Simulation-Driven Approach for Business Rules Discovery

Biljana Bajić-Bizumić[1], Irina Rychkova[2], and Alain Wegmann[1]

[1] École Polytechnique Fédérale de Lausanne (EPFL),
Lausanne, Switzerland
[2] Centre de Recherche en Informatique
Université Paris 1 Panthéon - Sorbonne,
90, rue Tolbiac, 75013 Paris, France
{biljana.bajic,alain.wegmann}@epfl.ch,
irina.rychkova@univ-paris1.fr

Abstract. Business rules are everywhere. Some of these rules are implicit and thus poorly enforced, others are written but not enforced, and still others are perhaps poorly written and obscurely enforced [1]. In this work, we propose an interactive, simulation-driven approach for the discovery of business rules. The rules are first specified in a natural language, then translated to the Alloy specification language. The Alloy Analyzer tool is used as a platform for rule simulation and discovery: it provides a domain specialist with an instant feedback, helping her to detect the issues with the existing business rules and to discover new rules in a systematic way.

Keywords: Business rules, Business Rule Discovery, Alloy, Requirements Elicitation.

1 Introduction

This paper illustrates how business rules for the order processing activity at Générale Ressorts SA can be discovered using the Alloy Analyzer tool [2]. The order processing example is academic, but it is grounded on a real problem: many insurance companies have a strategy for leveraging on late payments to maximize their return. In other industries, it can take up to three years between the ordering of parts and the payment. These terms are not captured as business rules, but they could be. When the IT systems make these payment terms explicit, management has to face many annoying facts in their business strategy. Another example: very often processes are defined in a strict way to make sure that the interest of the company is protected. But this is feasible only for second and third tier customers. Strategic customers always bypass the rules. When a rule is defined explicitly - the exceptions are not considered until a problem occurs with a key customer.

We claim that systematic capturing of business rules and the analysis of issues created by rules will help companies to define and improve their strategies.

X. Franch and P. Soffer (Eds.): CAiSE 2013 Workshops, LNBIP 148, pp. 111–123, 2013.

However, systematic handling of the rules expressed in a natural language is challenging: "Capturing the logic of an entire business would require probably many thousands of rules; a smaller subsystem, perhaps several hundreds" [3]; they can be specified by different analysts, can be inherited from the previous process versions, and can reflect different policies or strategic decisions.

The goal of this paper is to illustrate how Alloy can assist in the systematic discovery of business rules and the issues related to them. Our approach is based on rule modeling and simulation in Alloy Analyzer [2]. The use of formal specifications and model checking techniques allows us not only to *discover* the new rules but also to validate their consistency.

The remainder of this paper is organized as follows: In Section 2, we discuss our motivation and present the related works; In Section 3, we present our example - the Order Processing, specified for Générale Ressorts SA. We also introduce the Alloy specification language and discuss how business rules can be specified with this language. In Section 4, we present the Alloy model for Order Processing. In Section 5, we illustrate the BR discovery with Alloy Analyzer . We generalize our approach in a form of four steps to interactive BR discovery. In Section 6, we present our conclusions.

2 Motivation and Related Work

According to [3], "A business rule is a compact statement about an aspect of a business. It's a constraint, in the sense that a business rule lays down what must and must not be the case. At any particular point, it should be possible to determine that the condition implied by the constraint is true in a logical sense; if not, a remedial action is needed."

Many research and industrial publications are focused on challenges associated with business rules (BRs). In industry, vendors such as ILOG (currently a part of IBM), FICO Blaze Advisor and Pega Systems, Inc. have been developing business rule engines (BRE) since the late 1980s and are now leaders in the emerging BRE segment [4,5,6]. In academia, the computer sciences and engineering outlets have been active in business rule research [7,8,9], with extensive studies in rule programming, meta-modeling, rule mining, rules engines, business user interfaces and their role in services oriented architectures (SOA). Furthermore, joint academic and industry developed Object Management Groups (OMG) Semantics of Business Vocabulary and Business Rules (SBVR) standards (released in September 2006), which is intended to provide standards surrounding BR structure, terminology, classifications and meaning in BR authoring and repositories [10].

