The Role of Invariants in the Co-evolution of Business and Technical Service Specification of an Enterprise

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

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

Abstract. We explore invariants as a linking mechanism between the business and technical service perspectives: From the business perspective, invariants can be used to model (business) requirements of an enterprise; from the technical perspective, invariants express the properties that must hold during the execution of a service.
We propose an approach to enterprise service design that can be described as an iterative introduction and a modification of invariants in response to the evolution of business and/or technical service specifications. We formalize the service specifications in Alloy and demonstrate how each design iteration can be simulated, visualized and validated with the Alloy analyzer tool. We illustrate our findings with the example of Order Creation service.

Keywords. Enterprise Modeling, Service Design, Service Simulation, Alloy, Declarative Specification, Model Checking, Business Rules

1 Introduction

A considerable gap between business and technical worlds (often referred to as the business/IT alignment problem [1]) represents a serious issue for implementing the "co-evolution" of business and technical specification of a service in service design and development. Therefore, we explore the invariants as a linking mechanism between business and technical service perspectives.

We propose a method for agile service specification that extends Systemic Enterprise Architecture Method (SEAM) [2]. SEAM models can be used by business and technical specialists to visually *describe* an enterprise system, its structure and services it provides. We propose a method that allows us to *simulate and validate* visual service specifications defined in SEAM. It defines five design activities (design, simulation and simulation-based testing, analysis and anomaly resolution, validation, refinement) that can be performed sequentially

or iteratively, forming a *design spiral*, similarly to [4]. Within this spiral, an initial service model evolves in response to the changing business requirements and also makes these requirements evolve by revealing flaws and inconsistencies in them. This way, the partial specification is validated, verified and improved.

We illustrate our method with the example of an "Order Creation" service, specified for Générale Ressorts SA - the Swiss manufacturer of watch springs. This example is based on the consulting project we conducted with this company. The SEAM model for "Order Creation" and the transformation of this model to Alloy remains beyond the scope of this article.

This paper (a) explores the power of Alloy beyond the technical domain (b) investigates how invariants can be used as a linking mechanism between business and technical service perspectives for improved business/IT alignment.

The remainder of this paper is organized as follows: In Section 2 we explain our motivation and discuss the related works. In Section 3, we present the Alloy language and discusses the role of invariants in service design. In Section 4, we introduce our method for service design. In Section 5, we illustrate this method on the case study. In Section 6, we present our conclusions.

2 Motivation and Related Work

Since the first methods dealing with enterprise modeling (EM) that emerged in 1970s, a multitude of enterprise modeling approaches have been developed. e3Value [5] provides an ontology to conceptualize and visualize eBusiness idea and to be able to do an analysis and profitability assessment of the eBusiness model for all parties involved. The i* framework [6] focuses on modeling properties such as goals, beliefs, abilities, commitments; and on modeling strategic relationships. Enterprise Knowledge Development (EKD) [7] is a multi-model, participatory EM approach that involves a model for conceptual structures, and interlinked sub-models for goals, actors, business rules, business processes and requirements to be stated. Business Motivation Model (BMM) [8] models several concepts from goals, down to processes and technologies. The methodology that focuses more on business processes is Dynamic Essential Modeling of Organizations (DEMO) [9], which models, (re)designs and (re)engineers organizations.

SEAM [2] integrates the main principles of the well known EM methods by proposing three different types of models: SAR (business value between different stakeholders-similar to e3Value), goal-belief (goals and beliefs of the stakeholders and their relation-similar to i*), and behavior model (services and processes that implement them-similar to DEMO). In this work, we extend SEAM with the spiral design process that allows simulation and validation of SEAM models in the early stage of the design. This way, the examples of the partial specification could help the designer to realize what constraints are missing in the model. i* uses the similar approach the Formal Tropos [10] to do the model-checking of the models defined in i*. However, it focuses on the agent properties such as goals, beliefs and abilities.

Invariants have been used both in the business and technical world: to represent and check constraints [11], to model business rules [12], process invariants related to beliefs [13] etc. In requirements engineering, KAOS methodology uses invariants for object specification, domain properties specification, and indirectly for goal specification [14]. In this work, we use invariants as a a pivotal concept in improving business/IT alignment and in supporting the co-evolution of technical and business specifications.

