Towards Ontology Guided Translation of Activity-Centric Processes to GSM

Julius Köpke[1,2], Jianwen Su[1]

[1] Department of Computer Science, UC Santa Barbara, USA, `su@cs.ucsb.edu`
[2] Alpen-Adria Universität, Klagenfurt, Austria, `julius.koepke@aau.at`[**]

Abstract. There exist two major modeling paradigms for business process modeling: The predominant activity centric one and the artifact centric paradigm. Both are suitable for modeling and for executing business processes. However, process models are typically designed from different perspectives. Current translation methods operate on the syntactic level, preserving the point of view of the source process. The results of such translations are not particularly useful, understandable and insightful for stakeholders. In this paper we motivate the need for ontology-guided translations by comparing the results of a purely syntactic translation with a manual translation. We discuss shortcomings of the generated solutions and propose an ontology-based framework and sketch corresponding translation method for the generation of semantic translations, which allow to incorporate the point of view of the target modeling paradigm.

Key words: Process Translation, Artifact Centric BPM, Guard Stage Milestone, GSM, Semantic Process Abstraction

1 Introduction

In business process modeling, traditionally the activity- or control-flow perspective achieved the major attention, while the data perspective was addressed in a much lesser degree. This resulted in the predominant activity-centric modeling paradigm, where the data perspective is treated as an implementation issue rather than a modeling concern.

Data-centric approaches in general and the artifact-centric modeling paradigm in particular have emerged and gained an increasing momentum in the last decade. The essence is an integrated modeling method for both perspectives. As a result both competing paradigms are attracting users. When inter-organizational cooperations are concerned it is likely going to happen that interorganizational processes between companies that use activity-centric and the ones using artifact-centric modeling methods need to be established. Process views [7, 6, 14] have proven their usefulness to support activity-centric inter-organizational processes. Naturally, a new kind of "translating views" that allow the seamless

[**] Research conducted while visiting UC SB and supported by the Austrian Science Fund (FWF) under grant J-3609.

interoperation of activity-centric and artifact-centric processes to allow cross organizational interoperability is worth exploring.

In this paper we consider a particular sub-problem, namely "meaningful" translation between activity-centric and artifact-centric processes. We argue that in an inter-organizational and therefore heterogeneous setting existing translation approaches [8, 15, 21] are not desirable since they only address the syntactic aspect rather than taking into consideration domain knowledge. Therefore, we aim for a "good" artifact-centric representation of some activity-centric source process, where "good" means using the modeling capabilities of the target language and, since we are in an inter-organizational setting, agreed terms of the application domain.

In this paper we initiate the study by focusing on the translation from executable activity-centric processes to declarative Guard Stage Milestone [12] (GSM) models. In this scenario the activity-centric representation (e.g. BPEL) also specifies the data perspective for the execution of the process. However, the activity-centric process does not define any explicit relation between process variables and relevant data entities of the business domain, nor there is any information about relevant stages of business objects. As a consequence, a simple syntactic translation approach results in an executable target process that does not relate to real-world business entities and their states. Additionally, the target process is not taking advantage of GSM's capability of stage hierarchies.

The core idea of our approach to overcome these limitations is to make relevant domain knowledge accessible to the translation process by using *ontologies* and *semantic annotations*. Since ontologies are gaining popularity in the business world (e.g. in the tourism sector the open travel alliance ontology or in the insurance industry, the ACORD ontology to name only two), we think that the required additional effort to provide these descriptions is reasonable. As a consequence the translation of a business process remains automatic, while we can generate meaningful translations that refer to agreed terms of the domain regarding both, the activity- and the data-perspective including suitable abstractions thereof.

The contributions of this paper are the following. We present a novel syntactic translation approach for block-structured processes with data that produces hierarchic GSM models based on the hierarchy of the input process in Sect. 2.3. Then we discuss the weaknesses of syntactic translations in Sect. 2.4. We introduce a framework for the definitions of relevant semantics based on semantic annotations and a reference ontology in Sect. 3. A corresponding novel semantic translation method is sketched in Sect. 4. Finally, Sect. 5 discusses related work and Sect. 6 concludes the paper.

2 Motivations

We illustrate the need for domain knowledge to achieve meaningful translations by examining the output of a syntactic translation approach in comparison to a translation that could be provided by a domain expert.

