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 in the center of attention, both in the business world and in the IT world. Very often in a real systems, there is hundreds and thousands of rules to be captured. 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.
In this work, we propose an interactive, simulation-driven approach for business rules discovery. The rules are specified in natural language and then translated to the Alloy declarative language. They are then simulated using the Alloy Analyzer tool and the new or implicit rules are detected. Finally, the model is adjusted to synchronize the existing rules with the new rules. This way, the domain specialist together with the designer can interactively discover new business rules and check their consistency and validity.

Key words: Business rules, Business Rule Discovery, Alloy, Requirements Elicitation

1 Introduction

Business rules are everywhere. Every enterprise process, task, activity, or function is governed by rules. However, 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 business rules discovery. This approach will provide a domain specialist with an instant feedback helping her to reason about the existing business rules, to discover new or implicit business rules and, eventually, to improve their enforcement.

In addition, "capturing the logic of an entire business would require probably many thousands of rules; a smaller subsystem, perhaps several hundreds" [2]. They are often made by different people at different time, driven by different business ideas in mind. The rules consistency or the need to validate that a new rule does not conflict with the existing rules is a real problem. Our approach is based on rule modeling and simulation in Alloy Analyzer [3]; use of formal specifications and model checking techniques allows us not only to *discover* the new rules but also to preserve validate their consistency.

We illustrate our approach on the example of the Order Processing of Générale Ressorts SA: we formalize the existing business rules in Alloy, then we simulate the Alloy model and analyze the generated instances. We discover a cluster of instances that is derived from the Alloy model and is not explicitly covered by the existing business rules. We generalize the property of this cluster as a specific business case that, in fact, must be documented (restricted) with a corresponding business rule.

We show that the new rule added to the model makes the model inconsistent and, thus, requires adjustments. We provide the solution, first, in Alloy, and then, explain its meaning in business terms.

The reminder of this paper is organized as follows: In Section 2, we discuss our motivation and present the related works; In Section 3, we introduce our example - the Order Processing, specified for Générale Ressorts SA; then we formalize this example in Alloy and illustrate the BR discovery with Alloy Analyzer; In Section 4, we generalize our approach and formulate our findings as XX steps to interactive BR discovery based on formal model checking; in Section 5 we present our conclusions.

2 Motivation and Related Work

According to [2], "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 (BR). 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 an 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 orientated 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 [2] the main phases are: discovery, definition, review, maintenance. By [1] the main phases are: discovery, analysis, design, authoring, validation, deployment. In [11] these phases are: plan, capture, organize, author, distribute, test, 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 impacting 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 to realize what constraints are missing in the model. We propose the approach based on Alloy simulation that helps to discover new BR in an interactive way, while ensuring the consistency and validity of the overall set of BR.

Although BR traditionally are expressed in 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; 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 Business Rules Discovery with Alloy Analyzer

In this section, we present the process of interactive discovery of business rules, which is based on formal model checking and implemented using the Alloy Analyzer tool [3]. We start with introducing our example - the order processing, specified for Générale Ressorts SA. First, we specify the order processing and its associated business rules in natural language; then, we introduce Alloy specification language [16] and model our example in Alloy. Finally, 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.

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 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 may take months after the product is delivered[1]. Therefore, flexible business rules for order processing and customer transactions management are essential for GR. Générale Ressorts.

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

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 order-to-cash cycle (Fig. 1 [18]).

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 - a part; the confirmed order is then prepared and delivered to customer. As stated above, 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.
   BR1.4 ...
2. *Order delivery*
   BR2.1 Every confirmed customer order must be eventually delivered to the customer.
   BR2.2 ...
3. *Accounting*
   BR3.1 Every confirmed customer order must be eventually paid by the customer.
   BR3.2 ...
4. *Customer management*
   BR4.1 Every customer record must contain one customer name.
   BR4.2 Every customer record must contain at least one valid billing address.
   BR4.3 Every customer record must contain at least one valid shipping address.
   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.
   BR4.8 ...

