Semantic Machine Learning for Business Process Content Generation

Avi Wasser[1] and Maya Lincoln[2]

[1] University of Haifa, Israel
awasser@haifa.ac.il
[2] ProcessGene Ltd.
maya.lincoln@processgene.com

Abstract. Business process modeling is considered a manual, labor intensive task. It requires significant domain expertise and may be prone to errors or inconsistencies due to reliance on human factors. Hence, automation through reuse of predefined process models is becoming a common practice for generating new models. In this work we extend a previously proposed generation method by adding semantic learning capabilities that opt to improve the quality of generated business process models. The learning mechanism analyzes, in real-time, the linguistic relationships between process descriptors and adjusts them according to human inputs that are accumulated during the modeling process. To demonstrate the method we present a case-study from the food manufacturing industry. To estimate the applicative value we further experimented the method on a real-life process repository, showing that the learning mechanism increases the effectiveness of the previously suggested method for automating the design of new business process models.

Keywords: Business process model design, Business process repositories, Business process semantic similarity, Machine learning.

1 Introduction

Business process modeling is considered a manual, labor intensive task. It requires significant domain expertise and is prone to errors or inconsistencies due to reliance on human, non-machine assisted activities. Hence, automating the reuse of predefined process models is becoming a common practice for creating new business process models. Research in this field has focused on structured reuse of existing building blocks and pre-defined patterns that provide context and sequences [5]. The work in [11] established a method for designing new business process models from process repositories, based on semantic similarity. This method guides business analysts that are non domain experts, by suggesting process steps that are relevant for the realization of the process goal. The business logic for such suggestions is extracted from process repositories through the analysis of existing business process model activities. Each activity is encoded automatically as a *semantic descriptor* using the Process Descriptor Catalog ("PDC") notation, suggested first in [12] and elaborated in [11].

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 74–91, 2012.

This work aims to take the framework presented in [11] several steps forward by: (1) proposing a machine learning mechanism that will take into account the designer preferences at each design phase and adjust (in real-time) the suggestions made by the automated design mechanism at the next design phases; and (2) applying the suggested framework on real-life processes. Our work presents the following innovations: (a) it provides a generic, real-time, machine learning mechanism for the design of new business process models; (b) it equally utilizes objects and actions for machine learning: we make use of all activity linguistic components (object, actions and their qualifiers) concurrently, without special focus on objects (as object centric methods do) or on actions (as activity-centric methods do); and (c) it significantly extends the descriptor space model [11] to enable the ongoing update of its underlying business logic as a result of learning, turning it into a *learning* descriptor space.

The proposed extended method can assist process analysts in designing new business process models while making use of knowledge that is encoded both in the design of existing, related process models, and also from the accumulated knowledge of the human designer. The extended framework is illustrated throughout the paper using an example based on real-life processes from the food manufacturing industry. We also use this process-repository to demonstrate a case study, and use it as a basis for experiments that measure the effectiveness of the proposed machine learning framework.

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 present the semantic descriptor notion [11] as background to this work. In Section 4 we elaborate the descriptor space concept presented in [11] to support the learning framework. We describe the extended automation method for designing new business process models in Section 5. Section 6 introduces the case study and our experiments and empirical analysis. We conclude in Section 7.

2 Related Work

Research on automated process model generation mainly focuses on supporting the design of alternative process steps within existing process models [16,5,6,1]. The identification and choice of relevant process components are widely based on the analysis of linguistic components - actions and objects that describe business activities. Most existing languages for business process modeling and implementation are activity-centric, representing processes as a set of activities connected by control-flow elements indicating the order of activity execution [19,10]. Other works are action-centric, analyzing the connectivity and relationships between actions [17].

In recent years, an alternative approach has been proposed, which is based on objects (or artifacts/entities/documents) as a central component for business process modeling and implementation. This relatively new approach focuses on the central objects along with their life-cycles. Services (or tasks) are used to specify the automated and/or human steps that help move objects through their

life-cycle, and services are associated with artifacts using procedural, graph-based, and/or declarative formalisms [8]. Such object-centric approaches include artifact-centric modeling [14,2], data-driven modeling [13] and proclets [18].

Although most works in the above domain are either object, action or activity centric, only few works combine the three approaches in order to exploit an extended knowledge scope of the business process. The work in [9] presents an algorithm that generates an information-centric process model from an activity-centric model. The works in [12,11] present the concept of business process descriptor that decomposes process names into objects, actions and qualifiers. In our previous review (see [11]) we identified only a few works that addressed the design of *new* models. The work presented in [13], for example, utilizes the information about a product and its structure for modeling large process structures. [15] presents a method for designing new manufacturing related processes based on product specification and required design criteria. The work in [6] supports modeling recommendations based on the interpretation of process descriptions.

Focusing on our previous work in [11], we realized that although it presented a method for new process model design based on business process descriptor analysis, it didn't apply a machine learning mechanism to provide a real-time improvement of the suggested framework based on human inputs. We did find some works that involve learning as means for achieving other business processes repository utilization targets. For example, some works suggest frameworks for better understanding business process models that apply learning during simulations [3] or runtime [4], and the work in [7] uses reinforcement learning to solve a resource allocation optimization problem.

