High-Level Design and Analysis of Business Processes
The Advantages of Declarative Specifications

I. Rychkova, G. Regev, A. Wegmann
Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

***Abstract*—Business process modeling techniques, such as BPMN, encourage the early specification of the exact order in which the activities of the process will be executed. However, a business process may be exposed to different environments and subjected to many conditions in which a sequence cannot be identified at design time.**

We present declarative business process specifications that can be used to align optional process customizations, as well as process redesign, with the business strategy of the organization. These specifications complement the traditional (imperative) business process model by specifying the process independently from a particular environment.

***Index Terms*—Alloy, formal verification, Business Process Modeling, refinement.**

I. Introduction

ALIGNING business processes with business strategy is an important preoccupation in modern organizations. This alignment is made simpler if an adequate level of abstraction for business process representation is used. A business process can be defined as "a set of partially ordered activities aimed at reaching a well-defined goal." [1]. The keyword *partial* alludes to the problem of defining, ahead of time, the exact order in which the activities will be executed. Indeed a business process may be subjected to many conditions in which this order cannot be identified at design time. The exact sequence of activities is therefore quite impossible to predict [1]. Even a simple sale process has been shown to incorporate optional execution orders depending on, among other aspects, cultural and legal considerations [2]. The example given in [2] describes an on-line store that needs to adapt its sale process to local customs in different countries. The sequence of execution between payment and order fulfillment needs to be adapted to different local preferences. In the United States for example, payment by credit card is most often required before goods are shipped. In some European countries, e.g. Switzerland, customers are used to paying for goods after they have been received.

Organizations have a marked tendency to limit their interpretations of their environment [3]. These interpretations constrain their business processes at the early phases of their design [4]. Modeling techniques, such as BPMN [5] and use cases [6], also encourage modeling details at an early stage. As a result, in many cases, an organization will commit to one of the execution paths (e.g. paying before sending the goods) and later, handle the second one (sending the goods before receiving the payment) as an exception. The number of exceptions, however, often results in tangled processes containing many exceptions. This has two related consequences. First of all, the alignment between the strategy of the organization (i.e. selling on-line) and its detailed business processes is not apparent. Second, the flexibility of the processes themselves [7] is limited because they become difficult to manage and change.

In this paper, we propose a technique that complements imperative business process specifications with declarative specifications. This declarative specification enables designers to describe the actions that a business process needs to contain, but not their sequence. It omits the specification of the control flow between the actions thus keeping the process design independent from constraints imposed by an environment in which this process will be implemented. The control flow, often specific to a given environment, is later modeled in an imperative specification. Our technique includes checking the conformance of the imperative and the declarative specifications.

Presented technique can improve the alignment of the business process with the business strategy of an organization by giving a synthesis of a set of business processes (abstracting the control flow) while maintaining a rigorous relationship with the detailed process. Flexibility may also be enhanced because alternative paths are modeled as separate business processes conforming to an overall process, thereby helping organizations to tailor them to different environments without losing the overall view.

This technique is a new addition to SEAM (Systemic Enterprise Architecture Method) [8]-[10]. We illustrate our technique with the example of an on-line book store: The company wants to design a global view on its sale process in order to maintain the alignment between the different customizations of this process for different countries and to simplify the design of these customizations. We illustrate a business process redesign task using the same example and show how declarative specifications help designers to understand the relation between the redesigned process and the initial one.

We formalize the concepts of the SEAM modeling language using first-order logic with the Alloy specification language [11]. This enables us to check our models using the Alloy Analyzer [12].

In Section II we briefly present the SEAM method. We give an overview of the modeling concepts of SEAM and its underlying theory. In Section III we describe the example of the on-line book store and a SEAM declarative specification of the book store sale process. In this section we also illustrate how the sale process redesign can be rigorously modeled using declarative business process specifications. In Section IV we briefly introduce the Alloy specification language [11] and provide the Alloy semantics for the SEAM declarative specification. We complete this section with the validation of the declarative specification for the sale process using the Alloy Analyzer. In Section V we present the relevant related work. In Section 6 we outline what we envision as future work.

II. Declarative Business Process Specifications

A. The SEAM Hierarchical Model

SEAM is an Enterprise Architecture (EA) method that uses hierarchical modeling of systems, including business and IT systems. A SEAM model contains a set of specifications structured in an organizational level hierarchy.

In a SEAM specification, a system is represented by a working object. The working object can be seen as a whole where its construction is hidden or as a composite that reveals its components. The views as a whole and as a composite belong to two adjacent organizational levels. A SEAM model is usually represented graphically.

Fig. 1 illustrates four organizational levels and their representation in SEAM. These levels are:

- **The market segment level**, in which the organization of interest is modeled as a value network [13], a network of companies serving a customer (which also can be seen as being part of a value network). The value network is represented as a whole;
- **The business level**, in which the company of interest is represented as a whole, collaborating in inter-organizational business process with its partners (suppliers) within the value network. The company of interest and all its partners are represented as wholes and described by their responsibility within the inter-organizational business process [9] and the data they operate with;
- **The operational level**, where the company of interest is represented as a composite. The employee and IT system are represented as components of the company. They collaborate in a business process. The IT system is represented as a whole and is described by its responsibility within the business process and the data it operates with;
- **The IT level**, where the IT system is represented as a composite, i.e. a set of collaborating applications, seen as wholes.

To verify that a collaboration of components in one organizational level is consistent with the definition of the working object as a whole in the upper organizational level, a relationship between these levels must be made. In this work, we analyze the relationship between the market segment and the business organizational levels and verify that the business process defined for the value network (inter-organizational business process) is aligned with the strategy defined in the market segment level.

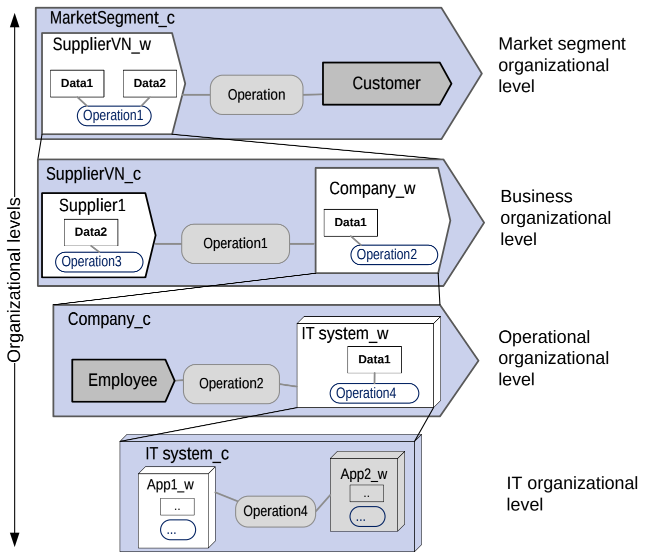


Fig. 1. Organizational levels.

B. A Process Specification in SEAM

Fig. 2 illustrates a SEAM working object (S1) seen as a whole (S1_w) and as a composite (S1_c), respectively. A working object as a whole has properties and localized actions (LA). Properties represent the state of the working object. A localized action changes the state of the working object by modifying its properties (Fig. 2).

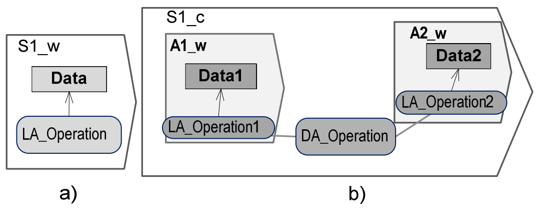


Fig. 2. SEAM notation.