3.2 Alloy

Alloy [19] 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 [3] 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

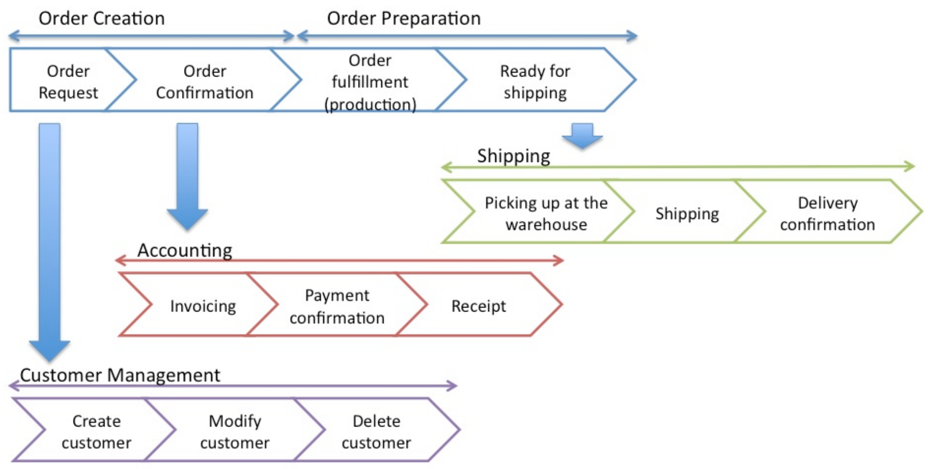


Fig. 1: Order-to-cash cycle - Adapted from [18]

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. [20]

The syntax of Alloy is similar to the syntax of OCL the Object Constraint Language for UML [21]. Data structures are represented with **signatures** labelled by the keyword sig and **fields**.

Alloy reusable expressions (i.e. functions) and constraints (i.e. facts, predicates and assertions) [16] can be used to reason about data structures and to define the relationships between them.

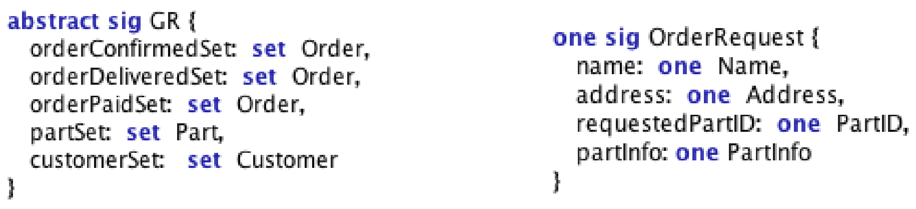
In our previous work [22] we defined the iterative process for service designe where the Alloy signatures, facts and predicates were used for service specification. In this work, we extend the use of Alloy constructs: in particular, we use Alloy constraints for modeling and validation of Business Rules.

By their definition, business rules are intended to assert business structure or to control or influence the behavior of the business [23]. Structural rules define the business information model. A behavioral rule, on the other hand, 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.

Fact is a model constraint that permanently holds. We use *Alloy facts* to specify 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* - is a structural business rule that can be modeled as an Alloy fact associated with a corresponding data structure specifying customer. **Predicate** is a constraint that holds in specific context or for a specific part of the model only. 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.* - this rule has a clear scope where it must be applicable. It can be modeled as an *Alloy predicate.* Whereas some business rules can be seen as restrictions or *prohibitions* and, 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.* **Assertion** is a property that the designer believes should be implied from the model and can check if it can be deduced from the other (permanent or contextual) constraints.

3.3 Alloy Specification for Order Processing

Order Processing in Alloy The data structure for the order processing is modeled using Alloy signatures as illustrated in Fig. 2.



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

(a) Signature GR

```
one sig OrderRequest {
  name: one Name,
  address: one Address,
  requestedPartID: one PartID,
  partInfo: one PartInfo
}
```

(b) Signatures *OrderRequest*

Fig. 2: Design - Alloy Signatures

Alloy signatures (**sig**) can be abstract or concrete, can have explicit cardinalities (e.g. only **one** *OrderRequest* object can be treated by the service at a time), 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 example `name: one Name` indicates that the *OrderRequest* object has an attribute *name* of the type *Name*. We use *one* and *set* cardinalities for the fields to distinguish "one to one" from "one to many" relations.