Our method is based on Alloy, a lightweight formal specification language. The area of Alloy application is very large[1]. To the best of our knowledge, all current Alloy applications in the domain of EM target technical specialists. In this paper, we present an agile EM method where Alloy diagrams serve as a means for communicating and evaluating both business and technical design decisions. Within our approach, the role of Alloy diagrams is two-fold: They provide an instant visual feedback to a designer that suggest new constraints to be added; They represent design artifacts for validation and can drive improvements of both technical and business specifications (like UML, BPMN).

3 Foundations

3.1 Alloy

Alloy [3] is a declarative specification language for expressing complex structural constraints and behavior based on first-order logic.

The Alloy Analyzer [15] is a tool for the automated analysis of models written in the Alloy 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 tries to find a model instance - a binding of the variables to values making the formula true [16].

Alloy reusable expressions (i.e., functions) and constraints (i.e., facts, predicates and assertions) [17] can be used to reason about data structures. **Fact** is a model invariant: a constraint that holds permanently. **Predicate** is a constraint that holds in a specific context or for a specific part of the model only. It can be seen as a *contextual invariant*. **Assertion** is a property that the designer believes should be implied from the model; he can check if it can be deduced from the other (permanent or contextual) constraints.

In our design process we use signatures, facts and predicates, first for partial and then for refined service specification; we use assertions in order to validate desired properties of our model.

3.2 The Role of Invariants

In computer science, an invariant is a condition that must hold during the execution of a program. Along these lines, in our design process, an invariant defines a

[1] `http://alloy.mit.edu/alloy/applications.html`

condition that must hold for all model instances that result from simulation. We define the role of invariants in our design process as follows: First, they implement the constraints required by business specification. For example, "The order can be placed for the existing parts only"; Second, they enable the designer to efficiently manage the model complexity by assuming that some of its properties always hold during an execution. For example, "To simplify the model, let's consider that the part's id provided by a customer is always correct " (i.e., exists in the database).

These roles correspond to the business and technical perspective. Therefore, in this approach we use them as a linking mechanism between these two worlds to restrict a model prohibiting some (invalid) instances identified during simulation (not necessarily covered by the explicit business specification).

4 Service Design Spiral

We introduce the five activities of our design approach, which can be performed sequentially or iteratively, forming the loops of a spiral as shown in Fig. 1a.

4.1 Model Design

We define a partial model of a service in Alloy: we specify its data structures, the initial predicate and make initial assumptions about our model defining model invariants. These invariants replace the properties required by the business specification and are used to control the model complexity.

4.2 Model Simulation

We simulate our partial model by using the Alloy Analyzer tool. Technically, a partial model written in Alloy represents a logical formula; model simulation means searching for a model instance that satisfies this formula. If it exists, it indicates that the formula is consistent (i.e., no contradictory constraints are specified). In our design process, we first check our model for consistency, and then test if it corresponds to the requirements and if there are some anomalies by studying the random set of model instances generated by Alloy Analyzer.

4.3 Model Analysis and Anomaly Resolution

There are two types of anomalies that can be observed: anomalies due to underspecification and anomalies due to overspecification.

Underspecification means that some model instances that are prohibited by the specification still appear during the simulation. In this case, we restrict the model by adding new invariants.

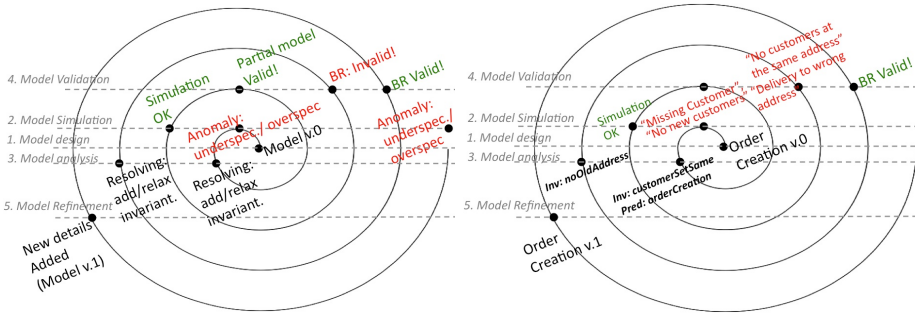
Overspecification means the opposite: some expected model instances are not observed during the simulation. The modeler then has to relax invariant, i.e. to replace an Alloy fact "X always holds" with an Alloy predicate "X holds when..." that can be activated in specific parts of the model only.