A working object as a composite specifies a distributed action (DA) between components of the working object. These components are also modeled as working objects (Fig. 2-b). The keyword *Distributed* stands for a distribution of responsibilities between components, answering the question, “Who does what?” The responsibilities are modeled as localized actions.

The specification in Fig. 2-b can be read as follows: *“To perform LA_Operation at S1, the collaboration DA_Operation of component working objects A1 and A2 is required. A1 participates in DA_Operation by performing the localized action LA_Operation1 that changes Data1. LA_Operation1 is the responsibility of A1 in*

DA_Operation. The responsibility of A2 in DA_Operation is represented by the localized action LA_Operation2 that changes Data2."

The distributed action DA_Operation is a declarative specification of a business process within S1. It defines the actions to be performed by components A1 and A2 (i.e. LA_Operation1, and LA_Operation2), but does not prescribe the order in which these actions will be performed. Many execution paths are valid for a given distributed action. The selection of one of them is the business process designer's choice. When a designer commits to a concrete control flow, the specification is no longer declarative; it is transformed into a traditional imperative business process model. We call it a ***customization***.

C. Formal Semantics for SEAM Specifications

To rigorously reason about graphical specifications, we define a formal semantics for SEAM. This semantics is based on first-order logic (FOL). It enables the mapping of a SEAM specification to the Alloy specification language [11] for further validation.

SEAM **property** P_i is specified in FOL as a set whose elements are instances of this property. **A state space** of a working object is a Cartesian product of properties of this working object:

$$\Sigma = P_1 \times .. \times P_n \qquad (1)$$

At any moment of time a working object is characterized by its state. **A state** of a working object seen as a whole is defined by a vector $\vec{X} = (p_{1_1}, .., p_{n_m})$ whose components $p_{i_j} : P_i$ are instances of properties this working object hosts.
Here $i = 1..n, j = 1..m$, where P_i is a property and *m* is a number of instances of this property in $\vec{X}$.

For every action A of the working object we define a **precondition** and a **postcondition**. Precondition A_{pre} specifies a set of states $\Sigma_{A_{pre}} \subseteq \Sigma$ where action A can be executed. Postcondition A_{post} specifies a set of states $\Sigma_{A_{post}} \subseteq \Sigma$ of the working object after action A was executed. Precondition and postcondition are modeled as predicates over state space Σ : $A_{pre}, A_{post} : \Sigma \to \{true, false\}$. A postcondition applied to some state $\vec{X}$ - $A_{post}(\vec{X})$ - evaluates to 'true' iff $\vec{X}$ belongs to the set of post-states of action A denoted by $\Sigma_{A_{post}}$. The same is valid for a precondition.

$$\begin{aligned} &\forall \vec{X} \mid A_{pre}(\vec{X}) \Leftrightarrow \vec{X} \in \Sigma_{A_{pre}} \\ &\forall \vec{X} \mid A_{post}(\vec{X}) \Leftrightarrow \vec{X} \in \Sigma_{A_{post}} \end{aligned} \qquad (2)$$

An action **invariant** is a condition that holds <u>before and after</u> the action execution. In other terms, during the action execution, the working object must be found only in states, specified be the action invariant A_{inv} (e.g. a cash value cannot be negative during a sale action). These states are *allowable* states for the action. **Global invariants** S_{inv} specify allowable states for the working object during its entire lifecycle, i.e. any action it might perform. Invariants are formalized as predicates over state space Σ : $S_{inv}, A_{inv} : \Sigma \to \{true, false\}$.

Action A defines a transition of the working object from state $\vec{X}_{pre}$ to state $\vec{X}_{post}$ (pre- and post-states respectively). In FOL, a SEAM action as a whole is specified as a formula that defines a relation between a pre- and a post- state:

$$A : \Sigma \times \Sigma \to \{true, false\} \qquad (3)$$

We specify the SEAM action using logical implication between precondition and postcondition:

$$A(\vec{X}_{pre}, \vec{X}_{post}) : A_{pre}(\vec{X}_{pre}) \to A_{post}(\vec{X}_{post}) \qquad (4)$$

If at a given state $\vec{X}_{pre}$ the precondition A_{pre} of the action A holds, then the working object will be transited to a state $\vec{X}_{post}$, for which the postcondition of A - A_{post} - holds.

If at a given state $\vec{X}$ preconditions and invariants of some actions $A_1, A_2, ..A_n$ hold, then these actions are called available actions for the working object at a given state. The action definition in (4) can be read as follows: *If a state of the working object is such that the action A is available, then the working object will be transited to one of the states specified by the postcondition of A - A_{post}.*

Preconditions, postconditions and invariants explicitly relate actions with properties within a working object. This is visible in a SEAM specification through the **action-property relations**.

Actions are specified declaratively. The action specification abstracts out how the transition from pre- to post- state is made. An imperative specification, in contrast, makes explicit the intermediate states (if any) between the pre- and the post-states.

D. Refinement of SEAM Specifications

The relationships between working objects in different organizational levels are captured by the notion of ***refinement***, adopted from software engineering [14]. In software engineering, a program specification development is considered as a sequence of step-wise refinements. Along these lines, SEAM model development can be considered as a step-wise refinement of its graphical specifications [15]. More precisely, refinement in SEAM specifies a transition from one organizational level, where the working object is presented as a whole, to another organizational level, where the same working object is presented as a composite. A specification of a working object as a whole is usually called ***abstract,*** and a specification of a working object as a composite is called ***concrete***. We say that a concrete specification refines the abstract one. A relation between the state spaces of the working object specified as abstract and the working object specified as a concrete is called a ***refinement relation***.

Let us consider a working objects W seen as a whole, and specified on the state space Σ_a with a localized action A_a, and a working object W', seen as a composite, and specified on the state space Σ_c with a distributed action A_c.

Definition .

Given a refinement relation R between the state spaces $R:\Sigma_c \to \Sigma_a$, *W'* is called a ***correct refinement*** of *W* if and only if for each run of the concrete action A_c of *W'*, which starts at $\overrightarrow{X}_{c\,pre} \in \Sigma_c$ and terminates at $\overrightarrow{X}_{c\,post} \in \Sigma_c$, there exists a run A_a of *W*, which starts at $\overrightarrow{X}_{a\,pre} \in \Sigma_a$ such that $\overrightarrow{X}_{a\,pre} = R(\overrightarrow{X}_{c\,pre})$ and terminates at $\overrightarrow{X}_{a\,post} = R(\overrightarrow{X}_{c\,post})$.

The definition above can be expressed as follows:

$$\begin{aligned}&\forall \overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post} \in \Sigma_c \mid \\ &A_c(\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}) \Rightarrow A_a(R(\overrightarrow{X}_{c\ pre}), R(\overrightarrow{X}_{c\ post}))\end{aligned} \quad (5)$$

Formula (5) says that for every pair of states $\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}$ of the concrete specification, whenever action $\boldsymbol{A_c}$ starts with an initial state $\overrightarrow{X}_{c\ pre}$ and terminates at a final state $\overrightarrow{X}_{c\ post}$, there exists a pair of states of the abstract specification $R(\overrightarrow{X}_{c\ pre}), R(\overrightarrow{X}_{c\ post})$ and a run of an abstract action $\boldsymbol{A_a}$, where $R(\overrightarrow{X}_{c\ pre})$ is its initial state, and $R(\overrightarrow{X}_{c\ post})$ is its final state respectively.