In this work we elaborate research in this domain by: (a) proposing a learning mechanism that will take into account the designer preferences at each design phase and adjust (in real-time) the suggestions made by the design assistant framework at next design phases; and (b) elaborating the descriptor space concept to support learning frameworks.

3 The Semantic Descriptor Model

In the Process Descriptor Catalog model ("PDC") [12] 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. 1, 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,"[1] can be used to automatically decompose process and activity names into *process/activity descriptors*.

For example, the activity "Manually mix wheat flour" generates an activity descriptor containing the action "mix," the action qualifier "manually," the object "flour" and the object qualifier "wheat."

[1] `http://nlp.stanford.edu:8080/parser/index.jsp`

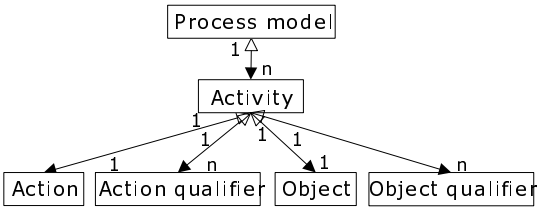

Fig. 1. The activity decomposition model

In general, given an object, o, an object qualifier, q_o, an action, a, and an action qualifier, q_a, a descriptor, d, is denoted as follows: $d = (o, q_o, a, q_a)$. A *complete action* is an action with its qualifier, and similarly, a *complete* object is an object with its qualifier. We denote by $a(d)$ the complete action part of the descriptor, *e.g.*, "manually mix" in the example, and similarly, $o(d)$ denotes the complete object part. In addition, $q_o(d)$ denotes the object qualifier part, $o_{thin}(d)$ denotes the object part of the descriptor (without its qualifiers), and $a_{thin}(d)$ denotes the action part.

3.1 A Descriptor Model for Process Design

The PDC model of [12] was enhanced by [11] to support automated process design. The extended model has two basic elements, namely objects and actions, and four taxonomies are delineated from them, namely an *Action Hierarchy Model* (AHM), an *Object Hierarchy Model* (OHM), an *Action Sequence Model (ASM)* and an *Object Lifecycle Model* (OLM). The business action and object taxonomy models organize a set of activity descriptors according to the relationships among business actions and objects both hierarchically and in terms of execution order, as detailed next.

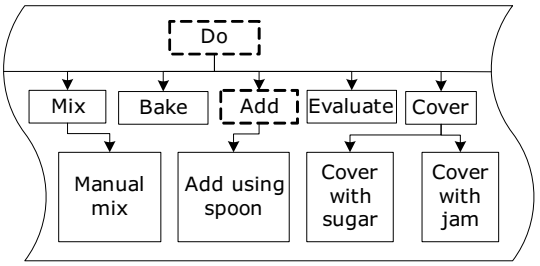

Fig. 2. Segment of an action hierarchy model

The hierarchical dimension of actions and objects is determined by their qualifiers. To illustrate the hierarchical dimension, a segment of the action hierarchy model of a bakery is presented in Fig. 2 and a segment of the object hierarchy

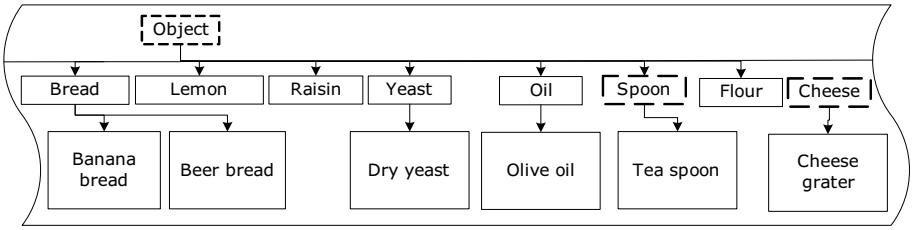

Fig. 3. Segment of an object hierarchy model

model of a bakery is presented in Fig. 3. In the action hierarchy model, for example, the action "Mix" It is a subclass (a more specific form) of "Manual mix," since the qualifier "Manual" limits the action of "Mix" 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 "Add" was not represented without qualifiers in the bakery process repository, and was completed from the more detailed action: "Add using spoon" by removing its action qualifier ("using spoon") (see Fig. 2). This type of objects and actions, namely: *artificial* objects and actions, are marked with a dashed border. In addition, a root node "Do" is added to any action hierarchy model and a root node "Object" is added to any object hierarchy model.