4.4 Model Validation

We make assertions about our model in order to test some desirable properties and business rules. Alloy Analyzer validates our assertion by searching for a **counterexample**: a model instance for which our assertion does not hold. If no such counterexample is found, then our assertion is valid within a given value domain. In the opposite case, the model has to be revised.

4.5 Model Refinement

In this activity, we implement new business requirements and extend our partial model. We introduce new elements in a data structure and specify new constraints. Refinement increases both the model complexity and its level of details, bringing it closer to its business specification. The complete design process is illustrated in Fig. 1a.



(a) Service design process spiral.

(b) Design process for "Order Creation" service.

Fig. 1. Spiral design process

5 "Order Creation" Design: The Case Study

In this section, we present an example of using our method for enterprise design, which involves both a technical and a business expert working together to define a complete service specification while maintaining business/IT alignment. We implement our service design process spiral step by step (Figure 1b).

5.1 Case Study: 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 [18]. We illustrate our process by applying it to the design of the "Order Creation" service for Générale Ressorts SA (GR). "Order Creation" is a part of an "Order Processing"; it is followed by "Order Delivery" and "Accounting" (order-to-cash cycle).

An overview of "Order Creation" service is: "The company gets a request from a customer (*OrderRequest*-with customer *name*, *address*, *partID* and *partInfo*[2]) for manufacturing a specific watch component identified by its ID (*partID*). A company agent (*OrderEntryPerson*) identifies the customer and the part to be manufactured by entering the customer's *name* and the *partID* into the enterprise information system (*EIS*). The process terminates with a creation and confirmation of a customer order (*OrderConfirmed*) in the EIS."

We specify the following business rules for our process:

- BR1: The created order must include the complete part specification (to be used for the order fulfillment) and the complete customer details (to be used for product delivery);
- BR2: The order can be confirmed only when the customer exists in the system;
- BR3: The order can be placed for the existing parts only;
- BR4: The company has to guarantee "no faulty delivery".

5.2 Order Creation: Model Design

The data structure for the "Order Creation" service is modeled using Alloy signatures:

```
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 {}
```

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 (as classes and attributes in object-oriented (OO) languages). We can also define additional constraints on the initial data structure with the invariants.

We express the behavior in terms of a *state transition:* we define a **pre-state** that describes the state of a system before the service has been performed and the **post-state** that describes the condition that must hold for the system upon the service termination - the service result. Note, that following the declarative modeling paradigm, we do not specify *how* the service will change the system's state.

We model the "Order Creation" service as a corresponding predicate in Alloy.

[2] We put in *italic* the names that will appear in the Alloy models.

```
1.pred orderCreation(aGR_pre:one GR_pre,aGR_post:one GR_post){
2. one aCustomer: Customer | one aPart: Part | one aOrderConfirmed: OrderConfirmed |
3.
4. aPart=findPartByPartID[aGR_pre.orderRequest.requestedPartID,aGR_pre.partSet] and
5. aCustomer= findCustomerByName[aGR_pre.orderRequest.name,aGR_pre.customerSet] and
6. aOrderConfirmed=createOrderConfirmed[aPart,aCustomer] and aGR_post.orderConfirmed=
7. aOrderConfirmed and aGR_post.orderConfirmedSet=aOrderConfirmed+aGR_pre.orderConfirmedSet}
```

This predicate shows a transition between *GR_pre* and *GR_post* states; these states are indicated as predicate parameters (line 1). In this predicate, the variables are declared (line 2), the customer and the part are found in the set (lines 4-5) and the order is created (line 6) and added to the set (line 7), as described in the case study.

5.3 Order Creation: Model Simulation and Anomaly Resolution

We attempt to simulate this model in Alloy Analyzer: to check our model for consistency and to test the random set of model instances to check for overspecification and underspecification anomalies.

Example 1. "Missing Customer" anomaly Fig. 2 illustrates an anomaly in our model behavior: In a pre-state we have *Customer0*, in a post-state we have *Customer1*. As we show exactly one execution of the service "Order Creation", we expect both the *customerSet* and the *partSet* to remain the same in pre- and post-state. However, the generated instance suggests the opposite.

