Modeling and Query Patterns for Process Retrieval in OWL

Abstract. Process modeling is a core task in software engineering in general and in web service modeling in particular. The explicit management of process models for purposes such as process selection and/or process reuse requires flexible and intelligent retrieval of process structures based on process entities and relationships, i.e. process activities, hierarchical relationship between activities and their parts, temporal relationships between activities, conditions on process flows as well as the modeling of domain knowledge. In this paper, we analyze requirements for modeling and querying of process models and present a pattern-oriented approach exploiting OWL-DL representation and reasoning capabilities for expressive process modeling and retrieval.

1 Introduction

Process models are used in various applications like business process modeling, modeling of system processes and also the combination and interaction of web services are described as processes. In order to use existing processes it is necessary to support basic process model management tasks like the retrieval of processes [16].

There are various approaches for modeling processes and corresponding retrieval methods like a keyword search or retrieval methods based on data in- and output or process properties which are mainly process annotations. Retrieval with respect to the activities of a process and especially to the execution order and dependencies of them requires the consideration of the control flow of a process (cf. Section 2.2,[12]).

The problem of process retrieval based on control flow information depends on the internal process structure. The structure includes sequences, choices, parallel activities as well as activity specialization. Therefore, search capabilities for processes must include reasoning facilities that are able to match the heterogeneous requirements that exist at different levels of granularities of process descriptions between a query and a process. Moreover, this matching capability needs to distinguish modalities of matches, i.e. whether a specific condition on a process (part) must be always fulfilled or only for some of its instances. In order to tackle the described challenges, we consider two main tasks:

- A formal description of the control flow of a process which explicitly models the various hierarchical and ordering relationships between activities.
- A characterization of query and reasoning tasks in order to retrieve processes with specified control flow characteristics and modalities.

OWL-DL allows for the description of terminology of processes and activities. DL reasoning enables matching of queries and processes (cf. Section 4). The available reasoning procedures account for the aggregation of activities and for different modalities.

OWL-DL and DL have been used for process modeling and retrieval before (cf. [4, 11, 16, 17]). However, in these process models, the control flow is either not represented at all or represented by syntactic means that do not allow for reasoning as needed or the process models are too weak to express realistic control flows. A comprehensive survey is given in Section 6.

To remedy this situation, this paper provides a threefold contribution. First, we analyze modeling and querying requirements for process retrieval in general (Section 2). Second, we provide patterns for modeling process control flow in OWL-DL (Section 3). Third, we demonstrate how to exploit these semantic-based models for expressive retrieval of (parts of) process models by reasoning in OWL-DL (Section 4). The retrieval capabilities of our model are evaluated in Section 5.

2 Process Retrieval Problems

This section outlines requirements for process modeling and retrieval motivated by a retrieval scenario.

2.1 An Example Process Model as UML Activity Diagram

We use the widespread and standardized UML-Activity Diagrams for graphical process modeling. Figure 1 depicts three different process models describing *SalesOrder* processes with different characteristics. The elements of UML Activity Diagrams are described in Table 2. Activities are the tasks which are executed by the process. An edge is the connection from an activity to the follower activity.

The activities in the diagrams from Figure 1 are decomposed into more fine-grained descriptions which are depicted in Figure 2. The *Reorder* process contains a decision in combination with a condition. If the article is a standard article an internal reorder activity is executed otherwise an external reorder activity.

2.2 Requesting Process Descriptions

We consider a situation where a customer wants to use an online shop to buy a product. The selling procedure of the online shop is specified by a salesorder process description. Depending on the individual preferences of the customer, he may want to select a shopping service based on properties of the process control flow which is realized by the shop. For instance, he may want to search for a shop (Q) and expects a concrete response (R) fulfilling his request:

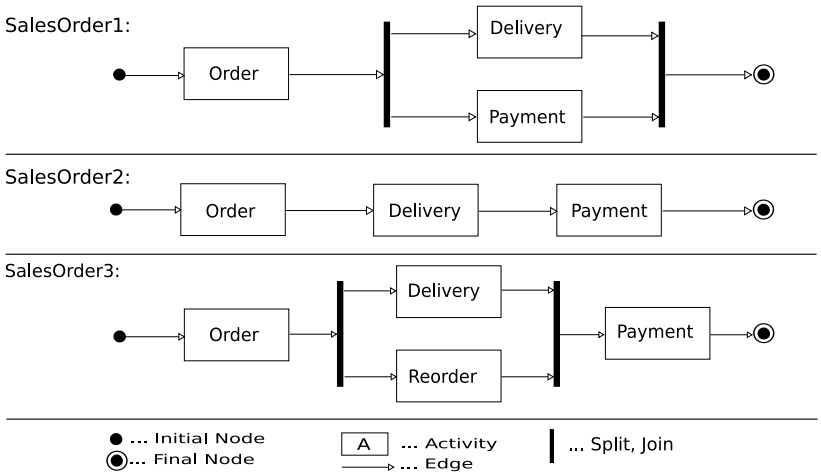

Fig. 1. UML-Activity Diagrams for Sales Order Processes.