For the order processing example, we specify a system - GR - as an Alloy signature illustrated in Fig. 2a that is 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;

[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).

orderDeliveredSet - the set of orders (subset of created and confirmed orders), which are delivered to their customers; *orderPaidSet* - the set of orders (subset of created and confirmed orders), which are paid by the customers.

Similarly to [24], 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.

The objective of our model - is to assist a business analyst in discovery of the (implicit of missing) business rules associated with order processing: to do so, we are going to investigate how the status of a customer orders is changing 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. The statuses cannot be canceled, i.e. once the order is paid, it cannot be "unpaid", etc.

Once the data structure is defined, we specify how the order processing will be executed (behavior). Following the description from Section 3.1, we represent the order processing activity as a concurrent[3] execution of order creation, order payment and order delivery processes.

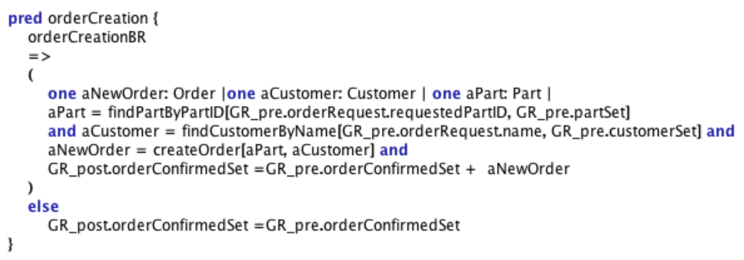
Order processing and its three component processes are modeled as Alloy predicates. These predicates show a transition between *GR_pre* and *GR_post* states. The proposed specifications are *"black box"* - they do not show *how* the corresponding processes are executed but only the *final result* of their execution visible in GR (i.e. how the GR attributes*orderConfirmedSet*, *orderPaidSet* and *orderDeliveredSet* will "look like" upon the process termination).

The predicates specify the conditions that must hold in GR upon order creation/delivery/payment. For order creation this condition is: considering order creation business rules are respected (*orderCreationBR*), the new order must be created and added to the corresponding order set - *orderConfirmedSet* - in the post-state. For order delivery it is: Considering a customer with an undelivered order in his order history (i.e. the order, which is not in the system's *orderDeliveredSet*) in pre-state, this order must be delivered and added to the corresponding set in post-state. For order payment it is: Considering a customer with an paid order in his order history (i.e. the order, which is not in the system's *orderPaidSet*) in the pre-state, this order must be paid and added to the corresponding set in post-state.

Along those lines, the *orderProcessing* predicate (Fig. 3d) specifies that, upon the order processing termination, three processes (order creation, order delivery

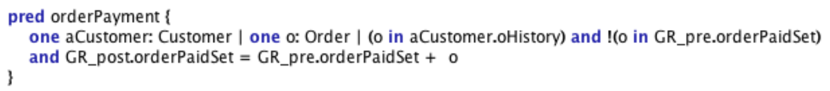
[3] We exploit the declarative approach to activity modeling and avoid the preordering of processes within order processing.

and order payment) must be accomplished. In Alloy, this corresponds to a logical conjunction of *orderCreation*, *orderPayment* and *orderDelivery* predicates.



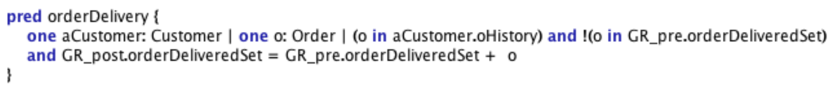
```
pred orderCreation {
   orderCreationBR
   =>
   (
      one aNewOrder: Order |one aCustomer: Customer | one aPart: Part |
      aPart = findPartByPartID[GR_pre.orderRequest.requestedPartID, GR_pre.partSet]
      and aCustomer = findCustomerByName[GR_pre.orderRequest.name, GR_pre.customerSet] and
      aNewOrder = createOrder[aPart, aCustomer] and
      GR_post.orderConfirmedSet =GR_pre.orderConfirmedSet +  aNewOrder
   )
   else
      GR_post.orderConfirmedSet =GR_pre.orderConfirmedSet
}
```

