extended for mortgage loan, e.g., to $((w \geq 3 \wedge w < 5) \wedge (a \geq 40) \vee (a \geq 40 \wedge 0.7l < a))$, which is permitted for mortgage loans (cf. Figure 14). Note: The annotation of discretionary choice conditions is not shown in Figures 10 and 16.

Ways to auto-complete and auto-correct discretionary choice conditions are: (1) to restrict choice to permission, (2) to extend choice by the respective conditions of superclasses , and (3) to resolve conflicts in case multiple choice conditions overlap by (a) using priorities for activities, (b) by using a default master activity, or (c) by restricting each choice condition to non-overlapping parts.

### 3.5 Weak and Strong Business Rules

Specialization and the decision-scope approach as introduced are inherently monotonic as any potential conflict of rules is resolved by precedence of rules at a superclass over those at a subclass. In case the same rules apply for all subclasses apart from a few exceptions, monotonicity requires to restructure the class hierarchy or to redundantly define the same set of rules at several subclasses. Non-monotonic default rules may be a reasonable design alternative, providing such they are used in limited cases and are identified as such. In the legal context at which we looked for guidance to the decision-scope approach, lower level provisions may override higher-level provisions in identified cases (e.g., general world-wide sales conditions may state that warranty is for one year only, unless local consumer law prescribes otherwise). An approach for supporting monotonic and non-monotonic rules together has been developed in the context of authorization rules in database systems by introducing weak and strong authorization rules. We carry this idea over to the specialization of business rules in general.

Strong business rules are handled as described so far. Weak business rules may be overridden or canceled partially or fully at subclasses. In case of conflict between a strong and a weak business rule, the strong business rule takes precedence.

Strong and weak business rules come in positive and negative from. A weak positive rule of a superclass may be overridden by a different weak positive rule at the subclass or canceled by a negative (strong or weak) rule at the subclass. Whether overriding or cancelation is chosen will depend on the extent of difference to the superclass: A major deviation from the superclass will usually be handled by indicating a different weak positive rule at the subclass, a minor deviation by a negative rule. Whether a strong or weak negative rule is used depends on whether the rule may again be overridden or canceled at a subclass or not. Complementary, a weak negative rule at the superclass may be overridden by another weak negative rule at the subclass or be canceled by a positive (strong or weak) rule at the subclass.

The notions of strong and weak could be conceptually used for preconditions (positive or negative) and activation conditions (positive or negative) for activities. In practice, we have (so far) only encountered use cases for strong and weak activation conditions.

**Example 12.** Figure 20 visualizes the same business rules as Figure 9 but extended with a weak activation condition $(\tilde{\lightning} f)$ to forward a loan application to the board in case the credit worthiness is relatively high and the loan amount is not very low.

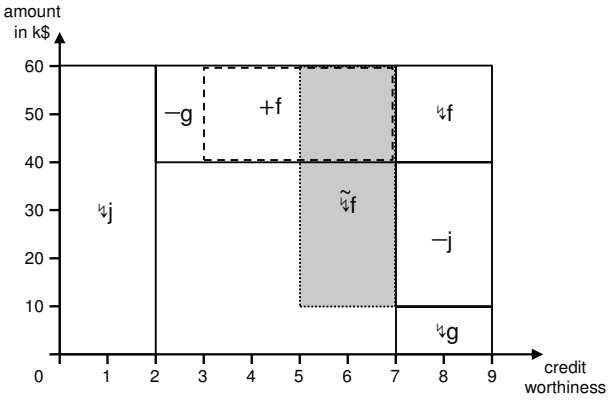


Figure 20: Business process LOAN: Weak and Strong Business Rules

$$\tilde{O}^a_o = \begin{cases} \tilde{\lightning}^a_o \wedge \neg D^a_o & \text{if } \tilde{\lightning}^a_o \text{ is defined} \\ \mathit{false} & \text{otherwise} \end{cases}$$

$$\tilde{D}^a_o = \begin{cases} -\tilde{\lightning}^a_o \wedge \neg O^a_o & \text{if } -\tilde{\lightning}^a_o \text{ is defined} \\ \mathit{false} & \text{otherwise} \end{cases}$$

