A Knowledge-based Approach to the Configuration of Business Process Model Abstractions

Shamila Mafazi[1], Wolfgang Mayer[2], Georg Grossmann[2], and Markus Stumptner[2]

University of South Australia, Adelaide, SA, 5095, Australia
[1]shamila.mafazi@mymail.unisa.edu.au
[2] {firstname.lastname}@unisa.edu.au

Abstract. Methods for abstraction have been proposed to ease comprehension, monitoring, and validation of large processes and their running instances. To date, abstraction mechanisms have focused predominantly on structural aggregation, projection, and ad-hoc transformations.
We propose an approach for configuration of process abstractions tailored to a specific abstraction goal expressed as constraints on the abstraction relation and process transformation operators. Our framework goes beyond simple structural aggregation and leverages domain-specific properties, taxonomies, meronymy, and flow criteria. In this paper we outline the constraint-based framework and its underlying inference procedure. We show that our approach can handle most of the common process analysis use cases.

Keywords: business process abstraction, business process management, process configuration

1 Introduction

Models of business processes and operational procedures are increasingly being used in modern organizations, and the size and complexity of processes and their models can often be large. Development processes in large technology-focused organizations can easily span more than one thousand process steps [10]. As a result, process models have become difficult to understand and manage, as they may not be specified in full in order to enable flexible executions. However, such flexibility comes at a price: it is no longer easily possible to reason about executions based on a single process model. Although learning methods have been developed to reconstruct process models from execution logs [5], the resulting processes are often very specific and can be difficult to comprehend in full. Therefore, methods for business process abstraction are desired that enable process analysts to tailor large models to their specific analysis task at hand.

Methods for abstraction have been proposed to ease comprehension, monitoring, and validation of large processes and their running instances. To date, abstraction mechanisms have focused predominantly on structural aggregation and projection. Collapsing "similar" entities in a process model into one abstract element and projecting away irrelevant entities are among the most common forms of simplification employed for abstraction. Similarity and relevancy of process entities is often defined ad-hoc using process structure, clustering techniques, and user-specified selection criteria [4]. Clustering techniques, statistical methods, and ad-hoc criteria are commonly used to devise a concise summary representation that reflects certain aspects of the larger process.

Although structural aggregation can lead to considerable simplification of large process models, the resulting model may not show all required elements or aggregate elements together that would be better kept separate. However, these measures fail to take into consideration the purpose of the abstraction for the user.

We propose an approach to computing abstractions of business process models tailored to conducting selected common business process analysis tasks. We address this problem by imposing constraints on the abstraction relations that relate concrete and abstract process models such that the abstract process model induced by the abstraction relation is guaranteed to include the information needed to assess selected properties of the process. Rather than relying on cumbersome explicit specification of relevant process elements, we combine a questionnaire-driven approach to eliciting constraints for common analysis tasks with explicit specification of additional constraints a user may have.

As a result, significance and granularity of an abstract model can be explicitly controlled and adjusted to suit a given task. Furthermore, the granularity need not be uniform across the entire model; different abstraction operators can be applied to different regions of the process model.

Although techniques for parameterizing the granularity of the resulting abstractions have been introduced in order to compensate for current techniques' inability to devise representations that are fit for the user's objective [8], to the best of our knowledge, no explicit means to control abstractions is available to non-experts in formal process analysis.

Our method can be seen as configuration of process models, where configuration applies to the abstraction operators used in devising process rather than the process model itself. In contrast to classic configuration where one chooses between alternative instantiations of given variation points within a parametric process model, our approach takes a detailed process model without explicit variation points and derives simplified variations thereof. Hence, our configuration method controls the operators applied within the abstraction process rather than the underlying process model.

In this paper we make the following contributions:

- a knowledge-based framework for configuring purposeful abstractions;
- a framework for specifying constraints on the abstraction;
- a method to infer the process elements (nodes, data, labels) that need to be retained in a conforming abstraction;
- a method to compute abstractions conforming to the abstraction goal.

The subsequent sections are structured as follows. Our process model and abstraction framework are introduced in section 2, our constraint-based abstraction framework and configuration mechanism are described in sections 3 and 4, respectively. Abstraction operators are modeled in section 5 and our method of synthesizing conforming abstractions is summarized in section 6, followed by discussion of related work in section 7.

2 Process Model Abstractions

Different users of a process model are usually interested in observing a process model at different levels of details. This requires creation of different abstract process models from one model. However, not all abstract views of a process are equally desirable, as useful abstractions should be tailored to the user's needs. In this work, we pursue this aspect of process abstraction by constraining abstractions such that certain user-selected properties of the underlying concrete process are maintained in its abstract view.

