A Method for Verifying the Consistency of Business Rules Using Alloy

Denilson dos Santos Guimarães
Post-Graduate Program in Informatics (PPGI)
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
denilson.guimaraes@ppgi.ufrj.br

Priscila M.V.Lima and Alexandre Luiz Correa
Post-Graduate Program in Informatics (PPGI)
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

Abstract—Agile enterprises have been investing heavily in the identification, representation and documentation of business rules (BRs), so that they can be known, understood and used all over the enterprise. In this respect a question of great concern in the development of information systems (IS) is the early identification of BR specification errors. This is due to its impact on the IS final cost, which can be considerable if the error identification occurs at a later time when the BRs have already been deployed. This paper presents a method for the verification of the logical consistency of a set of BRs. The method is particularly useful when the business domain encompasses a variety of types of BR specification errors. The set of BRs to be checked for consistency is translated into an Alloy Model, which the Alloy Analyzer uses as input to generate examples and counterexamples. The output of the Alloy Analyzer is then used for the identification of inconsistencies among the business rules. The proposed method makes it possible to have an early insight into inconsistencies that may exist in the rule model, allowing for a substantial reduction in rework costs.

Business Rules; Verification; KnowledgeBase; Alloy; SBVR; ABRD

I. INTRODUCTION

Business rules (BR) represent one of the most effective ways to express business knowledge and, nowadays, agile enterprises are reaching to identify, represent and document them, so they can be known and clearly used all over the enterprise [10]. Thus, identifying mistakes in business rules is increasingly becoming a question of great concern for the development of information systems (IS), impacting on their final costs, especially whenever this identification occurs at a later time, increasing these figures to hundreds of billions of dollars per year.

The typical inconsistencies in business rules specification are: redundancy, overlapping, conflict and incompleteness. Redundancy occurs when two rules have the same set of conditions (with conditions possibly arranged in a different order) and the same conclusion. Then, one of them is said to be redundant [2]. Overlapping occurs when one rule is wholly or partly contained within another [6] or when two rules have the same conclusion and one rule has a more restrictive condition. The more restrictive rule can be subsumed by the less restrictive rule on the grounds that whenever the former succeeds, the latter also succeeds [2]. Conflict occurs when two or more rules produce contradictory results [6] or their conditional parts consist of the same set, but the conclusions are mutually exclusive [2]. Incompleteness occurs when all the information necessary to produce a conclusion does not exist. This may be caused by gaps left inadvertently, uncertain knowledge or lost track of the grown knowledge base [2].

According to Hodges [8], beliefs can be expressed by declarative sentences. A set of beliefs is called consistent if there is some possible situation in which all the sentences are true.

Alonzo Church proved in 1936 that it is impossible to prove a theorem by a systematic method for testing the consistency of sets of sentences written in first-order language [8]. However, it’s still possible to prove the inconsistency of these sets.

Similarly, Edsger W. Dijkstra stated that program testing can be used to show the presence of bugs, but never to show their absence [7]. Unfortunately, most bugs in code elude testing.

In order to test the consistency of a BR set, we make a simple assumption: all inconsistencies will involve a simple pair of rules. In this case, it will be necessary to check the consistency of pairs of sentences. The computational complexity of this calculation will be in the order of the quadratic growth, i.e., its values grow proportionally to the square of the function argument, represented by \( \Theta(n^2) \). For instance, when \( n \) represents one thousand sentences, the number of operations will be in the order of one million [15]. This figure will certainly grow even more, when the consistency checking involves groups of three or more sentences.

In order to facilitate this great number of necessary operations, it's recommended to use a different approach, like the instance finding, proposed by the analysis underlying the Alloy tool. Rather than attempting to construct a proof that an assertion holds, it looks for a refutation, by checking the assertion, looking for a particular case in which the assertion is
found not to hold. That case is reported as a counterexample [7].