Q1: Which shop allows me to pay after delivery?
R1: Process that executes *Delivery* before *Payment*.
Q2: Which shop allows me to pay by cash?
R2: SalesOrder that offers cash payment as payment method.
Q3: Which shop accepts only cash payment?
R3: SalesOrder process that only offers cash payment.

In fact, quite often an online shop is requested that fulfills multiple constraints simultaneously. Besides the customer, the service provider may have further queries in order to find an appropriate process.

Q4: Which process executes at least all activities of *SalesOrder*2?
R4: *SalesOrder*3 process contains all activities of *SalesOrder*2 and an additional activity.
Q5: Which process runs are instances of *SalesOrder*2 and *SalesOrder*3?
R5: Process runs which are instances of both processes simultaneously.

2.3 Requirements for Process Modeling

In order to facilitate process retrieval we derived the requirements for process modeling from the demonstrated queries. Table 1 describes the derived requirements with respect to the questions. (1) A query must consider the execution order, e.g. that one activity happens (possibly indirectly) before another activity. (2) Process specializations and (3) activity decomposition involve the terminology, e.g. that one must consider the different specializations of payment types. (4) Queries have to cover modality, e.g. that a property like the type of payment is unavoidable in all possible process enactments. (5) Instance queries are relevant for all modeling dimensions. These requirements may be combined, e.g.

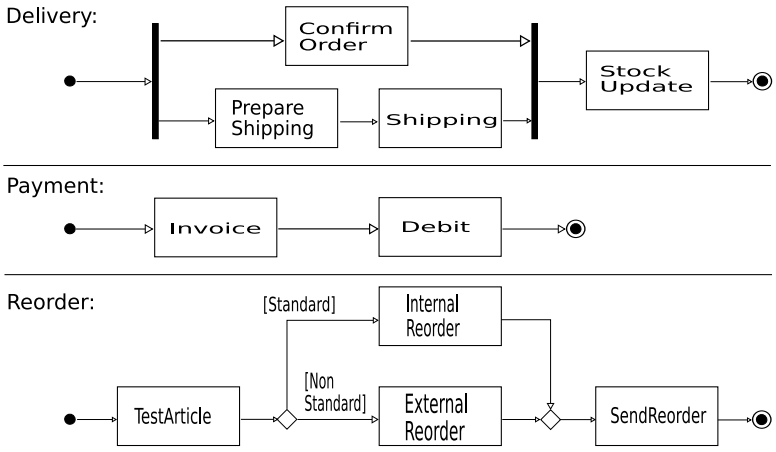

Fig. 2. UML-Activity Diagrams for the Subprocesses.

queries concerning the execution order have also to cover terminological information. Obviously, these queries are general and imprecise process descriptions, e.g. only a description of relevant parts of the process. Hence the process retrieval has to account for incomplete process descriptions.

Requirements	Questions
Order of Activities and Subactivities	Q1
Specialization, Inheritance	Q4
Activity Decomposition	Q2, Q3
Modality	Q2, Q3
Instance Queries	Q5

Table 1. Requirements derived from the queries.

3 Process Modeling in OWL

We have investigated two important design decisions for modeling processes in OWL-DL. The first method models occurrences of activities and their relative positioning in the control flow. The advantage is that each step can be modeled separately and - as we see later - some advantages with regard to refinements are implied. The second method models a process as one complex DL expression capturing all the steps. In this second model it is not fully possible to capture all temporal and refinement relationships simultaneously. However, the advantage of the second approach is that there are more possibilities for retrieval queries than in the first approach. In this paper we only focus on the second approach.

3.1 Design Principles and Transformation

A process model (sometimes called a process template) describes the set of all process runs it allows, i.e. a concept in OWL-DL. Process runs are instances in OWL-DL. A process run is a composition of activity instances. We translate the language primitives of the UML-Activity Diagram into an OWL-DL[1] representation. Table 2 lists our translation rules. A, B, C and D are activities, P is a process name.

Construct	UML Notation	DL Notation
1. Start	●	$Start_i$
2. End	◉	End_i
3. Activity	A	A
4. Edge	⟶	TO_i
5. Process P (Axiom)	● A ◉	$P \equiv Start_i \sqcap \exists_{=1} TO_i.$ $(A \sqcap \exists_{=1} TO_i.End_i)$
6. Flow	A B	$A \sqcap \exists_{=1} TO_i.B$
7. Decision	A, B, C, D	$A \sqcap \exists_{=1} TO_i. \quad ((B \sqcup C)$ $\sqcap \exists_{=1} TO_i.D$
8. Condition	A [Cond] B	$A \sqcap \exists_{=1} TO_i. \quad ((B \sqcap \kappa_{Cond}) \sqcup$ $(Stalled \sqcap \neg\kappa_{Cond}))$
9. Fork and Join	A, B, C, D	$A \sqcap \exists\, TO_i.(B \sqcap \exists_{=1} TO_i.D)$ $\sqcap \exists\, TO_i.(C \sqcap \exists_{=1} TO_i.D)$ $\sqcap = 2\, TO_i$
10. Loop	A, B	$Loop_j \sqcap \exists_{=1} TO_i.B,$ $Loop_j \equiv A \sqcap \exists_{=1} TO_j.$ $(Loop_j \sqcup End_j)$

Table 2. Transformation to Description Logic.