An expression for the correct refinement in (5) is equivalent to (6), where the refinement relation R' is defined as a predicate and returns 'true' if its arguments are related states:

$$\begin{aligned}&R': \Sigma_c \times \Sigma_a \to \{true, false\}; \\ &\forall \overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post} \in \Sigma_c, \overrightarrow{X}_{a\ pre}, \overrightarrow{X}_{a\ post} \in \Sigma_a \mid \\ &\begin{pmatrix} R'(\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{a\ pre}) \wedge R'(\overrightarrow{X}_{c\ post}, \overrightarrow{X}_{a\ post}) \wedge \\ A_c(\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}) \end{pmatrix} \Rightarrow \\ &A_a(\overrightarrow{X}_{a\ pre}, \overrightarrow{X}_{a\ post})\end{aligned} \quad (6)$$

This refinement is illustrated in Fig. 3. A_c correctly refines A_a if, when A_c makes a transition from its pre-state $\overrightarrow{X}_{c\ pre}$ to its post-state $\overrightarrow{X}_{c\ post}$, A_a is also making a transition from its pre-state $\overrightarrow{X}_{a\ pre}$ to its post-state $\overrightarrow{X}_{a\ post}$, and these states are related by R.

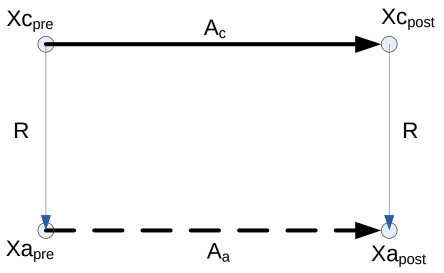


Fig. 3. The refinement in SEAM

The proposed formal semantics permit a validation of the SEAM declarative specifications and a validation of the refinement (i.e. a transition from one specification to another).

III. Example: A Sale Process For The On-Line Book Store

In this section we illustrate the declarative business process specifications with the example of a sale process for an on-line book store. We also clarify the relationships between these declarative specifications and traditional imperative business process models.

A. *The On-Line Book Store Description*

The On-Line Book Store (BS) is a company that collaborates with a publisher (P), and a bank (B) to sell books to customers. BS manages requests from customers via internet. A sale begins when a customer logs into www.BS.com using an id (customerID) and requests a book using a book id (bookID). If the requested book is available in the publisher's inventory and if the customer's rating in the data base of the bank is good then the sale is *successful*. The successful sale terminates when the book is delivered by the publisher to the customer and the payment for the book is received by the bank from the customer.

If the ordered book is not available or the customer's rating is not good, we assume that no action is executed (the cash and the inventory remain unchanged).

B. *The Successful Sale: Process Design.*

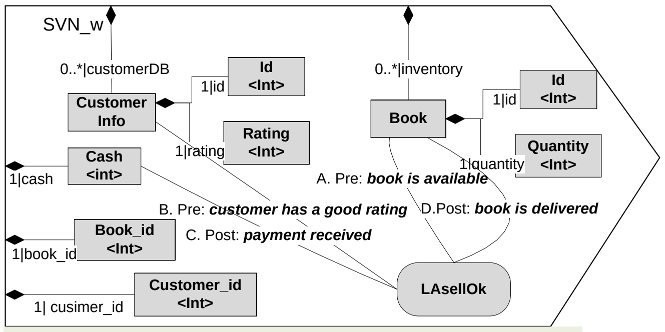
The company wants to design different customizations of its sale process for different countries by maintaining a global view of this process.

For the sake of simplicity, we limit our discussion to the specification of the successful sale. We do not specify the case where the payment is not received or the book is not delivered.

Localized Action sellOk

In Fig. 4 the On-Line Book-store value network is modeled as a working object seen as a whole - SVN_w. The successful sale process is modeled as a localized action LAsellOk of this working object. LAsellOk specifies the strategic goal of the value network: *To perform a sale by guarantying that if a book is available and if a customer has a good rating then this book will be delivered and paid by the customer.*

Action-property relations are used on the diagram in Fig.4 to specify pre- and post-conditions of LASellOk. In a legend for Fig.4 we present a formal specification of pre- and post-conditions for LASellOk written in the Alloy specification language.



For all book: Book, buyer: CustomerInfo holds:

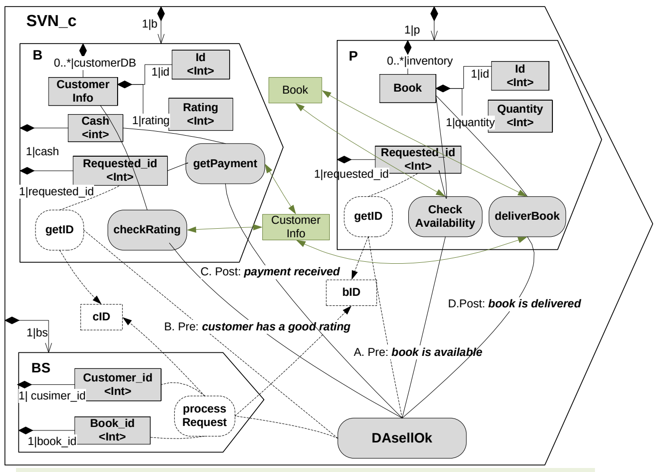
PRE	POST
A. (book.id = bookID) and (book in bInventory) and (book.quantity>0)	C. (book.quantity= book.quantity- 1)
B. (buyer.id = customerID) and (buyer in customerDB) and (buyer.rating > 0)	D.(cash = cash + 1)

Fig. 4. Localized Action SellOk.

Distributed Action DAsellOk

To relate the strategic goal of the value network with the specification of a business process that supports this goal, we represent the On-Line Book-store value network as a collaboration between the bank, the publisher and the book store – the participants in the value network. In Fig. 5 the On-Line Book-store value network is modeled as a working object seen as a composite - SVN_c. The Action DAsellOk in Fig.5 specifies how the responsibilities in a successful sale are distributed between the value network participants. It is therefore called a distributed action. The bank, the publisher and the book store are modeled as working objects seen as wholes. The responsibilities are modeled as localized actions of the corresponding working objects: for example, the fact that the bank checks the customer's rating is modeled by localized action checkRating within the B working object.

To specify the communication between the book store, the bank and the publisher, we define additional actions preocessRequest and getID, and properties cID, bID in Fig. 5. These actions and properties serve for information exchange between working objects and are not specific to the successful sale process; we show them without shading and place the relations between them and another actions and properties as dashed lines.



For all sharedBook:one Book, sharedCustomer: one CustomerInfo holds:

PRE	POST
A. p_checkAvailability[p_bInventory, p_requestedID, sharedBook]	C. p_deliverBook[p_bInventory, p_requestedID, sharedBook, sharedCustomer]
B.b_checkRating[b_customerDB, b_requestedID, sharedCustomer]	D.b_getPayment[b_cash, sharedCustomer]

Fig. 5. Distributed Action DAsellOk.

SEAM uses **shared properties** to specify distributed actions. Shared properties bind localized actions and represent a common knowledge that is maintained by the working object as a composite. In our example, sharedBook and sharedCustomer are shared properties. They represent the information used by the bank, the publisher, and the book store to manage their tasks within the successful sale process of the value network.