Instance finding has far more extensive coverage than traditional testing and it tends to be much more effective at finding bugs, because most bugs have small counterexamples. In other words, only a small counterexample is necessary to prove that an assertion is invalid. To make the instance finding feasible, it’s needed to define a scope that limits the size of instances considered. Even a small scope usually defines a huge space of instances. Systems that fail on large instances almost always fail on small ones with similar properties, even if such small instances don’t occur in practice. So, by checking all small instances, the large ones will be effectively checked also. This is called by Jackson as the "small scope hypothesis" [7].

Different approaches to the automated verification of a set of BRs have been presented in the literature but all of them, at a certain point, demand complex implementation or have logic limitations, making their use non-intuitive or limited to skilled programmers, with a non-novice knowledge of mathematical logic and proving methods.

This article intends to present an easy-to-use method that is suitable to the early verification of the consistency of a BR rule set. The method can precisely show the inconsistency generated by a BR specification. Moreover, the inclusion of new rules or the updating of the existing ones can be easily absorbed by the verifying procedure. This turns the method into a robust and reliable option for BRs analysts.

The remainder of this paper is organized as follows. Section 2 presents the conceptual framework. Section 3 describes the conceptual method to verify a set of BRs using the Alloy tool. Section 4 illustrates the method with an example. Finally, Section 5 presents the final remarks.

A. Related Work

EVA Project [1] is a reference in the verification and validation of BRs. The use of meta-knowledge (i.e. knowledge about the knowledge) describes restrictions necessary to validate the redundancy, consistency, completeness and correction of a knowledge base. Besides being very complete (with 28 different criteria), this project was extended to non-monotonic logic, bringing it closer to the real world. The processing output is, consequently, complex. The method proposed in this paper explores the utilization of a “model finder”, whose output is based on intuitive examples and counterexamples.

PREPARE method [2] uses predicate/transition nets to represent knowledge. These nets are special classes of Petri nets and verification is done through syntactic patterns of recognition. These nets are a graphic representation of predicate logic. Consequently, these graphic representations don’t support universal quantifiers, as the Alloy tool does.

NuSMV2 [3] is a tool that extends the original SMV of CMU. With a very efficient BDD algorithm, it was used successfully in hardware and software verification. Unfortunately, its programming language doesn’t support more complex structural types, such as relational, for example. This forces the developer to codify relations using arrays of primitive types. The Alloy tool offers the developer relational, arithmetic and logic types.

NIET method [4] is particularly critical with inference machine usage. It reduces the significance of these machines to merely correct the ordination of BRs. It proposes an IDE as repository, verification, validation and code generation in different languages of BRs. Through this method, the ordination solution is turned into a development concern. There is great concern with the performance and cost of the inference machines and with their replacement by an IDE, proposed by this method. The adopted strategy in our work doesn’t require the usage of any IDE and performance concerns aren’t addressed in our work.

CPKSA [5] is study that proposes the use of group decision, with an algorithm that has a confidence factor. It’s only suitable in situations with BRs conflicts that cannot be resolved.

NL2AlloyviaOCL [16] is a tool that generates Alloy expressions from English text statements and a corresponding UML model. But it doesn’t have a method to analyze the consistence of BRs generated using this tool.

There are several commercial BR management and simulation tools, but they do not focus on the early verification checking of BR models.

II. CONCEPTUAL FRAMEWORK

The method presented here for verifying a set of business rules, in order to allow for the early identification of inconsistencies into these rules, requires the specification of a representation of BR, the development of the Alloy Model and the use of a BR Cycle.

A. BR Representation

BR are declarations that constrain, derive and give conditions for existence, representing the knowledge of the business. BR are not descriptions of a process or processing. Rather, they define the conditions under which a process is carried out or the new conditions that will exist after a process has been completed [6]. BR can be represented using Semantics of Business Vocabulary and Rules (SBVR) [12] or using business language. SBVR use is suitable for business experts, since it allows the representation of business vocabulary and rules using controlled natural language. Business language is more intuitive to business stakeholders and business analysts, although it may be used in an unstructured way. Both representations generate the business vocabulary, composed by terms and facts about them, which will be used to formulate the BR.