The relation between an activity and the follower activity is represented by roles in DL, i.e. TO [2]. In order to allow for process composition and refinement in combination with cardinality restrictions the roles for each process (TO_i) are distinguished from each other. All roles TO_i are defined as subroles of TO in order to simplify query formulations. Therefore we may use TO for the retrieval of all processes. TOT is a transitive superrole of TO (cf. Table 3) which is similar to the relation pattern in DOLCE plan extension [9]. The combination of both roles enables process retrieval with respect to direct and indirect activity order.

A process is either atomic or composed. An atomic process is a non-decomposable activity. A composed process is built using axioms (No.5). An axiom defines a process starting with $Start_i$ followed by a sequence of composed activities. The last activity is the concept End_i. The follower relation TO_i refers

[1] We use the DL language $\mathcal{SHOIN}(D)$ which corresponds to OWL-DL.

[2] For sake of a short name, the role name TO is used instead of a more meaningful name like $FollowedBy$.

to an edge in the process diagram. A flow (No.6) is translated to DL with the expression $A \sqcap \exists\, TO_i.B$. This means that the process flow defines an execution of activity A which is directly followed by the activity B. The disjointness of A and B in a flow is here not required which accounts for a larger result set in the retrieval (cf. 5).

Decisions (No.7) are modeled in DL as a concept union $\exists\, TO_i.(B \sqcup C)$ representing a non-deterministic choice between two possible following activities. Both activities reach an activity D. The cardinality restrictions guarantee that there is only one follower activity. A condition can be assigned to a flow (No.8). Adding disjoint conditions to each flow of a decision leads to deterministic decisions. These deterministic decisions are used to model exclusive decisions (exclusive or), i.e. exactly one path after the decision is executed.

Disjointness of the activities B and C is also not required. We describe conditions as set of states that may be reached when the condition is fulfilled. The notation of No.8 describes that directly after the activity A there is an activity B and the condition $Cond$ holds, i.e. the activity B is only executed if the condition is satisfied. We add an additional activity *Stalled* if the condition is not satisfied. The activity *Stalled* means that the control flow of the process is stopped due to a violation of the condition. All conditions are subclasses of a general concept *Condition* (Table 3).

Statement	DL Axiom
Stalled has no follower	$Stalled \sqcap \exists\, TO.Activity \sqsubseteq \bot$
All activities are subclasses of *Activity*	$A, B, C, D \sqsubseteq Activity$
Conditions are subclasses of the concept *Condition*	$\kappa_j \sqsubseteq Condition$
Domain and Range of TO is *Activity*	$\exists\, TO.\top \sqsubseteq Activity$ and $\top \sqsubseteq \forall\, TO.Activity$
TO is superrole of all TO_i	$TO_i \sqsubseteq TO$
TOT is the transitive superrole of TO	$TO \sqsubseteq TOT$, and $TO^+ \sqsubseteq TOT$
$Start_i$ is the first activity End_i is the last activity	$Start_i \sqcap \exists\, TO_i^-.Activity \sqsubseteq \bot$ $End_i \sqcap \exists\, TO_i.Activity \sqsubseteq \bot$

Table 3. Process Model Axiomatization.

A parallel execution (No.9) starts with a fork and ends with a join. The fork is described by the explicit statement that there exist multiple follower sequences, which is described by an intersection of the sequences. The join of sequences like for decisions and parallel executions is represented by activity (D) in the complex DL expression. Loops (No.10) are described by a subprocess $Loop_j$ which contains the activities of the loop (A) and the follower activity is either $Loop_j$ or End_j. This construct is a special decision with a self reference. In case of a nested loop structure, i.e. there is a loop within another loop, the transformation is straightforward. The inner loop is treated like a single activity of the outer loop, e.g. activity A in No.10 could be an inner loop. The inner loop is specified by a separate corresponding definition.

For each process definition we require that $Start_i$ has no predecessor, End_i has no follower which is described in the second part of Table 3. In order to model process refinements these conditions are only valid for this (sub-) process. Therefore the start and end activities are distinguished from each other, using levels (i) for each process. Further axioms like domain and range restrictions, subclass relations between activities and conditions and the role hierarchy for the follower relations (TO_i) are described in the first part of Table 3.

In our OWL-DL model there are limitations in the accuracy. Some model constructs are non-deterministic, e.g. we use a disjunction in DL to model a non-deterministic control flow description and we do not require a general disjointness of the activities of a process. However, this supports query expressions using general and incomplete process descriptions with a maximal possible result set.