(a) *OrderCreation* process



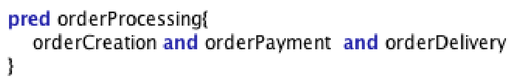
```
pred orderPayment {
   one aCustomer: Customer | one o: Order | (o in aCustomer.oHistory) and !(o in GR_pre.orderPaidSet)
   and GR_post.orderPaidSet = GR_pre.orderPaidSet +  o
}
```

(b) *OrderPayment* process



```
pred orderDelivery {
   one aCustomer: Customer | one o: Order | (o in aCustomer.oHistory) and !(o in GR_pre.orderDeliveredSet)
   and GR_post.orderDeliveredSet = GR_pre.orderDeliveredSet +  o
}
```

(c) *OrderDelivery* process



```
pred orderProcessing{
   orderCreation and orderPayment  and orderDelivery
}
```

(d) *OrderProcessing* activity

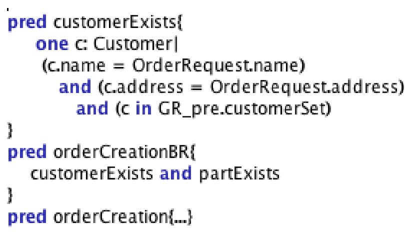
Fig. 3: Order Processing activity as a combination of order creation, order delivery and order payment processes

Business Rules To illustrate our approach, we use subset of the business rules given in Section 3.1. The business rules we select are: *BR1.1, BR1.2, BR2.1, BR3.1* and *BR4.7*. These business rules are shown in Fig. 4.

The business rules *BR1.1* and *BR1.2* are modeled as predicates, as they have an explicit scope - order creation process. In Alloy, *BR1.1* states that the customer has to be registered in the *customerSet* in pre-state - before we start creating the order. This predicate (*customerExists*) is called in predicate *orderCreation* via *orderCreationBR* predicate (Fig. 3a). The predicate *customerExists* together with its call in *orderCreation* claims that the order can be created only if the customer is registered in the *customerSet*. *BR1.2* states that for order creation requested part should exist in the *partSet* in the order creation pre-state. Whenever we run *orderCreation* the rule will be automatically respected, as the new order will be created only if the customer exists in the system. Otherwise, the *orderConfirmedSet* will remain the same (Fig. 3a).

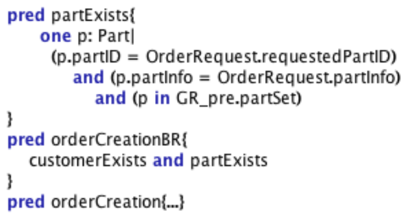
BR2.1 is modeled as a fact. The fact claims that when we do delivery of order, all the existing orders will be eventually delivered in post-state. Similarly, *BR3.1* is represented as a fact that claims the same for order payment. These rules have a global scope.

Finally, *BR4.7* is modeled as an assertion: we have to ensure that the strategic customer can always create new order despite of any other conditions. This is the "necessity" rule that we want always to hold. We write it as assertion, which can be checked in the model. 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*. If the assertion is valid, it means that this business rule is respected in the system. In case we get counterexamples, the rule is not respected and we need to revise the business rules.



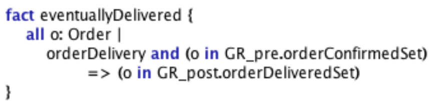
```
pred customerExists{
    one c: Customer|
     (c.name = OrderRequest.name)
       and (c.address = OrderRequest.address)
         and (c in GR_pre.customerSet)
}
pred orderCreationBR{
   customerExists and partExists
}
pred orderCreation{...}
```

(a) BR1.1 - A customer order can be created and confirmed only for the customers registered in EIS.