The Process Customization

The distributed action DAsellOk is a declarative business process specification that defines the conditions and the results of the process but does not impose any constraints on how this process will be conducted in a particular environment.

Considering that the on-line book store wants to pursue international markets, namely US and European markets (including Switzerland), different process customizations have to be designed [2].

In the US, most on-line orders are paid by a credit card and shipped only after the payment is received. A customization of the sale process for the US market is illustrated in Fig.6-a. This customization is modeled as a BPMN business process diagram (BPD).

In countries such as Switzerland most mail order companies and on-line stores have traditionally trusted customers enough to deliver ordered goods without an obligation to pay in advance. A payment form is shipped with the purchase and customers can then use it to pay for their purchases in a post office or through their bank [2]. For the Suisse market, the sell process should be customized allowing for the delivery prior to (or simultaneously with) the payment procedure as illustrated in Fig. 6-b.

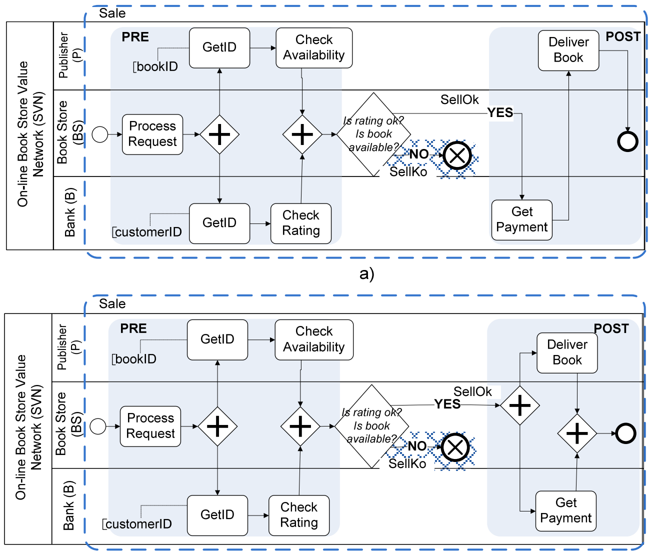


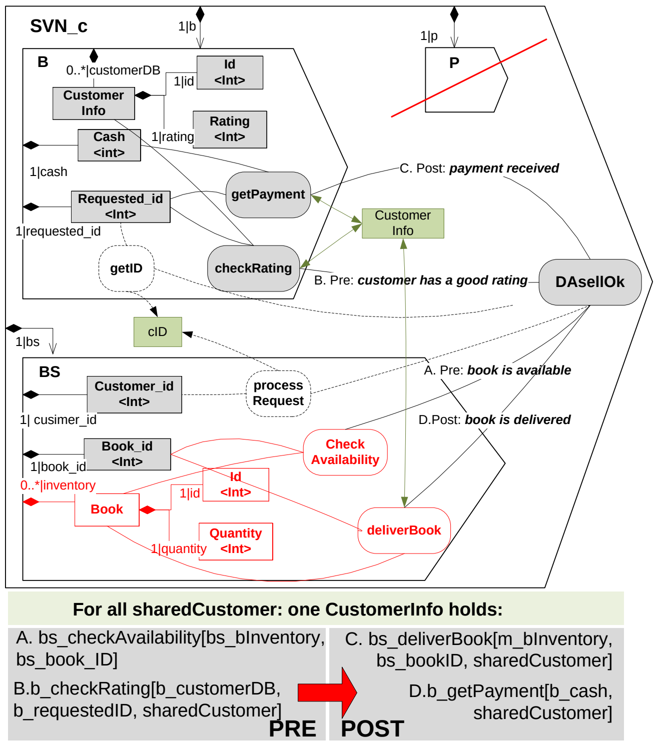
Fig. 6: On-line book store value network performing Sale:
a. the process customization for US;
b. the process customization for Switzerland

The distributed action DAsellOk relates business process customizations illustrated in Fig. 6 with the strategic goal of the on-line book store value network, specified as a localized action in Fig. 4.

C. The Successful Sale: Process Redesign.

The second business process modeling task that can benefit from an additional declarative specification layer is a business process redesign. A decision of the company to redesign its business process (or processes) can be based on different internal or external factors, e.g. the emergence of new technologies or new products, the change of a political situation, the competitive landscape etc. Considering our example, let's imagine that the on-line book store discovered that its shipment service suffers from chronic delays and is found unsatisfactory by the customers. The on-line book store decides to maintain its own inventory and to provide the shipment service by itself instead of outsourcing this service to the publisher.

Although the strategic goal of the value network remains the same, the value network itself is reorganized and, as a consequence, a business process redesign is required. The redesign of a successful sale can be rigorously modeled using a declarative specification that reflects a new distribution of responsibilities between participants of the reorganized value network. We specify a new (redesigned) distributed action for sellOk in Fig. 7. In this specification, the book inventory modeled as a set of books, and the localized actions checkAvailability and deliverBook become a part of the BS working object specification. Working object P that represents the publisher in our specification is removed.



SVN_c
1|b
1|p
B
P
0..*|customerDB
1|id
Id <Int>
Customer Info
Rating <Int>
Cash <int>
1|rating
1|cash
Requested_id <Int>
1|requested_id
getPayment
checkRating
getID
Customer Info
C. Post: payment received
B. Pre: customer has a good rating
DASellOk
cID
1|bs
BS
Customer_id <Int>
process Request
1| cusimer_id
A. Pre: book is available
D.Post: book is delivered
Book_id <Int>
1|book_id
Check Availability
0..*|inventory
1|id
Id <Int>
Book
Quantity <Int>
1|quantity
deliverBook
For all sharedCustomer: one CustomerInfo holds:
A. bs_checkAvailability[bs_bInventory, bs_book_ID]
B.b_checkRating[b_customerDB, b_requestedID, sharedCustomer]
C. bs_deliverBook[m_bInventory, bs_bookID, sharedCustomer]
D.b_getPayment[b_cash, sharedCustomer]
PRE
POST

Fig. 7 Distributed action for redesigned sale.

The distributed action DAsellOK in Fig.7 is consistent with the localized action LAsellOk in Fig.4 because the latter specifies only the work to be done, but not the distribution of this work. This illustrates an integration of two declarative specifications of the sale process: the initial one and the redesigned one.

Based on the redesigned distributed action, new process customizations for US and Switzerland are modeled in Fig. 8. The redesigned distributed action DAsellOk relates the business process customizations illustrated in Fig. 8 with the strategic goal of the on-line book store value network, specified as a localized action in Fig. 4.