3.2 Process Transformation to OWL

Based on the described transformation pattern from UML process models to OWL, we describe the *SalesOrder* processes from Section 2 in OWL-DL. Figure 3 depicts the axioms that describe the *SalesOrder* processes. For being more readable, we omit here the cardinality restrictions in all process definitions and the *Stalled* activity for violated conditions.

$$
\begin{aligned}
SalesOrder1 \equiv\ & Start_1 \sqcap \exists\, TO_1.(Order \sqcap \exists\, TO_1.(Delivery \sqcap \exists\, TO_1.End_1) \\
& \sqcap \exists\, TO_1.(Payment \sqcap \exists\, TO_1.End_1)) \\
SalesOrder2 \equiv\ & Start_2 \sqcap \exists\, TO_2.(Order \sqcap \exists\, TO_2. \\
& (Delivery \sqcap \exists\, TO_2.(Payment \sqcap \exists\, TO_2.End_2))) \\
SalesOrder3 \equiv\ & Start_3 \sqcap \exists\, TO_3. \\
& (Order \sqcap \exists\, TO_3.\ (Delivery \sqcap \exists\, TO_3.(Payment \sqcap \exists\, TO_3.End_3)) \\
& \sqcap \exists\, TO_3.(Reorder \sqcap \exists\, TO_3.(Payment \sqcap \exists\, TO_3.End_3))) \\
Delivery \equiv\ & Start_4 \sqcap \exists\, TO_4.(ConfirmOrder \sqcap \exists\, TO_4.\ (StockUpdate \sqcap \exists\, TO_4.End_4)) \\
& \sqcap \exists\, TO_4(PrepareShipping \sqcap \exists\, TO_4.(Shipping \sqcap \\
& \exists\, TO_4.(StockUpdate \sqcap \exists\, TO_4.End_4))) \\
Payment \equiv\ & Start_5 \sqcap \exists\, TO_5.(\ Invoice \sqcap \exists\, TO_5.(Debit \sqcap \exists\, TO_5.End_5)) \\
Reorder \equiv\ & Start_6 \sqcap \exists\, TO_6.(TestArticle \sqcap \exists\, TO_6.\ ((InternalReorder \sqcap Standard \sqcup \\
& ExternalReorder \sqcap \neg Standard) \sqcap \exists\, TO_6. \\
& (SendReorder \sqcap \exists\, TO_6.End_6)))
\end{aligned}
$$

Fig. 3. The Sales Order Processes in DL.

$$
\begin{aligned}
CreditCardPayment &\sqsubseteq Payment \\
CashPayment &\sqsubseteq Payment \\
CashPayment \sqcap CreditCardPayment &\sqsubseteq \bot
\end{aligned}
$$

Fig. 4. Domain Knowledge Axioms.

Domain knowledge (Figure 4) contains terminological information about activities and subactivities of the processes, e.g. a credit card payment is a subclass of payment. Figure 5 contains further process definitions also without cardinality restrictions. *CreditCardPayment* and *CashPayment* are specializations of the *Payment* activity. *SalesOrder*2_*Credit* and *SalesOrder*2_*Cash*

$$
\begin{aligned}
SalesOrder2\_Credit \;\equiv\; & Start_2 \sqcap \exists\, TO_2.(Order \sqcap \exists\, TO_2. \\
& (Delivery \sqcap \exists\, TO_2.(CreditCardPayment \sqcap \exists\, TO_2.End_2))) \\
SalesOrder2\_Cash \;\equiv\; & Start_2 \sqcap \exists\, TO_2.(Order \sqcap \exists\, TO_2. \\
& (Delivery \sqcap \exists\, TO_2.(CashPayment \sqcap \exists\, TO_2.End_2)))
\end{aligned}
$$

Fig. 5. Additional SalesOrder-Processes in DL.

refine the $SalesOrder2$ process using the subconcepts $CreditCardPayment$ and $CashPayment$ instead of $Payment$. In the remainder of this paper, we refer to the set of axioms from Table 3, Figures 3, 4 and 5 as the knowledge base KB.

3.3 Relations between Processes

Specialization and refinements are orthogonal relationships between processes. A specialization (or sometimes called an extension) is a relationships between processes in which the specialization of a process consists either of additional activities or some of the activities are specialized. An activity specialization or a subactivity is a subclass of the more general activity. Specializations are described as inheritance relationships in [20]. There is a distinction between four different inheritance relationships. One of them is the *projection inheritance* which refers to the notion of specialization in our process model. A formal definition is given in Definition 1.

As in [20] we use the term hidden activity to refer to an activity in a process which is not executed, i.e. this activity would be removed from the flow. The edges to and from the hidden activity remain in the process as a valid path but without these activity. For instance $SalesOrder3$ is a specialization of $SalesOrder2$. The additional activity is $Reorder$. If $Reorder$ is hidden in $SalesOrder3$ the processes are equivalent, i.e. they have the same set of process runs. The path via $Reorder$ is still an existing path, but without any activity. Therefore a hidden activity is also called an activity without effect.

Definition 1. *A process P' is a specialization or extension of P if the following conditions hold: (1) Each activity from process P is either an activity in P' or there is a subactivity in P' with respect to the terminology. The subactivity is subsumed by the activity from P. (2) If all additional activities in P' are hidden then P' realizes the same executions as the general process P. Additional activities of P' are activities that are neither in P nor subactivities of them are in P.*

These definition refers to definitions like in [5, 20]. In our model, we use DL concept subsumption to analyze process specializations. If a process specialization contains a parallel execution instead of a sequential or it contains additional activities or additional conditions, this specialization is subsumed, since it is modeled as concept intersection. These DL concept subsumptions conform to Definition 1, e.g. $SalesOrder3$ is subsumed by $SalesOrder2$ and $SalesOrder3$ is a specialization with respect to the definition.