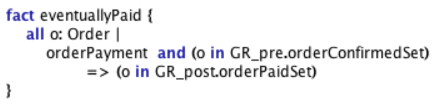
```
pred partExists{
     one p: Part|
      (p.partID = OrderRequest.requestedPartID)
        and (p.partInfo = OrderRequest.partInfo)
          and (p in GR_pre.partSet)
}
pred orderCreationBR{
   customerExists and partExists
}
pred orderCreation{...}
```

(b) BR1.2 - A customer order can be created and confirmed only for the parts existing in the product catalog.



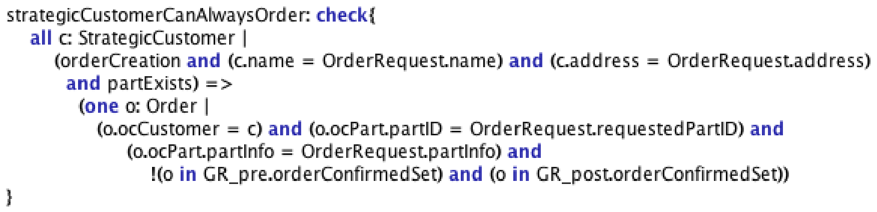
```
fact eventuallyDelivered {
   all o: Order |
     orderDelivery and (o in GR_pre.orderConfirmedSet)
          => (o in GR_post.orderDeliveredSet)
}
```

(c) BR2.1 - Every confirmed customer order must be eventually delivered to the customer.



```
fact eventuallyPaid {
   all o: Order |
     orderPayment  and (o in GR_pre.orderConfirmedSet)
          => (o in GR_post.orderPaidSet)
}
```

(d) BR3.1 - Every confirmed customer order must be eventually paid by the customer.



```
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))
}
```

(e) BR4.7 - Strategic customers must always be able to submit the order with GR.

Fig. 4: Initial Business Rules

3.4 Business Rules Discovery for Order Processing

Discovering New Business Rules Our approach to BR discovery is based on simulation. The idea is to simulate the Alloy model representing the subset of our system of interest (its structure, behavior and initial business rules modeled as Alloy constraints) and to analyze instances generated by the Alloy Analyzer tool. These instances, in our case, represent the scenarios of order processing enabled by our created model. The objective of model simulation is twofold: 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 for discovering (possibly) missing BRs. We call this phase a business rule discovery.

Simulating order creation process, we find the instances where the customer can make new order, even if he did not pay his previous orders that have been already delivered. This scenario is illustrated in Fig. 5. The scenario shows a regular customer (grey parallelogram on top) creating an order. This customer is associated with 2 orders (*Order0* and *Order1*) in his history (black rectangles). *Order 1* is already delivered and unpaid. Despite this, he was able to create the new order *Order0*. We know that this is the new order because we can see in pre-state view of the system that it was not in the *orderConfirmedSet* in pre-state and has appeared in post-state in Fig. 5. This could potentially bring to a company a lot of unpaid orders and short or long-term loses. The domain specialist decides weather he needs to define new BR to restrict this behavior to make sure the interest of the company is protected. In case he does, the designer translates the business rule into Alloy.