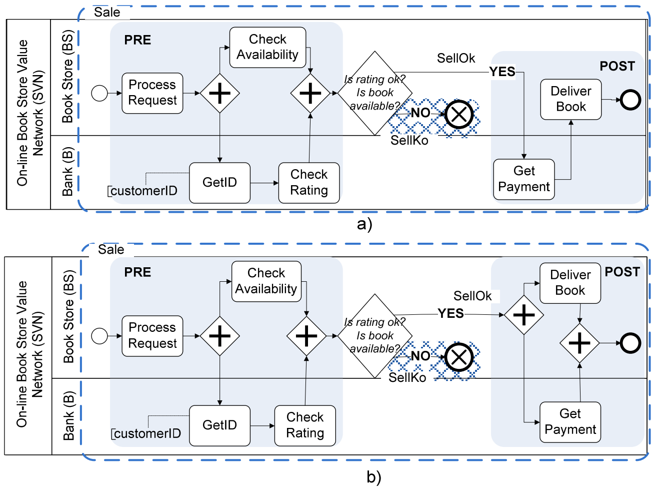
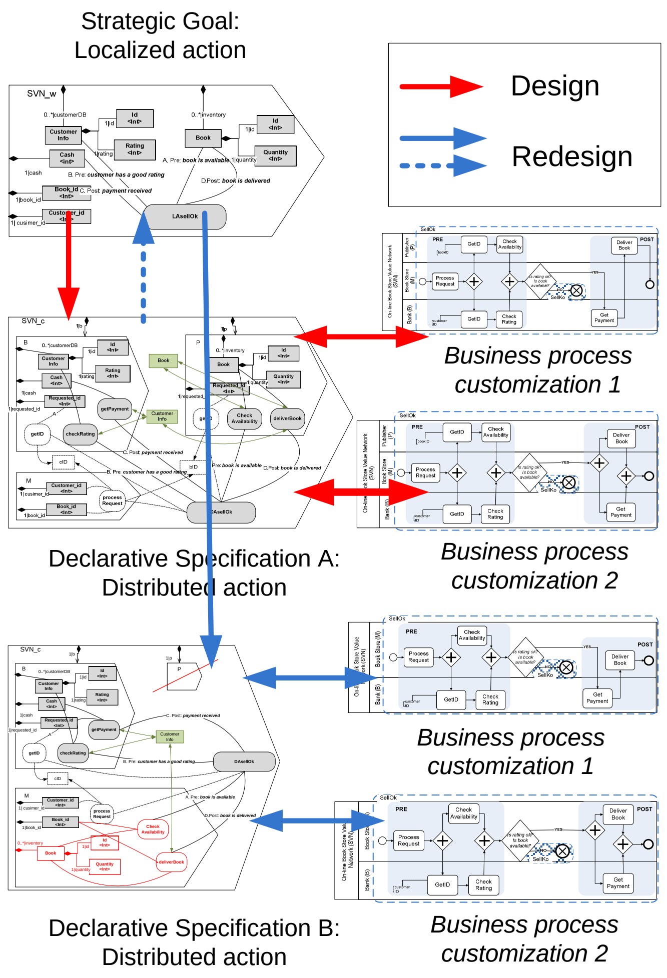


Fig. 8 On-line book store value network performing Sale:
a. the process customization for US (redesigned);
b. the process customization for Switzerland (redesigned)

Fig. 9 presents an overview of the design and redesign of the successful sale business process and shows how design and redesign tasks can be related via declarative specifications.



Strategic Goal:
Localized action
Design
Redesign
Business process customization 1
Declarative Specification A:
Distributed action
Business process customization 2
Business process customization 1
Declarative Specification B:
Distributed action
Business process customization 2

Fig. 9. Business process design and redesign schema using declarative SEAM specifications.

IV. Validation of Declarative Specifications in Alloy

We call a transition from the localized action specified for the working object seen as a whole to the distributed action specified for the same working object seen as a composite a *specification refinement*. In this section we demonstrate how SEAM specifications and a refinement between these specifications can be validated in Alloy.

A. *Alloy Specification Language*

Alloy is a declarative specification language developed by the Software Design Group at MIT - http://Alloy.mit.edu/. Alloy is a language for modeling systems as complex structures with constraints and behavior based on first-order logic. The syntax of Alloy is similar to the syntax of OCL – the Object constraint language for UML. However, Alloy is a fully declarative, whereas OCL combines both declarative and imperative (operational) elements.

Unlike a programming language, a declarative Alloy model describes the effect of a behavior and does not reveal its mechanism. This modeling technique allows for the creation and analysis of partial models and is beneficial when, for example, a modeler has a limited knowledge about the system and develops an abstract system specification. Alloy specification language belongs to the class of formal specification languages like Z, VDM, B, etc; its main benefit is the possibility of a fully automated analysis of its models.

For the automated analysis of models written in Alloy, an Alloy Analyzer [12] is used. The Alloy Analyzer is the model checker for Alloy: given a logical formula and a data structure that defines the interpretation 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 evaluates the formula to 'true'. A logical formula may correspond to some property of the modeled system or its behavior.

In this work we are dealing with the latter case: We model the actions performed by a system as Alloy formulae with variables representing the system states before and after the action. Using Alloy Analyzer, we verify if the action specifies a legal state transition.

The second analysis presented in this work and performed with the Alloy Analyzer is refinement checking between the SEAM localized action and the SEAM distributed action. To check that one (refined) action specification A' correctly refines another (abstract) action specification A, we assert that A' implies A in Alloy. The Alloy Analyzer negates the assertion, and looks for a model, which, if found, will be a counterexample to the claim. The absence of a counterexample automatically means a validity of a claim.

In the rest of this section we illustrate how the mapping between SEAM and Alloy languages is done and present the analysis of Alloy specifications obtained in more details.

B. *Specification of localized and distributed actions sellOk using Alloy*

We begin with a mapping of the SVN_w specification shown in Fig. 4 and the SVN_c specification shown in Fig. 5 to Alloy. Technically, the mapping of SEAM specifications to Alloy is based on the XSLT transformation of the XML file, which contains the SEAM specification, to the Alloy specification file.

We specify the working object SVN_w using an Alloy signature (the analogy of a class in the object-oriented paradigm). The properties of a working object are represented by Alloy relations (the analogy of fields in the object-oriented paradigm). To avoid confusion between the term "relation" in Alloy and in SEAM, we call Alloy relations "fields" later on in the text.

```
sig SVN_w{
customerID: one Int,            - customer ID
bookID: one Int,                - book to buy
customerDB: one CustomerDB,      - customer data base
bInventory: one Inventory,      - book inventory
cash: one Int                   - cash
}
```

Here a book inventory (*Inventory*) is modeled as a set of books and a customer database (*CustomerDB*) is modeled as a set of customer info records:

```
sig Inventory{content: set Book}
sig CustomerDB{content: set CustomerInfo}
```

The property *CustomerInfo* is specified as an Alloy signature with two fields: *id* and *rating*. Respectively, the property *Book* is specified as a signature with the fields *id* and *quantity*:

```
sig CustomerInfo{
 id: one Int,
 rating: one Int}  - rating>0 - good; <0 - bad;

sig Book{
 id: one Int,
 quantity: one Int}  - number of books available
```

We model SEAM actions as Alloy predicates. In SEAM, an action defines a transition of a working object from one state (pre-state) to another (post-state). The SEAM action specification from (4) uses a pre-state and a post-state as parameters and can be rewritten as follows:

$$A(\vec{X}_{pre}, \vec{X}_{post}) = A(p_{1_{pre}}, .., p_{n_{pre}}, p_{1_{post}}, .., p_{n_{post}}) \qquad (7)$$

Components $p_{1_{pre}}, .., p_{n_{pre}}, p_{1_{post}}, .., p_{n_{post}}$ define values of properties of the working object before and after the action happen respectively.

Along these lines we use indexes **_pre**, **_post**, and **_prepost** to model parameters of the Alloy predicate:

- all parameters indexed with _pre correspond to the properties of the working object before the action and define a pre- state of this working object $\vec{X}_{pre}$;
- all parameters indexed with post- correspond to the properties of the working object after the action happens and define the post-state $\vec{X}_{post}$ of this working object;
- index **_prepost** specifies parameters that are not modified by the action. These parameters correspond to the properties that make a part of both $\vec{X}_{pre}$ and $\vec{X}_{post}$.