A process containing a decision like $Reorder$ is not subsumed by a process without the decision. For instance the $Reorder$ process without $InteralReorder$

activity does not subsume the *Reorder* process, since the decision is modeled as a concept union and therefore define a more general concept. However, decisions are not specializations with respect to our definition, since hidden additional activities do not lead to the same process. The path of the hidden activity (e.g. $InternalReorder$) still exists, and therefore the activity $ExternalReorder$ could be omitted using the other path (with no effect) which is still available. For loops the DL subsumption also confirms to the definition. Adding a $Loop_j$ to a process is like adding a single activity to the more general process.

In [21] process specialization is defined using execution set semantics. The execution set of a process contains all process runs. Process P' is a specialization with respect to the minimal execution set semantics, if it contains at least all process runs from the general process. This definition is valid for processes P' containing decisions compared to a process P without decision. Under the maximal execution set the specialization consists of a subset of the process P. This definition refers to all other primitives except decisions that satisfy the concept subsumption $P' \sqsubseteq P$. Therefore this model can not be referred to only one of these specialization notions.

A refinement is an equivalent representation with another granularity (cf. [21]). The same process is described in a more fine-grained representation. The inverse of a refinement is an abstraction. A refinement replaces activities of a process with corresponding subprocesses, e.g. *Payment* is replaced in a *SalesOrder* process as described in Figure 2. A subprocess can also consist of a single activity.

4 Semantic Query Patterns for Process Retrieval

In Section 3 we have used the expressiveness of OWL to describe the control flow of processes, activity decomposition and specialization of activities and processes (Figure 4,5). In order to satisfy the requirements from Section 2.3 we express queries in DL and use DL reasoning to query the process models and derive process information and relationships.

Queries are general and incomplete process descriptions which specify the requested core functionality of the process. The query result contains all processes of the KB satisfying the query specification. The retrieval of a process with respect to the query depends on the axioms in the KB, i.e. the process information. We apply two different inference strategies in order to demonstrate how information or missing information in the KB may lead to different query results, depending on the applied inference method.

A strong inference method is the entailment of concept subsumption, i.e. the query subsumes the retrieved processes. The subsumption is used only in one direction. The result processes are specializations of the query. A weaker inference method is the satisfiability of concept conjunction, i.e. to test whether a process and the query process may have a common process run. This weaker condition leads to a higher number of results. The result processes are candidates for the query process. Missing information, e.g. missing disjointness axioms leads

to a higher number of positive results. In general, adding further axioms to the KB reduces the result set for this inference method.

We consider three non-disjoint query patterns with respect to the control flow of a process. The first pattern mainly considers the retrieval of processes with respect to the execution order. The second pattern considers terminological queries. Queries for the modality are demonstrated with the third pattern. The query pattern consist of three components: query, inference and result. The *query* input P is a process description. *Inference* is the description of the applied reasoning strategy, i.e. entailment or satisfiability. For sake of a shorter presentation, we only depict the applied inference for one process (here $SalesOrder1$), but this is performed for all processes in the KB.

Pattern for Execution Order A query for question Q1 from section 2.2 is described below. The query result contains all processes from the KB which conform to the described execution order.
Query: Which processes execute $Payment$ after $Delivery$?
$\{P \equiv \exists\, TOT.(Delivery \sqcap \exists\, TOT.Payment)\}$
Inference: Test entailment of concept subsumption:
$KB \models SalesOrder1 \sqsubseteq P, \ldots$
Result: $\{SalesOrder2, SalesOrder2\_Cash, SalesOrder2\_Credit, SalesOrder3\}$

The result contains all processes that execute $Delivery$ before $Payment$ with an arbitrary number of activities between them. The result includes also processes which specialize the $Payment$ activities. If the query should only retrieve processes with directly connected activities, the transitive role TOT is replaced by the role TO in the query. For instance the following query searches for all processes with $Payment$ as direct follower of $Order$.
Query: Which processes execute $Payment$ directly after $Order$?
$\{P \equiv \exists\, TOT.(Order \sqcap \exists\, TO.Payment)\}$
Inference: Test entailment of concept subsumption:
$KB \models SalesOrder1 \sqsubseteq P, \ldots$
Result: $\{SalesOrder1\}$

Pattern for Process Terminology This pattern uses the terminological knowledge. This covers the extension of processes and the specialization of activities. The following query searches for all processes in which the $Delivery$ activity is executed before the $Debit$ activity. In all described processes there is at least the $Invoice$ step between these activities.
Query: Which processes execute $Delivery$ before $Debit$?
$\{P \equiv \exists\, TOT.(Delivery \sqcap \exists\, TOT.Debit)\}$
Inference: Test entailment of concept subsumption:
$KB \models SalesOrder1 \sqsubseteq P, \ldots$
Result: $\{SalesOrder2, SalesOrder2\_Cash, SalesOrder2\_Credit, SalesOrder3\}$
The query refers to the refinement of $Payment$, since only the decomposed process contains the $Debit$ activity. If the query only uses the TO relationship the result-set is empty.