Different approaches propose different phases in business rules management life cycle (BRMLC). In [3] the main phases are discovery, definition, review and maintenance. By [1] the main phases are: discovery, analysis, design, authoring, validation, deployment. In [11] these phases are: plan, capture, organize, author, distribute, test and apply. In this paper, we focus on discovery (i.e. capturing) phase of BRMLC. The goal of discovery phase is to identify the potential business rules affecting the domain segment in development.

To see the effect of the introduced business rules, the designer must go through a number of phases, such as analysis, design, authorizing, validation and deployment. It would be much more effective, if the designer could get instant, visual feedback on how new business rules influence the behavior of a process. Examples of the company behavior could help him find what constraints are missing in the model. To discover new BR in an interactive way and ensure the consistency and validity of the overall set of BR, we propose the approach based on Alloy simulation.

Although BRs traditionally are expressed in a natural language [12], the works presented in [13] and in [10,14] report on other forms of BR formalization. In [13], the diagramatic language is used and in [10,14,15] the rules are specified with formulas in modal logic. In this work, we use an Alloy specification language (based on first order logic) and propose a technique for BR discovery based on model simulation and analysis in the Alloy Analyzer tool.

3 Modeling Business Rules with Alloy

In this section, we introduce our working example, the order processing, specified for Générale Ressorts SA. First, we specify the order processing and its associated business rules in a natural language; then, we discuss how this example can be specified in Alloy [16]. In the following sections we present the Alloy model for order processing and illustrate how the business rules for order processing can be interactively analyzed and discovered using the Alloy Analyzer tool [2].

3.1 Case Study: Order Processing in Générale Ressorts

Générale Ressorts SA is the market leader in watch barrel springs and a first-class manufacturer of tension springs, coil springs, shaped springs and industry components [17]. Générale Ressorts SA works with thousands of customers and strives to ensure the highest quality both for its products and for its customer services.

Order processing is one of the strategic activities in Générale Ressorts SA: it covers a complete order life cycle, from order creation to payment and delivery. Whereas the company constantly improves its technological processes in order to shorten the production cycle, the payment can take months after the product is delivered[1]. Therefore, flexible business rules for order processing and customer transactions management are essential for GR.

Order processing includes the following processes: order creation, order preparation, shipping and accounting. It is also closely related to the customer management processes in the company. The whole process, from the moment the customer makes an order to the delivery and the accounting is known as the order-to-cash cycle.

[1] The "shipping after payment confirmation" policy is not acceptable for this industry in general and for Générale Ressorts SA in particular.

In this paper, we define the (simplified) order processing activity that focuses on order creation, delivery and payment only: A customer submits an order request for manufacturing a watch component (part); the confirmed order is then prepared and delivered to customer. As stated above, the payment for the confirmed customer orders is a necessary condition to finalize the overall order processing transaction for a given order, though it is not required for order delivery.

Below, we present list of business rules related to order processing:

1. *Order creation*
   BR1.1 A customer order can be created and confirmed only for the customers registered in the enterprise information system.
   BR1.2 A customer order can be created and confirmed only for the parts existing in the product catalog.
   BR1.3 If an order request from a new customer is received, this customer has to be registered in the enterprise system.
2. *Order delivery*
   BR2.1 Every confirmed customer order must be eventually delivered to the customer.
3. *Accounting*
   BR3.1 Every confirmed customer order must be eventually paid by the customer.
4. *Customer management*
   BR4.1 Every customer record must contain one customer name.
   BR4.4 Every customer record must be associated with a previous orders history.
   BR4.5 A customer whose transactions with GR is equal or superior to XX XXX euro per year receives a status of strategic customer at GR.
   BR4.6 A customer whose transactions with GR is inferior to XX XXX euro per year receives a status of regular customer at GR.
   BR4.7 Strategic customers must always be able to submit the order with GR.

3.2 Alloy

Alloy [18] is a declarative specification language developed by the Software Design Group at MIT. Alloy is a language for expressing complex structural constraints and behaviour based on first-order logic.