2.1 Activity-Centric Process Model

We base our activity-centric source process model on the essentials of BPEL and inter-organizational view approaches [7, 6, 14], following block-structured processes which can be represented as trees. Supporting the typical control-flow constructs sequence (SEQ), parallel (PAR), loop ($LOOP$) and decision nodes (XOR). In the usual graph based representation PAR, $LOOP$, XOR-blocks are represented by corresponding *split*− and *join*− nodes. See examples on the left of Fig. 1. To enforce the block-structure in the graph representation, each split node must have its corresponding join node. Activities are executed by activity-steps, which may occur as leaf nodes in the tree-representation. The data perspective is modeled by declaring process variables (literal or XML-Type) and by specifying, which activity-step reads and writes to what variables and by defining conditions of XOR- or LOOP blocks in form of boolean expressions over process variables. For example the XOR-split node in Fig. 1 may have the Boolean condition $\$a > 10$ *or* $\$b = 1$, where $\$a$ and $\$b$ are variables of the process. The activity step A may be defined to read $\$b$ and write to $\$a$.

2.2 Guard Stage Milestone (GSM)

We briefly discuss the essentials of Guard Stage Milestone (GSM) relevant for this work here and refer the reader to [12, 4] for all details. A process is modeled in form of artifacts, where each artifact has a data schema holding data attributes and state attributes, and a life-cycle definition. GSM life-cycles are based on guards, stages and milestones. In the graphical representation (see Fig. 1), guards are depicted as diamonds, stages as rounded boxes with optional labels and milestone are depicted as circles. Guards, define when a particular stage becomes active, milestones define, when a stage is completed. Stages can be nested, where stages at leaf-level contain service calls for task execution. Guards and milestones may have labels and are specified by sentries. Sentries are defined in form of Event Condition (over the data schema) Action (ECA) Rules of the form *"on event if condition"*, *"on event"* or *"if condition"*. Events may be internal (such as achieving of a guard or a milestone) or external such as the completion event of a service call. Achieving events of sentries are denoted by the prefix + and their invalidation by the prefix −, respectively.

2.3 A Syntactic Translation Approach

Given a block structured activity centric process model as input, our syntactic translation is based on a set of transformation rules that transform each block-type to its GSM representation as shown in Fig. 1.

Translation Rules *A sequence* $< A, B >$ is transformed to two stages A' and B', where the guard of B' gets activated after the milestone of A' is reached ($B'.Guard = on{+}A'.Milestone$). If A refers to an activity-step, then A' contains a service call and the milestone of A' is reached on the completion event of

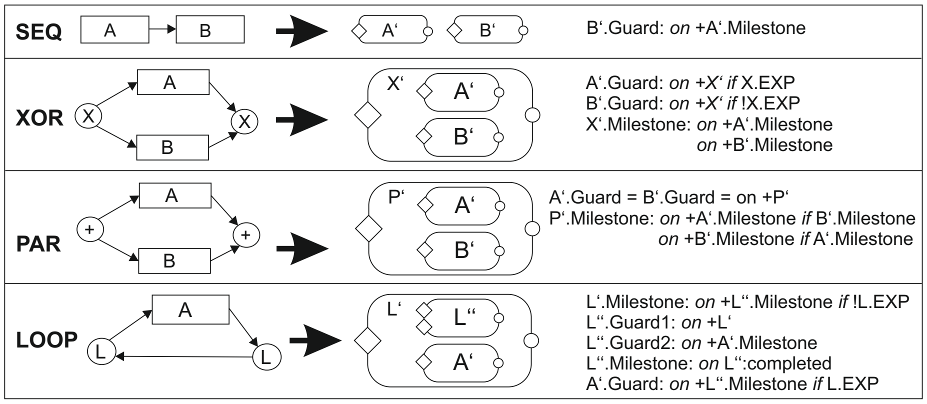

SEQ
A
B
A‘
B‘
B‘.Guard: on +A‘.Milestone
XOR
X
A
B
X‘
A‘
B‘
A‘.Guard: on +X‘ if X.EXP
B‘.Guard: on +X‘ if !X.EXP
X‘.Milestone: on +A‘.Milestone
on +B‘.Milestone
PAR
+
P‘
A‘.Guard = B‘.Guard = on +P‘
P‘.Milestone: on +A‘.Milestone if B‘.Milestone
on +B‘.Milestone if A‘.Milestone
LOOP
L
L‘
L“
L‘.Milestone: on +L“.Milestone if !L.EXP
L“.Guard1: on +L‘
L“.Guard2: on +A‘.Milestone
L“.Milestone: on L“:completed
A‘.Guard: on +L“.Milestone if L.EXP

Fig. 1. Syntactic Transformation Rules

the service in A. Otherwise, the milestone of A' is reached after achieving the milestone of the nested stage.