We adapt the process model of Smirnov et al.[15] for our purposes and furnish the model with explicit representations of data- and domain-specific properties attached to tasks:

Definition 1 (Process Model). *A tuple (N, F, P, ϕ, D_P) is a process model, where N is a finite set of nodes partitioned into tasks N_t and gateways N_g, $F \subseteq N \times N$ is the flow relation such that (N, F) is a connected graph, P is a finite set of properties of tasks, D_P is a finite set of property values of tasks, and $\phi : N \times P \mapsto D_P$ is a function that maps each property of a node to its value. For brevity, we write $n.p$ for $\phi(n, p)$. Let M denote the set of all process models.*

The set of properties P comprises common domain-specific properties, predicate valuations, and information derived from executions of process instances. Common properties include roles, resources, timing information, and used and modified data flow information. Domain-specific predicates are boolean properties expressing facts such as "is on a critical path". Information derived from executions indicate aggregate information, for example execution frequencies or number of running instances of a task.

Given a concrete model m of a business process, an *abstract view* of m is a process model $\hat{m}$ that retains "significant" entities of m and omits insignificant ones. In our framework, *entities* comprise the nodes, flows, and properties associated with nodes in a given model. We write Ω_m to denote the set of entities in m where $\Omega_m \subseteq N \cup F \cup \{n.p | n \in N, p \in P\}$.

Which entities are considered significant is largely determined by the purpose of the abstract model and hence should be defined flexibly based on the goals of the analyst. We will therefore use an abstract predicate $sign \subseteq \Omega_m \cup \Omega_{\hat{m}}$ to capture the significant entities.

Whereas insignificant entities can be either *eliminated* from in the abstraction or *absorbed* into an abstract entity, the significant elements are to be retained. The correspondence between significant entities of m and their abstract counterpart in $\hat{m}$ is given by an *abstraction relation* $R \subseteq \Omega_m \times \Omega_{\hat{m}}$.

Definition 2 (Process model abstraction). *A business process model abstraction is a function $\alpha : M \mapsto M$ that transforms a model m into a model $\hat{m} = \alpha(m)$ with correspondence relation R_α such that*

- *$\forall \hat{\omega} \in \Omega_{\hat{m}}\ sign(\hat{\omega})$ is true,*
- *$\forall \hat{\omega} \in \Omega_{\hat{m}} \exists \omega \in \Omega_m\ (\omega, \hat{\omega}) \in R_\alpha$,*
- *$\forall \omega \in \Omega_m\ sign(\omega) \rightarrow \exists \hat{\omega} \in \Omega_{\hat{m}} : (\omega, \hat{\omega}) \in R_\alpha$, and*
- *α preserves local composition of m in $\hat{m}$.*

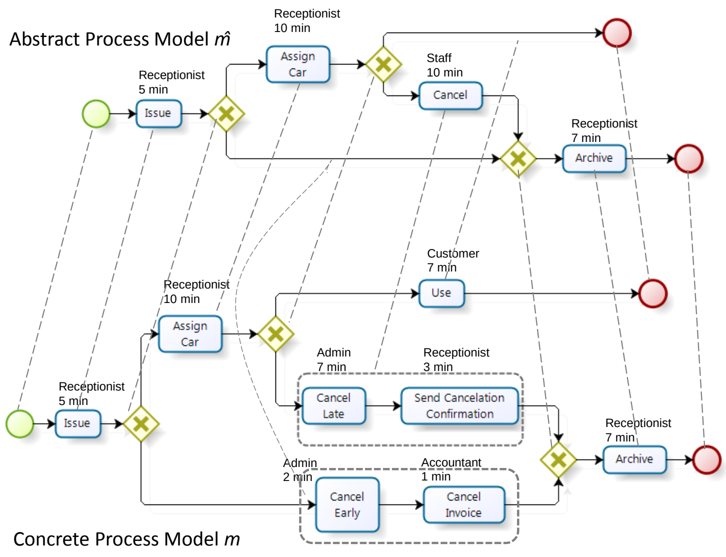

Fig. 1. Example Process Model (bottom), Abstract Model (top), and Correspondence Relation

The first three conditions ensure that all retained entities in the abstraction are significant, are justified by the existence of at least one entity in the concrete process, and that all significant concrete entities have a corresponding element in the abstract model. The fourth condition restricts correspondences to meaningful maps that preserve the local structural composition of m in $\hat{m}$. We require that each concrete entity maps to at most one abstract counterpart. Where each abstract property attaches to the abstraction of the concrete nodes belonging to the concrete properties. Also the abstract flow relation reflects the flow in the concrete process model:

- $\forall \omega \in \Omega_m \, \forall \hat{\omega}, \hat{\omega}' \in \Omega_{\hat{m}} \, (\omega, \hat{\omega}) \in R_\alpha \wedge (\omega, \hat{\omega}') \in R_\alpha \rightarrow \hat{\omega} = \hat{\omega}'$,
- $\forall n.p \in \Omega_m \forall \hat{n}.\hat{p} \in \Omega_{\hat{m}} \, (n.p, \hat{n}.\hat{p}) \in R_\alpha \rightarrow (n, \hat{n}) \in R_\alpha$,
- $(\hat{m}, \hat{n}) \in \hat{F} \rightarrow \exists m, n \in N \, (m, n) \in F^* \wedge (m, \hat{m}) \in R_\alpha \wedge (n, \hat{n}) \in R_\alpha$.