NOTE: the inputs and outputs in our diagrams (e.g,, *OrderRequest* and *OrderConfirmed* in Fig. 2) are depicted with black rectangles; customer data (*Customer*, *Name*, *Address*) and part data (*Part*, *PartID*, *PartInfo*) are depicted with parallelograms and diamonds, respectively. We depict the pre-state (prior to the order creation service execution) and post-state (upon the service termination) of the GR company with "houses" and the corresponding labels: *GR_pre, GR_post*.

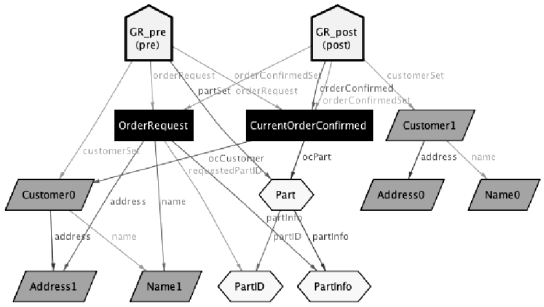


Fig. 2. Anomaly due to Underspecification: "Missing Customer"

This anomaly indicates that some constraints, which should prevent the customer set and the part set from changing during the service execution, have to be specified. Thus, it is an anomaly due to the **underspecified** model.

In fact, the declarative specification principles oblige us to explicitly state the elements that must remain "unchanged" during the state transition. Therefore, we need to add an invariant that states that the *customerSet* in post-state is the same as the *customerSet* in pre-state. The same applies to part set.

```
fact customerSetSame{ GR_post.customerSet = GR_pre.customerSet}
```

In order to validate that we have resolved the "Missing Customer" anomaly, we create an Alloy assertion that claims that *for all Order Creation executions (i.e., model instances), the customer set will remain the same in pre- and post-states of GR.*

```
customerPrePostSame: check{
 all aGR_pre:GR_pre,aGR_post:GR_post |
 orderCreation[aGR_pre, aGR_post] => aGR_post.customerSet=aGR_pre.customerSet}
```

Checking this assertion, we find no counterexamples.

```
Executing ‘‘Check customerPrePostSame’’ Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1
Symmetry=20 1014 vars. 109 primary vars. 1750 clauses. 32ms.
No counterexample found. Assertion may be valid. 12ms.
```

This confirms the assertion validity (for a given model scope). We repeat the simulation until all anomalies are resolved ("design loop" in Fig. 1).

5.4 Order Creation: Model Validation and Anomaly Resolution

We check the validity of each of the business rules from Section 5.1, using Alloy assertions. We show an example of BR4 validation ("no faulty delivery").

Example 2. "Delivery to the Wrong Address" anomaly As *OrderConfirmed* is used for delivery, to ensure "no faulty deliveries" (BR4), we check that the customer and part data in the confirmed order are exactly the same as in the requested order. The assertion "orderConfirmedCorrect" is defined to validate this BR:

```
orderConfirmedCorrect: check {
    all aGR_pre:GR_pre,aGR_post:GR_post,oReq:OrderRequest, oCurrent:CurrentOrderConfirmed |
    orderCreation[aGR_pre, aGR_post] => (oCurrent.ocCustomer.name=oReq.name and
    oCurrent.ocCustomer.address=oReq.address and
    oCurrent.ocPart.partID=oReq.requestedPartID and oCurrent.ocPart.partInfo=oReq.partInfo)}
```

When we run the assertion, we obtain the counterexamples. Fig. 3 shows an example of the incorrect delivery: the order is created on the correct customer's name, but the delivery address associated with this name does not correspond to the address provided in the *OrderRequest*. Therefore, the part can be delivered to the wrong address. The anomaly observed is due to model **underspecification**.

In order to resolve the detected anomaly, we add a new invariant "noOldAddress" that states that we cannot have a customer in the system with the name given in the requested order, but with an old/invalid address and vice versa:

```
fact noOldAddress{all c:Customer | c.address=OrderRequest.address<=>c.name=OrderRequest.name}
```

If we check now the assertion "orderConfirmedCorrect", we get the result "No counterexample found. Assertion may be valid.", meaning that this assertion holds in a given domain, and all orders will be delivered to the correct customers to the correct address.