We write the following Alloy specifications of pre- and post- states for localized action LAsellOk in Fig.4:

```
bInventory_pre: one Inventory,
customerDB_prepost: one CustomerDB,
customerID_prepost: one Int,
bookID_prepost: one Int,
cash_pre: one Int;
```
$\Leftrightarrow \vec{X}_{pre}$

```
bInventory_post: one Inventory,
customerDB_prepost: one CustomerDB,
customerID_prepost: one Int,
bookID_prepost: one Int,
cash_post: one Int
```
$\Leftrightarrow \vec{X}_{post}$

The Alloy code below specifies the LAsellOk localized action as a corresponding Alloy predicate. Lines 1-7 in this code correspond to the action's precondition; lines 8-14 – to its postcondition. The predicate LAsellOk holds when its precondition implies its postcondition.

```
pred LAsellOk [bInventory_pre, bInventory_post: one
Inventory, customerDB_prepost: one CustomerDB,
customerID_prepost, bookID_prepost, cash_pre,
cash_post: one Int] {
1.(all requested_book: Book, buyer: CustomerInfo|
2.((requested_book.id = bookID_prepost) and
3.(requested_book in bInventory_pre.content) and
4.(requested_book.quantity>0) and
5.(buyer.id = customerID_prepost) and
6.(buyer in customerDB_prepost.content) and
7.(buyer.rating > 0) ) =>
8.((one b_post: Book |
9.  (b_post.id = requested_book.id) and
10. (b_post.quantity= requested_book.quantity- 1)
   and
11. (bInventory_post.content =
   bInventory_pre.content - requested_book + b_post)
   and
12. //(customerToDeliver.id = bookDeliveredToID)
13. (cash_post = cash_pre + 1 ) )
14. //  (buyer.id = paymentFromID)
   ) )}
```

The specification of the localized action LAsellOk in Alloy can be read as follows:
For all buyers and requested books (line 1)*: the precondition of LAsellOk holds if the values of their id fields are equal to the values of bookID and customerID respectively* (lines 2,5), *and the requested book exists in the inventory* (line 3), *and is available* (line 4), *and a buyer exists in the customer DB* (line 6), *and has a good rating* (line 7). *The postcondition stands that there exists a book_post* (line 8) *that corresponds to the requested book* (line 9) *and its quantity is equal to the quantity of the requested book decreased by one* (line 10), *and the book inventory after the action (bInventory_post) is equivalent to the inventory before this action (bInventory_pre) with the requested book substituted by the book_post* (line 11), *and the cash value after the action is augmented by one unit* (line 13). We also need to specify that the requested book is delivered to the proper buyer, and that the payment is received from the proper customer (lines 12, 14). For the sake of simplicity we do not model it in this example.

We specify the working object SVN_c from the SEAM specification in Fig.5 as follows:

```
sig SVN_c{
b: one B,
p: one P,
bs: one BS}
```

The three fields of this signature represent three component working objects:

```
lone sig B{          - the bank
customerDB: set CustomerInfo,
cash: one Int,
requestedID: one Int }

lone sig P{          - the publisher
bInventory: set Book,
requestedID: one Int }

lone sig BS{           - the book store
customerID: one Int, //customer ID
bookID: one Int //book to buy
}
```

The localized actions of component working objects are modeled as the following Alloy predicates:
`pred p_checkAvailability[..]{..}` – the publisher checks if the requested book is available;
`pred b_checkRating[..]{..}`- the bank checks if a rating of the customer is good;
`pred p_deliverBook[..]{..}` – the publisher delivers the book to the customer;
`pred b_getPayment[..]{..}`- the bank receives payment from the customer.

The following predicates specify communication between the book store, the bank, and the publisher, as do so the corresponding localized actions in Fig. 5:
`pred bs_processRequest[..]{..}`- the book store gets request and externalizes the requested book id and the customer id for the rest of the network.
`pred p_getID[..]{..}` – the publisher gets the requested book id;
`pred b_getID[..]{..}`- the bank gets the customer id.

The distributed action DAsellOk binds the localized actions of the component working objects. Without any other specific information, we write a declarative specification of a distributed action as a conjunction of formulae representing localized actions. This stipulates that the action terminates successfully if and only if all of its components terminate successfully:

$$\overrightarrow{X}_{pre}, \overrightarrow{X}_{post} \in \Sigma;$$
$$DA(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}) = \qquad (8)$$
$$\left(LA_1(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}) \wedge .. \wedge LA_m(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post})\right)$$

Here $LA_1(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}),..,LA_m(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post})$ are localized actions that represent responsibilities of component working objects within the distributed action, modeled as predicates.
A distributed action does not specify its own precondition, postcondition, and invariant: it inherits them from the localized actions it invokes.
A partial ordering of localized actions within the distributed actions can be defined:

$$\overrightarrow{X}_{pre}, \overrightarrow{X}_{post} \in \Sigma;$$
$$DA(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}) = \qquad (9)$$
$$\left(LA_1(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}) \wedge .. \wedge LA_m(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post})\right) \Rightarrow$$
$$\left(LA_{m+1}(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post}) \wedge .. \wedge LA_n(\overrightarrow{X}_{pre}, \overrightarrow{X}_{post})\right)$$

Here, the fact that predicates $\left(LA_1 \wedge .. \wedge LA_m\right)$ hold implies the fact that predicates $\left(LA_{m+1} \wedge .. \wedge LA_n\right)$ hold. The first group can be considered as 'responsible' for a precondition A_{pre} of an action from (4), whereas the second group – for its postcondition A_{post}.

The Alloy code below specifies the DAsellOk distributed action as an Alloy predicate. This action is obtained as a refinement of a localized action LAsellOk. Lines 1-7 in this code correspond to the precondition of a localized action LAsellOk from the listing above; lines 8-9 – to its postcondition.

```
pred DAsellOk[p_bInventory_pre, p_bInventory_post:
one Inventory, p_requestedID_prepost: one Int,
b_customerDB_prepost: one CustomerDB,
b_requestedID_prepost: one Int, b_cash_pre,
b_cash_post: one Int,
bs_customerID_prepost, bs_bookID_prepost: one Int]{
  1. ( one cID,bID: Int |
  2. bs_processRequest[bs_bookID_prepost,
      bs_customerID_prepost, bID,cID] and
  3. p_getID[bID, p_requestedID_prepost] and
  4. b_getID[cID, b_requestedID_prepost]) and
  5. all  sharedBook:one Book, sharedCustomer: one
      CustomerInfo|
  6. (p_checkAvailability[p_bInventory_pre,
      p_requestedID_prepost, sharedBook] and
  7. b_checkRating[b_customerDB_prepost,
      b_requestedID_prepost, sharedCustomer] )
      =>
  8. (p_deliverBook[p_bInventory_pre,
     p_bInventory_post,p_requestedID_prepost,
     sharedBook, sharedCustomer] and
  9. b_getPayment[b_cash_pre,b_cash_post,
     sharedCustomer])}
```

Prefixes **p_**, **b_**, **bs_** in the names of predicates specifying localized actions and in the names of predicate parameters specifying properties refer to the component working objects these localized actions or properties belong to (e.g. p_bInventory specifies the book inventory, which is the property of the publisher).