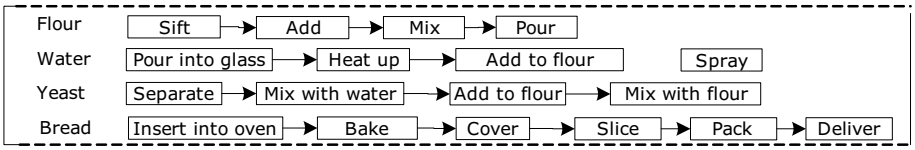

Fig. 4. Segment of an action sequence model of a bakery

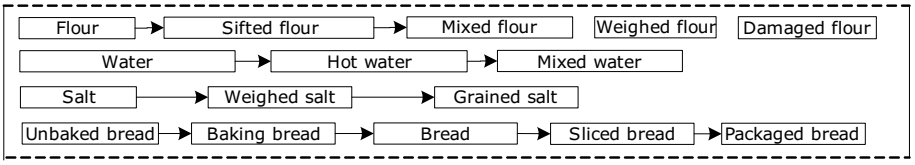

Fig. 5. Segment of an object lifecycle model of a bakery

In the action sequence model, each object holds a graph of ordered actions that are applied to that object (see illustration in Fig. 4). For example, the object "Flour" is related to the following action sequence: "Sift" followed by "Add," "Mix," and finally "Pour."

In the object lifecycle model each object holds a graph of ordered objects that expresses the object's lifecycle, meaning - the possible ordering of the object's

states. In other words, an OLM is a graph of ordered complete objects that expresses the possible ordering of the object's states (see illustration in Fig. 5). For example, the object "Flour" is part of the following object lifecycle: "Flour"–> "Sifted flour"–>"Mixed flour."

Note that ASM and OLM are defined as sets of sequences and not as a single sequence, since different unconnected processes in the repository may involve the same object, and therefore contribute a different sequence to these models. We denote the procedure for creating an ASM and an OLM for a complete object, o, as: $create_{ASM}(o)$ and $create_{OLM}(o)$, respectively.

4 The Learning Descriptor Space

In this section we extend the Descriptor Space (DS) concept presented in [11] to express learned information regarding a designer's preferences and knowledge.

Based on [11], 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 describes 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. Therefore, each space coordinate represents an activity as a quadruple $AC = \langle o, q_o, a, q_a \rangle$.

Once constructed, the descriptor space includes all the possible combinations of descriptor components, forming a large and diversified set of possible descriptors. It includes several "virtual" combinations - that did not originally exist in the original process repository. These virtual combinations, together with existing activities, form a significantly extended repository that is used for the automated design of new business processes as well as for the learning mechanism. For every two coordinates in the descriptor space a *distance function* is defined as a linear combination of changes within each of its dimensions. Therefore, four specific distance measures are defined as follows.

Definition 1. ***Object distance* (OD)***: Let o_i and o_j be two objects, OD_{ij} is the minimal number of steps connecting o_i and o_j in the object lifecycle model.*

In a similar way *Action distance*, AD, is defined, calculated based on the action sequence model. For example, the action distance between "Sift" and "Mix" when acted on "Flour" is 2 (see Fig. 5).

Definition 2. ***Object hierarchy distance* (OHD)***: Let o_i and o_j be two objects, OHD_{ij} is the minimal number of steps connecting o_i with o_j in the object hierarchy model.*

In a similar way *Action hierarchy Distance*, AHD, is defined, calculated based on the action hierarchy model.

Definition 3. ***Object learned proximity* (OLP)***: Let o_i and o_j be two objects in the object lifecycle model, OLP_{ij} is a constant positive number representing the learned proximity between o_i and o_j in the object lifecycle model. OLP is calibrated to 0 at the beginning of the first design step, and is updated according to*

the learning mechanism presented in Section 5. Higher values of OLP represent learned proximities.

In a similar way: (1) *Action learned proximity*, ALP, is defined, calculated based on the action sequence model; (2) *Object hierarchy learned proximity*, $OHLP$, is defined, calculated based on the object hierarchy model; and (3) *Action hierarchy learned proximity*, $AHLP$, is defined, calculated based on the action hierarchy model.

OD, AD, OHD and AHD are combined with OLP, ALP, $OHLP$ and $AHLP$ to generate a specific distance function between any two activities AC_i and AC_j, as follows (brackets are for readability only):

$$Dist(AC_i,AC_j)=(OD_{ij}-OLP_{ij})+(AD_{ij}-ALP_{ij})+(OHD_{ij}-OHLP_{ij})+(AHD_{ij}-AHLP_{ij}) \quad (1)$$

It is worth noting that the hierarchy distances (OHD and AHD) can always be calculated since the hierarchy models that they rely on are bidirectional trees. However, the distances OD and AD can be undefined in some cases (*e.g.*, when the two objects are not connected in the object hierarchy model, or when the two actions are not acted upon the same object and therefore do not take part in the same action sequence). In these cases the above distance components contribute a *no-connection* distance to the overall distance function. This distance is an application-specific tunable parameter.

In general, it is possible to navigate within the descriptor space (hence, move from one descriptor to another) in a meaningful way. This navigation enables us to move up to more general or drill down to more specific action and object scopes as well as to navigate to: (a) preceding and succeeding actions that act on the descriptor's object and (b) advance to a successor (more advanced) state of the object's current state or recede to a predecessor (less advanced) state.

5 Method for Automated Generation of Business Process Content

In this section we extend the method presented in [11] by adding a learning mechanism.

The design assistance method relies on an underlying process descriptor space and at any phase, based on the user's decision, it either refines an existing process activity or suggests a next process activity. Based on each such user decision, the design assistant learns more about the relationships between the involved actions and objects and adjusts their distances in the descriptor space accordingly.