B. Alloy Model

Alloy is a language for describing structures and a tool for exploring them. An Alloy Model is a collection of constraints that describes (implicitly) a set of structures. Alloy’s tool, the Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model, by generating sample structures (here called examples), and to check properties of the model by generating counterexamples [11].

The gross structure of an Alloy Model consists of [7]:

382
• Signature declarations, labeled by the keyword `sig`, each one representing a set of atoms, and may also introduce some fields, each representing a relation.
• Constraint paragraphs, labeled by the keywords `fact, fun` and `pred`, which record various forms of constraints and expressions.
• Assertions, labeled by the keyword `assert`, that record properties that are expected to hold.
• Commands, labeled by the keywords `run` and `check`, with the instructions to the Alloy Analyzer.

The signature declarations set up a classification hierarchy which can be extended into disjoint subsets of these sets, using the keyword `extend`. Marking an extended signature declaration using the keyword `abstract`, indicates that the set has no elements of its own that do not belong to its extensions [7].

A `fact` paragraph records a constraint that is assumed always to hold. An `assert` paragraph introduces a constraint that is intended to follow from the facts of the model. A `pred` paragraph defines a constraint, in the form of a logical predicate. The `run` command instructs the Alloy Analyzer to attempt to find a solution (or example) to the constraints, usually grouped together in a test predicates. The `check` command tells the Alloy Analyzer to find a counterexample to an assertion [7].

C. BR development cycle

For developing business rules-based applications, it’s recommended using a well documented and structured approach. The Agile BR Development methodology (ABRD) is the industry’s first free, vendor-neutral methodology and provides a full rule lifecycle, from discovery to governance, by using an “agile”, iterative approach. ABRD activities fall into five categories described below, each of these being run multiple times as the cycle is executed [13]:

- Rule Discovery: aims to develop simple modeling artifacts (like rule descriptions, business entity diagrams and business process maps) and is an iterative process, that will identify a subset of rules and document them, as opposed to spending months figuring out all the rules up front and producing a huge document [14].
- Rule Analysis: the goal is to understand the meaning of the rule as stated by the business person and subject matter experts and to remove any ambiguity and semantic issue, preparing the rules for the future implementation [14].
- Rule Design: the purpose is to take a first complete pass through the development process, confront the main design issues and lay the groundwork for future refinements, including architecture aspects and initial prototyping [14].
- Rule Authoring: looks at the general issues surrounding expressing BR in a technology-independent way, particularly addressing those related to the business vocabulary itself and to the rule structures [14].
- Rule Validation: develops unit tests to assess the rule outcome and executes the tests, until all tests succeed, ensuring that the rules accurately reflect the business intent [14].
- Rule Deployment: addresses common issues around rule deployment, packaging and integration, looking at major deployment and integration considerations, such as transaction support, scalability and data access [14].

The method proposed in this paper is particularly applicable at the Rule Analysis activity.

D. The Verification of BR

Assuring the quality of BR falls into two general areas: validation and verification [9].

Validation means ensuring the correctness of BR with respect to the business purpose, with the goal to assure that when they are applied, the results will be appropriate in all relevant circumstances. Validation is largely a matter of human inspection and can be achieved by using diagrams, test scenarios and individual analysis of each business rule. Rule validation is about checking whether there are the right BR [9].

Verification, on the other hand, means assessing fitness with respect to logical consistency and involves discovering BR (usually two or more in combination) that exhibit some anomaly. Rule verification is concerned about whether the BR are correct [9].

In the method proposed in this paper, we will use the Alloy tool to support and automate the verification activity described above. We will focus on the discovery of redundant, overlapping and conflicting rules. The incomplete rules are out of the scope of the method, since it pertains to the validation activity. We will treat the problems of the overlapping and the conflicting rules in the same manner, as they are implicitly related. The mapping of BR artifacts into Alloy structures is not being addressed in this paper, because it would exceed the limit of pages allowed for the paper.