Consider the example process models in Figure 1, where the model in the lower half depicts the concrete process and the upper half shows the abstract model. The correspondence relation for tasks is indicated by dashed lines; the correspondences for flows are left implicit. Assuming that all elements performed by role *Receptionist* in m are significant, the abstraction satisfies the condition of Definition 2 as well as the three constraints stated above. For illustration, assume that tasks *Cancel Late* and *Send Cancellation Confirmation* each have a property *Duration*, then the constraints on R_α ensure that property *Duration* of abstract task *Cancel* is an abstraction of only the concrete tasks' property.

3 Abstraction Specification

According to Smirnov et al.[15] business process abstraction consists of three aspects: the *why*, the *when* and the *how* aspect. The *why* aspect captures the reasons for building an abstraction of a process model (fragment), the *when* aspect describes the conditions under which an element of a process model needs to be abstracted, and the *how* aspect relates to the concrete transformation mechanism to devise an abstraction. Whereas an extensive body of work covers the *how* aspect, comparatively little work is available to address the remaining aspects.

Our work aims to address the *why* and *when* aspects. We assume that a specification of the information, its granularity, and predicates whose truth values shall be preserved by the abstraction can be elicited, represented formally, and exploited to guide a search procedure to infer suitable abstractions. Let Γ be such a specification, formulated over the entities in a given process model m. Specifically, we are interested in abstract models $\hat{m} = \alpha(m)$ satisfying Γ.

By making the abstraction criterion explicit, the *why* aspect of process abstraction is captured, which can be translated into conditions for *when* it is admissible to abstract different entities. We define the significance predicate such that the entities are preserved which are required to ensure that criterion Γ is fulfilled on the abstract model. Building on prevalent structural rewriting mechanisms, we provide generic operators on properties and their values in order to automatically eliminate or aggregate entities, and furnish the abstract model with a suitable representation of aggregated information. The application of operators is restricted such that the resulting abstract model retains the significant entities and predicates.

An abstraction criterion may be composed of the following specification primitives:

- $sign(\omega)$ for $\omega \in \Omega_m$;
- $\omega = \omega'$ for $\omega, \omega' \in \Omega_m \cup \Omega_{\hat{m}}$;
- $(n, n') \in F^* \cup \hat{F}^*$, $n, n' \in N_t \cup \hat{N}_t$, $n, n' \in N_g \cup \hat{N}_g$;
- $n.p \oplus c$, where n, p, and c are a node, a property and a constant drawn from D_P, respectively, and $\oplus$ is a relational operator (e.g. $\prec$, $\preceq$, $=$, $\neq$, $\dots$);
- $(\omega, \hat{\omega}) \in R_\alpha$;
- negation, conjunction, disjunction, universal and existential quantification.

This language is expressive enough to capture many interesting properties, including domain-specific predicates and some aggregate instance information. The starred F^* and $\hat{F}^*$ denote the transitive closure of the flow relation.

For example, one could be interested only in the expensive tasks in the process model in Figure 1, where the value of *Fee* exceeds some threshold \$\$: $\Gamma = x.\mathit{Fee} \geq \$\$ \rightarrow \exists x\, (x, \hat{x}) \in R_\alpha \wedge x.p = \hat{x}.p \wedge (x.p, \hat{x}.p) \in R_\alpha$ for $p \in \{\mathit{Fee}, \mathit{Label}\}$. Capturing this explicitly in Γ, significance predicate and aggregation operators can be found. The example formula implies that all "expensive" tasks will retain their precise labels and fee information, whereas all other tasks and properties can potentially be abstracted away (subject to maintaining the generic abstraction constraints and well-formedness of the resulting abstract process).

While this example may seem trivial, our approach generalizes to more involved situations. For example, if execution times shall be retained, but labels of some tasks need

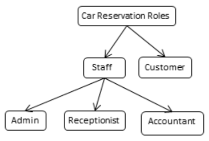

Fig. 2. Role Hierarchy

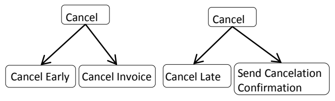

Fig. 3. Task Meronymy

not be, our approach allows us to absorb otherwise insignificant tasks into other tasks, but prevents us from eliminating the task entirely, which would result in its contribution to execution time being lost. Similarly, the model abstractions that may be applied in devising an abstraction would be restricted to aggregate the property of sequence of nodes using the *sum* function but not, for example, *max* function. Furthermore, data flow in the model may impose restrictions on significance of non-local process entities.