The design assistant is illustrated in Fig. 6. The design process starts when a process designer defines the name of the new process model. This name is decomposed into a process descriptor format. For example, a new process named: "Bake raisin bread," will be transformed into the following process descriptor: object="bread," action="bake," object qualifier="raisin," action qualifier="null."

Based on the process descriptor input, the design assistant produces options for the first process activity (see Section 5.1). The process designer reviews the

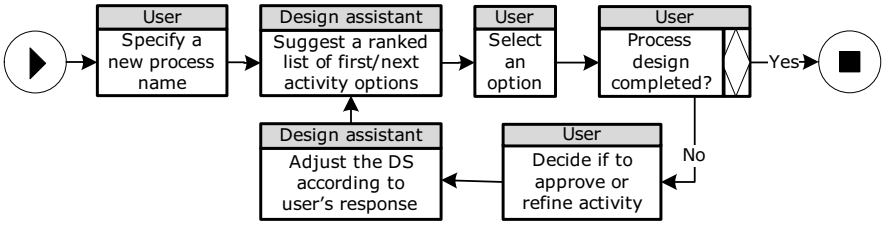

Fig. 6. The design assistant mechanism

output option list, and either selects the most suitable first activity for the newly designed process, or suggests an alternative. At any next phase the designer either requests to refine the current activity (see Section 5.2) or advance to design the next activity (see Section 5.3). Each time the design assistant is requested to suggest activities as part of the design process it outputs a list of options, sorted and flagged according to the option's relevance to the current design phase and based on the current descriptor space (see Section 5.4). Based on the designer's preferences regarding the most suitable activity from the option list and whether to refine or proceed to the next activity, the design assistant deduces new knowledge regarding relationships between actions and objects in the descriptor space and adjusts its distances accordingly (see Section 5.5).

After selecting the most suitable process activity from the suggested list, the designer examines the newly designed process model to determine if it achieves the process goals. If goals are achieved, the design is terminated; else - the design procedure continues until the process goal is achieved.

5.1 Suggesting the First Process Activity

To suggest the first process activity, the design assistant searches the target object and its more specific objects within the object hierarchy model. It then creates first activity suggestions in the format of activity descriptors comprised of the retrieved objects and the first action that acts upon them in the action sequence model. Continuing the example above, the following first activity options will be suggested (see Fig. 3 and Fig. 4): "Insert bread into oven" and "Insert banana bread into oven."

5.2 Refining the Currently Suggested Process Activity

A refinement can be performed by five orthogonal methods. To illustrate each of these methods we will show how the action "Cover bread" can be refined.

Action and Object Refinement. To refine the reference action, the design assistant navigates the descriptor space by drilling *down* the action hierarchy to more specific actions. It then combines the retrieved, more specific, actions with the reference object. The refinement of objects is done in a similar manner.

By applying an action refinement to our example's reference activity, the refinement option: "Cover bread with sugar" is retrieved (see Fig. 2).

Action and Object Generalization. The generalization method is similar to the action and object refinement method, only this time the design assistant navigates the descriptor space by moving *up* the action and the object hierarchical dimension, respectively.

Advance an Action or an Object State. To advance the object's state within an activity, the design assistant navigates the descriptor space by moving *forward* in the object lifecycle sub-dimension. In a symmetrical manner, to advance an activity's action, the design assistant moves forward in the action sequence sub-dimension of the descriptor space. In our example the object "Sliced bread" represents a more advanced state of the object "Bread" (see Fig. 5) and the action "Slice" follows the action "Cover" in the action sequence applied on "Bread" (See Fig. 4). Therefore, the following two refinement suggestions are constructed: "Cover sliced bread" and "Slice bread."

Recede to a Less Processed State of the Object or to a Former Action. The receding method is similar to the advancing method, only this time the design assistant navigates the descriptor space by moving *backwards* in the object lifecycle and action sequence sub-dimensions. For example, the action "Bake" is acted on "Bread" before this object is covered (before the action "Cover" is applied) (see Fig. 4), hence creating the option: "Bake bread."

Move to a Sibling Action or Object. In order to move to a sibling action, the design assistant moves horizontally within the action hierarchical sub-dimension. By fixing the reference action's level, it retrieves sibling actions for this action. Moving to a sibling object is conducted in a similar manner. Continuing our example, a navigation to sibling actions to "Cover" retrieves a list of activities that includes: "Mix bread" and "Evaluate bread" (see Fig. 2).

5.3 Suggesting the Next Process Activity

This step can be achieved in two alternative ways: either by advancing to a later action that acts on the currently accepted (reference) object, or advancing to a sibling object combined with the reference activity's action. To demonstrate this step, consider the activity following "Add yeast to flour." The design assistant finds in the action sequence model the option: "Mix yeast with flour" (see Fig. 4). In addition, sibling objects to "Yeast" are also retrieved from the object hierarchy model, creating additional options such as "Add raisin" and "Add lemon" (see Fig. 3).