The Alloy Analyzer [2] is a tool for the automated analysis of models written in the Alloy specification language. Given a logical formula and a data structure that defines the value domain for this formula, the Alloy Analyzer decides whether this formula is satisfiable. Mechanically, the Alloy Analyzer attempts to find a model instance - a binding of the variables to values - that makes the formula true. [19]

The syntax of Alloy is similar to the syntax of OCL (the Object Constraint Language) for UML [20]. In the following lines, the Alloy keywords are marked in bold. Data structures are represented with signatures (**sig**) and fields. Logic of Alloy language combines the quantifiers of first-order logic ($\exists$ (**one**), $\forall$ (**all**), etc.) with the arithmetic operators ($+, -.$ =,etc.), set operators ($\cup$, $\cap$, $\subset$ (**in**), etc.), relational and logical operators ($\neg$ (**not**/!), $\wedge$ (**and**/&&), $\vee$ (**or**/||), $\Rightarrow$ (**implies**/=>), etc.).

There are three types of constraints specified in Alloy: Fact (**fact**) is a model constraint that permanently holds; Predicate (**pred**) is a constraint that holds in specific context or for a specific part of the model only; Assertion (**assert**) is a property that the designer believes should be implied from the model and can check (command **check**) if it can be deduced from the other (permanent or contextual) constraints. In the examples given in the paper, assertion is presented

in a shorter form **assertionName: check**. In our previous work [21] we defined the iterative process for service design where the Alloy signatures, facts and predicates were used for service specification. In this work, we extend the use of Alloy constructs: in particular, we use Alloy constraints for modeling and validation of business rules.

Representation of Business Rules in Alloy
By their definition, business rules are intended to assert business structure or to control or influence the behavior of the business [22]. Structural rules define the business information model. Whereas, a behavioral rule is about how the business reacts to business events. They are specified when something happens at the boundaries of the system [10]. In this approach we deal with both kind of business rules.

According to the rule classification from [22],[10], we distinguish two categories of rules: structural business rules and behavioral business rules. We also distinguish between behavioral rules that have a global scope (represent system invatiants) and those that have limited scope (must hold for a given process, activity or context). We use *Alloy facts* to specify the rules that must hold for entire model (i.e. structural business rules and behavioral rules that are system invariants). For example, *"Every customer record must contain at least one valid billing address.*" We use Alloy predicates to model the business rules with a clear scope or context, for example, *"If an order request from a new customer is received, this customer has to be registered in the enterprise system."*

Whereas some behavioral rules can be seen as restrictions or *prohibitions*, thus modeled with Alloy facts and predicates, other business rules have a different nature, for example, *"Strategic customers must always be able to submit the order with GR"*. This business rule is not a restriction, but a *necessity* - a property that has to be ensured or provided despite of any other conditions.[2] We model this type of business rules using *Alloy assertions*.

4 Alloy Specification for Order Processing

4.1 Order Processing: Data Structure

The data structure for the order processing is modeled using Alloy signatures as illustrated below.

```
abstract sig GR {
   orderConfirmedSet: set Order,
   orderDeliveredSet: set Order,
   orderPaidSet: set Order,
   partSet: set Part,
   customerSet: set Customer
}
```

```
one sig GR_pre extends GR {
   orderRequest: one OrderRequest
}
one sig GR_post extends GR {}
```

[2] This distinction has been already proposed in [15], where two modal operators are defined: necessity (with its negation: possibility) and obligation (with its negation: prohibition).

Alloy signatures (**sig**) can be abstract or concrete, can have explicit cardinalities and can contain one or multiple fields. Each field indicates a relation to a corresponding object type and can be considered as an analogy of attributes in object-oriented (OO) languages. For the order processing example, we specify a system - GR - as an Alloy signature illustrated above, characterized by the following fields:

partSet - the set representing all parts (watch components) that can be ordered;
customerSet - the set of customers registered in the GR information system;
orderConfirmedSet - the set of orderes created and confirmed in GR;
orderDeliveredSet - the set of orders (subset of created and confirmed orders) delivered to their customers;
orderPaidSet - the set of orders (subset of created and confirmed orders) paid by the customers.

Similarly to [23], we adapt the state-oriented perspective and specify the execution of order processing in terms of a *state transition:* we define a **pre-state** - GR_pre - that describes the state of a system (GR) before the order processing has been performed and the **post-state** - GR_post - that describes the condition that must hold for the system upon the activity termination.

Once the data structure is defined, we specify how the order processing will be executed (behavior).

