Content-Based Validation of Business Process Models

Maya Lincoln, Avigdor Gal

Abstract. In this work we present a methodology for content-based validation of business process models, focusing on existing organizational policies. This methodology goes beyond structural notation and proposes to automatically extract business logic from process repositories as a basis for content validation. Each process activity is encoded automatically as a descriptor, containing objects, actions, and qualifiers. The collection of all process descriptors formulates a taxonomy model of action sequence, object lifecycle and object and action hierarchies that are used to support the validation procedure. We propose a stepwise method for context-based validation that includes deficiency identification (using existing descriptors as a reference), validation score calculation and generation of a ranked list of corrections. We illustrate our approach using three types of validation errors that may occur as a result of an atomic change (addition or deletion) of an activity in a process model. We developed a software tool that implements the suggested methodology and we report on an empirical evaluation, showing that the utilization of the taxonomy model makes it possible to effectively validate the design of new process models and the modification of existing ones. The proposed method can guide business analysts that opt to validate business process models, by marking contextually non-valid segments and suggesting possible corrections.

Keywords: Business process models, Validation, Object and action sequences.

1 Introduction

In recent years, researchers have become increasingly interested in developing methods and tools for the verification, validation, and compliance of business process models. Business process verification mainly focuses on avoiding errors at the structural level of process models, for example by checking the existence of deadlocks or incorrect data links during the design stage (see for example [37,28,29,23,26]). Business process validation takes into consideration the content, data, and context layers ("content") of process models aiming to evaluate the correctness of related business process logic (see for example [45,19,20,8,13]). Finally, Business process compliance addresses both verification and validation issues aiming to ascertain whether a process model complies with a reference pattern that can refer either to structure or content (see for example [4,9,34,7,30]).

Our work presents content-aware validation framework, that aims to complement the validation and compliance domains. Current research in the validation domain focuses on two major streams: (1) semantics based validation and (2)
object lifecycle and action sequence validation. Our work extends both streams in the following way: first, it does not require a pre-definition of related semantics, but it rather relies on the natural language of the specific process model. Second, it deals with action sequences and object lifecycles simultaneously and not separately as in previous works. Our contribution to the compliance domain is in the *dynamic* construction and adjustment of patterns - avoiding the need to design and maintain external, static rules.

In order to demonstrate the need for content-aware validation analysis that focuses on the content of business process models, consider the following example.

Example 1. Consider two manufacturing organizations - one that uses a *Make to Order* (MtO) production policy while the other deploys *Make to Stock* (MtS) production policy. Both manufacturers use an ERP or a BPM system to support the enactment of their ongoing business. Using high level abstraction, the execution pattern of the MtS process starts with *forecasting and planning*, then *sourcing and procurement*, followed by *manufacturing*, and then *inventory and warehouse management*, moving on to *sales order processing*, and concluding with *shipping and transportation*. The MtO process, on the other hand, would have the following flow: *sales order processing* → *planning* → *sourcing and procurement*, etc. Hence, the inclusion of any forecasting related activities in the MtO process would be redundant or even erroneous. Furthermore, aiming to process sales orders in the context of MtS prior to completing a forecasting activity would be problematic if not impossible. Such validation issues propagate into lower level process modeling and will not be identified by structural verification methods, nor by semantic/data based validation methods. Hence, the need for content-based validation becomes apparent.

In this work we suggest a methodology for content-based validation, taking into account existing organizational regulations and practices. We propose to *automatically* extract business logic from process repositories using the PDC model [15,16,12]. Each activity is encoded automatically as a *descriptor* that represents objects, actions, and qualifiers. The collection of all process descriptors formulates a taxonomy model of action sequence, object lifecycle, and object and action hierarchies that are used to support the validation process. We propose a three step method for content-based validation, namely: deficiency identification (using existing descriptors as reference), validation score calculation and generation of a ranked list of corrections. We then instantiate our approach using three types of validation errors that may occur as a result of an atomic change of addition or deletion of an activity in a process. Two of these errors are related to execution order, namely: an *invalid action sequence error* and an *invalid object lifecycle error*. The third error, namely: an *unknown descriptor combination warning* is related to new knowledge within the repository.

We developed a software tool that implements the suggested methodology. We report on an empirical evaluation of the procurement industry case study based on the Oracle Applications ERP process repository, showing that the
utilization of the taxonomy model makes it possible to effectively validate the
design of new process models and the modification of existing ones.

The proposed method can support business analysts that opt to validate new
process models or to change existing models, by marking contextually non-valid
process segments and suggesting possible corrections - ranked according to their
relevance to the designer’s intention. Our contribution includes:

- Proposing a content-based validation framework for business process models
  that does not require any additional preparations within a given process
  repository (e.g. additional semantics or pre-defined patterns) and is agnostic
to the process model type (e.g. BPMN, Petri net, YAWL, etc.).
- Automatically extracting organizational business logic from process reposi-
tories as a basis for content validation.
- Presenting how to handle a list of three validation error types that can be
deduced from the descriptor model.
- Providing an empirical analysis of the proposed method and showing its
  benefits in identifying content-based validation errors.

The rest of the paper is organized as follows: we present related work in Section 2,
positioning our work with respect to previous research. In Section 3 we provide a
brief overview of the PDC model. Then, we describe our content-based validation
methodology in Section 4 and illustrate it with three error types in Section 5.
Section 6 introduces the software tool and our empirical analysis. We conclude
in Section 7.