Figure 21: Weak Activation Conditions: Cancelation, class $o$ without superclass

Figures 21 and 22 summarize the interplay between strong and weak activation conditions, negative and positive. The figures show how weak obligations and weak disobligations for invoking an activity $a$ of class $o$ without a superclass or with superclass $\hat{o}$, resp., are derived from activation conditions (weak or strong, positive or negative) and have to be read in conjunction with Figure 17.

**Example 13.** Figure 23 shows a negative strong activation condition $(-\lightning f)$ for activity forward of class LOAN canceling partially the inherited positive weak activation condition for a rather low loan amount in case the detachable income $i$ is lower than the monthly rate $r$ and the loan amount is below $30k$. It also visualizes the resulting weak obligation to invoke activity forward for a private loan $(\tilde{O}f)$.

In semi-automatic decision making, an activity $a$ will be automatically invoked for a business case of concrete class $o$, if it is strongly obliged or if it is permitted and weakly obliged.

$$\tilde{O}^a_o = \begin{cases} \tilde{\lightning}^a_o \wedge \neg D^a_o & \text{if } \tilde{\lightning}^a_o \text{ is defined} \\ \tilde{O}^a_{\hat{o}} \wedge \neg(D^a_o \vee \tilde{D}^a_o) & \text{if } \lightning^a_o \text{ not defined} \end{cases}$$

$$\tilde{D}^a_o = \begin{cases} -\tilde{\lightning}^a_o \wedge \neg O^a_o & \text{if } -\tilde{\lightning}^a_o \text{ is defined} \\ \tilde{D}^a_{\hat{o}} \wedge \neg(O^a_o \vee \tilde{O}^a_o) & \text{if } -\tilde{\lightning}^a_o \text{ not defined} \end{cases}$$

Figure 22: Weak Activation Conditions: Overriding and Cancelation, subclass $o$, superclass $\hat{o}$