4.2 Order Processing: Modeling Behavior

Order processing and its three component processes are modeled as Alloy predicates. These predicates specify a transition between *GR_pre* and *GR_post* states. The proposed specifications are represented as *"black box"*: they do not show *how* the corresponding processes are executed but only the *final result* of their execution visible in GR (i.e. how the GR attributes *orderConfirmedSet*, *orderPaidSet* and *orderDeliveredSet* will change upon the process termination). For example, the *orderCreation* predicate (lines 1-10) declares that *the new order must be created and added to the orderConfirmedSet in the post-state GR_post (i.e. upon the order creation termination).* Along these lines order delivery and payment are specified (lines 11-18).

The *orderProcessing* predicate (lines 19-20) specifies that upon the order processing termination, three processes (order creation, order delivery and order payment) must be accomplished. In Alloy, this corresponds to a logical conjunction of *orderCreation*, *orderPayment* and *orderDelivery* predicates.

4.3 Order Processing: Business Rules

To complete our model of order processing from Section 3.1, we model the following business rules in Alloy: *BR1.1, BR1.2, BR2.1, BR3.1* and *BR4.7.*

The business rules *BR1.1* and *BR1.2* have an explicit scope - the order creation process. According to our BR taxonomy presented in 3.2, these rules are modeled with Alloy predicates *customerExists* and *partExists*:

```
//BR 1.1:  A customer order can be created only for the customers registered in EIS.
pred customerExists{
    one c: Customer|
    (c.name = OrderRequest.name)
    and (c.address = OrderRequest.address)
    and (c in GR_pre.customerSet) }
//BR 1.2: A customer order can be created only for the parts existing in the product catalog.
pred partExists{
    one p: Part|
    (p.partID = OrderRequest.requestedPartID)
    and (p.partInfo = OrderRequest.partInfo)
    and (p in GR_pre.partSet)}
//BR... <other BR for Order creation>
pred orderCreation {
    customerExists and partExists and ... =>   .... }

1. pred orderCreation {
2.     customerExists and partExists and ... =>
3.     (one aNewOrder: Order| one aCustomer: Customer| one aPart: Part |
4.     aPart =
5.     findPartByPartID[GR_pre.orderRequest.requestedPartID,GR_pre.partSet]
6.     and aCustomer=
7.     findCustomerByName[GR_pre.orderRequest.name,GR_pre.customerSet]
8.     and aNewOrder=createOrder[aPart, aCustomer] and
9.     GR_post.orderConfirmedSet=GR_pre.orderConfirmedSet+ aNewOrder)
10.    else GR_post.orderConfirmedSet=GR_pre.orderConfirmedSet }
11.pred orderPayment {
12.    one aCustomer: Customer | one o: Order | (o in aCustomer.oHistory)
13.    and !(o in GR_pre.orderPaidSet)
14.    and GR_post.orderPaidSet=GR_pre.orderPaidSet +  o }
15.pred orderDelivery {
16.    one aCustomer: Customer | one o: Order | (o in aCustomer.oHistory)
17.    and !(o in GR_pre.orderDeliveredSet)
18.    and GR_post.orderDeliveredSet=GR_pre.orderDeliveredSet +  o }
19.pred orderProcessing{
20.    orderCreation and orderPayment and orderDelivery }
```

BR1.1 states that before we start order creation (pre-state), the customer (order requestor) has to be registered in the *customerSet*. Along these lines, *BR1.2* states that for order creation requested part should exist in the *partSet*. This predicate is called within its scope - *orderCreation*. As a result, a new order will be created upon *orderCreation* only if the specified rules are respected.

The business rules *BR2.1* and *BR3.1* must hold for the entire model of order processing. They are modeled with Alloy facts[3] *eventuallyDelivered* and *eventuallyPaid*:

```
//BR2.1: Every confirmed customer order must be eventually delivered to the customer.
fact eventuallyDelivered {
    all o: Order |
    orderDelivery and (o in GR_pre.orderConfirmedSet)
    => (o in GR_post.orderDeliveredSet }
//BR3.1: Every confirmed customer order must be eventually paid by the customer
fact eventuallyPaid {
    all o: Order |
    orderPayment and (o in GR_pre.orderConfirmedSet)
    => (o in GR_post.orderDeliveredSet }
```

These facts claim that when delivery (payment) of orders is provided, all the existing orders will be eventually delivered (paid) in post-state.

[3] Alternatively, if the scope of the model is larger (i.e. it covers not only Order Processing but other activities of GR), these BR can be modeled with predicates as in the previous example.