2 Related Work

To position our work within the vast research on business process validation, we
introduce a meta-categorization of related works and then locate our contribution
within these categories.

We have grouped current related works into three major research fields (see
Fig. 1): (1) Business Process verification - addressing the question of “does this
process model work?” This research stream largely focuses on avoiding errors
at the structural level of the process model, e.g. deadlocks or incorrect data
links; (2) Business process validation - addressing the question: “does this pro-
cess model works well?” To answer this question it is required to examine the
content, data and context layers of process models to find out whether the pro-
cess logic is correct. For example, a process model that contains the activity “Pay
for the received goods” before the activity “Receive goods” can be verified but
may not be valid; (3) Business process compliance. This topic can address both
verification and validation issues, and the main question it addresses is “does
this process complies with a reference model/pattern?”

2.1 Business Process Verification

Works such as [37,28,29,23,26] focus on avoiding errors at the structural level.
Special emphasis was put on correctness of change operations [25], mainly dealing
with inserting additional activities or moving activities to other model locations [28]. In addition, significant research was carried out on defining the soundness criterion and its derivatives, e.g. [6, 22, 24, 36, 1], aiming to check whether proper execution completion is possible or even guaranteed. Our work does not focus on these aspects (structural layer) but rather on the content and context layers of business process models.

2.2 Business Process Validation

To assure that a process model indeed behaves as expected, it is necessary to take into account what the individual activities in the process actually achieve when they are executed [45, 19].

One line of research in this domain focuses on semantics correctness. [17, 18] use a notion of semantic correctness that builds on annotations to tasks as being mutually exclusive or dependent. In the first case they cannot co-occur in a trace, in the second case they must appear in a certain order. For semantic correctness the process must comply with the annotations. This approach provides somewhat similar features as linear temporal logic [39]. Other works apply semantic validation techniques: techniques that take the annotations and the underlying ontology into account in order to determine whether the tasks are consistent with respect to each other, and with respect to the underlying workflow structure [44, 45, 8]. This domain also includes data flow analysis (e.g. [32]), where dependencies are examined between the points where data is generated, and where it is consumed; some ideas related to this are implemented in the ADEPT system [27]. Data flow analysis builds on compiler theory [2] where data flows are typically examined for sequential programs mostly. Our work is also in the domain of business process validation, yet we rely on the natural language of the model.

Another research line deals with objects and actions as a basis for data validation. The work in [13, 31] deal with compliance of business process models to a set of given reference object life cycles. The work in [33] defines a concept of action patterns, which are closely related to semantic content of a process model. We position our work at this less attended domain of process validation. Works
in this field have mainly focused on pre-defined validation rules or patterns, while our work does not require any predefined pattern or rule to determine validation. In addition, our method deals with action sequences and object lifecycles simultaneously and not separately as was done in previous works.

2.3 Business Process Compliance

Works on Business process compliance have focused on examining whether a given process model is compliant with a certain reference model/pattern. On the technical level, the workflow pattern initiative has identified various patterns for the specification of control flow [41], data flow [30], and resources [30] in workflow management systems. The work in [35] deals also with the planning layer by formalizing process patterns using UML concepts. These compliance works have focused on the structural level of process models, while another line of works focuses on the combination of data and structure [4,44,9,34,7]. The frameworks in [19,7,3], for example, provides general compliance criteria for assessing the compliance of processes with semantic constraints. In addition, some compliance works were aimed at supporting specific purposes, for example: correcting process models at design time [10], verifying changes in existing models [43], identifying compliance in the context of process mining [38], and identifying violations of execution order compliance rules [4].

Our proposed model does not use pre-defined patterns. Rather, it creates a content-aware validation methodology that extracts business logic from actual business process repositories. The aim to automatically extract organizational content distinguishes our work from the previous contributions, and creates a platform for developing enabling applications for content-based validation.

2.4 Related Techniques

Various different labeling schemes and their impact on model understanding were analyzed in [21] and textual labels were used for matching and comparing process models in [11,42]. Our use of the method proposed in [16,15] to decompose activity labels enables us to deduce useful relationships for the task in hand. In addition, the work in [14] improves the label quality of activities automatically, using NLP methods. We use also NLP to decompose activity labels automatically, while relying on the same understanding of activity label required components (namely: object, action and qualifiers).

3 The Descriptor Model

In this Section we describe, for completeness sake, the descriptor model, based on [16,15]. In Section 4 we show how this model supports process model validation. We illustrate the model using Example 1.

The Workflow Management Coalition (WFMC) [5] defines business process as a “set of one or more linked procedures or activities which collectively realize
a business objective or policy goal.” An example of such business process model is the “Evaluate Supplier Performance” process model, presented in Fig. 2 using YAWL [40] with two slight visual representation modifications, convenient for our needs: (a) roles were added at the top of each activity; and (b) predecessor and successor processes are presented as nested activities at the beginning and at the end of the workflow.

![Fig. 2. An example: the “Evaluate Supplier Performance” process model.](image)

In the Process Descriptor Catalog model (“PDC”) [16,15] each activity is composed of one action, one object that the action acts upon, and possibly one or more action and object qualifiers, as illustrated in Fig. 3, using UML relationship symbols. Qualifiers provide an additional description to actions and objects. In particular, a qualifier of an object is roughly related to an object state. State-of-the-art Natural Language Processing (NLP) systems, e.g., the Stanford Parser, can be used to automatically decompose process and activity names into process/activity descriptors.