The next query searches for all extensions of the process $SalesOrder2$, as described in question Q4.
Query: Which processes extend $SalesOrder2$?
$\{P \equiv SalesOrder2\}$
Inference: Test entailment of concept subsumption:
$KB \models SalesOrder1 \sqsubseteq P, \ldots$
Result: $\{SalesOrder2\_Credit, SalesOrder2\_Cash, SalesOrder3\}$
The extensions $SalesOrder2\_Credit$ and $SalesOrder2\_Cash$ are generalized by $SalesOrder2$ since they contain a payment activity which is defined as a specialization of $Payment$. The extension relation between $SalesOrder2$ and $SalesOrder3$ is inferred since $SalesOrder3$ contains all activities from $SalesOrder2$ and additional activities ($Recorder$). For the entailment the same TO_i roles are required.

Pattern for Process Modality This pattern considers queries which express the modality of processes that refer to questions like Q2 and Q3 and also terminological query Q4 from Section 2.2. The first query searches processes that offer a credit card payment.
Query: Which process offers $CreditCardPayment$?
$\{P \equiv \exists\, TOT.CreditCardPayment\}$
Inference: Test entailment of concept subsumption:
$KB \models SalesOrder1 \sqsubseteq P, \ldots$
Result: $\{SalesOrder2\_Credit\}$

There is only one process definition with an explicit statement that the payment method is $CreditCardPayment$. The next query searches for all processes that only offer credit card payment. Using concept subsumption there is no process that fulfills this condition, i.e. is subsumed by the query process. Since in no process definition the condition is explicitly stated. However, if the inference is changed to the weaker satisfiability of concept intersection, there are processes which are candidates for satisfying the query. The p in the query refers to a process run which is not in the KB.
Query: Which process only offers $CreditCardPayment$?
$\{P \equiv \forall\, TOT.(\neg Payment \sqcup CreditCardPayment)\}$
Inference: Test satisfiability of concept intersection:
$KB \cup \{p : (SalesOrder1 \sqcap P)\}, \ldots$
Result: $\{SalesOrder1, SalesOrder2, SalesOrder2\_Credit, SalesOrder3\}$

The query description P describes that if there is a $Payment$ activity in the process it must be a $CreditCardPayment$. The formula is valid for all SalesOrder processes except for the $SalesOrder2\_Cash$ processes since there is an axiom in the KB that the payment methods $CreditCardPayment$ and $CashPayment$ are disjoint.

5 Evaluation

Dataset: We evaluated the process retrieval with a KB of 182 different process models. These process models are based on ten real world processes (business processes) from the SAP Business Workflow library[3], from the Enterprise Services (ES) Workplace at the SAP Developer Network (SDN)[4] and from the process library [5]. We used muliplicators in order to create from each basic process model between 16 and 25 different models. This muliplicator modifies the processes using the following operators: (1) Add and remove activities (2) Change of activity order, (3) Replace activity with sub- or superactivity and (4) Add and remove flow conditions. These process models were transformed into OWL. The process models contain unconditional decisions (70%), conditional decisions (65%), split/fork (40%) and loops (25%). The DL expressivity is $\mathcal{SHIQ}$.

Methodology: For the evaluation we used the Pellet 2.0.0 reasoner in Java 1.6 running on a computer with 1.7 GHz CPU and 1 GB RAM. The retrieval is evaluated on four knowledge bases with different size. The first contains 52, the second contains 107, and the third and fourth contain 182 process models. In the last two KB the size of the processes (number of activities) is different. Due to this generation method the KB contains many similar process models that only differ in some activities or their ordering.

Depending on the size of the KB the evaluation consists of 25 until 35 queries like the demonstrated queries from Section 4. In the evaluation each query invokes the reasoner without caching. The queries covering activity ordering, terminology and modality. The queries for activity order also considered activities within non-sequential process parts, like in parallel executions or decisions. For terminology, we queried also for flow conditions and their complement.

Result: The evaluation result is depicted in Table 4. The first column refers to the number of processes in the KB. The second column contains the average number (`Av.`) of activities in a process and the third column the maximal (`Max.`) activities of a process. The number of axioms of the ontology is described in column four. The columns five and six describe the retrieval time (average and maximum) for simple queries in milliseconds (`ms`). Simple queries contain only one activity or a negated activity. All other queries with multiple activities (at least three) are referred to as complex queries, the retrieval time is in the columns seven and eight. The size of the result set (number of process models) of the queries using concept subsumption is 9.3 for the smallest KB, 9.8 for the KB with 107 models, 13.7 and 14.6 for the knowledge bases with 182 models. The retrieval time for all processes which satisfy the concept intersection with the query process (weak reasoning condition) are outlined in column nine (`Sat.`).

The performance evaluation from Table 4 indicates the following results. The influence of the size of the process models is negligible, as described in No.3, 4.

[3] http://help.sap.com
[4] http://esworkplace.sap.com/sdn
[5] http://bpt.hpi.uni-potsdam.de/Prozessbibliothek/WebHome

No.	KB Size	Process Size		Axioms	Simple Query Time [msec.]		Complex Query Time [msec.]		Concept Sat. Time [msec.]
		Av.	Max.	Av.	Av.	Max.	Av.	Max.	Av.
1	52	12.7	19	238	1420	1504	1508	1820	1390
2	107	12.9	19	416	1684	1720	1608	1754	1603
3	182	12.9	21	568	4548	4606	4391	4590	4141
4	182	21.2	28	587	4793	4890	4460	4557	4178

Table 4. Retrieval Evaluation Result.