To facilitate the abstraction of data properties and other non-structural aspects of the business process, we assume that the value domain D_p of each property (including the label of nodes) $p \in P$ forms a (finite-height) (semi-)lattice with partial order $\prec_p$, where $x \prec_p y$ denotes that x is more precise (or has more information) than y. We use $\top_p$ to denote the least element of D_p, which provides no information. In this case, the property can be omitted.

For example, let us revisit the model in Figure 1. An example of the (semi-)lattice for the *Role* properties is shown in Figure 2. The lattice for roles indicates that roles *Receptionist* and *Admin* are specializations of role *Staff* and are therefore candidates for role abstraction.

For example, one could be interested in distinguishing *Customers* from *Staff* but not the precise staff roles. This could be captured in Γ as a constraint on the *Role* property of nodes. As a result, any value r of property *Role* that satisfies $r \prec \mathit{Staff}$ would be abstracted to value *Staff*.

We impose one more constraint on R_α: any admissible R_α must satisfy that no information can be gained in the abstract model. That is, $(\omega, \hat{\omega}) \in R_\alpha \rightarrow \omega \preceq \hat{\omega}$ must hold for all property entities $\omega, \hat{\omega}$.

4 Abstraction Configuration

Although the method of constraining valid abstractions is powerful, direct exposure of the formal framework to business analysts is rarely feasible in practice. Therefore, we employ knowledge-based configuration mechanisms to elicit appropriate partial abstraction specifications. We use a variant of the questionnaire method of process configuration [6], which interacts with the user in terms of simple domain-specific questions in order to construct the formal domain representation from the user's answers. Different from previous work, our configuration model does not rely on established variation points within the process model, but rather aims to construct a formula that constrains the admissible abstraction relations and operators that can be used to construct it. No explicit library of processes and variation points specific to the process under consideration is needed.

We envision our process abstraction configurator to provide a wizard-like interaction where process analysts may select the information and predicates they wish to retain in the abstraction, and define domain-specific value lattices, aggregation- and structural transformation operators. Underlying our configurator is a catalog of abstraction constraint templates, which can be selected and its parameters instantiated by the user.

Definition 3 (Configuration Model). *A configuration model is a triple $(\mathcal{C}, \mathcal{O}, G)$, where $\mathcal{C}$ is a catalog of abstraction aspects, $\mathcal{O}$ is a library of abstraction operators (defined in section 5), and G is a finite set of boolean propositions.*

The catalog contains configuration options and associated abstraction constraints, the library of abstraction operators defines the transformations that can potentially be applied to the process model, and the set of propositions allows one to restrict the set of applicable operators based on choices made for aspects in the catalog. We first describe the catalog and defer discussion of the operators until the next section.

Definition 4 (Abstraction Aspect Catalog). *An abstraction aspect catalog is a set of templates $(Q, X, C[X, G])$ where Q is a configuration option, X is a set of parameter variables, and $C[X, G]$ is a formula template parametric in X specifying the abstraction constraints associated with Q in terms of the process model, and abstraction operator constraint in terms of assignments to G. Each placeholder variable $x \in X$ can be assigned a predicate or domain value from the process model (subject to resulting in a well-formed formula $C[X, G]$). The configuration criterion Γ is simply the conjunction of all constraints $C_i[x_i, G_i]$ of selected Q_i with binding $X_i = x_i$.*

As an example, let the configuration option Q_1 be 'Get a process view from all the interactions between two specific roles'. By selecting this configuration option, the parameter variables are set as: $X = \{Role_1, Role_2\}$. The values for the roles are requested and assigned as $Role_1 = Admin$ and $Role_2 = Accountant$. The configuration imposes constraints on the abstraction relation: a task n must be retained in the abstraction if its $Role$ property valuation matches either $Role_1$ or $Role_2$, and there is a flow from n to another task n' that has property $Role$ set to the remaining given role. Formally, the abstraction criterion Γ can be expressed as

$$\begin{aligned}\forall n_1, n_2 \in N_t : (n_1, n_2) \in F^* \\ \wedge ((n_1.Role = Role_1 \wedge n_2.Role = Role_2) \\ \vee (n_1.Role = Role_1 \wedge n_2.Role = Role_2)) \\ \rightarrow (n_1, n_1) \in R_\alpha \wedge (n_2, n_2) \in R_\alpha.\end{aligned}$$