For example, in Fig. 2, the activity “Add supplier to the approved supplier list” generates an activity descriptor containing the action “add,” the action qualifier “to the approved supplier list,” the object “supplier” and the object qualifier “selected.”

![Fig. 3. The activity decomposition model.](image)

The descriptor model has two atomic elements, namely objects and actions, and four taxonomies, namely an action hierarchy model, an object hierarchy model, an action sequence model and an object lifecycle model. The business action and object taxonomy models organize a set of activity descriptors according

---


2 http://nlp.stanford.edu:8080/parser/index.jsp
to the relationships among business actions and objects both longitudinally (hierarchically) and latitudinally (in terms of execution order), as detailed next.

![Diagram of action hierarchy model](image1)

**Fig. 4.** A segment of the action hierarchy model extracted from the procurement processes.

![Diagram of object hierarchy model](image2)

**Fig. 5.** A segment of the object hierarchy model extracted from the procurement processes.

The longitudinal dimension of actions and objects is determined by their qualifiers. To illustrate the longitudinal dimension of the procurement workflows, a segment of the action hierarchy model is presented in Fig. 4 and a segment of the object hierarchy model is presented in Fig. 5. Consider the complete action (the action and its qualifier) “Manual check.” It is a subclass (a more specific form) of “Check” in the action hierarchy model, since the qualifier “Manual” limits the action of “Check” to reduced action range. It is worth noting that some higher-hierarchy objects and actions are generated automatically by removing qualifiers from lower-hierarchy objects and actions. For example, the action “Search” was not represented without qualifiers in the procurement processes repository, and was completed from the more detailed action: “Search in database” by removing its action qualifier (“in database”) (see Fig. 4). In Section 5 we will show how such elements assist in validating process models by enriching the underlying process repository range. This type of objects and actions are marked with a dashed border.
To illustrate the latitudinal dimension of the procurement process repository, a segment of the action sequence model is presented in Fig. 6 and a segment of the object lifecycle model is presented in Fig. 7. Latitudinally, each object holds: (a) a graph of ordered actions (an “action sequence”) that are applied to that object. For example, the object “New supplier details” is related to the following action sequence: “Receive” followed by “Examine,” “Approve” and “Document” (see Fig. 6); (b) a graph of ordered objects that expresses the object’s lifecycle, meaning - the possible ordering of the object’s states. This sequence is built by locating the same object with different qualifiers along the process diagram. For example, the object “New supplier” is part of the following object lifecycle: “New supplier” → “Examined new supplier” → “Certified supplier”/“Rejected supplier” → “Listed supplier” (see Fig. 7).

Based on the activity decomposition model, it is possible to visualize the operational range of a business process model as a descriptor space comprised of related objects and actions. The descriptor space is a quad-dimensional space describing a range of activities that can be carried out within a process execution flow. The coordinates represent the object dimension, the action dimension, and their qualifiers. For example, the activity “Update approved supplier form with new details” can be represented by the following coordinate: ⟨update, with new details, form, approved supplier⟩. This coordinate represents an actual activity in the business process model: “Maintain suppliers data.”

Once constructed, the descriptor space includes all the possible combinations of descriptor components, forming a much larger and diversified set of possi-
ble descriptors. Hence it includes several “virtual” combinations— that did not originally exist in the original process repository. These virtual combinations, together with existing activities, form an expanded repository that is used for validating process models.

For every two coordinates in the descriptor space a distance function is defined, which represents a linear combination of changes within each of its dimensions. The definition of the distance metric is detailed in [15], allowing us to quantify the cost of navigation within the descriptor space. As an example, consider the two descriptors (requisition, for approval, select, null) and (requisition, null, submit, manually). To navigate from the first descriptor to the second, we first move one step up in the object hierarchy (object hierarchy distance=1) from the object “Requisition for approval” to the object “Requisition” (see Fig. 5). Then, we recede two steps from the action “Select” in the action sequence of “Requisition” (action distance=2), resulting with the action “Submit” (See Fig. 6). Finally, we drill down one step within the action hierarchy (action hierarchy distance=2), and retrieve the action “Manually submit,” and by that we reach the target descriptor. In total, the distance between the two above coordinates is 4.

4 The Content-Based Validation Methodology

Framework

Business logic and business rules, as reflected in an existing process repository, are valuable assets of an enterprise. Therefore, the suggested validation framework aims at seeking consistency of new knowledge (e.g., changes) with existing knowledge and alert if inconsistencies are revealed. In doing so, the method is conservative with respect to existing knowledge and indifferent to new knowledge that has no representation in the existing repository. Some inconsistencies can reflect new valid knowledge in an enterprise modus-operandi (e.g., changes in the execution order of activities). Nevertheless, the suggested framework will also alert such inconsistencies, not because they are necessarily wrong, but in order to enable the designer to correct either the new knowledge or the existing repository if required. Also note that the repository is assumed to be updated with changing organizational policies and know-how.

The validation of changes in existing business process repositories, as proposed in this paper, is based on the compliance of a change with the business logic and business rules as automatically extracted from the existing process repository (before the changes was applied). We refer to the business logic and business rules of the organization as the content of the validation process.