With the rule *BR4.7*, we ensure that the strategic customer can always create a new order despite of any other conditions. This is the "necessity" rule according to our taxonomy from 3.2; we model this rule with an Alloy assertion.

```
//BR4.7 Strategic customers must always be able to submit the order with GR.
strategicCustomerCanAlwaysOrder: check{
    all c: StrategicCustomer |
    (orderCreation and (c.name = OrderRequest.name) and
    (c.address = OrderRequest.address) and partExists) =>
    (one o: Order |
        o.ocCustomer = c and
         o.ocPart.partID = OrderRequest.requestedPartID and
         (o.ocPart.partInfo = OrderRequest.partInfo) and
          !(o in GR_pre.orderConfirmedSet) and
            (o in GR_post.orderConfirmedSet)) }
```

The assertion claims that whenever there is an order request for the strategic customer for the existing part, the new order is created in *EIS*. This assertion can be checked for our model and must be always valid, which means that this business rule is respected in the system. In the case we get counterexamples, the rule is not respected and we need to revise the model.

5 Business Rules Discovery and Validation with Alloy Analyzer

In this section, we demonstrate how the Alloy Analyzer tool can assist in interactive discovery and validation of business rules that have been missing/omitted/implicit in the initial business specification of Order Processing. We terminate this section by generalizing our approach in a form of four steps an analyst needs to accomplish.

5.1 Order Processing: Model Simulation and Business Rules Discovery

Our approach to BR discovery is based on simulation. The objective of model simulation is two-fold: first, to check our model for consistency (absence of contradictory constraints in business rules); and second, to test the random set of model instances generated by Alloy Analyzer. These instances, in our case, represent the scenarios of order processing enabled by our created model.

The Alloy Analyzer generates the model instances in a form of visual diagrams. Examining these diagrams, an analyst identifies the scenarios indicating implicit, missing, or incorrect business rules. We call this phase a business rule discovery.

Simulating the Alloy model for order processing, we investigate how the status of a customer orders changes during the order processing activity. For a given order, this status can be identified by analyzing the *orderConfirmedSet*, *orderDeliveredSet* and *orderPaidSet* of GR. Note that the same order can be in one or multiple sets at a time. For example, if the order is in the *orderPaidSet* - it is paid. Consequently, if the order is not in *orderPaidSet* in pre-state, but

is added into *orderPaidSet* in post-state upon termination of a given activity, it means that it has been paid.[4]

Simulating our model, we find the instances where the customer can make a new order, while some of his previous orders (order history) that have been already delivered remain "unpaid". This scenario is illustrated in Fig. 1. The scenario shows a regular customer (parallelogram on top) creating an order. This customer is associated with 2 orders: *Order0* and *Order1* (black rectangles). *Order 0* is a newly placed order (not in the *orderConfirmedSet* in pre-state); *Order 1* is an old order that is delivered but unpaid (the status is indicated in the bottom of the corresponding rectangle). In the post-state, the new order *Order0* is confirmed (added into *orderConfirmedSet*) and placed in the oder history, meaning that it was accepted by the system.

The presence of this scenario need to be analyzed by a domain specialist as it potentially can bring to a company a lot of unpaid orders and short or long-term loses. The domain specialist decides whether he needs to define new BRs to restrict this behavior and to protect the interest of the company.

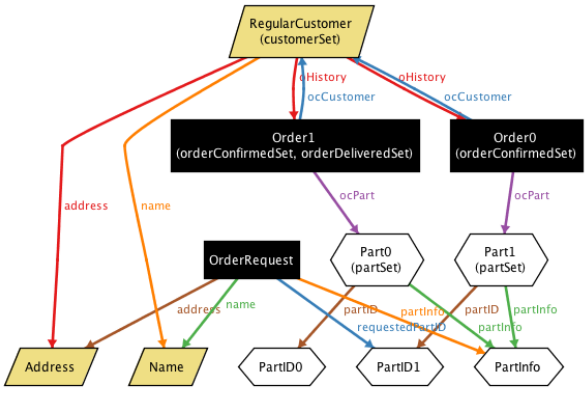

Fig. 1. Order Accepted with Unpaid Orders in the History

5.2 Order Processing: Model Simulation and Business Rules Analysis

We provide a domain specialist with an instant feedback, helping him to reason about the existing business rules, to interactively discover new or implicit business rules and, eventually, to improve their enforcement. Once the new rule is specified by the domain specialist, it can be translated to Alloy for further model analysis. The new business rule covering this business case is:

– BR4.8. New order from a customer can be accepted only if all delivered orders in the customer's order history are paid.