5.4 Preparing a Set of Output Options

The design assistant assesses the output options in each navigation phase and combines an ordered option list to assist the user in selecting the most suitable

option. The design assistant sorts the options according to their relevance to the current design phase based on two considerations. First, on proximity to the design phase reference coordinate - which represents the last selected activity when suggesting a refined or next activity, or to the targeted process descriptor when suggesting the first process activity. Second, the design assistant considers to what extent was it changed comparing to actual activities that were part of the underlying process repository. Therefore, the construction of the ordered option list is conducted according to the following three stages: (a) sort by proximity to the reference activity; (b) internally sort by similarity to processes in the repository; and (c) flag each option, as further detailed below.

Sort by Proximity to the Reference Activity. The design assistant calculates the distance between the reference coordinate and each of the list options (see definition 1), and sorts the list in an ascending order - from the closest to the most distant option.

Internally Sort by Similarity to Processes in the Repository. The design assistant also takes into account the extent to which a proposed activity was changed in comparison to actual activities in the underlying process repository. For this purpose the design assistant distinguishes between three change levels: (a) *No change*- the suggested activity is represented "as is" within the underlying business process repository. These options are not marked by any flag; (b) *Slight modification* - there is an actual activity in the underlying business process repository containing the same object and action with different qualifiers. These options are marked with "~"; (c) *Major change* - the object and action within the suggested activity were not coupled in any of the activities within the underlying business process repository. These options are marked with "M".

According to the example presented in Section 5.3, several options were generated as candidates for next activities to be conducted after the activity "Add yeast to flour." Most of these options were produced by combining the action "Add to flour" with siblings of the object "Yeast," hence having the same distance from the reference activity. Nevertheless, these options can further be differentiated. For example, "Add lemon" is an actual activity in the bakery process repository, and therefore is flagged as such. Nevertheless, "Add oil" has no representation in this repository, but since "Add olive oil" does, this option is flagged by "~." Since there is no descriptor that combines the action "Add" and the object "Spoon" in this repository, the option "Add spoon" is flagged by "M."

Flag Each Option. After assessing each option's relevance to the current navigation phase and sorting the option list accordingly, the design assistant tags each option with both the numerical distance value and the change level. For example, the option "Add oil" from the example above will be flagged "[2,~]."

5.5 Applying a Learning Mechanism

At each design step the process designer is provided with a list of optional next activities, and is required to make the following two decisions: (1) select the most suitable descriptor, d_s, from the option list; and (2) decide whether to refine the selected descriptor or to accept it and proceed to the design of the next activity. The learning mechanism receives the two above designer decisions and deduces new conclusions regarding the underlying business rules and business know-how encoded in the descriptor space. As a result, the learning mechanism adjusts the descriptor space, which is then used as a basis for producing the next/refined optional activity list. The learning mechanism analyzes the following designer decisions as detailed next.

Selecting Artificial Activities. As presented in Section 3, some actions and objects in the descriptor space are artificial (automatically generated in the action and object hierarchy models), and hence are not represented in any of the process repository activities. In case the designer selects a descriptor that contains an artificial action or object, the learning mechanism deduces that these are "real-life" semantic elements and amends the descriptor space as follows.

1. Representing the artificial action or object as a real one in its hierarchy model. For example, in case the designer selects the activity "Use grater," the object "Grater" becomes a real object in the object hierarchy model, and its dashed line is replaced by a regular one (see Fig. 3).
2. Generating a new action sequence for the selected activity's object. Following the above example, a new sequence in the action sequence model is automatically generated for the object "Grater," including one action, "Use." Formally, this suggestion can be given by:

$$create_{ASM}(o(d_s)) \tag{2}$$

3. Automatically updating the object lifecycle model by adding a representation of the selected activity's object. Following the above example, the object "Grater" is added to the OLM as a single-object lifecycle. Formally, this suggestion can be given by:

$$create_{OLM}(o(d_s)) \tag{3}$$

Selecting Virtual Activities. Some options in the suggested descriptor list are marked as "Slight modification" or "Major change," hence representing a virtual activity in the descriptor space (e.g. "Add oil," "Add spoon"). In case the process designer selects such options, the learning mechanism deduces that the complete action can be applied to the complete object in "real-life" processes. the descriptor space is therefore amended as follows.

1. Adding $a(d_s)$ to the action sequence of $o(d_s)$. Following the above example, the action "Add" is added to the action sequence of "Oil" in the action sequence model. In order to update the ASM, we regenerate $ASM(o(d_s))$. Formally, this suggestion can be given by Eq. 2.

2. Updating the object lifecycle of $o(d_s)$. Formally, this suggestion can be given by Eq. 3.

Selecting Distant List Options. As mentioned above, each optional activity flag indicates the activity's numerical distance from the reference descriptor, d_r. Options with the minimal distance, $Dist_{min}$, are presented at the top of that list, followed by other options in an ascending distance order. For example, given the reference activity "Sift flour," the optional activity list starts with the options "[1] Add flour" followed by "[2] Mix flour" (see Fig. 4). In case the process designer selects an activity flagged by distance $Dist(d_s) > Dist_{min}$, it is possible to learn that the *actual* distance between the selected and reference activities is shorter than the one currently represented in the DS. Higher values of $Dist(d_s)$ represent a greater difference between the actual and current DS. In our example, the designer may select the activity "[2] Mix flour" although it is not the first option in the list.