We define two atomic change types, namely the Add and Delete. We consider a modification of an existing activity’s location and a modification of an existing activity’s descriptor (e.g. a change of its object) as a combination of a Delete change followed by an Add change. Also, more complex changes, such as the change of a chain of activities can be broken into these atomic operations and analyzed as such.
The content-based validation method examines an atomic change to an existing process model in terms of its validity, based on the business logic as expressed in an underlying process repository. This repository is represented as a process descriptor space (see Section 3). The suggested method can be used for guiding process designers in a step-by-step manner regarding the validity of each change they make in an existing process repository.

\begin{verbatim}
Input: A basic change to an existing process model
Procedure : If(basic change type == "Add" ||
  Decompose the name of the new activity ("Anew") into a process descriptor format
  Define a list of validation deficiencies
  Calculate the validation score
  Generate a ranked suggestion list for correcting the proposed change
Output: 1. A list of validation deficiencies
         2. The validation score
         3. A ranked suggestion list for correcting the proposed change
\end{verbatim}

Fig. 8. The validation method.

A template of the validation method is illustrated in Fig. 8. It is initiated when a process designer makes an atomic change to an existing process model. When the designer adds a new activity, the activity’s name is decomposed automatically into a process descriptor format, as detailed in Section 3.

The validation method examines the atomic change based on the content rules that are expressed within the descriptor space and that are represented within the four object and action taxonomies (Section 3). As a result, it produces the following validation outputs: (a) a list of validation deficiencies; (b) the validation score - representing the gravity of the error. This score is determined as a sum of penalty and compensation grades, while a lower score means better compliance between the change and the existing process repository. One of the main factors in the score determination is based on distances in the descriptor space, which reflect a measure of proximity between activities in operational terms; and (c) a ranked suggestion list for correcting the proposed change aimed at providing the designer with correction directions that are compliant with the existing process repository’s content. The set of correction suggestions is then being ranked according to suggestions’ proximity to the user’s required change (her “original intention”). For any combination of errors - the method tries correcting each error separately and produces suggestions for each error type.

5 Instantiation of the Content-Based Validation Method

It is possible to classify errors that can be deduced from the descriptor model into three types: (1) class A: errors related to execution order; these errors include: (a) an Invalid Action Sequence Error and (b) an Invalid Object Lifecycle Error;
(2) class B: alerts related to new knowledge. This class includes only one error: an Unknown Descriptor Combination Warning; and (3) class C: errors based on dependencies between descriptor components (e.g. the object qualifier “ready” is never followed by an object qualifier “rejected”, even if different objects are involved). In this paper we show how to identify and rank errors that belong to the first two classes. Errors from the third class can be found using statistics methods, and are not within the scope of this paper.

5.1 An Invalid Action Sequence Error (**Err-IAS**)

**Definition** An invalid action sequence error (**Err-IAS**) occurs whenever an atomic change in a process model creates a new action sequence of an object that is not compliant with the action sequence model of that object in the existing process repository. For example, the actions “Create,” “Submit” and “Review” are all applied sequentially to the object “Requisition” in the same process model within a given process repository. When deleting the activity “Submit requisition” from this process model - a new action sequence is created containing the action “Create” followed by “Review.” Since no such sequence of actions is applied to the object “Requisition” in the original process repository, it is considered a violation of this object’s action sequence (see Fig. 6). Similarly, this error type can also occur when adding a new activity “Approve requisition” before “Review requisition” (see Fig. 6).

**Error Identification** To examine whether an error of type **Err-IAS** occurred, the validation method follows the procedure as illustrated in Fig. 9. The atomic change that was made to the existing process model is inputted to this procedure in terms of change type (**Add/Delete** and the added/deleted activity (**A_{new}**/**A_{old}**) represented as an activity descriptor.

To search for **Err-IAS** occurrences in case a new activity, **A_{new}**, was added to the model, we first retrieve all action sequences of **A_{new}**’s object (denoted by \{AS_{current}\}). For example, in case **A_{new}**’s object is “Inventory type”, then \{AS_{current}\} = \{Define \rightarrow Verify \rightarrow Correct, Search \rightarrow Insert\} (see Fig. 6). We then define a group of all new adjacent action sequences created by **A_{new}** (denoted by \{AS_{new}\}). This group contains all action sequences in the changed process repository that are composed of the following tuple: (the action of **A_{new}**’s predecessor activity, **A_{new}**’s action, the action of **A_{new}**’s successor activity). For example, in case the new activity “Verify inventory type” is added between the activities “Search for inventory type” and “Insert inventory type”, the following new action sequence is created: \{AS_{new}\} = \{Search \rightarrow Verify \rightarrow Insert\} (see Fig. 6). Note that \{AS_{new}\} can contain more that one action sequence in case the new activity is adjacent or creates a split in the activity sequence (e.g., the new process is just before an “OR” split, and therefore there are several successor options). Based on the current and new action sequence sets, it is then possible to define \{AS_{violated}\} as a group of all members in \{AS_{new}\} that are not contained (not a sub-sequence) in any of the members in \{AS_{current}\}. If this group of violated sequences is not empty, each element in this group creates
Input: A basic change in an existing process model

Procedure:
// 1. Determine if Err-IAS occurred
Retrieve all action sequences for A_new/A_deleteds object in the current repository (\{AS\_current\})
Define a group of all adjacent action sequences created by A_new/A_deleteds (\{AS\_new\}/\{AS\_deleted\})
Set \{AS\_violated\} as a group of all members in \{AS\_new\}/\{AS\_deleted\} that are not a sub-sequence in any of the members in \{AS\_current\}
If \{AS\_violated\} is not empty: Err-IAS occurred