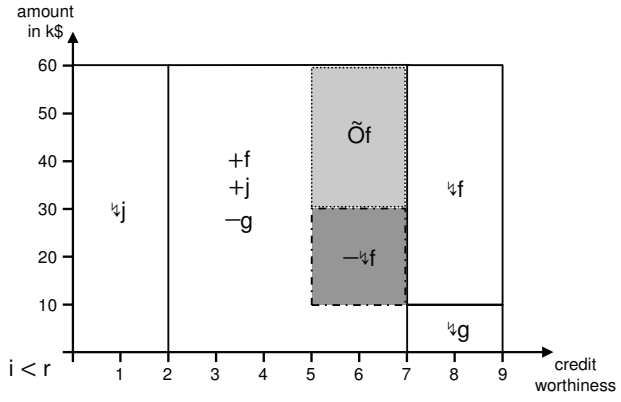


Figure 23: Business process PRIVATE-LOAN: Weak and Strong Business Rules

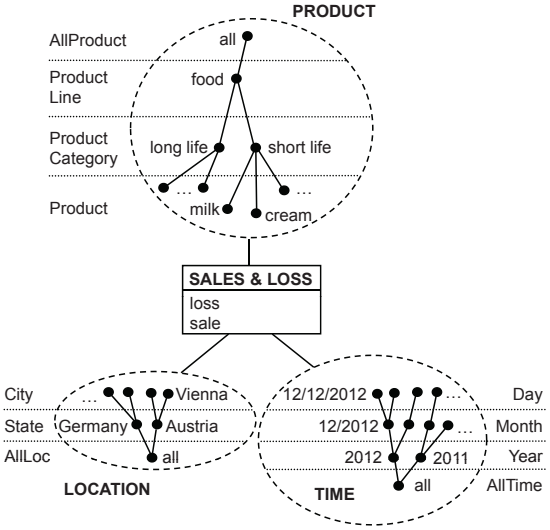


Figure 24: Data Warehouse Schema SALES&LOSS

## 4 Analysis Rules in Data Warehousing

In this section we describe how the decision-scope approach can be applied in data warehousing. Business rules in the context of data warehousing usually represent actionable knowledge on how to react on the presence or absence of (base or aggregated) facts in a data warehouse. We call such kinds of business rules *analysis rules*. They are used in active data warehousing for semi-automatic decision making, providing a feedback loop to transactional database systems in that rule actions are transactions in transaction databases. Analysis rules are useful as well in comparative data analysis where they guide users during an analysis process, provide judgements or alert managers upon detection of interesting comparative situations (which are given through the comparison of a fact group of interest vs. a fact group of comparison). For simplicity, we introduce the decision-scope approach for non-comparative situations.

### 4.1 Data Warehouses and Classes of Facts

A data warehouse consist of a set of facts and a set of dimensions. Each *dimension* consists of a leveled hierarchy of *roll-up* nodes. For simplicity we assume a linear hierarchy to explain the decision-scope approach: (1) The roll-up hierarchy of nodes is a tree, (2) Each node belongs to a level, (3) All leaf nodes belong to the same level, (4) Any two nodes of the same level roll-up to nodes of a common level. The set of facts, usually referred to as *cube*, consists of multi-dimensional points, whose coordinates are leaf nodes of the dimensions, and a set of measure values. We refer to the levels of the coordinates of a multi-dimensional point as *granularity*.

**Example 14.** Figure 24 depicts the schema of a simple data warehouse with dimensions PRODUCT, LOCATION, and TIME and cube SALES&LOSSES with measures loss and sold. The figure shows also sample nodes of dimension PRODUCT: nodes milk and cream at level Product, nodes longLife and shortLife at level ProductCategory, and node food at level ProductLine.

Base facts (whose point coordinates are at the bottom granularity) roll-up to aggregated facts (or roll-up facts). In this fashion the measure of an aggregated fact is derived from the base facts that roll-up to the aggregated fact, i.e, whose coordinates directly or indirectly roll-up to coordinates of the aggregated fact. From here on, we simply speak of "a point (or fact) *rolls-up to* another point (or fact)", when the point (or fact) directly or indirectly rolls-up to the other point (or fact). In top-down analysis, *drill-down* refers to the inverse of *roll-up*.

**Example 15.** Fact (Milk,Vienna,28-11-2012,5,20) rolls up to fact (shortLife,Austria,2012,8,27).

For applying the decision-scope approach to specialization of analysis rules, we consider a fact as object and the set of facts of a particular granularity (given by a level for each dimension) as a class. Furthermore, we can consider the class of facts of particular granularity $G$ that roll-up to a given multi-dimensional point $p$, denoted as $p[G]$. In this notation the class of facts at a particular granularity $G$ is the set of facts that roll-up to the root point $r$, consisting of the all-node from each dimension, $r[G]$. For brevity, a coordinate of a point or a level of a granularity is omitted if the coordinate is the root node of the dimension or the level is the root level, respectively. If a point $p$ rolls-up to $q$, then $p[G]$ is a subclass of $q[G]$. While class hierarchies with multiple inheritance can be constructed this way, we introduce the principles of the decision-scope approach in data warehousing for single inheritance. An extension to multiple inheritance requires to check the introduced consistency criteria for each superclass and it requires as well to apply the introduced auto-correction and auto-completion rules for each superclass.

**Example 16.** Class (food,Austria,2012) [ProductCategory,City,Year] could for example contain roll-up fact (shortlife,Vienna,2012,7,17).

### 4.2 Mono-granular Analysis Rules

We can apply the decision-scope approach as introduced for action rules defined over classes of business processes in the same way to analysis rules defined over classes of data warehouse facts if we consider only analysis rules that are defined over classes of facts of the same granularity. We call such analysis rules *mono-granular*.