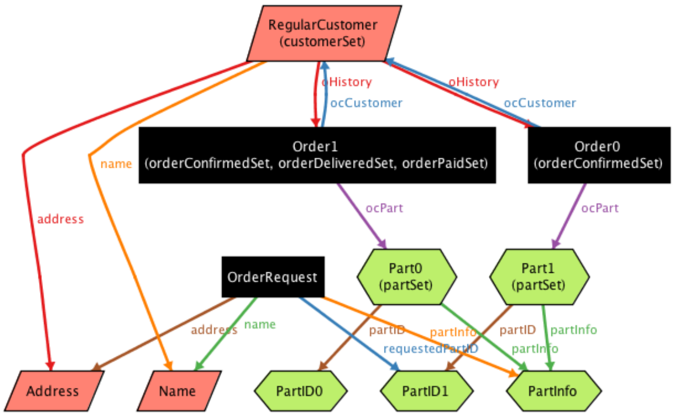


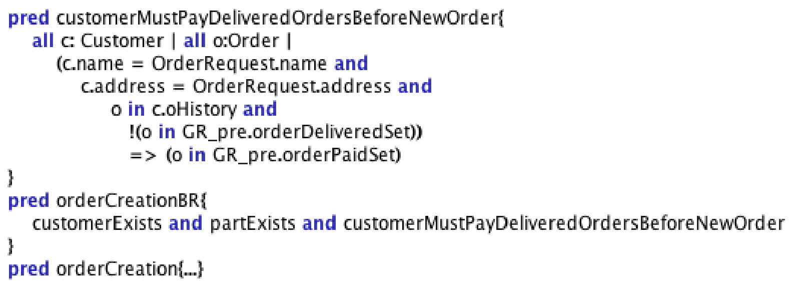
Fig. 5: Order Accepted with Unpaid Orders in the History

Business Rules Consistency Checking 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 missing rule is confirmed by a business (domain) specialist, it can be added to our Alloy model.

The new business rule covering this business case is:

– BR1.3. New order cannot be created for the customer order if there are delivered but unpaid orders in the customer's order history.

This business rule represented in Alloy is in Fig. 6. This business rule has an explicit context - order creation. We model it thus as a predicate. In predicate *customerMustPayDeliveredOrdersBeforeNewOrder*, we claim that the all orders from customer's history that are delivered has to be paid. This predicate is called in *orderCreation* predicate via *orderCreationBR*. With combination of the three predicates, we claim the if we do creation of new order for some customer, his past orders that have already been delivered have to be paid.



```
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)
}
pred orderCreationBR{
   customerExists and partExists and customerMustPayDeliveredOrdersBeforeNewOrder
}
pred orderCreation{...}
```

Fig. 6: BR1.3. New order cannot be created for the customer order if there are delivered but unpaid orders in the customer's order history.

As a result, simulating the model, we observe 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 running the assertion in Fig. 4e, we receive the result that it is not valid. The counterexamples show the cases where this rule is not respected (Fig. 7). In this example, we can see that the new order is not created for strategic customer, meaning *BR4.7* (Fig. 4e) is not valid. As strategic customers are of crucial importance for GR, they should by-pass the new rule and be able to order even if they have some unpaid orders.

Thus, by introducing the new rule, we have created a conflict with the existing rules and overspecified the model. Therefore, we need to relax the rules.

In order to resolve current conflict, GR defines separate rules for regular and strategic customers:

– BR7. For regular customers, the new order cannot be accepted if there are unpaid orders in the order history.

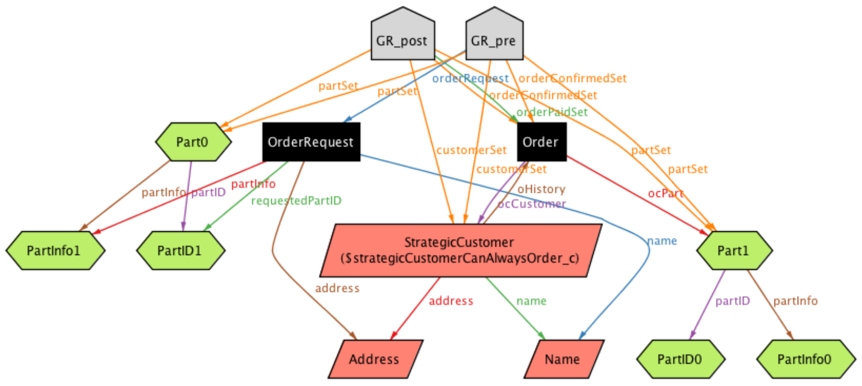
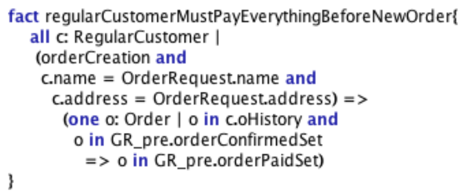