In both settings there are 182 process models in the KB but the average size of the process models in No.4 is about 60% higher. For simple queries the retrieval time is even lower. The retrieval time increases by the number of processes in the KB. As indicated in settings No.1 - 3 the retrieval time increases lightly from a KB with 52 processes compared with the KB with 107. If the KB is extended to 182 which is an increase of 70% the retrieval time is more than doubled. There is no performance difference between queries which contain activities either at the beginning of a process or at the end. The complexity of the query process (simple or complex) has also no influence. This indicates that only the number of process models in the KB affects the retrieval time.

The evaluation outlines the following qualitative results. (i) Processes form the KB are subsumed by the more general query process using the roles TO and TOT, except queries with decisions, since decisions are not considered as specializations and therefore not subsumed. (ii) However, if the query contains activities which are after a decision and before the merging of the flows the query has to cover either every sequence after the decision or use the disjoint flow conditions to retrieve the processes containing the decision. Since loops are special decisions, the retrieval of processes containing loops is realized in the same way.

(iii) The retrieval of refined processes depends on the refinement. If the refinement is realized by the decomposition of activities into subprocesses at the end of a process, the processes are retrieved using concept subsumption. Otherwise it requires two queries. For instance the query for a refined process $\exists\, TOT.(StockUpdate \sqcap \exists\, TOT.Payment)$ is transformed into a first query that selects the subprocess, the second query uses the start-activity of the refined process, e.g. $\exists\, TOT.(Start_i \sqcap \exists\, TOT.Payment)$, at which $Start_i$ is the start-activity (first activity) of the subprocess containing $StockUpdate$. The problem is that the order relation TO is not inferred for the subactivities (e.g. $StockUpdate$) with respect to the following activities of the original process (e.g. the relations from $StockUpdate$ to $Payment$).

(iv) More complex refinements, e.g. change of the original execution order in combination with activity specialization is not retrieved by concept subsumption.

Distinction to other Technological Approaches: (i) Another way of using OWL for process modeling is the annotation of process models, i.e. other process models like UML are enriched with OWL annotations. However, the aim of our model is a

generic process model in OWL with an explicit description of activity relations within a process. (ii) Non-ontological models like graph-based representations and queries [7] or tree-shaped diagram unravelling can manage the same ordering queries, but there is no reasoning benefit for process retrieval and a lack of representing terminological information, which is another aim of our approach.

Lessons Learned

Process Modeling: Modeling with cardinality restrictions leads to a more precise model. The difference between a decision and a parallel execution is emphasized with this restriction that there exists either exactly one follower or exactly n follower. There is a modeling tradeoff between accuracy and generic modeling. Generic descriptions account for a larger result set with respect to the demonstrated query mechanism. We do not model the join of decisions and parallel flows, which could be easily added with axioms like $D \sqsubseteq \exists\, TO_i^-.B \sqcap \exists\, TO_i^-.C$ for No.7 and 9 in Table 2. The join is described by the occurrence of the same activity (name) in all paths after the decision or split. This accounts for a maximal possible result set and a better identification of relationships between processes.

For the same reason we model decisions as union $\exists\, TO_i.(B \sqcup C)$ with only one TO_i relationship instead of $\exists\, TO_i.B \sqcup \exists\, TO_i.C$ in order to retrieve a process with this decision also for queries which contain both activities simultaneously, e.g. either B before C or vice versa. We do not require disjointness of activities in a flow and within decisions and parallel flows which could be simply added to the knowledge base by disjointness axioms.

Process Retrieval: Retrieval of refined processes is difficult due to the process definition using complex expressions. The relation from an activity to its successor is only defined for the process as a whole and not for the activities. However, in the demonstrated modeling methodology the reasoning complexity is lower than in the other modeling approach with explicit descriptions of the activity relations since the number of defined concepts in the ontology would be significantly higher. The query expressions are also more intuitive. Another benefit from the chosen modeling technique is the possibility to identify relationships between processes, e.g. process specializations. Since processes would be modeled by explicit relationships between activities, an activity can not occur in two different processes. Therefore it is required to use subclasses to denote the activities of the different processes.

6 Related Work

A comparison of our model with other process models is depicted in Table 5. The comparison contains modeling and retrieval.

For a core of our model, i.e. sequences only, this had already been accomplished by [14]. We significantly extend their model to account also for decisions, conditions, forks, joins, and loops. The matching of processes with different reasoning strategies is adopted from [13]. Process retrieval based on process information and annotation is realized in in [19]. A layered process representation is

Modeling Characteristic	[3]	[8]	[10]	[11]	[6, 16]	[19]	Our Model
Control Flow Modeling	y	y	n	y	n	n	y
Realization of Process Retrieval	n	n	y	y	y	y	y
Activity Terminology in Modeling and Retrieval	n	n	n	y	y	n	y
Process Refinement	y	y	y	n	y	y	y
Modality	n	n	n	n	n	n	y
State Change	n	n	n	n	n	y	n

Table 5. Comparison of Modeling Dimensions.

used. A service retrieval with additional functional information is described in [8, 15] Other applications use search algorithms for process retrieval, like [6, 16]. Similarity measure with respect to the process annotations and descriptions is used instead of logical reasoning. They use a process ontology and a corresponding query language. The retrieval in [10] also uses search algorithms instead of reasoning. DL based process models are described in [11] in order to enable process reuse. The model does not consider complex control flow.