Analysis rules are associated with a class of facts and determine, based on conditions over facts of the class, whether an action should be invoked or not. Thus, analysis rules specify a positive or negative activation condition, strong or weak. In most data warehouse settings initiated actions are always permitted

such that negative and positive preconditions of actions need not to be modeled (but this could be done as well if needed the same way as introduced for business processes).

Analysis rules are usually checked for a subset of facts for which they are defined based on external or application events. E.g., an informative action may be initiated at the end of a temporal period (month or year) after evaluating facts of this period once they have been loaded into the data warehouse, or a guidance action (suggesting further analysis steps) may be initiated for facts that are part of a current analysis situation (consisting of a set of facts selected in the current analysis step of an analysis session [31]).

Specialization along subset relationships between points at the same granularity is governed by the same set of rules as specialization between superclasses and subclasses of objects.

**Example 17.** Figure 25 depicts the conditions under which a particular Product is to be de-listed ($\lightning$) or not to be de-listed ($-\lightning$) in a particular State dependent on the percentage of the loss-to-sold ratio ($l/s$). The analysis rules are specified for product line food and specialized for product categories shortLife and longLife.

| *class* | $\lightning$ delist | $-\lightning$ delist |
|---|---|---|
| (food) [Product,State] | l/s $\geq$ 40 | l/s $<$ 10 |
| (longLife) [Product,State] | l/s $\geq$ 15 | l/s $<$ 10 |
| (shortLife) [Product,State] | l/s $\geq$ 40 | l/s $<$ 20 |

Figure 25: Mono-Granular Analysis Rules: Delist

### 4.3 Multi-Granular Rules: Prerogative Evaluation

In addition to the subset relationship between set of points at the same granularity, the roll-up hierarchy of points in multi-dimensional space can be used to evaluate analysis rules for the same action along drill-down relationships (the inverse to roll-up). One possible evaluation strategy is the *prerogative evaluation strategy* in which an action triggered for a higher level point implicitly implies the same action for each drill-down point. We have applied this principle in similar form for active data warehousing in the past [38, 39]. An alternative evaluation strategy will be presented in the subsequent subsection.

In the prerogative evaluation strategy, analysis rules for the same action are evaluated top-down. We assume at first that for one granularity at most one analysis rule is defined by a condition pair, consisting of a positive and negative activation condition. If one of the two condition applies for a roll-up fact, analysis is completed, whereby the indicated action is triggered if the positive activation condition is satisfied. Only if both conditions are not satisfied, i.e., for the situation that a fact falls into the open space of the condition scope, the analysis rules for drill-down granularities and drill-down facts at these granularities are considered in further rule evaluation.

Prerogative evaluation of analysis rules along roll-up hierarchies in top-down manner can be combined with evaluation along class hierarchies of points at the same granularity as described in the previous subsection. At each granularity all analysis rules defined for classes of points at that granularity are considered, and for each fact the most specific class is chosen. For simplicity we assume here that the class hierarchy of points has been defined such that each fact falls into a single most-specific class. Once a decision has been determined for a fact at a given granularity, analysis stops; if no decision could be made due to an open decision scope, analysis continues with drill-down facts at a lower granularity for which an analysis rule is defined. Again, we assume herein for simplicity that the hierarchy of granularities for which analysis rules for a given action are defined is a tree. The approach for message-to-method binding introduced for multiple inheritance of cooperation contracts [33], which define methods that are polymorphic in multiple receivers, can be carried over in order to extend the presented approach for the case of multiple inheritance.

**Example 18.** Figure 26 defines analysis rules at the granularity of (Product,State) and at the lower granularity of (Product,City) for action delist(PRODUCT,LOCATION). Figure 27 shows sample facts and for each fact whether action delist is trigged for the fact in the multi-granular evaluation strategy.

The analysis rule for (shortLife) [Product,State] is the most specific rule for roll-up fact (milk,Austria,20) but does not apply, neither positively nor negatively. Therefore, the analysis rules for action delist are evaluated for drill-down facts and the analysis rule specified for (shortLife) [Product,City] triggers for fact (milk,Vienna,48), de-listing milk in Vienna.