Fig. 7: Example - New Order Not Created For Strategic Customers Because of Unpaid Past Orders

We replace the rules *BR4.7* and *BR6* with the two new rules. These rules in Alloy are shown in Fig. **??**. *BR7* claims that if the order is created for the regular customer, all his past orders must be paid. *BR8* claims that if the order is created for strategic customer, there can be maximum 2 unpaid orders for that customer.



```
fact regularCustomerMustPayEverythingBeforeNewOrder{
    all c: RegularCustomer |
    (orderCreation and
     c.name = OrderRequest.name and
       c.address = OrderRequest.address) =>
         (one o: Order | o in c.oHistory and
           o in GR_pre.orderConfirmedSet
             => o in GR_pre.orderPaidSet)
}
```

Fig. 8: BR7 - Regular Customers Must Pay Past Orders Before Placing New Order

If we check now the assertion in Fig. 4e, we get no counterexamples, showing that the rule is valid again. If we run order creation process we can obtain the instances showing that the strategic customer can order even with unpaid orders in his history (Fig. 9).

Option 2: BR1: For regular customers, the confirmed order must be delivered only upon the payment received; BR2: For large accounts, the confirmed order must be delivered; BR3: For large accounts, the confirmed order must be eventually payed; .... BR1000045: the large accounts must always be able to submit the order.

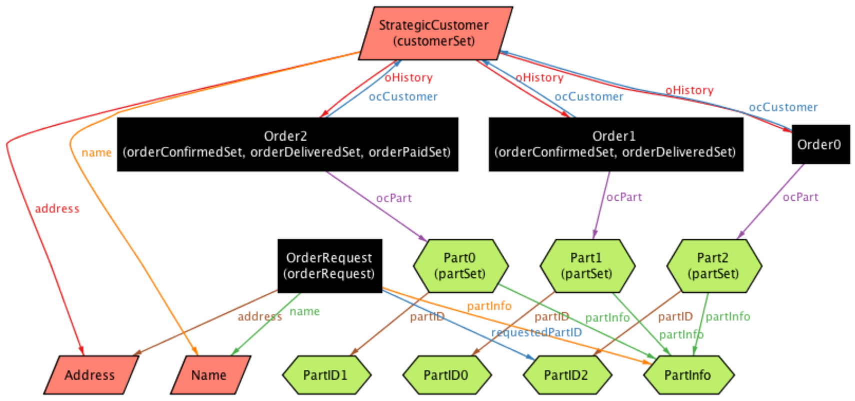

StrategicCustomer (customerSet)
oHistory
ocCustomer
Order2 (orderConfirmedSet, orderDeliveredSet, orderPaidSet)
Order1 (orderConfirmedSet, orderDeliveredSet)
Order0
name
address
ocPart
OrderRequest (orderRequest)
Part0 (partSet)
Part1 (partSet)
Part2 (partSet)
partInfo
partID
requestedPartID
Address
Name
PartID1
PartID0
PartID2
PartInfo

Fig. 9: Strategic Customer Can Make New Order With Unpaid Orders

4 An Interactive Discovery and Validation of Business Rules: ???

In this section, we explain the general process for interactive automated discovery of business rules with Alloy Analyzer. The basic steps of our simulation-driven business rule discovery approach are:

4.1 BR Specification in the Natural Language

The business analyst specifies the rules in the natural language. These are the initial business rules he elicited from different roles in the company. Very often these set of rules is not covering some implicit rules. The following steps help him detect the new business rules.

4.2 Specification in Formal Language

The designer translates the business rules in natural language to Alloy declarative language. He also specify the whole model in Alloy (including data structure and behavior), so that he can detect how the business rule influence the company behavior. The business rules in Alloy can be represented as Alloy facts, predicates or assertions depending on the type of the business rule, as explained in Section 3.2. This way, the designer has the power to express different scope of business rules, which can be very useful.