// 2. Determine if Err-IOL occurred
Set "OL\_new/deleted" as A\_new/deleteds lean object
Retrieve all object lifecycles that involve OL\_new/deleted in the current repository (\{OL\_current\})
Define a group of all adjacent new object lifecycles created by A\_new/deleted (\{OL\_new\}/\{OL\_deleted\})
Set \{OL\_violated\} as a group of all members in \{OL\_new\}/\{OL\_deleted\} that are not a sub-sequence in any of the members in \{OL\_current\}
If \{OL\_violated\} is not empty: Err-IOL occurred

// 3. Determine if Warn-UDC occurred
If (basic change type == "Add")
   If A\_new is not represented in the original process model: Warn-UDC occurred
   If at least one real activity in the current business process repository contains both
      A\_new's lean object and A\_new's lean action:
         Warn-UDC is classified as a slight modification
      else: Warn-UDC is classified as a major change
   }

Output: The list of validation deficiencies

Fig. 9. Defining the validation deficiencies.

A validation error Err-IAS. Following our example, \{AS\_violated\} = \{Search \rightarrow Verify \rightarrow Insert\}.

An activity deletion can also cause validation errors. To locate Err-IAS errors in this case, we first retrieve all action sequences related to the deleted activity’s (A\_deleted) object in the current repository (\{AS\_current\}). We then define a group of all adjacent action sequences created by A\_deleted (\{AS\_deleted\}). This group contains all action sequences in the changed process repository that are composed of the following tuple: (the action of A\_deleted’s predecessor activity; the action of A\_deleted’s successor activity). For example, if the activity “Verify inventory type” was deleted then \{AS\_deleted\} = \{Define \rightarrow Correct\} (see Fig. 6). We then set \{AS\_violated\} as a group of all members in \{AS\_deleted\} that are not a sub-sequence in any of the members in \{AS\_current\}. As in the add case, if \{AS\_violated\} is not empty: Err-IAS occurred.

Calculating the Validation Score A penalty of $K_{IAS}$ units is assigned with an Err-IAS error type. For add changes there is an additional penalty calculated by the distance (in the descriptor space sense) between the erroneous activity and a valid state. A lower distance means that the two activities are close in terms of operations (they “do” similar activities”) and vise versa. In addition, a compensation can reduce this penalty in the following case. To achieve a compensation, the following steps are carried out: (a) try to locate the nearest object in the object hierarchy that its action sequence model contains a direct sequence of the following adjacent actions (an action sequence that the new or deleted ac-
tively caused): (i) the direct predecessor action of the new/deleted activity; (ii) the new activity’s action in case of an add change; and (iii) the direct successor action of the new/deleted activity; (b) if such an object is located - calculate the hierarchy distance between this object and the object of the added/deleted activity within the object hierarchy model. Compensation is then assigned in an inverse proportion to the object hierarchy distance. It is worth note that $K$’s are set as parameters, and can be tuned with user feedback.

**Generating a Ranked Suggestion List for Correcting the Proposed Change.** To correct the proposed change, in case a new activity was added to the process model, one of a few options are available. First, the new activity can be moved to the nearest location that is compliant with the required action sequence and does not cause an error in the object’s lifecycle of the new activity. Alternatively, the action can be changed to a valid one - from the action lifecycle of the given object. A third option is to change the object to the nearest object in the object hierarchy for which the action sequence is correct and finally, the closest valid activity to $A_{new}$ in the activity descriptor space can be identified. In case of an activity deletion, all successor activities that are located after the deleted activity are deleted and assign the same object with a more advanced action in the action sequence model.

To rank the list of correction suggestions we calculate the distance between the proposed change and the correction suggestions. For this purpose we use the distance function discussed in Section 3. This distance caters to our needs since it is calculated in terms of how many steps are required from one coordinate to the other. In case of an add change the distance is calculated between the added activity and the suggested correction in the model. Since the correction suggestions for delete changes involve the deletion of other activities as well - the distance is calculated as the number of deleted activities.

For example, in case a new activity “Correct an inventory type” is added before the activity “Verify an inventory type” then the following corrections suggestions will be generation (a) move the new activity so that it is located after the nearest activity named “Verify an inventory type” in the model. Since this change requires a movement of only one step in the action lifecycle of the object “Inventory type,” its ranking score is 1 (see Fig. 6); (b) change the action to “Define” resulting with an alternative new activity named “Define an inventory type,” with a ranking score = 1 (see Fig. 6); (c) change the object to “Category code,” since the action sequence “Correct” → “Verify” is applied on this object in its action sequence. This last option involves a movement of one step up and one step down in the object hierarchy model - resulting in a ranking score = 2 (see Fig. 5); (d) all activities with distance = 1 between them and the new activity are adequate in this step. Since any movement to an alternative object or action in the action and object hierarchy models requires a minimum of two steps (see Figs. 5 and 4), we can move in this example one step forwards or backwards in the object lifecycle and action sequence models resulting for example in the activities “Verify inventory type” and “Correct new inventory type” (see Figs. 7 and 6). Continuing this example, in case of an activity deletion, the suggested correction
will be to delete the nearest activity named: “Verify an inventory type” in the process model and the following activity named “Correct an inventory type,” resulting with a ranking score=2 (see Fig. 6).