Analysis rule (shortLife) [Product,State] applies to roll-up fact (cream,Austria,20) de-listing cream in Austria which implicitly implies that cream is no longer sold in every city in Austria. Therefore, analysis is completed and no longer checked for drill-down facts such as (cream,Vienna,15).

| *class* | $\lightning$ delist | $-\lightning$ delist |
|---|---|---|
| (food) [Product,State] | l/s $\geq$ 40 | l/s $<$ 10 |
| (longLife) [Product,State] | l/s $\geq$ 15 | l/s $<$ 10 |
| (shortLife) [Product,State] | l/s $\geq$ 40 | l/s $<$ 20 |
| (food) [Product,City] | l/s $\geq$ 45 | l/s $<$ 10 |
| (longLife) [Product,City] | l/s $\geq$ 25 | l/s $<$ 10 |
| (shortLife) [Product,City] | l/s $\geq$ 45 | l/s $<$ 20 |

Figure 26: Multi-Granular Analysis Rules: Delist

| *point* | l/s % | delist |
|---|---|---|
| (milk,Austria) | 20 | |
| (cream,Austria) | 50 | $\lightning$ |
| (milk,Vienna) | 48 | $\lightning$ |
| (cream,Salzburg) | 52 | |
| (cream,Vienna) | 15 | |

Figure 27: Multi-Granular Prerogative Evaluation on Sample Facts

### 4.4 Multi-Granular Rules: Presumed Evaluation

An alternative evaluation strategy to the prerogative strategy is the *presumed evaluation strategy*. In the presumed strategy an action (typically an informative one) is not triggered again for drill-down facts to avoid information load, unless it is justified by a negative activation condition.

**Example 19.** Figure 28 defines analysis rules at the granularity of (Product,State) and at the lower granularity of (Product,City) for action alertMg(PRODUCT,LOCATION) . Figure 29 shows sample facts and for each fact lists whether action alertMg is triggered for the fact in the multi-granular evaluation strategy.

The analysis rule for (shortLife) [Product,State] is the most specific rule for roll-up fact (milk,Austria,20) but does not apply, neither positively nor negatively. Therefore, the analysis rules for action delist are evaluated for drill-down facts and the analysis rule specified for (shortLife) [Product,City] triggers for fact (milk,Vienna,48) alerting the manager for milk in Vienna.

The analysis rule (shortLife) [Product,State] applies to roll-up fact (cream,Austria,20) alerting the manager for cream in Austria, with the alert presumed to include all drill-down facts. This avoids raising an alert for every city in Austria and shields the manager from information overload, since presumably if the aggregated fact has a strikingly unusual measure value, so will usually the drill-down facts from which the aggregated measure is computed, e.g. fact (cream,Salzburg,52). But different to prerogative evaluation, an exception is reported if the alert should not have been raised due to a negative activation condition for a drill-down fact (e.g., for drill-down fact (cream,Vienna,15)). An exception alert will be raised in such a case.

| *class* | $\lightning$ alertMg | $-\lightning$ alertMg |
|---|---|---|
| (food) [Product,State] | l/s $\geq$ 40 | l/s $<$ 10 |
| (longLife) [Product,State] | l/s $\geq$ 15 | l/s $<$ 10 |
| (shortLife) [Product,State] | l/s $\geq$ 40 | l/s $<$ 20 |
| (food) [Product,City] | l/s $\geq$ 45 | l/s $<$ 10 |
| (longLife) [Product,City] | l/s $\geq$ 25 | l/s $<$ 10 |
| (shortLife) [Product,City] | l/s $\geq$ 45 | l/s $<$ 20 |

Figure 28: Multi-Granular Analysis Rules: Alert-Manager

| *point* | l/s % | alertMgr |
|---|---|---|
| (milk,Austria) | 20 | |
| (cream,Austria) | 50 | $\lightning$ |
| (milk,Vienna) | 48 | $\lightning$ |
| (cream,Salzburg) | 52 | |
| (cream,Vienna) | 15 | $-\lightning$ |