The catalog allows for convenient elicitation of user's requirements based on common abstraction goal patterns. Table 1 shows how 11 of the 14 common use cases for process abstraction presented by Smirnov et al[15] can be expressed in our framework. Most constraints restrict which tasks and properties may be abstracted, and whether insignificant tasks shall be eliminated or aggregated. In the first group of uses cases (1–4), a process view respecting one or more properties of a task, such as *resources and data objects*, is required. For this purpose the properties of all tasks are compared with the user specified property P. Tasks satisfying property P over property A are retained in the abstraction, whereas others are eliminated. In the second use case, tracing a task, the effect of a task in the process model needs to be assessed. For this purpose a process view containing the tasks which are reachable from the interesting task is produced. The constraint ensures that all tasks x' reachable from a given task x are retained in the abstraction. For instance-related use cases (5–7), we currently require a pre-processing stage, where the tasks in the process model are furnished with aggregate property information derived from the instances. For example, an property representing execution frequencies or cumulative case costs could be added. For use case 9, adapt process model for an external partner, the tasks which need to be presented to the external partner are selected. The selected tasks are considered as significant, hence they need to be retained while the rest of the tasks are aggregated. The first constraint ensures that selected tasks are retained in the abstraction, whereas the second constraint ensures that no insignificant tasks are eliminated from the model (although such tasks may be aggregated with other insignificant tasks). In use case 10, a process view respecting the data dependencies of the tasks is required. For this purpose those tasks which make use of the data objects of interest are considered as significant and must be retained in the abstraction while the rest of the tasks are considered as insignificant and can be eliminated from the abstract model. For use case 13, a process view respecting user specified property(s) is required. Different from use cases 1–4, in this process view the insignificant tasks (tasks without interesting property(s)) are aggregated and presented as a composite task in the process view. Hence the constraint prohibiting the elimination of insignificant tasks must be imposed in addition to the constraint capturing use cases 1–4.

Three use cases cannot directly be expressed in our framework. In use case 14, *Retrieve coarse grained activities*, a view over the coarse-grained tasks are required but not a view over the process model. This requires inferring the coarse-grained activities, i.e, abstraction hierarchies and meronymy, from the detailed process model. In contrast, our approach relies on given abstraction hierarchies and meronymy to compute abstractions. In use case 12, the user needs to control the abstraction level gradually while in our approach the process model is abstracted until all the user specified criteria are met. Finally, use case 8 requires to infer possible executions of the process model given a specification of a case instance. Extensions to our framework would be required in order to infer transitions that are potentially enabled or blocked based on guard conditions and values in the given case instance.

5 Abstraction Operators

Once abstraction constraints have been set, the concrete process model m can be transformed into a customized process view $\hat{m}$. In our framework, this amounts to constructing an abstraction function α and its induced R_α such that all abstraction constraints are satisfied when applying α on m. We employ generic search techniques to compose α from individual model transformation operators selected from a library of abstraction operators.

Preserving Relevant Tasks (Use cases 1–4)
Q_1 : Retain a task if property [A] satisfies [P] $C_1[A,P] = \forall x \in N_t \, [P](x.[A]) \rightarrow (x,x) \in R_\alpha$
Tracing a Task (Use case 11)
Q_2 : Retain a task if it is reachable from the node [x] $C_2[x] = \forall x' \in N \, (x,x') \in F^* \rightarrow (x',x') \in R_\alpha$
Preserving Relevant Process Instances (Use cases 5–7)
Q_1 and Q_2, based on pre-processed model
Adapt Process Model for an External Partner (Use case 9)
Q_3 : Retain selected tasks in set T $\forall x \in T \, (x,x) \in R_\alpha$ Q_3' : Aggregate insignificant tasks: $\forall x \in N \, sign(x)$
Trace Data Dependencies (Use case 10)
Q_4 : Retain a task if it uses data property [P] $C_4[P] = \forall x \in N_t \forall p \in [P] \, HasProperty(x,p) \rightarrow (x,x) \in R_\alpha \wedge (x.p, x.p) \in R_\alpha$
Get Process Quick View Respecting a Property (Use case 13)
Q_1 and Q_3'

Table 1. Representation of Use Cases in [15]

Abstraction operators are model transformations that rewrite the concrete model's entities into their abstract counterparts. Traditionally, work on business process abstraction focuses predominantly on structural transformations, where rules specify how fragments in a model shall be transformed into an abstract (smaller) fragments in the abstract model. Our work extends this approach to data properties.

Similar to constraints on the abstraction relation, which limit the information retained in the abstraction, the selection of abstraction operators is subject to constraints imposed by the configuration model that ensure abstract data values are given meaningful values consistent with the purpose of the abstraction.

Definition 5 (Abstraction Operator). *An abstraction operator is a tuple (R, S, V, W) where R, S are fragments of a process model ("patterns") with common variables V, and W is a boolean expression over propositions G (in the configuration model) and V, governing the application of the operator. If R matches with binding σ in a model m, and W is satisfiable, a model $m' = m[R\sigma \mapsto S\sigma]$ is obtained by replacing the matched part $R\sigma$ in m with the replacement fragment $S\sigma$. Substitute S may contain data transformation functions that compute the aggregate value for properties in the abstract model. Operators include sum, min, max, avg, for numeric properties, and least upper bound and greatest lower bound operators (if defined) on properties' value lattices.*

Our library of abstraction operators currently comprises:

- Projection operators that eliminate tasks/flows;
- Entity abstraction rules that transform labels and properties of individual tasks. These operators abstract property values according to the corresponding lattices of domain values;

- Structural rewrite rules that transform the process structure and re-arrange tasks and flows;
- Aggregation rules that aggregate values of properties of multiple tasks. Separate rules exist for properties of different type, and different aggregation functions may need to be used for sequence, choice, parallel, and loop constructs.