5.2 An Invalid Object Lifecycle Error (Err-IOL)

Definition An invalid object lifecycle error (Err-IOL) occurs whenever an atomic change in a process model creates a new object lifecycle that is not compliant with the object lifecycle model in the existing process repository. This error type is relevant for both change types. For example, the objects “New supplier,” “Examined new supplier” and “Certified supplier”/”Rejected supplier” are sequential within a given process repository. When deleting the activity “Contact examined new supplier” from this process model - a new object lifecycle is created containing the object “New supplier” followed by “Certified supplier”/”Rejected supplier.” Since no such sequence of objects is represented in the original process repository, it is considered a violation of an object lifecycle (see Fig. 6). Similarly, this error type can also occur when adding a new activity “Contact certified supplier” before “Interview new supplier” (see Fig. 6).

Error Identification To determine if an error of type Err-IOL occurred due to a change within a given process model, the following procedure is conducted (as illustrated in Fig. 9). In case of an add change, it is required first to set $OL_{new}$ as $A_{new}$’s object without its qualifiers. Continuing the previous example, in case the new activity is named “Verify inventory type” then $OL_{new} = 'type';$ Then retrieve all object lifecycles that involve $OL_{new}$ in the existing process repository before applying the new change ($\{OL_{current}\}$). In our example $\{OL_{current}\}$ contains only one object lifecycle: $\{\text{New-inventory-type} \rightarrow \text{Inventory-type} \rightarrow \text{Deprecated-inventory-type}\}$ (see Fig. 7). Then, we define a group of all adjacent new object lifecycles created by $A_{new}$ ($\{OL_{new}\}$). This group contains all object sequences in the changed process repository that are composed of the following tuple: (the object of $A_{new}$’s predecessor activity; $A_{new}$’s object; the object of $A_{new}$’s successor activity). For example, in case the activity “Verify inventory type” was added to the model after the activity “Define a new inventory type” and before the activity “Define a depreciated inventory type;” then $\{OL_{new}\} = \{OL_{current}\}$. We then Set $\{OL_{violated}\}$ as a group of all members in $\{OL_{new}\}$ that are not a sub-sequence of any of the members in $\{OL_{current}\}$. In case $\{OL_{violated}\}$ is not empty, then we deduce that Err-IOL occurred. In our example, since $\{OL_{new}\} = \{OL_{current}\}$, then $\{OL_{violated}\}$ is empty and therefore we deduce that the suggested change does not create an Err-IOL.

The examination whether Err-IOL occurred in case of an activity deletion is conducted in a similar manner to the addition of an activity, except for the phase in which the group of all adjacent new object lifecycles, $\{OL_{new}\}$ is calculated. In this case $\{OL_{new}\}$ contains all object sequences in the changed process repository that are composed of the following tuple: (the object of $A_{deleted}$’s predecessor activity, the object of $A_{deleted}$’s successor activity).

Calculating the Validation Score Once Err-IOL is identified, it generates a penalty of $K_{IOL}$ units. In case of an add change there is an additional penalty
calculated by the distance (in the descriptor space sense) of moving this activity to a new location within the process model that does not involve Err-IOL.

**Generating a Ranked Suggestion List for Correcting the Proposed Change** The list of corrections in case of an *add* change is generated similarly to the procedure described for Err-IAS (see Section 5.1). In case of a *delete* change, delete all the activities that are located after the deleted activity and have more advanced objects in the object lifecycle model. Ranking the generated suggestion list is conducted similarly to the ranking method of Err-IAS correction suggestions (see Section 5.1).

### 5.3 An Unknown Descriptor Combination Warning (Warn-UDC)

**Definition** When the combination of a newly added descriptor components is not represented in the original process model, an unknown descriptor combination warning (Warn-UDC) occurs. In this case we distinguish between two types of changes according to their level of magnitude. A *slight modification* means that there is an actual activity in the underlying business process repository containing the same object and action with different qualifiers. A *major change* means that the object and action within the suggested new activity were not coupled in any of the activities within the underlying business process repository. For example, “Document signed contract” is an actual activity in the procurement process repository. Nevertheless, “Compare conditions” has no representation in this repository, but “Compare vendor conditions” does, and therefore the “Compare conditions” descriptor is considered “artificial” although it represents only a slight change. Since there is no descriptor that combines the action “Compare” and the object “Contracts” in this repository, the option “Compare contracts” is also an “artificial” descriptor that represents a more significant change in the original repository. It is worth noting that this error type is relevant only for *add* type of changes.

**Error Identification** To determine if an error classified as Warn-UDC occurred we check whether $A_{new}$ is represented in the original process model. If not - Warn-UDC occurred. To determine the type of Warn-UDC we check if there is a descriptor in the current business process repository that contains the combination of both $A_{new}$’s object without its qualifiers and $A_{new}$’s action without its qualifiers. If such an activity is found, Warn-UDC is classified as a slight modification. Otherwise, Warn-UDC is classified as a major change (see Fig. 9).