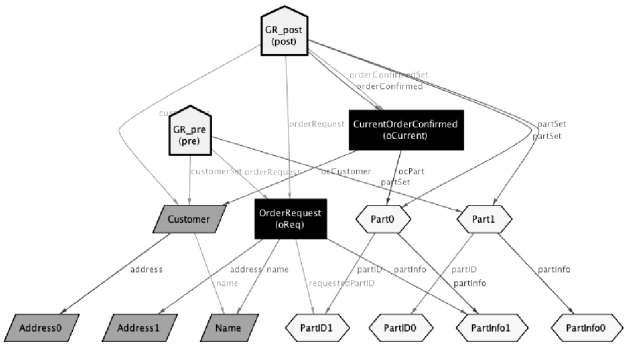


Fig. 3. Anomaly due to Underspecification: "Delivery to the Wrong Address"

We continue "debugging" the model by running the simulations, checking if we have introduced some new unwilling behavior. We repeat the process for other BRs. After validating all BRs and finding no anomalies, we conclude that the designed model meets its business requirements at a given level of details.

5.5 Order Creation: Model Refinement

At the refinement, we can add new data structures and behavior to our model. Then, we resolve all added anomalies, if any, in the "design loop". The next step is to check if the BRs still hold by repeating the "BR validation loop" until all the BRs hold. The refinement specifies a new iteration on the spiral (Fig. 1). The designer can continue refining the model until the desired level of detail is achieved. The design process we propose will ensure that, upon each iteration, the model remains consistent and has no anomalies. Refinement of "Order Creation" service will not be considered in this paper. The resulting design process of "Order Creation" (Fig. 1b) represents an instance of the spiral process illustrated in Fig. 1a.

6 Conclusion

We have presented a lightweight, interactive and visual method for service design that supports the co-evolution of technical and business service specifications of an enterprise. In particular, we have explored the power of Alloy formal method beyond the technical domain and how it can be used as a toolbox for both technical and business specialists.

The evolution of service model in Alloy can be seen as an iterative introduction and modification of logical *invariants*. Invariants represent the assumptions about business or technical properties of a modeled service and, consequently, play the role of a linking mechanism between business and technical perspectives.

This work has illustrated how Alloy can be used as a design environment for both technical and business specialists. For now, we expect that the Alloy diagrams are interpreted and analyzed by designers and business analysts. These specialists trace the observed scenarios back to the specification for its improvement. Automated interpretation and traceability between scenarios generated by Alloy and their specifications (business requirements, business rules, etc) is a subject of our future research.

References

1. Luftman, J., Papp, R., Brier, T.: Enablers and inhibitors of business-it alignment. Communications of the AIS **1**(3es) (1999) 1
2. Wegmann, A.: On the Systemic Enterprise Architecture Methodology (SEAM). In: ICEIS. (2003)
3. 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)
4. Boehm, B.: A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes **11**(4) (August 1986)
5. Gordijn, J., Akkermans, J.: Value-based requirements engineering: Exploring innovative e-commerce ideas. Requirements engineering **8**(2) (2003) 114–134
6. Yu, E.: i* - an agent- and goal-oriented modelling framework. http://www.cs.toronto.edu/km/istar/ (page visited 2013)
7. Kirikova, M., Bubenko, J.A.: Enterprise modelling: improving the quality of requirements specifications. (1994)
8. Montilva, J., Barrios, J.: Bmm: A business modeling method for information systems development. the Clei Electronic Journal **7**(2) (2004)
9. Dietz, J.L.: Understanding and modelling business processes with demo. In: Conceptual ModelingER99. Springer (1999) 188–202
10. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model checking early requirements specifications in tropos. In: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, IEEE (2001) 174–181
11. Reynolds, M.C.: Lightweight modeling of java virtual machine security constraints. In: Abstract State Machines, Alloy, B and Z. Springer (2010) 146–159
12. Kilov, H., Simmonds, I.: Business rules: from business specification to design. Springer (1998)
13. Regev, G., Bider, I., Wegmann, A.: Defining business process flexibility with the help of invariants. Software Process: Improvement and Practice **12**(1) (2007) 65–79
14. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. Software Engineering, IEEE Transactions on **26**(10) (2000) 978–1005
15. Jackson, D.: Alloy Analyzer tool. http://alloy.mit.edu/alloy/ (2013)
16. Rychkova, I.: Formal Semantics for Refinement Verification of Enterprise Models. PhD thesis, EPFL (2008)
17. Jackson, D.: Software Abstractions- Logic, Language and Analysis. MIT Press (2011)
18. GR: Generale ressorts site. http://www.generaleressorts.com/ (2013)
19. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly **28**(1) (2004) 75–105