C. Validation of Declarative Specifications using Alloy Analyzer 4.0

Specifications written in Alloy can be automatically analyzed using the Alloy Analyzer [12]. The Alloy Analyzer tool can generate examples of the working object and counterexamples to claims made about this working object and its behavior.

To validate if Alloy specifications of sellOk are consistent, we execute corresponding predicates in the Alloy Analyzer [12]. The Alloy Analyzer examines a predicate and looks for the possibility to ***instantiate*** this predicate, i.e. to find a set of values that evaluates this predicate as *true*. If such an instance is found, then the predicate is consistent on the test space provided by the analyzer. If no instance found, then the predicate is inconsistent, and the specification may contain contradictory constraints. Note that the predicate consistency (as well as inconsistency) is checked only on the limited test space. An example of the execution trace in the Alloy Analyzer is provided below:

```
Executing "Run LAsellOk "
Solver=sat4j Bitwidth=4 MaxSeq=4 Symmetry=20
3605 vars. 561 primary vars. 8156 clauses. 80ms.
Instance found. Predicate is consistent. 55ms
```

D. Validation of Refinement From LA to DA using Alloy Analyzer 4.0

To relate the designed business process of successful sale to the strategic goal of the on-line book store, we have to guarantee:

1) The correct refinement from the localized action LAsellOk to the distributed action DAsellOk;

2) The correct mapping between the declarative specification DAsellOk and the imperative business process specifications (i.e. BPMN diagrams) that specify process customizations.

To check if the distributed action DAsellOk correctly refines the localized action LAsellOk in our example, we use the definition of refinement from (5)(6). We consider the distributed action DAsellOk a concrete specification and the localized action LAsellOk an abstract specification. We rewrite (6) as an Alloy assertion that specifies the correct refinement from abstract to concrete specification:

assert DA_LA{
all $\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}, \overrightarrow{X}_{a\ pre}, \overrightarrow{X}_{a\ post}$ |
(R_LA_to_DA ($\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{a\ pre}$) and
R_LA_to_DA($\overrightarrow{X}_{c\ post}, \overrightarrow{X}_{a\ post}$)and
DAsellOk($\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}$)) =>
LAsellOk($\overrightarrow{X}_{a\ pre}, \overrightarrow{X}_{a\ post}$) }

Here $\overrightarrow{X}_{c\ pre}, \overrightarrow{X}_{c\ post}, \overrightarrow{X}_{a\ pre}, \overrightarrow{X}_{a\ post}$ stand for pre- and post-states at concrete and abstract specifications respectively. R_LA_to_DA is a refinement function that relates state spaces of the SVN_w and SVN_c. We provide the complete specification of this refinement function:

```
pred R_LA_to_DA[p_bInventory_t: one Inventory,
p_requestedID_t: one Int, b_customerDB_t: one
CustomerDB, b_requestedID_t: one Int,
b_cash_t: one Int,
bs_customerID_t, bs_bookID_t: one Int,
// concrete
bInventory_t: one Inventory,
customerDB_t: one CustomerDB, customerID_t,
bookID_t, cash_t: one Int // abstract
]{
p_bInventory_t = bInventory_t
p_requestedID_t = bookID_t
b_customerDB_t = customerDB_t
b_requestedID_t = customerID_t
b_cash_t = cash_t
bs_customerID_t = customerID_t
bs_bookID_t = bookID_t
```
} ⇔ R[$\overrightarrow{X_c}, \overrightarrow{X_a}$]

To validate an assertion, the Alloy Analyzer looks for a counterexample, i.e. a set of values that evaluates this assertion to *false*. If such a counterexample is found then assertion is invalid. In our case it also means that the refinement is incorrect. If no counterexample is found, then the assertion is valid and the refinement is correct. An example of the execution trace is provided below:

```
Executing "Check DA_LA"
Solver=sat4j Bitwidth=4 MaxSeq=4 Symmetry=20
5352 vars. 593 primary vars. 16733 clauses. 618ms.
No counterexample found. Assertion may be valid.
1166ms.
```

The Alloy language and analyzer are known to be used for industrial purposes, i.e. for modeling and verification of the large-scale systems [survey-tbd]. Based on this, we conclude that our approach is scalable and limited only by the size of a SEAM model.

The mapping between SEAM distributed actions, modeled declaratively, and imperative business process diagrams modeled in BPMN can be done in two steps: (1) definition of a control flow for SEAM distributed actions modeled declaratively; (2) mapping of obtained ***imperative*** specifications in SEAM to BPMN.

To pursue the first step, intermediate states of the system and their order should be defined:

$$\overrightarrow{X}_{pre}, \overrightarrow{X}_{t_1}, .., \overrightarrow{X}_{t_l}, .., \overrightarrow{X}_{t_k}, \overrightarrow{X}_{post} \in \Sigma \mid \quad (9)$$
$$pre \le t_1 \le .. \le t_l \le .. \le t_k < post;$$

Intermediate states and their order specify a control flow for a distributed action. The resulting specification is ***imperative***. The conformance of the imperative specification with the declarative specification in SEAM is formally verified.

The imperative specification of a distributed action contains the information required for the mapping to BPMN. In our future work we will address a more detailed discussion about the imperative specifications in SEAM and about the mapping between SEAM and BPMN using these specifications.

V. RELATED WORK

The possibility of customizing a business process while taking into account an environment where this business process is instantiated is a part of the more general problem of flexibility. This problem was identified in [16] and [17] in general and in the context of WfMS respectively: Knoll and Jarvenpaa [16] introduce the term of flexibility as a form of alignment between organizations and their IT systems in turbulent environments, and they point out that “The principle of “flexibility” explicitly assumes that the world is too dynamic for a static order between different organizational components.” The authors recognize three types of flexibility in the context of IT: flexibility in functionality, in use and in modification. Heinl et al. [17] illustrate the necessity of flexibility in workflow management applications and identify two classes of flexibilities: by selection and by adaption. Flexibility by selection implies that more than one valid interpretation of a workflow type exists and might be selected based on a concrete situation. Flexibility by adaption defines new variants of workflow execution when flexibility by selection is not sufficient. Flexibility by selection covers the topic of business process customization, whereas flexibility by adaption is related to the process redesign considered in our work.

Another stream of research, e.g. [1] and [18] favors what we refer to as declarative business process modeling. In [1] the representation of a business process as a trajectory in a state space is introduced. The authors attempt to declaratively describe the dynamics of a business process by defining a notion of a valid state and planning rules that make a state valid. Van der Aalst in [18] presents a case handling paradigm to cope with business process flexibility. In contrast to workflow management, case handling aims to describe what *can* be done to achieve a business goal but not what *should* be done and *how*.

The flexibility of a business process is usually understood as the capability to accept changes without losing identity [19]. Hence, this capability is not always beneficial, because some changes can be contradictory to the strategy of an organization. In [20] invariants for business processes are introduced and formalized. Invariants define an identity of an organization that must remain unchanged. Rittgen [21] proposes the notion of Collaboration Model to capture the stable part of a business process model. The part of the model that is flexible is addressed in business process rules. In [22] authors discuss a variability applied to business process modeling and propose modeling a family of business processes adaptable for different environments and organizations. The authors define common and variable parts for an entire family based on the fact that all of the processes are designed to achieve the same goal but in a different way.

In [23] the definition of flexibility is grounded in the notion of a process goal, which defines a set of final states of the process, and on the theory of coordination, which describes dependencies between processes.