In response to such designer selections, the learning mechanism reacts as follows. First, it calculates the difference between the distance components - OD, AD, OHD and AHD of $Dist_{min}$ and those of $Dist(d_s)$. We denote this difference as the actual gap, AG. In our example the actual gap between OHD of the first list option, $d_{first} = (flour, null, add, null)$, and the selected option, $d_s = (flour, null, mix, null)$, is: $AG_{OHD}(d_{first}, d_s) = 0$ (See Fig. 3), and the actual gap between their action distance is: $AG_{AD}(d_{first}, d_s) = AD(d_s) - AD(d_{first}) = 2 - 1 = 1$ (see Fig. 4).

Second, the learning mechanism corrects the learned proximities related to each of the four distance components (OLP, ALP, $OHLP$ and $AHLP$ (see Section 4)) by adding them a numerical value proportional to their actual gap. Additions to learned proximities in case of "next activity" decisions are more significant than additions in case of "activity refinement" decisions, since the first indicates an *exact* match while the second indicates *proximity* only. Formally: the object learned proximity between $o(d_r)$ and $o(d_s)$, $OLP_{new}(o(d_r), o(d_s))$, in case of a "next activity" decision, is corrected as follows: $OLP_{new}(o(d_r), o(d_s)) = OLP(o(d_r), o(d_s)) + AG_{OD}(d_{first}, d_s) * h_{next}$, where h_{next} is a tunable parameter that can be optimized in a future work. The calculation in case of an "activity refinement" is similar, using the tunable parameter $h_{refine} < h_{next}$. ALP, $OHLP$ and $AHLP$ are updated similarly. In continuous to our example, assuming the designer chooses to move to the next activity, "Mix flour," and assuming $ALP(o(d_r), o(d_s)) = 0$ and $h_{next} = 0.1$, we calculate the new action learned proximity as follows: $OLP_{new}(o(d_r), o(d_s)) = 0 + 1 * 0.1 = 0.1$.

Selecting Refined List Options. In some cases the next activity is selected after several refinement steps. After such design phases, that involve n refinements, the learning mechanism shortens the distance between the previous (first reference) selected descriptor, d_r, and the new (currently selected one), d_s, as follows: $Dist_{new}(d_r, d_s) = Dist(d_r, d_s) - n * h_{next}$.

6 Case Study and Experiments

6.1 Case Study: An Example for Designing a New Process Model

To illustrate the proposed framework we present two short examples from the field of bakeries. The bakery process repository covers bakery activities starting from the raw material procurement, through the manufacturing of baked goods and terminating as the baked goods are sold to the customer. The newly designed processes are related to the bakery field, but are not covered by the process repository. The first new process, "Bake a chocolate cake," extends the process repository by baking a new product. The second new process, "Sell baked goods via the Internet" extends the process repository by offering an additional service to customers, that eliminates the need for their arrival to the store. Using these examples we will show how the learning mechanism can be utilized to guide and improve the design of new processes. In both examples h_{next} was set to 0.6 and h_{refine} was set to 0.5.

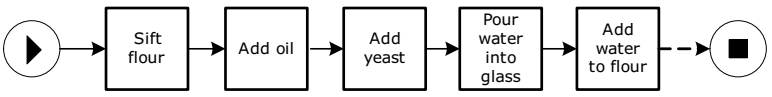

Fig. 7. The new designed process diagram for "Bake a chocolate cake"

The first example supports the design of a new business process for: "Bake a chocolate cake." The generated output (new process model) of this example is illustrated in Fig. 7 as a YAWL (Yet Another Workflow Language) diagram. The design process starts when the (human) process designer inserts the following process descriptor: (action="bake", action qualifier=null, object="cake", object qualifier="chocolate") to the process assistant and determines that the first activity is: "Sift flour." Respectively, the process assistant searches the descriptor space, looking for next activity possibilities. The result set includes the following activities (see Sections 5.3 and 5.4): "[1] Add flour," "[2] Sift oil" and "[2,M] Sift yeast." The designer selects the option "Add flour" and decides to refine it. After one refinement the activities "Add oil" and "Add yeast" are selected, and in the next design phase after four refinements the activity "Pour water into glass" is selected. As a result, the process designer suggests an option list for the next activity that starts with the option: "[1] Heat up water." This first option is selected for refinement and as a result an option list is suggested, containing the option: "[1] Add water to flour." This option is selected as a valid option in the process model. Therefore, its distance from the original reference activity, "Pour water into glass" was shortened from 2 into $2 - 2 * 0.6 = 0.8$. In contrast to the original descriptor space (Fig. 4), the activity "Add water to flour" is now more closer to "Pour water into glass" than the activity "Heat up water." This automatically acquired knowledge seems reasonable, since cake preparation processes do not usually require warm water as in the case of bread preparation processes. After

12 additional design phases, the process design reaches the phase of preparing a jam cover for the cake. In this case the activity "Pour water into glass" was selected again and this time the activity "Add water to flour" was suggested *before* (and instead of) the activity "Heat up water," saving one refinement step.