Figure 29: Multi-Granular Presumed Evaluation on Sample Facts

## 5 Conclusion

In this paper we have outlined the general principles of the decision-scope approach to specialization of business rules. We have shown that the decision-scope approach used by law hierarchies and hierarchical organizations for decision making along hierarchies of laws or organizational levels can be beneficially applied to the specialization of business rules. We have presented a set of consistency criteria that govern the decision-scope approach and introduced auto-correction rules to adjust inconsistent business rules in that conflicts are resolved in favor of the superclass, like in the real world a higher-level authority overrules a lower-level.

In particular, we have exemplified how the decision-scope approach can be incorporated into business process modeling and how it can be used in capturing actionable knowledge in data warehousing. We currently work in the SemCockpit [32] project on employing the decision scope approach for specialization of guidance and judgment rules in the context of comparative data analysis in business intelligence.

## References


[1] The Business Rules Manifesto – the Business Rules Group, http://www.businessrulesgroup.org/, 2006.

[2] Semantics of Business Vocabulary and Business Rules(SBVR), v1.0, http://www.omg.org/spec/SBVR/1.0/PDF. Technical report, January 2008.

[3] Jürgen Angele, Michael Kifer, and Georg Lausen. Ontologies in F-Logic. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 45–70. Springer Berlin Heidelberg, 2009.

[4] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 21–43. Springer Berlin Heidelberg, 2009.

[5] Twan Basten and Wil M. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.

[6] Elisa Bertino, Giovanna Guerrini, and Isabella Merlo. Trigger inheritance and overriding in an active object database system. *IEEE Trans. Knowl. Data Eng.*, 12(4):588–608, 2000.

[7] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Trans. Inf. Syst.*, 17(2):101–140, 1999.

[8] Peter Bichler and Michael Schrefl. Inheritance of business rules. In *Grundlagen von Datenbanken*, pages 20–24, 1995.

[9] Graham M. Birtwistle, Ole-Johan Dahl, Bjørn Myrhaug, and Kristen Nygaard. *Simula BEGIN*. Auerbach/Studentlitteratur, 1974.

[10] Alexander Borgida, John Mylopoulos, and Harry K. T. Wong. Generalization/specialization as a basis for software specification. In M. L. Brodie, J. Mylopoulos, and Schmidt J. W., editors, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer, 1984.

[11] Ronald J. Brachman. What is-a is and isn't: An analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30–36, 1983.

[12] Christoph Bussler. Public process inheritance for business-to-business integration. In *Proceedings of the Third International Workshop on Technologies for E-Services*, TES '02, pages 19–28, London, UK, UK, 2002. Springer-Verlag.

[13] Fabian Büttner and Martin Gogolla. On generalization and overriding in UML 2.0. In *UML 2004 Modeling Languages and Applications. UML 2004 Satellite Activities*. Springer, 2004.

[14] Luca Cardelli. A semantics of multiple inheritance. *Inf. Comput.*, 76(2/3):138–164, 1988.

[15] Giuseppe Castagna. Covariance and contravariance: Conflict without a cause. *ACM Trans. Program. Lang. Syst.*, 17(3):431–447, 1995.

[16] Roland Ducournau. "Real World" as an argument for covariant specialization in programming and modeling. In *OOIS Workshops*, pages 3–12, 2002.

[17] Georg Grossmann, Michael Schrefl, and Markus Stumptner. Design for service compatibility – behavioural compatibility checking and diagnosis. *Software and Systems Modeling*, 2012, DOI 10.1007/s10270-012-0229-0.

[18] O. M. G. Group. UML Specification, Version 2.0, 2006.

[19] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[20] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[21] Holger Herbst, Gerhard Knolmayer, Thomas Myrach, and Markus 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*, pages 29–46, New York, NY, USA, 1994. Elsevier Science Inc.