Providing other types of semantics (including formal semantics) for visual models was recognized as a useful way to increase model precision and to automate model verification. Baar and Marcović [24] introduce a proof technique for the semantic preservation of refactoring rules for UML[25] class diagrams and OCL constraints. This technique is implemented in the RoclET tool. In [26] formal semantics of Petri nets are defined for BPMN models. A mapping between BPMN and Petri Net is implemented as a tool that generates Petri Net Markup Language specifications for further static analysis.

In spite of their effectiveness, approaches based on a formal validation and verification using theorem proving are rarely used in practice due to the high cost. However, we want to point out the following work:

In [27] the UML2Alloy tool for modeling and analysis of discrete event systems is presented. UML2Alloy is based on MDA [OMG] and implements research results that attempt to formalize UML[25] using Alloy. This is remarkable because it results in the integration of semi-formal UML and formal Alloy languages within one tool.

VI. Conclusion

In this paper, we have presented declarative business process specifications as a mechanism to integrate different customizations and redesigns of a business process. Declarative specifications focus on the definition of a business process and on its alignment with the organization's strategic goals. They omit the definition of the process control flow thus keeping the process design independent from constraints imposed by an environment in which this process will be implemented.

Once a control flow is selected for a process based on a specific environment, the declarative specification can be mapped to an imperative business process model.

In the future we envision that the work described in this paper will enable us to link the SEAM modeling tool SeamCAD [28] and BPMN tools [5]. In particular, we want to automatically generate imperative BPMN models from SEAM models defined in SeamCAD.

References

[1] M. Khomyakov, and I. Bider, "Achieving Workflow Flexibility through Taming the Chaos". OOIS 2000 - 6th international conference on object oriented information systems. Springer, 2000, pp.85-92. Reprinted in the Journal of Conceptual Modeling, August 2001: http://www.inconcept.com/JCM/August2001/bider.html, accessed December 2007.

[2] G. Regev, and A. Wegmann, "Regulation Based Linking of Strategic Goals and Business Processes", Proceedings of the 3rd BPMDS Workshop on Goal-Oriented Business Process Modeling, GBPM'02, London, September 2002.

[3] K. E.Weick, "The Social Psychology of Organizing", second edition, McGraw-Hill. 1979

[4] M.M. Narasipuram, G. Regev, K. Kumar, A. Wegmann, "Business Process Flexibility through the Exploration of Stimuli", accepted for publication, International Journal of Business Process Integration and Management (IJBPIM), 2008

[5] Business Process Modeling Notation (BPMN) Version 1.0, OMG Final Adopted Specification, February 6, 2006.

[6] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, "Object-Oriented Software Engineering: A Use Case Driven Approach", (ACM Press) Addison-Wesley, 1992.

[7] G. Regev, P. Soffer, and R. Schmidt, "Taxonomy of Flexibility in Business Processes", proceedings of the seventh workshop on Business Process Modeling, Design and Support (BPMDS'06), 2006.

[8] A.Wegmann, "On the systemic enterprise architecture methodology (SEAM)", proceedings of International Conference on Enterprise Information Systems 2003, (ICEIS 2003), Angers, France.

[9] A. Wegmann, G. Regev, I. Rychkova, L-S. Lê, "Business-IT Alignment with SEAM for Enterprise Architecture", proceedings of the 11th IEEE International EDOC Conference (EDOC 2007), Annapolis, Maryland, 15-19 October 2007.

[10] A. Wegmann, P. Julia, G. Regev, O. Perroud, I. Rychkova, "Early Requirements and Business-IT Alignment with SEAM for Business", proceedings of the 15th IEEE International Requirements Engineering Conference, New Delhi, India, October 15-19th, 2007.

[11] D. Jackson, "Software Abstractions: Logic, Language, and Analysis", MIT Press. Cambridge, MA. March 2006. ISBN 0-262-10114-9

[12] Alloy Analyzer 4.0, http://Alloy.mit.edu/Alloy4/

[13] Stabell, C. B. and Fjeldstad, Ø. D., "Configuring value for competitive advantage: on chains, shops, and network", Strategic Management Journal 19(5): p. 413 – 437, 1998

[14] N. Wirth, “Program development by stepwise refinement”, Communications of the ACM, 14:221–227. 1971.
[15] I. Rychkova, A. Wegmann, “Refinement propagation. Towards automated construction of visual specifications”, proceedings of International Conference on Enterprise Information Systems (ICEIS) (2007)
[16] K. Knoll, S.L. Jarvenpaa, “Information technology alignment or “fit” in highly turbulent environments: the concept of flexibility”, proceedings of the 1994 computer personnel research conference on Reinventing IS.
[17] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke, “A Comprehensive Approach to Flexibility in Workflow Management Systems”, proceedings of the international joint conference on work activities coordination and collaboration, 1999, San Francisco, California, USA, February 22-25, 1999, ACM 1999, pp79-88
[18] W.M.P. van der Aalst, M. Weske, D. Grünbauer, “Case Handling: A New Paradigm for Business Process Support”, Data Knowl. Eng. 53(2) (2005) 129–162
[19] G. Regev, A. Wegmann, “A Regulation-Based View on Business Process and Supporting System Flexibility”, proceedings of the CAiSE 2005 Workshops, p. 91-98.
[20] G. Regev, I. Bider, A. Wegmann, “Defining business process flexibility with the help of invariants”, Special Issue on Design for Flexibility . Published Online: 27 Sep 2006.
[21] P. Rittgen, “ Supporting Planned and Ad-Hoc Changes of Business Processes”, proceedings of seventh workshop on Business Process Modeling, Development, and Support (BPMDS'06), Luxembourg, 5 – 6 June 2006.
[22] C. Rolland, N. Prakash, “On the Adequate Modeling of Business Process Families”, proceedings of the eighth workshop on Business Process Modeling, Development, and Support (BPMDS'07), 11-15 June 2007, Trondheim, Norway
[23] P. Soffer, “On the Notion of Flexibility in Business Processes”, proceedings of sixth workshop on Business Process Modeling, Development, and Support (BPMDS'05), Porto, Portugal June 13-14 2005
[24] T. Baar and S. Marković, “A Graphical Approach to Prove the Semantic Preservation of UML/OCL Refactoring Rules”, Irina Virbitskaite and Andrei Voronkov, editors. Perspectives of Systems Informatics, 6th International Andrei Ershov Memorial Conference, PSI 2006, Akademgorodok, Novosibirsk, Russia, June 27-30, 2006, Proceedings, LNCS 4378, pp. 70-83, Springer, 2007.
[25] Unified Modeling Language: Superstructure, version 2.1.2, OMG, November 2007.
[26] R. M. Dijkman, M. Dumas, and C. Ouyang, “Formal Semantics and Analysis of BPMN Process Models”, preprint version, QUT | ePrints Archive, http://eprints.library.qut.edu.au/ 2007.
[27] Behzad Bordbar and Kyriakos Anastasakis, “UML2Alloy: A tool for lightweight modelling of Discrete Event Systems”. IADIS International Conference in Applied Computing 2005. In Nuno Guimarães and Pedro Isaías (es.), IADIS International Conference in Applied Computing 2005. Volume 1., Algarve, Portugal, IADIS Press, 2005. 209-216
[28] L.S. Lê and A. Wegmann, “SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture”, 39h IEEE Hawaii International Conference on System Sciences, 2006