The following approaches are focused on modeling and analyzing without retrieval. The OWL-S process model [1] is an OWL ontology for process description. Flow sequences are modeled as ordered lists. Like in OWL-S, processes are described as service interactions in WS-BPEL [2]. A XML messaging protocol is defined in order to specify the interoperability of services. The process specification language (PSL) [18] provides a mathematical model for activities, relations between activities and for data exchange. In [3] the control flow is analyzed according to the OWL-S process model. A combination of OWL and petri nets is demonstrated in [17]. OWL is only used to annotate the process models.

7 Conclusion

In this paper, we described process control flow modeling and retrieval in OWL-DL. For an application scenario we outlined the requirements of process retrieval with respect to the internal process structure. In order to fulfill these requirements it is necessary to describe explicitly the execution order of a process and to express the modality and terminology of process activities and structures. This also requires reasoning support to identify process extensions and terminological relationships between activities. Based on a process description with UML Activity Diagram the process models were transformed into OWL-DL. This contained a general description of the transformation and the design principles. The query patterns for process retrieval with respect to the control flow also cover the relationship between processes and demonstrate the usage of terminology and modality in the retrieval. In the evaluation the modeling and retrieval results, strengths and weaknesses are discussed.

References

1. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S, 2004.
2. Web Services Business Process Execution Language V. 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.
3. A. Ankolekar, M. Paolucci, and K. Sycara. Spinning the OWL-S Process Model - Toward the Verification of the OWL-S Process Models. In *Proc. of ISWC Workshop on Semantic Web Services*, 2004.
4. M.A. Aslam, S. Auer, J. Shen, and M. Herrmann. Expressing Business Process Models as OWL-S Ontologies. In *Proc. of Int. Workshop on Grid and Peer-to-Peer based Workflows (GPWW)*. Springer, 2006.
5. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *J. Log. Algebr. Program.*, 47(2):47–145, 2001.
6. A. Bernstein and M. Klein. Towards High-Precision Service Retrieval. In *Proc. of ISWC*. Springer, 2002.
7. D. Bildhauer and J. Ebert. Querying Software Abstraction Graphs. In *Working on Query Technologies and Applications for Program Comprehension*, 2008.
8. S. Ferndriger, A. Bernstein, J.S. Dong, Y. Feng, Y.-F. Li, and L. Hunter. Enhancing Semantic Web Services with Inheritance. In *Proc. of the ISWC*, 2008.
9. A. Gangemi, S. Borgo, C. Catenacci, and J. Lehmenn. Task Taxonomies for Knowledge Content D07. In *Metokis Deliverable*, pages 20–42, 2004.
10. Y. Gil, P.A. Gonzalez-Calero, J. Kim, J. Moody, and V. Ratnakar. Automatic Generation of Computational Workflows from Workflow Templates Using Distributed Data and Component Catalogs, 2008.
11. A. Goderis, U. Sattler, and C. Goble. Applying DLs to workflow reuse and repurposing. In *Description Logic Workshop*, 2005.
12. A. Goderis, U. Sattler, P. Lord, and C. Goble. Seven Bottlenecks to Workflow Reuse and Repurposing. In *Proc. of ISWC*, LNCS, pages 323–337, 2005.
13. S. Grimm, B. Motik, and C. Preist. Variance in e-Business Service Discovery. In *Proc. of the ISWC Workshop on Semantic Web Services*, 2004.
14. H. Hirsh and D. Kudenko. Representing Sequences in Description Logics. In *Proc. of AAAI*, 1997.
15. D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding Semantic Matching of Stateless Services. In *Proc. of AAAI*, 2006.
16. C. Kiefer, A. Bernstein, H.J. Lee, M. Klein, and M. Stocker. Semantic Process Retrieval with iSPARQL. In *Proc. of ESWC*. Springer, 2007.
17. A. Koschmider and A. Oberweis. Ontology Based Business Process Description. In *EMOI-INTEROP*, 2005.
18. C. Menzel and M. Grüninger. A formal Foundation for Process Modeling. In *Proc. of Int. Conf. on Formal Ontology in Information Systems*, pages 256–269, 2001.
19. M.Wolverton, D. Martin, I. Harrison, and J. Thomere. A Process Catalog for Workflow Generation. In *Proc. of ISWC*, LNCS, 2008.
20. W.M.P. van der Aalst. Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In *Petri Net Technology for Communication-Based Systems*, pages 383–408, 2003.
21. G. Wyner and J. Lee. Defining Specialization for Process Models. *Organizing Business Knowledge: The Mit Process Handbook, MIT Press*, 2003.