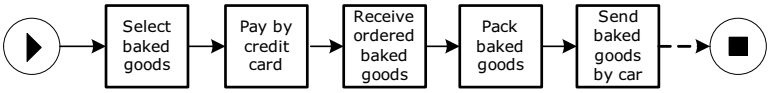

Fig. 8. The new designed process diagram for "Sell baked goods via the Internet"

The designer is now interested to design the new business process: "Sell baked goods via the Internet." The design process is conducted in a similar manner to the one presented above and results in the process diagram presented in Fig. 8. An interesting observation in this design process is the learning usage of the activity "Send baked goods by car." While the original business process repository contained the action "Send by car" applied only to the object "damaged flour," the terminating activity combines this action with the object: "baked goods." This was achieved by the following design phases: after accepting the activity: "Pack baked goods," the designer asks for next activity suggestions and receives an option list. Knowing that in order to fulfill the process goal, the last activity should involve a sending by car action, she selects the option: "Send damaged goods by car" and asks to refine it since it does not provide the exact required activity. Since the objects "Damaged goods" and "Baked goods" are siblings in the object hierarchy model, one of the refinement options is: "[2,M] Send baked goods by car," although it represents a major change to the underlying process repository. The designer approves the new process design as a complete design that fulfills the process goal, and the learning mechanism deduces that the action "Send by car" can be applied on "baked goods" in real-life scenarios and updates the descriptor space accordingly. This new learned knowledge assists in improving the design process of other new process models such as: "Sell baked goods via phone" and "Donate baked goods to poor families."

6.2 Experiments

We now present an empirical evaluation of the proposed method effectiveness. In order to evaluate the learning mechanism's contribution, we implement the evaluation method used in [11]. 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 "no-connection" distance (defined in Section 4) was set to 500; h_{next} was set to 0.2 and h_{refine} was set to 0.1.

Data. We chose a set of 12 real-life processes from the bakery process repository, comprising: (a) five processes from the core "Baking" category, with 45 activities altogether; (b) three processes from the "Sales" category, with 22 activities altogether; and (c) four processes from the "Maintenance" category, with 39 activities altogether. The "Baking" data set contains core and industry-specific activities, while the "Maintenance" data set represents a combination of industry-specific as well as more industry agnostic activities. The most generic activity collection is represented in the "Sales" domain, which shares many of its activities with the food manufacturing industry. Using the selected 12 processes we created a "process repository database."

Evaluation Methodology. To evaluate the suggested method we conducted 12 experiments, each repeated twice: first without applying the learning mechanism (a reference experiment) and second by applying the learning mechanism (a full method experiment). 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) run the design assistant mechanism in a stepwise manner. At each phase we try to identify an activity ("goal activity") that is compatible with the removed process, according to the following steps: (1) if the goal activity's linguistic components are represented in the Process Repository Database, run the "find next activity" algorithm (see Section 5.3). If the output list contains the goal activity - continue to reconstruct the next goal activity. Else, run the "activity refinement" algorithm (see Section 5.2). If the option list produced by the refinement step does not include the goal activity, choose the activity that shares the largest amount of common descriptor components with the goal activity as a basis for an additional refinement. If, after 10 successive refinements, the required activity is still not represented by one of the output options, it is inserted manually as the next process activity and the design process is continued by locating the next activity; (2) else (the goal activity's linguistic components are not represented in the Process Repository Database), the next goal activity is inputted manually by the experimenter. In full method experiments, at the end of each such phase the learning mechanism is applied, correcting the current descriptor space as an input to the next design phase (see Section 5.5).

Results and Analysis. Table 1 presents a summary of the experiment results. Each experiment of creating a new 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 goal process. This means that we aim at recreating the goal activities from a partial set of activity descriptors. On average, for 83.3% of the goal activities, all descriptor components were contained both in the goal process and in another process (see column #3). This was the case despite the relatively small experiment size (11 processes, whereas the entire bakery process repository includes around 70 processes), highlighting the amount of similarity one would expect when designing new processes based on an existing repository.

For the remaining 16.7%, at least one descriptor component was missing. In such a case, the activity was inserted manually during the design process. It is worth noting that for the 83.3% of activities that had the potential of reconstruction from the database, 100% were reconstructed successfully using our method (see Table 2).