4.3 Simulation and Analysis

The designer simulate the model with the initial business rules using Alloy Analyzer tool. The objective of model simulation is twofold: 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 for different kind anomalies. If underspecification anomaly appears - the "unwanted/unexpected" behavior is identified - the analyst should confirm that it is correct to have this instances or detect is as a problem. In the second case an explicit rule need to be specified for the cluster of "unwanted/unexpected" instances.

4.4 Adding New BR

The business analyst specifies the new business rule in natural language. The designer translates the business rule to Alloy language and simulate again the model. New rule can potentially be in conflict with the existing one, can add unnecessary constraints to the model and overspecify the model. If the business analyst confirm this is the case, the business rules should be adapted.

4.5 Adjusting Model to New BR

In case the overspecification appears, the rule should be replaced by considering different cases when it should apply. The business analyst gives proposition of the changes of business rules. The designer translates relaxed business rules into Alloy and simulate the model to confirm with business analyst everything is correct now.

The steps 4.3, 4.4 and 4.5 are repeated until there is no anomalies detected and business analyst confirms that the system behaves by his expectations.

5 Conclusion

We have presented an interactive approach for business rule discovery based on simulations in Alloy Analyzer tool. Business rules are represented as Alloy facts, predicates or assertions, depending on the type of business rule (necessity/possibility, obligation/restriction; or maybe to put our categories allowance/obligation/restriction/possibility - but maybe this is the same :) if the necessity is the same as allowance). This enables to define precise scope of business rule - do they apply to the whole model, just to the given context or they should always be implied from the model.

The business analyst defines the initial business rules in natural language, which the designer translates to Alloy formal language. The designer also specifies the model of the company. The model with the initial business rules in then simulated using the Alloy Analyzer tool. During the simulation, the designer checks if the business rules are consistent and checks together with business analyst if some anomalies appear in the behavior. In case they appear, business analyst suggest new rules in natural language, which the designer translates to the formal language. Finally, the model should be adjusted to the new rule, since the inconsistencies could appear between the existing and the new business rule.

This way, the business analyst can interactively analyze the existing business rules and discover the new and implicit business rules.

The next step in our approach is to make the language for business rules specification closer to the business analyst. One way to this is to use Attempto Controlled English (ACE), a controlled natural language, i.e. a rich subset of standard English designed to serve as knowledge representation language. in this way, we could allow the analyst to express texts precisely, and in the terms of their respective application domain and perhaps to avoid having the designer as an intermediate step of business rule discovery.

References

1. Boyer, J., Mili, H.: Agile Business Rule Development: Process, Architecture, and JRules Examples. Springer (2011)
2. Morgan, T.: Business rules and information systems: aligning IT with business goals. Addison-Wesley Professional (2002)
3. Jackson, D.: Alloy Analyzer tool. http://alloy.mit.edu/alloy/ (2013)
4. ILOG, I. http://www-01.ibm.com/software/websphere/ilog/ (2013)
5. Advisor, F.B. http://www.fico.com/ (2013)
6. Pegasystems. http://www.pega.com/ (2013)
7. Berstel-Da Silva, B.: Verification of business rules programs. (2012)
8. Nagl, C., Rosenberg, F., Dustdar, S.: Vidre–a distributed service-oriented business rule engine based on ruleml. In: Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International, IEEE (2006) 35–44
9. Orriëns, B., Yang, J., Papazoglou, M.: A framework for business rule driven service composition. Technologies for E-Services (2003) 14–27
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: Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, IEEE (2008) 113–113
12. Ross, R.G.: Principles of the business rule 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 business rules. Advances in Enterprise Engineering I (2008) 1–15
16. Jackson, D.: Software Abstractions- Logic, Language and Analysis. MIT Press (2011)
17. GR: Generale ressorts site. http://www.generaleressorts.com/ (2013)
18. OTC: Order-to-cash cycle. http://www.three2tango.com/general/business/order-to-cashotc.html/ (2013)
19. Jackson, D., Schechter, I., Shlyakhter, I.: ALCOA: The Alloy constraint analyzer. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland (June 2000)
20. Rychkova, I.: Formal Semantics for Refinement Verification of Enterprise Models. PhD thesis, EPFL (2008)

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