A XOR node with the sub-blocks A and B is translated to an anonymous stage X' with two sub stages A' and B', where the guards of A' and B' are defined over the opening event of X' and the xor-expression of the input block. The guard of A' refers to the xor expression, the one of B' to its negation. The milestone of X' is achieved, when either the milestone of A' or B' are achieved.

Parallel Blocks with the sub-blocks A and B are translated to an anonymous stage P' with the sub stages A' and B'. The guards of A' and B' are open as soon as the guard of P' becomes active. The milestone of P' is achieved when the milestones of A' and B' are both achieved.

A Loop-Block L with the sub-block A is translated to an anonymous stage L', containing the sub stages A' and L''. The stage L'' is used to determine if the loop needs to be executed or repeated, while A' holds the loop body. L'' is activated, when L' becomes active or when the milestone of A'' is achieved. The milestone of L' is reached if the milestone of L'' is achieved and the loop condition $L.EXP$ is $false$. In order to comply with the flip-once [4] restriction of GSM, the milestone of L'' is defined over the task completion event of the dummy task in L''. The same approach is used, if an XOR-block contains empty branches.

2.4 Weaknesses of Syntactic Translations

We will now discuss the properties of the translation approach based on the partly simplified example process for order processing shown in Fig. 2. Part a) of the example shows the activity-centric input process. Part b) shows the translation of a) based on the proposed syntactic translation algorithm. Part c) shows a GSM version of a) potentially created by a domain expert from scratch. Our syntactic translation shown in b) has a number of disadvantages in comparison to c):

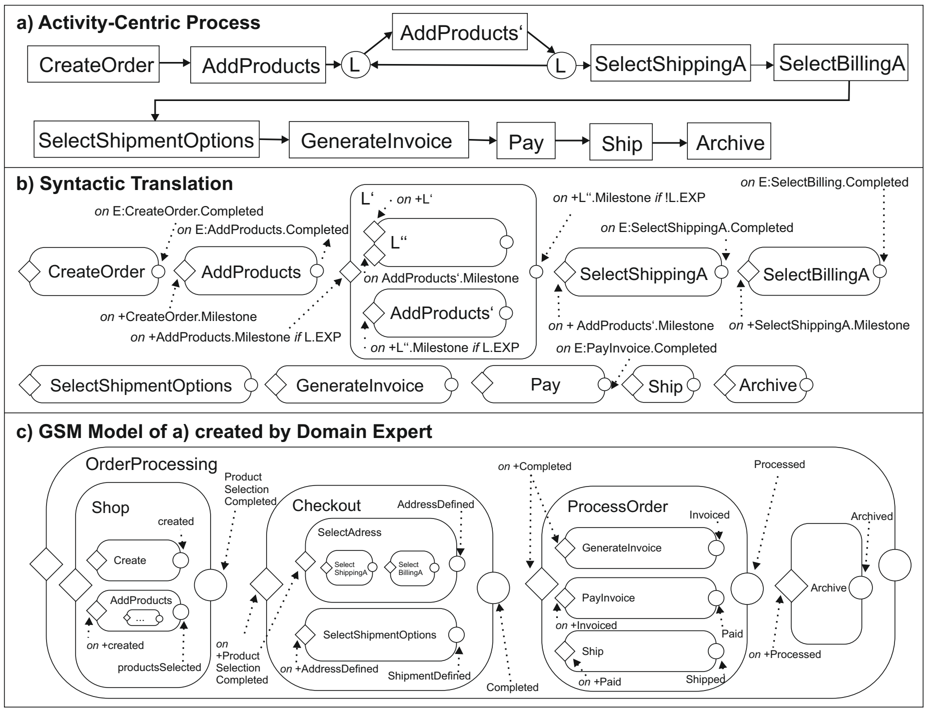


Fig. 2. Input Process a), Syntactic Translation b), GSM Process of Domain Expert c)