Table 1. Experiment results

Column #	1	2	3	4	5	6	7
Column name	# of total processes in DB	# of total activities in DB	% of goal activities represented in the DB	% improvement in avg. # of steps per design phase	% improvement in avg. location of correct option in 'next activity'	% improvement in avg. location of correct option in 'refine activity'	% improvement in avg. location of the correct option per design phase
Avg.-all	12	106	83.3%	12.7%	32.9%	26.4%	28.5%
Avg.-Baking	5	45	85.2%	17.8%	38.6%	30.1%	31.0%
Avg.-Sales	3	22	80.8%	11.6%	34.2%	25.4%	27.7%
Avg.-Maintenance	4	39	82.7%	7.0%	24.8%	22.5%	25.9%

In addition, Table 1 shows that by applying the learning mechanism the following measures were improved. First, on average, the number of iterations required for reconstructing a goal activity (see column #4) was improved by 12.7%. The design of Sales processes required less steps than the design of Baking and Maintenance processes, and therefore was less improved (7% vs. 17.8 and 11.6% on average, respectively). It should be noted that the location of the goal activity was improved significantly in the ranked list of suggested activities (average improvement: 28.5%, see column #7). This location was even better improved at phases that did not involve refinement (average improvement: 32.9%, see column #5); and was a little lower in steps in which a refinement was required (26.4% on average, see column #6). This may be due to the fact that refinement steps include a much larger amount of alternatives, and the tunable parameter h_{refine} is lower than h_{next} (hence, learning is slower in refinement steps). Again it should be noted that results within the Baking category were better than results within the Sales and Maintenance categories - probably due to the larger amount of activities representing each of the Baking processes, which enables more learning opportunities. Another reason may be the similarity between Baking processes which enables applying the learning results from one process to others as well.

Table 2 analyzes the difference in the number of refinements that are needed to design the correct goal activity due to the usage of the learning mechanism.

Table 2. Distribution of successful predictions vs. the number of required refinements

# of refinements	0	1	2	3	4	5	6	7
% of difference in the # of successful predictions	11%	17%	10%	1%	-10%	-9%	-7%	-9%

For each number of refinements (0-7), we record the percentage of cases where this number of refinements was needed: (a) when the learning mechanism is not applied; and (b) when the learning mechanism is applied. Then, we calculate the difference between the results in both cases. We observe, for example, that by applying the learning mechanism, the ability of the system to reconstruct the goal activity after one refinement was improved by 17%. In total, it can be observed that the learning mechanism reduced the number of refinements required to reach the correct goal activity. These results clearly demonstrate that the learning mechanism improves the speed and efficiency of the design method. As hypothesized earlier - a larger database would probably yield even better results.

To summarize, we have shown the usefulness of using the learning mechanism in identifying activities for a new business process. We also showed the mechanism to be effective in the given experimental setup, both in terms of improvement in the number of design steps and in the number of refinements that are needed.

7 Conclusions

We proposed a learning mechanism to improve a machine-assisted design method. Such a mechanism saves design time and supports designers in creating new business process models. The proposed method provides 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 newly generated processes; and (2) an extended activity decomposition model to include an elaborated set of business data and logic (*e.g.*, roles and resources).

As a future work we intend to investigate further language semantics by using more advanced natural language processing techniques, as well as semantic distances between words. Finally, we intend to apply the learning technique we have developed to create new methods for workflow validation.

References

1. Becker, J., Delfmann, P., Herwig, S., Lis, L., Stein, A.: Towards Increased Comparability of Conceptual Models-Enforcing Naming Conventions through Domain Thesauri and Linguistic Grammars. In: ECIS (June 2009)

2. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
3. Cronan, T.P., Douglas, D.E., Alnuaimi, O., Schmidt, P.J.: Decision making in an integrated business process context: Learning using an erp simulation game. Decision Sciences Journal of Innovative Education 9(2), 227–234 (2011)
4. Ghattas, J., Soffer, P., Peleg, M.: A Formal Model for Process Context Learning. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 140–157. Springer, Heidelberg (2010)
5. Gschwind, T., Koehler, J., Wong, J.: Applying Patterns during Business Process Modeling. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 4–19. Springer, Heidelberg (2008)
6. Hornung, T., Koschmider, A., Lausen, G.: Recommendation Based Process Modeling Support: Method and User Experience. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 265–278. Springer, Heidelberg (2008)
7. Huang, Z., van der Aalst, W.M.P., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. Data & Knowledge Engineering 70(1), 127–145 (2011)
8. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
9. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
10. Leopold, H., Smirnov, S., Mendling, J.: On the refactoring of activity labels in business process models. Information Systems (2012)
11. Lincoln, M., Golani, M., Gal, A.: Machine-Assisted Design of Business Process Models Using Descriptor Space Analysis. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 128–144. Springer, Heidelberg (2010)
12. Lincoln, M., Karni, R., Wasser, A.: A Framework for Ontological Standardization of Business Process Content. In: International Conference on Enterprise Information Systems, pp. 257–263 (2007)
13. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
14. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
15. Reijers, H.A., Limam, S., Van Der Aalst, W.M.P.: Product-based workflow design. Journal of Management Information Systems 20(1), 229–262 (2003)
16. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting Flexible Processes through Recommendations Based on History. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
17. Smirnov, S., Weidlich, M., Mendling, J., Weske, M.: Action patterns in business process model repositories. Computers in Industry (2012)
18. Van der Aalst, W.M.P., Barthelmess, P., Eliis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. International Journal of Cooperative Information Systems 10(4), 443–482 (2001)
19. Wahler, K., Küster, J.M.: Predicting Coupling of Object-Centric Business Process Implementations. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 148–163. Springer, Heidelberg (2008)