**Calculating the Validation Score** Warn-UDC can generate a penalty of maximum $K_{UDC}$ units, according to the extent to which the added activity was changed compared to real activities that are represented in existing the process repository. The penalty in case of a slight modification, $K_{UDC_{Slight}}$, should be lower than the penalty in case of a *Major change*, $K_{UDC_{Major}}$ (e.g., $K_{UDC_{Slight}} = K_{UDC_{Major}}/2$). We argue that the maximal penalty for Warn-UDC should be lower than the penalty given as a result of Err-IAS and Err-IOL. This is due to the fact that we are faced with a new activity to which pattern of behavior we have no previous knowledge.
Generating a Ranked Suggestion List for Correcting the Proposed Change

To correct a Warn-UDC error type, the following correction options are generated (recall that Warn-UDC is relevant only for add cases). First, we search for all objects that this new action participates in their action sequence model and that when replacing them with the suggested new object - their object lifecycle is not violated. Secondly, we search for all actions that are optional according to the new object’s action sequence model, making sure that these suggestions do not violate the object lifecycle model. Ranking the generated suggestion list is conducted similarly to the ranking method of Err-IAS correction suggestions (see Section 5.1).

6 Empirical Analysis

In this section we introduce our empirical analysis. We describe the implemented validation component in Section 6.1. Our experiments and the associated analysis are reported in Section 6.2.

6.1 Implementation

We have developed Process Model Validation Assistant (PMVA), a system that implements the suggested method for content-based validation in process models. Given an atomic change within a process model, and based on an existing process repository, the PMVA identifies validation errors in process models and guides users in avoiding them. The system implements a client-server architecture. Server side logic is implemented in PHP using a MySql database. It uses a Natural Language (NL) parser - the Stanford Parser - as a Web service for decomposing sentences into linguistic components. The client runs within an Internet browser and is implemented in HTML and JavaScript, with AJAX calls to the server.

6.2 Experiments

We now present an empirical evaluation of the proposed method effectiveness. We first present our experimental setup and describe the data that was used. Based on this setup we present the implemented methodology. Finally, we present the experiment results and provide an empirical analysis of these results.

**Experiment Setup**

The Process Model Validation Assistant software (PMVA, see Section 6.1) was installed on a workstation running Windows XP, IIS6, PHP 4.8 and MySQL 5.0. This workstation served both as the server and the client, running Internet Explorer 7 as the application container and presentation layer. The error penalties were set as follows: $K_{IAS} = 6$, $K_{IOL} = 6$, $K_{UDCMajor} = 4$ and $K_{UDCSlight} = 2$.

**Data** We chose a set of 14 real-life processes from the Oracle Business Model (OBM)\(^3\), comprising of nine business processes from the “Procurement” category,

\(^3\) [http://www.oracle.com/applications/tutor/index.html](http://www.oracle.com/applications/tutor/index.html)
with 96 activities altogether, and five business processes from the “Inventory” category, with 31 activities altogether. The “Procurement” data set contains related, sequential activities and therefore represents a focused operational area. The “Inventory” data set represents an extended business area, featuring loosely coupled business logic. Using the selected 14 processes we created a “process repository database” (see Section 6.1).

Fig. 10. An example of a single experiment conduct.

**Evaluation Methodology** To evaluate the suggested method we conducted 14 experiments. At each experiment, a single process was removed from the database and then each of its activities were re-added to the database as a “new activity” placed in the original location of the removed process (see Figure 10 for illustration). After validating this add change, the same activity was removed again from the model creating a delete change that was also validated. Each experiment was conducted according to the following steps: (a) preparation: remove one of the processes from the database so that the database will not contain any of its descriptor components; (b) Locate the last activity in the predecessor process of the removed process (anchor activity) - as was defined in the original database (before the selected process was removed); (c) run the PMVA in a stepwise manner. At each phase we try to: (i) add an activity (new activity) from the removed process to the existing model - so that it follows the anchor activity. Then, the PMVA software validates this add change and outputs its validation results; (ii) delete that same activity from the model and validate this delete change using the PMVA.

For example, consider an experiment where Process$_2$ is removed from the database (illustrated in Fig. 10). At first, we try to add to the reduced model (the process model without Process$_2$) the new activity: A$_{1,2}$. Then we call the PMVA and receive the validation result of this process model change (an add change). We then delete A$_{1,2}$ from the process model (after recording it in the repository) and again, invoke the PMVA for calculating the validation result of a delete change. In a similar manner, in the following m-1 sub-experiments, we try to add each time a different activity from Process$_2$ as a new activity to the reduced model as a following activity to A$_{n,1}$ and then delete it from
this model, creating a delete change. For each such change (add and delete) the PMVA is invoked, calculating this change’s validation error parameters. Note that after each sub-experiment the reduced process model returns to its original form (a database without the removed process) so that the sub-experiments are independent and their execution order is not important.

Results and Analysis Table 1 presents a summary of the experiment results for add changes. Each experiment of adding a new activity to the existing process model was based on a database with the set of all activity descriptors in all process models, excluding the set of activity descriptors of one process. The experiment results presented in this table show that as the suggested new activity is further away from the anchor activity: (a) the validation error score grows; and (b) the lowest distance of the proposed change in the suggested corrections list increases, meaning that the error requires a larger number of correction steps. In addition, the first activity in the removed process (the correct activity) was found in the correction suggestions list at a higher location as the new activity was closer to the anchor activity (whenever the correct activity was not found in this list - a fine of 20 units was set instead). In other words, we found that as a newly suggested activity is less adequate in the existing process model (more distant from the anchor activity), the gravity of the validation error produced by the PMVA is higher and the suggested corrections require more changes in the original model.