For space reasons we cannot present the entire collection in detail. Figure 4 contains examples of property-related aggregation for properties of different types (numeric, set-valued, boolean). The bottom part shows the concrete fragments and the top part the abstract counterparts; X and Y represent variables to be matched and a,b,c represent placeholders for numeric, set-valued, and boolean properties, respectively.

Figure 4a indicates 2 tasks in a block. To aggregate the numeric properties of the two tasks, the operators such as *Max, Min, Avg, Sum* can be employed. Selecting an operator is completely case based. For example, assume a user is interested in tasks with high hand-off times. In this case, the operator *Max* needs to be selected to assign the maximum hand-off time to the composite task *XY*. Likewise, for the set-valued properties, an operator such as *union, aggregate meronymy, abstract label*, based on the configuration option in hand, can be selected. The operators for boolean properties of the tasks, include, *Or, And, Xor*. As an example, assume a user is interested in observing the tasks which are in a critical path, the operator *Or* can be employed which indicates the composite task is whether on a critical path or not. Figure 4b shows an abstraction operator for two tasks in a loop. For the numeric properties of these tasks, based on how many times the loop is executed, the result from the abstracting operator needs to be multiplied or widened to infinity, if an upper bound is not known. Figure 4c shows an abstraction operator for sequential tasks. In this case, where numeric properties typically are aggregated, set-valued properties are merged, and boolean properties are either merged or combined using logic operators to infer the property value associated with the abstract task.

Table 2 gives a list of operators currently defined in our library. The table on the right-hand side of the figure shows examples for formalization of three operators in our framework. Our formalization relies on a set G of propositions defined in the configuration model that is used to govern the application of certain abstraction operators. The elements of this set are determined by the selected configuration options and domain model and consists of propositions of the form $Enable(o, op, p)$, where o is the name of an abstraction operator, op is an aggregation operation, and p is a property. Together with a hierarchy of properties (with specialization ordering $\sqsubseteq$), the propositions are used to control which operators can be applied to certain operations. For example, abstraction operator *SumNumPropSeq* is only applicable if none of the configuration options prohibits its application. Whereas most operators are generic and can be applied to process models from any domain, domain-specific operators can be introduced to account for specific abstractions, such as the meronymy approach presented in [14].

6 Abstraction Computation

Conceptually, our abstraction method proceeds as follows. Starting with a given concrete process model m and configuration constraints Γ, we employ a search procedure

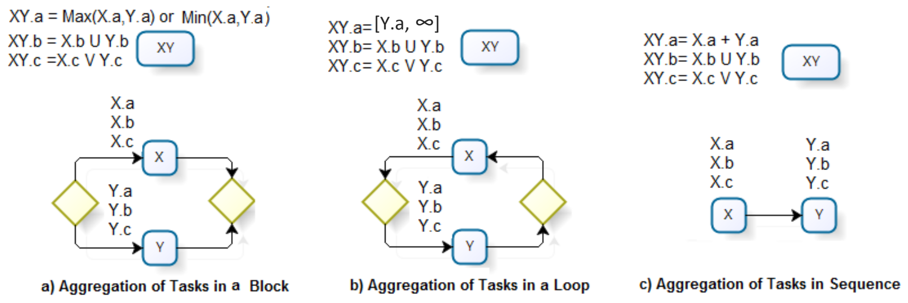

Fig. 4. Structural and Property Aggregation Operators

Operator	**Type**
Remove Task/Flow	Projection
Remove Property	Projection
Abstract Label	Entity
Abstract Property Value	Entity
Aggregate Sequence	Structural
Aggregate Concurrent	Structural
Aggregate Choice	Structural
Aggregate Loop	Structural
Aggregate Meronymy	Structural
Simplify Gateway	Structural
Shift Gateway	Structural
Aggregate Value (Seq)	Aggregation
Aggregate Value (Concurrent)	Aggregation
Aggregate Value (Choice)	Aggregation
Aggregate Value (Loop)	Aggregation

Logic Representation of the Operators

RemoveTask(x):

$\forall x \in N_t : \neg sign(x) \rightarrow$
$\nexists \hat{x} \in \hat{N}_t : (x, \hat{x}) \in R_\alpha$

AggregateTaskSeq(x,y):

$x, y \in N_t \wedge (x, y) \in F \rightarrow$
$\exists \widehat{xy} \in \hat{N}_t : (y, \widehat{xy}) \in R_\alpha$
$\wedge (y, \widehat{xy}) \in R_\alpha$

SumNumPropSeq(x,y,p):