1. Milestones and guards are defined on a solely technical level not relating to any agreed real-world states of data objects. For example the milestone for Pay in Fig. 2 b) is defined by the completion of the Pay task. In contrast, the domain expert has modeled a stage $PayInvoice$ with the milestone *paid* in Fig. 2 c), where *paid* is a well-known state of order objects in the domain and $PayInvoice$ is a defined activity in the domain.
2. While existing translation approaches produce completely flat GSM models [20, 21], [8], our algorithm faithfully preserves the structure of the input process regarding LOOP, PAR and XOR-Blocks. However, this still leads to a mostly flat output process in b). In contrast, the domain expert makes use of nested stages in GSM, which allows to structure the process based on abstract state transitions that are well-known in the domain. As a consequence, c) is much better understandable than b). It encloses the activity stages in the upper-level stages $Shop$, $Checkout$ and $ProcessOrder$. In addition, the stages are described by meaningful labels for stages, guards and milestones referring to agreed terms of the domain.
3. Our translation approach generates exactly one artifact for each activity centric process. This contradicts with the aim of GSM to identify and model key artifacts, where a process is possibly composed by the interplay of multiple different artifacts (e.g. $Order$, $Invoice$, $Payment$).

Only the last critics (3) is solved by other translation approaches [15, 8, 21]. No existing approaches are capable to address critics one and two. All other approaches generate flat GSM models and they either follow similar ideas for encoding control-flow into stages and guards as our approach or they assume to get the state of business objects as an input. Our aim is to automatically or semi-automatically generate models like *c*) from models like *a*). From a more generic perspective, we argue that the quality [18, 9] of solution c) is superior to the quality of solution b). The reason is the different expressiveness of the activity-centric model and the GSM model, leading to partial triangle mappings between the real world, process a) and process c).

3 Towards Semantic Translations

As motivated in Sect. 2.4 a meaningful translation from an activity-centric process to a GSM process requires domain knowledge, which may only exist informally. We now present a framework that explicitly provides this knowledge by using a reference ontology defining relevant business objects and their states and a taxonomy of actions. The provided information is supposed to be generic for a specific domain and can therefore be reused for all activity-centric process models of that domain. The activity-centric processes are linked to the reference ontology by semantic annotations [13].

3.1 Ontology of Business Objects and States

The reference ontology (RO) is encoded using the DL fragment of OWL [19] and defines concepts for each relevant business entity such as *order* or *invoice* in our example. It also describes what attributes (data-type properties) and what relations to other entities (object properties) the different business entities have. Besides the conceptualization of business entities, the ontology also defines their valid states. States are described as defined OWL classes, which allow to derive a state hierarchy automatically and to classify instance data. For example the state *paid* of an *order* is defined as an *order* for which there exists an invoice and for that invoice there exists a payment. This can be expressed as following DL expression: "$invoice \sqcap \exists\ hasInvoice.(Invoice \sqcap \exists has.payment)$". In order to allow reasoning over states and matching of states against pre- and post-conditions of tasks, we propose to use such general definitions, rather than defining the states based on status attributes (e.g. an order is paid, if some attribute has the value *paid*). A fragment of an ontology for our *order* example is shown in Fig. 3. For simplicity, we depict no hierarchy of states.

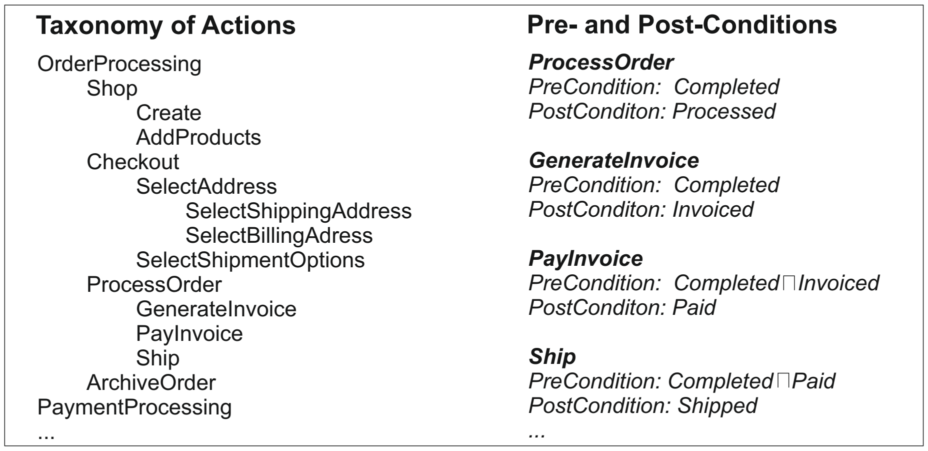
3.2 Semantic Annotations

The concepts in the ontology are linked to the activity-centric process by semantic annotations [13, 3, 22]. We propose to use annotation paths [13] for