The resulted error gravity directionality (from lower to higher) as the new activity is less adequate to the model can be explained as follows: when adding a more inadequate activity to the model, it usually violates its object and action taxonomies since it is less reasonable that the current taxonomies already represent the new object and action relations it poses. Moreover, it is more likely that such cases will: (a) involve an additional penalty related to the distance between the erroneous activity and a valid state in the descriptor space; (b) not gain any compensation that is given in case of similarities to other objects and actions (see Section 5). The compensation factor also explains the jump in the validation error score that occurred when the distance from the anchor activity grew from 3 to 4 (a jump from 6.7 to 11.4) (see Table 1). When the new activity was distant up to three activities from the anchor activity, a similarity based compensation was gained. From this point on, no such compensation was awarded and hence the jump in the total error gravity. In addition, more correction actions are required in such cases since there are more steps required for relocating the new activity in the model or modifying it descriptor to a valid one.

The experiment results for delete changes are presented in Table 3. We observe a decreasing validation error and smaller distances to suggested solutions. We can learn that as the deleted activity is further away from the anchor activity: (a) the validation error score reduces; and (b) the lowest distance in the suggested corrections list of the proposed change decreases (meaning that the error requires less correction steps). In other words, it was found that when an activity that is not highly related to a process model is deleted - the gravity of the validation error it causes is lower. This error gravity directionality can be
Table 1. Experiment results: distribution of validation error gravity vs. the location of the activity in the removed process - add changes.

<table>
<thead>
<tr>
<th>Distance from the anchor activity</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation error score (avg.)</td>
<td>3.8</td>
<td>4.6</td>
<td>6.2</td>
<td>6.7</td>
<td>11.4</td>
<td>12.3</td>
<td>17.1</td>
<td>17.4</td>
<td>23.0</td>
<td>35.2</td>
</tr>
<tr>
<td>Lowest distance in the suggested corrections list (avg.)</td>
<td>2.1</td>
<td>2.7</td>
<td>3.2</td>
<td>4.5</td>
<td>4.7</td>
<td>6.0</td>
<td>6.4</td>
<td>6.4</td>
<td>7.3</td>
<td>9.4</td>
</tr>
<tr>
<td>Location of the correct activity in the suggested corrections list (avg.)</td>
<td>2.2</td>
<td>3.4</td>
<td>5.1</td>
<td>12.1</td>
<td>15.3</td>
<td>16.0</td>
<td>18.2</td>
<td>18.4</td>
<td>20.0</td>
<td>20.0</td>
</tr>
</tbody>
</table>

explained as follows: when adding an inadequate activity to the model it is not connected tightly to the object and action taxonomies (e.g., if the new activity’s object is not related to the objects that are handled by the previous process (the process it was added to) - this addition does not change the object lifecycle model). Therefore, when deleting this activity from the model, it does not cause significant violations to the object and action taxonomies and hence the lower error score. In addition, less correction actions are required in such cases - since in delete changes - correction suggestions involve deleting activities that were related to the deleted activity. Since the number of such related activities is lower as the deleted activity is less connected to the the process model, less correction steps are required. This reason explains also the jump in the validation error score when the distance from the anchor activity grew from 4 to 5 (a jump from 18.1 to 12.5) (see Table 3). On average, in 87% of the experiments, the first five new activities that were added to the model involved an object that was also handled in the previous process. In such cases, the new activity extended both the object lifecycle and the action sequence taxonomies. Therefore, on their removal - these activities caused a higher validation error.

Table 3. Experiment results: distribution of validation error gravity vs. the location of the activity in the removed process - delete changes.

<table>
<thead>
<tr>
<th>Distance from the anchor activity</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation error score (avg.)</td>
<td>25.4</td>
<td>22.0</td>
<td>18.4</td>
<td>19.1</td>
<td>18.1</td>
<td>12.5</td>
<td>9.7</td>
<td>6.2</td>
<td>5.8</td>
<td>3.4</td>
</tr>
<tr>
<td>Lowest distance in the suggested corrections list (avg.)</td>
<td>3.1</td>
<td>2.9</td>
<td>2.9</td>
<td>2.7</td>
<td>2.4</td>
<td>2.1</td>
<td>1.5</td>
<td>1.4</td>
<td>1.1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

To summarize, we have shown the usefulness of using a descriptor repository in validating changes in an existing business process model. We also showed the method to be effective in the given experimental setup, both in terms of the gravity of the validation error it produces and in the correction suggestions it generates.

7 Conclusions

We proposed a methodology for content-based validation of business process models. The methodology includes deficiency identification (using existing descriptors as a reference), validation score calculation, and generation of a ranked list of corrections. The suggested methodology and accompanying software tool will save design time and will support non-expert designers in creating and checking new business process models.

We consider this work as a starting point that can already be applied in real-life scenarios, yet several research issues remain open, including: (1) an extended empirical study to further examine the quality of the validation results; (2) a learning mechanism that will take into account previous designer preferences.
and adjust (in real time) the validation mechanism; and (3) weight definition for the relationships presented in the object and action taxonomies, to be used in determining the relative strengths between the extracted business rules; (4) introducing a transactional concept which will allow to consider several changes in a particular content; (5) applying the methodology to richer models, for instance, containing non-sequential processes where branching depends on conditions. We also consider as a future work the investigation of further language semantics by using more advanced natural language processing techniques.

References