[4] The statuses cannot be canceled, i.e. once the order is paid, it cannot be "unpaid", etc.

This business rule has an explicit context - order creation. We model it with Alloy predicate *customerMustPayDeliveredOrdersBeforeNewOrder*.

```
pred customerMustPayDeliveredOrdersBeforeNewOrder{
    all c: Customer | all o:Order |
    (c.name = OrderRequest.name and c.address = OrderRequest.address and
    o in c.oHistory and !(o in GR_pre.orderDeliveredSet))
    => (o in GR_pre.orderPaidSet)}
```

This predicate claims that, for order creation, all orders from customer's history that are already delivered have to be paid. We modify the *orderCreation* predicate to add this BR.

```
pred orderCreation{
    customerExists and partExists and
    customerMustPayDeliveredOrdersBeforeNewOrder ... =>     .... }
```

As a result, simulating the model, we observe that there are no cases where the order is created and unpaid delivered orders exist for a given customer. However, when we check the validity of the business rule for strategic customer, by checking the assertion *strategicCustomerCanAlwaysOrder*, we receive the result that it is not valid. The counterexamples show the cases where this rule is not respected - no new orders are added in *orderConfirmedSet* for the given order request (Fig. 2). In this example, we see that the new order is not created for the strategic customer, invalidating *BR4.7*. As strategic customers are of crucial importance for GR, they should bypass the new rule and be able to order, even if they have some unpaid orders. In order to resolve current conflict, GR domain specialist modifies the rule 4.8:

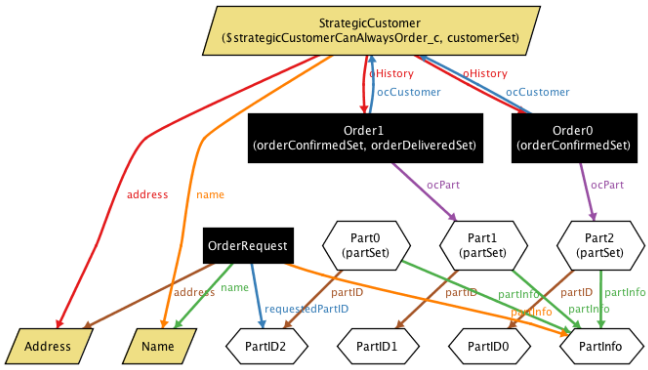

Fig. 2. Example - New Order Not Created For Strategic Customers Because of Unpaid Past Orders

– BR4.8 New order from regular customer can be accepted only if all delivered orders in the customer's order history are paid.

```
pred regularCustMustPayDeliveredOrdersBeforeNewOrder{
    all c: RegularCustomer | all o:Order |
    (c.name = OrderRequest.name and
    c.address = OrderRequest.address and
    o in c.oHistory and
    !(o in GR_pre.orderDeliveredSet))
    => (o in GR_pre.orderPaidSet)}
pred orderCreation {
    customerExists and partExists and customerMustPayDeliveredOrdersBeforeNewOrder ...=> ....}
```

If we check now the assertion *strategicCustomerCanAlwaysOrder*, we do not obtain any counterexamples, which means that the rule is valid again. If we run the order creation process, we can obtain the instances showing that the strategic customer can order even with unpaid orders in his history. Alternatively, the domain specialist can provide the rules for strategic customers, which limit the number of unpaid orders or their total amount.

5.3 Interactive BR Discovery Process: Four Steps

We generalize our approach and define the four steps that a business analyst, business rules designer and/or domain specialist can accomplish in order to systematically discover the business rules and to analyze the issues related to them.

1. The business analyst specifies the BR in a natural language;
2. The designer classifies the BR according to their scope and nature (see Section 3.2) and translates them to Alloy specification language. He also models the whole system of interest (a process, an activity etc) in Alloy, so that he can detect how the business rule influence the system behavior.
3. The designer simulates the model with Alloy Analyzer tool and examines the model instances. [5] Model instances reveal the issues with the existing business rules and indicate the missing or implicit rules. "Necessity" business rules can be validated or invalidated by checking corresponding Alloy assertions.
4. Once a new business rule is discovered, the business analyst specifies this rule in a natural language. Then this rule is added to the Alloy model for further simulations.