Reference Ontology Classes	**Example Class Definitions**
BusinessEntity	
Order	$BillingAddressSelected \equiv$
Invoice	$Order.State \sqcap \exists\ hasBillingAddress.Address$
Payment	$ShippingAddressSelected \equiv$
BillingAddress	$Order.State \sqcap \exists\ hasShippingAddress.Address$
ShippingAddress	$AddressDefined \equiv$
...	$BillingAddressSelected \sqcap ShippingAddressSelected$
States	$ShipmentDefined \equiv$
Order.State	$Order.State \sqcap \exists\ hasShippingOption$
Empty	$Invoiced \equiv$
ProductsSelected	$Order.State \sqcap \exists\ has.Invoice$
ProductSelectionCompleted	$Paid \equiv$
BillingAddressSelected	$Order.State \sqcap \exists\ has.(Invoice \sqcap \exists\ has.Payment)$
ShippingAddressSelected	$Completed \equiv$
AddressDefined	$AddressDefined \sqcap ShipmentDefined$
ShipmentDefined	$Shipped \equiv Order.State \sqcap \exists\ hasShippingStatus.'shipped'$
Completed	$Processed \equiv$
Invoiced	$Invoiced \sqcap Paid \sqcap Shipped$
Paid	...
Shipped	
Processed	
Archived	
...	

Fig. 3. Fragment of the Example Ontology

annotating variable declarations of the activity centric process with the reference ontology. Following the annotation path method, we can for example annotate a process variable, used to store an *ordernumber* with the annotation path $/order/has/orderNumber$, where *order* is a concept, *has* an object property and $orderNumber$ a concept in the reference ontology. If the variable stores some complex XML-Type, then the XML-Type itself can be annotated using the annotation path method [13]. The annotations of variable declarations also implicitly maps read and write declarations of activities to the corresponding ontology concepts. This already allows to (heuristically) map pre- and post-conditions of activities to states in the ontology. For example, we can derive that the post condition of the task *pay* refers to the *paid* concept ($Order.State \sqcap \exists\ has.(Invoice \sqcap \exists\ has.Payment)$). This is possible even without explicit pre- and post-conditions of *paid*, assuming that *pay* reads from a variable that is annotated with an *invoice* concept and it writes to a previously non-initialized variable that refers to a *payment* concept. Such inference is possible for most states which are defined over the existence of a general relation between data entities (as most states in the example are). However, more fine-grained state changes require explicit pre- and post-conditions of tasks. These may either be defined by directly annotating [11] the pre- and post-conditions of tasks with states of the ontology or they may be defined on the syntactical level, still allowing to infer the actual state in the ontology. For example the state *Shipped* is defined by the value *shipped* for the data-type property *hasShippingStatus* in the ontology. In this case read- or write accesses declarations of the activity *ship* are insufficient to infer the post condition state *shipped*. However, if the activity *ship* has the expression *a=“shipped”* as its post condition, where the variable *a* is annotated with the annotation path */order/hasShippingStatus*, we can infer



Taxonomy of Actions

- OrderProcessing
  - Shop
    - Create
    - AddProducts
  - Checkout
    - SelectAddress
      - SelectShippingAddress
      - SelectBillingAdress
    - SelectShipmentOptions
  - ProcessOrder
    - GenerateInvoice
    - PayInvoice
    - Ship
  - ArchiveOrder
- PaymentProcessing
- ...

Pre- and Post-Conditions

ProcessOrder
PreCondition: Completed
PostConditon: Processed

GenerateInvoice
PreCondition: Completed
PostConditon: Invoiced

PayInvoice
PreCondition: Completed⊓Invoiced
PostConditon: Paid

Ship
PreCondition: Completed⊓Paid
PostCondition: Shipped
...

Fig. 4. An Example Taxonomy of Actions

that the post condition is the *shipped* concept. Such pre- and post-conditions may be manually defined or heuristically derived from a log [22].