[22] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM Conferences (2)*, LNCS 5332, pages 1152–1163, Monterrey, Mexico, 2008. Springer.

[23] Walter L. Hürsch. Should superclasses be abstract? In *Proceedings of the 8th European Conference on Object-Oriented Programming*, ECOOP '94, pages 12–31, London, UK, UK, 1994. Springer-Verlag.

[24] Gerti Kappel and Michael Schrefl. Object/Behavior Diagrams. In *ICDE*, pages 530–539, 1991.

[25] Gerti Kappel and Michael Schrefl. Modelling object behavior: To use methods or rules or both? In *DEXA*, pages 584–602, 1996.

[26] Peter Kueng and Michael Schrefl. Spezialisierung von Geschäftsprozessen am Beispiel der Bearbeitung von Kreditanträgen. *HMD - Praxis Wirtschaftsinform.*, 185, 1995.

[27] Peter Lang, Werner Obermair, and Michael Schrefl. Modeling business rules with situation/activation diagrams. In *ICDE*, pages 455–464, 1997.

[28] Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.

[29] Milan Milanovic, Dragan Gasevic, and Luis Rocha. Modeling Flexible Business Processes with Business Rule Patterns. In *EDOC*, pages 65–74. IEEE Computer Society, 2011.

[30] Mukesh K. Mohania, Ullas Nambiar, Michael Schrefl, and Millist W. Vincent. Active and real-time data warehousing. In *Encyclopedia of Database Systems*, pages 21–26. 2009.

[31] Thomas Neuböck, Bernd Neumayr, Thomas Rossgatterer, Stefan Anderlik, and Michael Schrefl. Multi-dimensional navigation modeling using BI Analysis Graphs. In Silvana Castano, Panos Vassiliadis, Laks V. Lakshmanan, and Mong-Li Lee, editors, *ER Workshops*, volume 7518 of *Lecture Notes in Computer Science*, pages 162–171. Springer, 2012.

[32] Bernd Neumayr, Michael Schrefl, and Konrad Linner. Semantic Cockpit: An ontology-driven, interactive business intelligence tool for comparative data analysis. In *ER Workshops*, pages 55–64, 2011.

[33] Michael Schrefl, Gerti Kappel, and Peter Lang. Modeling collaborative behavior using cooperation contracts. *Data Knowl. Eng.*, 26(2):191–224, 1998.

[34] Michael Schrefl and Markus Stumptner. Behavior-consistent specialization of object life cycles. *ACM Trans. Softw. Eng. Methodol.*, 11(1):92–148, 2002.

[35] B. Silver. *BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A Structured Approach for Business Process Modeling and Implementation Using BPMN 2.0.* Cody-Cassidy Press, 2011.

[36] John Miles Smith and Diane C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst.*, 2(2):105–133, 1977.

[37] Markus Stumptner and Michael Schrefl. Configuring loopholes: Interactive consistency maintenance of business rules. In *Configuration – Papers from the Configuration Workshop at the 19th International Joint conference on Artificial Intelligence (IJCAI 2005)*, pages 37–39, 2005.

[38] Thomas Thalhammer and Michael Schrefl. Realizing active data warehouses with off-the-shelf database technology. *Softw., Pract. Exper.*, 32(12):1193–1222, 2002.

[39] Thomas Thalhammer, Michael Schrefl, and Mukesh K. Mohania. Active data warehouses: complementing OLAP with analysis rules. *Data Knowl. Eng.*, 39(3):241–269, 2001.

[40] W.M.P. van der Aalst. Inheritance of dynamic behavior in UML. In *Proceedings of the Second Workshop on Modelling of Objects, Components and Agents (MOCA 2002)*, pages 105–120, 2002.

[41] R.J. Wieringa, H. Weigand, J.-J.Ch. Meyer, and F.P.M. Dignum. The inheritance of dynamic and deontic integrity constraints or: Does the boss have more rights? *Annals of Mathematics and Artificial Intelligence*, 3(2-4):393–428, 1991.