III. THE METHOD

A. Method Steps

The proposed method for verifying the consistency of a set of BR consists of the following steps:

1. Identify the BRs and specify them in a structured way, using business language for representation.
2. Build a conceptual model, describing the business concepts and their relations.
3. Transform the rules into an Alloy Model, building a consistent set of BR, as described in Section III-B.
4. Using one rule at a time, introduce the new BR in the model, as described in step 2 of Section III-B, generating possible inconsistency situations. When this happens, update the set of BR, as described in Section III-C.
5. Repeat step 4, until no inconsistencies in BR are found and the set of BR returns to the previous consistent situation.

B. Generation of the Alloy Model

This section describes the steps necessary to generate the Alloy Model, as described in Section II-B, representing the
consistent set of BR that will be evaluated. The method consists of the following steps:

1. Represent in the Alloy Model every concept and relation built in step 2 of Section III-A, as sets of structures, in the form of Alloy paragraphs sig, extend and abstract, generating a Consistent BR Model (CBRM)
2. Each rule to be introduced in the model is converted to a paragraph pred.
3. Execute the run command of the Alloy, searching for valid examples of the BR model. If no valid example is found, the rule created an conflict and the conflicting rule must be found, the model must be adjusted.
4. Define an assertion paragraph, negating the same constraint used in previously step 2 and execute the check command of the Alloy, searching for invalid examples for the set of BR modeled. If an invalid example is found, then the rule is redundant and the model must be adjusted.
5. Since the rule is consistent and non-redundant with the BR model, it can converted to an Alloy paragraph fact, generating a new CBRM.

C. Generation of the Inconsistency Scenarios

Given a CBRM generated through the mechanism described in Section III-B, each new BR must be introduced in this model using the same concepts and the rules already represented in Alloy Model. The method allows the detection of the following inconsistency scenarios:

1. Conflicting rules (step 3 of Section III-B): when the Alloy Analyzer indicates that the model is inconsistent. In this case, the rule must be checked against every other possibly conflicting rule in the model. This can be done by creating a test predicate containing the disjunction of the rule and the possibly conflicting rule. The run command is executed again and, if the model becomes consistent, the conflict between the rules is proven.
2. Redundant rules (step 4 of Section III-B): The same procedure described above is repeated using a new test assertion containing the rule and the negation of the possibly redundant rule and the check command must be executed. If counterexamples are found, the redundancy is proven.

IV. EXAMPLE

This example focuses on the rules set of a fictional loan process, similar to the one presented in [6]. The verification will be executed following the proposed method, in order to find any inconsistencies that may occur in the set of rules.

A. Identify the BR

In this step, we specify all the business terms, facts and rules. These rules are listed below, structurally expressed in business language:

A loan exceeding $1,000 must be approved by a regular manager.

A new loan may not be offered to an overdrawn customer.

An overdrawn customer may not be offered a new loan.

B. Build the domain conceptual model

Figure 1 shows the class diagram representing the conceptual model of the loan process domain, proposed in [6].

C. Build the Alloy Model

At this stage of the method, the Alloy Model will be built, representing the conceptual model. For example, from Section IV-B, the concepts Customer, OverdrawnCust and UnderdrawnCust will be written in the Alloy Model, as follows:

abstract sig Customer { }

sig OverdrawnCust extends Customer { }
sig UnderdrawnCust extends Customer { }

The complete Alloy Model representing the conceptual model is shown in figure 2.

D. Build the BR Model

The consolidated rules will be written in sequence. For each pair of rules described in Section IV-A, the first one will be used to represent the consolidated rule set. For example, from Section IV-A, the rule “A new loan may not be offered to an
overdrawn customer” will be written in the Alloy Model, as a fact paragraph, as follows:

\[
\text{fact NoLoanOverdrawn\{no } z: \text{Loan} | z.c \text{ in OverdrawnCust}\}.
\]

In sequence, a simple constraint (as simple and simultaneously complete as possible, according to the “small scope hypothesis” described in Section I) will be defined. In this example, it will state that there will always be a non-empty set of Loans in our model. This constraint is now written in the Alloy Model as followed:

\[
\text{pred SomeLoans\{some } s: \text{Loan} | s \text{ in ApprovedLoan one } s: \text{Loan} | s \text{ in NotApprovedLoan}\}.
\]

According to the step 3 of the method, as presented in Section III-A, the run command is executed. The Alloy Analyzer informs that the model is consistent, showing instances that exemplify it, as shown in figure 3.