$x, y \in N_t \wedge (x, y) \in F \wedge$
$(x, \widehat{xy}) \in R_\alpha \wedge (y, \widehat{xy}) \in R_\alpha$
$\wedge p \sqsubseteq Numeric$
$\wedge \neg Enable(SumNumPropSeq, +, p) \notin G$
$\rightarrow \widehat{xy}.p = x.p + y.p$

Table 2. Abstraction Operators

to incrementally build an abstraction. An applicable abstraction operator r is selected and applied to m, yielding a transformed model m'. If structural aggregation was performed, additional rules to determine the property values of new task(s) are applied. Concurrently, the abstraction function and its correspondence relation are extended to account for the effects of r. This process repeats until an abstraction satisfying all constraints in Γ has been created and no further rule applications are possible. As a result, we obtain an abstraction function that transforms the given model m in a maximally abstract process model reflecting the relevant tasks and properties. If the intermediate results are recorded, this yields a hierarchy of abstractions of varying granularity. Although not all models in this hierarchy necessarily satisfy all abstraction constraints, navigating the abstraction hierarchy could be useful to “drill-down” in specific areas if needed (comparable to the approach in [12]). Incremental specification and adjustment of abstraction constraints based on initial abstract views remains a direction for future research.

If multiple operators are applicable, this approach may result in multiple possible abstractions. To steer our algorithm towards desirable abstractions, we employ a simple optimization method that aims to minimize both constraint violations and model complexity. When selecting an abstraction operator, we choose the operator that minimizes the sum $viol(\Gamma, \alpha, m) + size(\alpha(m))$, where $viol(\Gamma, \alpha, m)$ denotes the number of constraints in Γ that are violated by the current abstraction α when applied to m, and $size(\alpha(m))$ measures the number of elements ($|N| + |F|$) in the abstract model $\alpha(m)$. In addition, we maintain a worklist of the current best k abstraction functions. Currently, k is a user-set parameter.

For example, let us revisit the process in Figure 1. Assume that only tasks that are involving role "Receptionist" with *Duration*$>$ 3min are required to be shown in the abstraction.

Based on the given the abstraction constraints, the abstraction criterion Γ can be expressed as:
$\forall n \in N_t \wedge n.Role = Receptionist \wedge n.Duration > 3 \rightarrow (n, n) \in R_\alpha$

Considering the criteria, tasks *Use*, *Cancel Early* and *Cancel Invoice* are insignificant, as for example $Use.Role \neq Receptionist$. Aggregating the two tasks *Cancel Early* and *Cancel Invoice* does not result in a significant task either. Hence among others, operator "Remove Task" can be applied to these tasks to eliminate them from the process model. Tasks *Cancel Late* and *Send Cancellation Confirmation* are also insignificant but unlike *Cancel Early* and *Cancel Invoice*, aggregating these two tasks results in a significant task. Hence, operator *Abstract Property Value* can be applied to their role properties to lift the property value to the abstract value *Staff*. Now, operator *Aggregate Meronymy* can be applied (based on the meronymy in Figure 3), combining *Cancel Late* and *Send Cancellation Confirmation* into *Cancel*. The operator *SumNumPropSeq* is applied on the duration properties of the two tasks to add up these properties. Since the abstract task was formed by sequential composition, *Aggregate Value (Seq)* must be applied twice to infer the value for properties *Role* and *Duration* of the abstract task.

At this point, no operators are applicable that satisfy the abstraction constraints. Further simplification of properties and removal of tasks or flows would yield either an ill-formed process model or violate an abstraction constraint.

7 Related Work

The research presented in this paper complements the areas of business process model abstraction and process model configuration. Due to emerging various needs, several approaches have been proposed by which the size of a process model can be reduced. However, no single approach provides the same level of configuration ability as ours.

Many approaches for simplifying a given process model based on rewrite rules have been developed [15]. Rewriting approaches based on process fragments, process regions and patterns aim to simplify the structure of large processes by hierarchical aggregation. Various process visualization techniques rely on users selecting interesting tasks and eliminating the remaining tasks from the process model [2]. Pankratius et al. [11] proposed Petri Net based reduction patterns, including place and transition elimination and place and transition join, for abstraction. Liu et al. [9] cluster tasks in a process model, preserving ordering constraints, roles, execution frequencies, and process view for external partners. Since their main abstraction operation is aggregation, the clusters are aggregated into composite nodes. In both of these approaches [11, 9], the authors address the *how* component of the business process abstraction. Since the papers ignore the execution semantic of the process model and treat only tasks, but not the reachability criterion, as the abstraction objects, the process views related to the process instances (use cases [5-7]) cannot be captured by their techniques. Additionally, compared to our approach, their approach is not user interactive. Cardoso et al.[3] proposed reduction rules to synthesize process views respecting ordering constraints and roles. The paper concentrates on the *how* component of the process abstraction while only non-functional property values have been considered. Furthermore, their reduction technique is pattern based. Once a region matches one of their predefined patterns, the region is aggregated into a composite node. Hence, it is not always possible to aggregate an insignificant task, as forming a region for the task that matches the patterns, can be impossible.