6 Conclusion

In this paper, we have presented an interactive, simulation-driven approach for business rule discovery. Our approach is based on formal model checking with the Alloy Analyzer tool. We specify business rules as constraints in Alloy: three types of constraints (facts, predicates and assertions) can be used depending on the type of business rule (necessity/possibility, obligation/restriction) and on their scope.

[5] If no such instances are generated - this indicates the presence of contradictory rules in the model.

We have illustrated our approach with the example of order processing. We demonstrate how the order processing activity and the business rules for it can be modeled in Alloy. We also illustrated how the Alloy Analyzer tool can assist in interactive discovery and validation of business rules. Though formal methods are rarely considered in business, our findings demonstrate that, thanks to their rigor, these methods can support companies, enabling more systematic discovery, analysis and validation of their business rules.

The next step in our approach is to make the language for business rules specification closer to business analyst. One way to do this is to use Attempto Controlled English (ACE) [24], a controlled natural language, i.e. a rich subset of standard English designed to serve as knowledge representation language. This would enable analysts to express the business rules with rigor and in terms of their respective application domain.

References

1. Boyer, J., Mili, H.: Agile Business Rule Development: Process, Architecture, and JRules Examples. Springer (2011)
2. Jackson, D.: Alloy Analyzer tool (2013), http://alloy.mit.edu/alloy/
3. Morgan, T.: Business rules and information systems: aligning IT with business goals. Addison-Wesley Professional (2002)
4. ILOG, I (2013), http://www-01.ibm.com/software/websphere/ilog/
5. Advisor, F.B. (2013), http://www.fico.com/
6. Pegasystems (2013), http://www.pega.com/
7. Berstel-Da Silva, B.: Verification of business rules programs (2012)
8. Nagl, C., Rosenberg, F., Dustdar, S.: Vidre–a distributed service-oriented br engine based on ruleml. In: 10th IEEE International, EDOC 2006, pp. 35–44. IEEE (2006)
9. Orriëns, B., Yang, J., Papazoglou, M.P.: A framework for business rule driven service composition. In: Benatallah, B., Shan, M.-C. (eds.) TES 2003. LNCS, vol. 2819, pp. 14–27. Springer, Heidelberg (2003)
10. OMG: OMG: Semantics Of Business Vocabulary And Business Rules (SBVR) - Version 1.0. OMG Document Number: formal/2008-01-02 (2008)
11. Nelson, M.L., Rariden, R.L., Sen, R.: A lifecycle approach towards business rules management. In: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, pp. 113–113. IEEE (2008)
12. Ross, R.G.: Principles of the BR approach. Addison-Wesley Professional (2003)
13. Halpin, T.A., Morgan, A.J., Morgan, T.: Information modeling and relational databases. Morgan Kaufmann (2008)
14. Dietz, J.L.: Enterprise ontology: theory and methodology. Springer (2006)
15. Dietz, J.L.: On the nature of brs. Advances in Enterprise Engineering I, pp. 1–15 (2008)
16. Jackson, D.: Software Abstractions- Logic, Language and Analysis. MIT Press (2011)
17. GR: Generale ressorts site, http://www.generaleressorts.com/ (2013)
18. Jackson, D., Schechter, I., Shlyakhter, I.: ALCOA: The Alloy constraint analyzer. In: Proceedings of the 22nd ICSE, Limerick, Ireland (June 2000)
19. Rychkova, I.: Formal Semantics for Refinement Verification of Enterprise Models. PhD thesis, EPFL (2008)

20. OMG: OMG: Object Constraint Language - Version 2.2. OMG Document Number: formal/2010-02-01 (2010)
21. Bajic-Bizumic, B., Wegmann, I.R., Towards, A.: a invariant-based service design process. Technical report, EPFL (2013)
22. Hay, D., Healy, K.A.: Defining brs -what are they really. Final Report (2000)
23. Andersson, T., Bider, I., Svensson, R.: Aligning people to business processes experience report. Software Process: Improvement and Practice 10(4), 403–413 (2005)
24. ACE: Attempto Controll English (2013), http://attempto.ifi.uzh.ch/site/