D. Generate the Inconsistency Scenarios

Now, according to step 4 of the method, as presented in Section III-A, new rules will be introduced, representing the inconsistency scenarios to be evaluated (conflict and redundancy). For each pair of rules described in Section IV-A, we will include the second one in the consolidated rule set generated in step 3 of the method.

E. Execute the Alloy Analyzer

The first inconsistency scenario is generated with the inclusion of the conflicting rule “A loan exceeding $1,000 must be approved by a regular manager.”, described in Section IV-A, written as a pred paragraph, as follows:

\[
\text{pred RuleLoanRegularApprover\{all } x: \text{Loan} | x.v = V1000 \text{ implies } x.m \text{ in RegularManager}\}.
\]

According to step 4 of the method, as presented in Section III-A, the run command is executed. The Alloy Analyzer informs that the model is still consistent, finding instances that satisfy both conflicting rules. But, when the assert and check commands are executed, a counterexample is found, proving that the assertion is invalid, as shown in figure 4.

By updating the model, joining the conflicting rules, using the or connector and executing the run command again, the Alloy Analyzer informs that the model is now consistent. Executing the assert and check commands, no counterexample is found, proving that the assertion is valid. However, the instances found only exemplify the ambivalence of both rules written in the model, demonstrating the conflict generated, as shown in figure 5. This inconsistency must be resolved in the validation activity of the BR Cycle.

The next inconsistency scenario is generated with the inclusion of the redundant rule “An overdrawn customer may not be offered a new loan.”, described in Section IV-A, written as a pred paragraph, as follows:

\[
\text{pred NoLoanOverdrawn02\{all } od: \text{OverdrawnCust} | \#od.l = 0\}.
\]

According to step 4 of the method, as presented in Section III-A, the run command is executed. The Alloy Analyzer informs that the model is now inconsistent, not being possible to find any instances that satisfy both redundant rules. Executing the assert and check commands, no counterexample is found, indicating that the inclusion of the new rule didn’t affect the results of the execution of the original rules set, as shown in the figure 6.

By updating the model, denying the original rule, using the not connector and executing the run command again, the Alloy Analyzer informs that the model is now consistent, not being possible to find any instances that satisfy both redundant rules. Executing the assert and check commands, a counterexample is found, informing that the assertion is invalid and proving the redundancy, as shown in figure 7.
Figure 5. The Alloy Model with the conflicting scenario corrected and the snapshot of the execution of the Alloy Analyzer

Figure 6. The Alloy Model with the redundant scenario corrected and the snapshot of the execution of the Alloy Analyzer

V. CONCLUSIONS

The method proposed in this paper provides both the business analyst and developers with a tool that allows a rapid evaluation of the consistency of a set of BR, in a logical point of view. It also proposes an incremental form of process evaluation of a set of BR, as this set is continuously updated through the lifecycle of BR applications.

By using this method, the most typical inconsistencies in BR (overlapping, redundancy and conflict) can be found, showing the counterexamples found and making the detection and correction easier and faster.

Thus, a consistent set of BR can be generated more easily, since the automated rule verification activity, provided by this method, can guarantee the rule validation activity.

The next challenges are concentrated in evolving this method to make its using easier to non-technical people and provide ways to automate the identification of the rules that become inconsistent with the inclusion of new rules in the consistent set of BR.

ACKNOWLEDGMENT

Our thanks to Fabrício de Martino, for the related work research and the experiments with the Alloy tool.

REFERENCES