Bobrik et al.[1] aggregate information derived from running instances into a summary process model, including completion status, data properties, and timing information. In this paper only the *how* component is discussed. Also the paper does not discuss the property aggregation operations for different types of properties. Polyvyanyy et al. [13] defined abstraction criteria based on slider approach which separate significant from insignificant tasks, which are subsequently aggregated based on structural process patterns. Although the abstraction criteria can be extended to cover more abstraction scenarios, they are limited to those properties which have a quantitative measurement such as cost and execution duration.

Fahland et al.[5] proposed a simplification strategy for Petri nets that is based on unfolding and subsequent transformation and folding regardless of abstraction purposes. Overall most of the process model abstraction approaches focus on only the *how* component, reduce a process model based on predefined patterns, consider only a limited number of properties, and are not user interactive. In contrast, we take other process abstraction components into account, we do not restrict the preservation or aggregation of a task based on its region and the corresponding patterns, we provide an aggregation solution for properties with different types. Finally using a questionnaire, different needs of a user from abstracting a process model are taken into account.

In process model configuration literature, La Rosa et al. [8] introduce a questionnaire approach for system configuration. The questionnaire elicits facts about the desired process variant. Facts are associated with actions that adapt a given generic reference process to suit the users requirements. Gottschalk et al. [7] summarizes similar approaches for EPCs and YAWL, where tasks in the process are either blocked or hidden. In contrast, our approach does not rely on a reference process with variation points. Instead, we constrain the resulting abstraction relation and employ search techniques to compute suitable abstractions for tasks and data entities in the process model.

8 Conclusion

We presented a configuration method for generating tailored business process abstractions that satisfy user-selected abstraction criteria. Our method is based on imposing constraints on the abstraction relation, which is computed using a generic search procedure using a library of generic and domain-specific abstraction operators. Elicitation of relevant abstraction constraints is simplified by a questionnaire-based approach that hides much of the formal underpinnings of our method. Our abstraction approach goes beyond simple structural transformation and also considers data properties and flow aspects within the process model in the abstraction.

In this paper we focused on conceptual elaboration of our method. Immediate future work will focus on empirical evaluation of the approach on large business processes, and on incorporating preference orderings into our search and operator selection algorithms. Other avenues for research are incremental elicitation of abstraction constraints in the context of incremental process exploration and integration of process instance-based properties and further reachability-based criteria.

9 Acknowledgement

We would like to acknowledge that this research was supported by the Australian Research Council (ARC) under grant DP0988961.

References

1. Bobrik, R., Reichert, M., Bauer, T.: Parameterizable views for process visualization. Tech. rep., Centre for Telematics and Information Technology, University of Twente (2007)
2. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Proc. BPM. pp. 88–95. Springer (2007)
3. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1(3), 281 – 308 (2004)
4. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proc. APCCM. pp. 71–80. Australian Computer Society (2007)
5. Fahland, D., van der Aalst, W.: Simplifying mined process models: An approach based on unfoldings. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management, LNCS, vol. 6896, pp. 362–378. Springer (2011)
6. Gottschalk, F., La Rosa, M.: Process configuration in yawl. In: Hofstede, A.H.M., Aalst, W.M.P., Adams, M., Russell, N. (eds.) Modern Business Process Automation, pp. 313–382. Springer (2010)
7. Gottschalk, F., Wagemakers, T., Jansen-Vullers, M., van der Aalst, W., La Rosa, M.: Configurable process models: Experiences from a municipality case study. In: Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 5565, pp. 486–500. Springer Berlin / Heidelberg (2009)
8. La Rosa, M., van der Aalst, W., Dumas, M., ter Hofstede, A.: Questionnaire-based variability modeling for system configuration. Software and Systems Modeling 8, 251–274 (2009)
9. Liu, D.R., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems 28, 505 – 532 (2003)

10. Mayer, W., Killisperger, P., Stumptner, M., Grossmann, G.: A declarative framework for work process configuration. AI EDAM 25(2), 145–165 (2011)
11. Pankratius, V., Stucky, W.: A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In: APCCM. pp. 79–88. APCCM '05, Australian Computer Society (2005)
12. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: EDOC. pp. 325–331 (2008)
13. Polyvyanyy, A., Smirnov, S., Weske, M.: Business process model abstraction. In: Handbook on Business Process Management 1, pp. 149–166. Springer (2010)
14. Smirnov, S., Dijkman, R., Mendling, J., Weske, M.: Meronymy-based aggregation of activities in business process models. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) Conceptual Modeling ER 2010, LNCS, vol. 6412, pp. 1–14. Springer (2010)
15. Smirnov, S., Reijers, H., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases 30, 63–99 (2012)