3.3 Taxonomy of Actions

The taxonomy of actions (ToA) describes abstract, well agreed actions that result in state changes of a specific business entity. Such actions are organized in a part of hierarchy. In this paper, we assume that such a taxonomy is provided by the user. However, it may be created with the help of existing clustering techniques such as [10] or by employing domain ontologies such as the MIT process handbook [1].

Each action in ToA is annotated with pre- and post-conditions. Both are DL expressions over states in the ontology. The semantics of the ToA is the following: If some action b is defined as a child-action of some action a, then b is considered as a potential part of a. Action b may be used to achieve the post condition of a but it is not required to use b to achieve a in every case. Therefore, the ToA can be considered as a general glossary of actions which can be reused for different processes of the domain. An example ToA is shown in Fig. 4. In the example the action $ProcessOrder$ has the precondition $Completed$ and the post condition $Processed$. In the ontology $Completed$ is defined as $AddressDefined \sqcap ShipmentDefined$.

4 Ontology Guided Translations

After the inputs for an ontology assisted translation approach are defined in form of an annotated activity-centric process G, a reference ontology RO and a taxonomy of actions ToA, we can define what a desirable output of the translation is. The translation should use as much domain knowledge as possible. Therefore, obviously labels should be obtained from RO and ToA, for all matching stages, milestones and guards. Regarding the introduction of the state hierarchy as much hierarchic information from ToA should be added to the GSM process as possible. However, nesting should only be provided, when also groupings are performed. Therefore, nesting of a single stage into another stage is not considered as a desirable behavior.

4.1 Semantics-Based Translation Approach

We sketch how the proposed framework of an annotated process G, RO and ToA can guide the creation of a nested GSM process referring to agreed terms (guards, stages, milestones) of the domain. Our approach operates in three phases.

Phase 1: Syntactic Translation: First a syntactic translation is created that guarantees to preserve the behavior of the process in analogy to Sect. 2.3. We assume a 1:1 relation between activity-centric processes and artifacts. However, 1:n relations can be handled by first projecting the input process onto the different artifacts based on RO.

Phase 2: Mapping activities to states of RO and actions of ToA: Activities of the input process G are mapped with RO and ToA guided by the semantic annotations / data access definitions. This mapping has two purposes:

1) Setting labels for guards (mapped states of preconditions in RO) and milestones (mapped states of post condition in RO) of atomic GSM stages. E.g. Fig. 2 b), the label of the milestone of Pay can be set to *paid* because the post condition of the activity Pay matches the *paid* concept in RO (see Sect. 3.2 for details on this matching).

2) Mapping of each activity $a \in G$ based on pre- and post-conditions with actions in ToA using RO. An activity a potentially matches an action t in ToA, if $t.pre \sqsubseteq a.pre \wedge a.post \sqsubseteq t.post$.

Phase 3: Generation of Nested Stages: After the activities of G and therefore, also the atomic stages of the target process are matched to RO and ToA, we create a nesting of atomic stages guided by ToA. A set of activities $A = \{s_1, s_2, ..., s_n\}$ can be nested into a parent stage p from ToA, if p is a parent of every $s \in A$ in ToA and if A matches the pre- and post-conditions of p and there exists no set of activities A', where $A \subset A'$ and A' matches p. Assuming that the ToA is generic and applicable for different processes, not all sub-actions of an action in ToA are necessarily required to realize the parent action p. Therefore, we propose the following matching criteria based on pre- and post-conditions:

Matching Post-Conditions: We require that the steps of A produce at least the post condition of p. Therefore, the combined post condition of A must be an equivalence or subclass of the post condition of p.

Matching Pre-Conditions: The combined post condition of all preceding steps of $first(A)$ in G must be an equivalence or subclass of $p.precondition$, where

$first(A)$ defines the set of all steps $\subseteq A$ that are potentially executed first in G. Therefore, all preconditions for opening p are achieved in G before opening p.

Calculating Combined Post-Conditions: We propose to follow a similar approach for computing the combined post condition of multiple activities as [11, 22]. It depends on the control structures within the input process. We only discuss the case that A is a sequence in G here. The combined post condition of $< a, b >$ is *a.post* $\sqcap$ *b.post* unless *a.post* and *b.post* are disjoint. As [11, 22], we assume that contradicting effects of a are overwritten by b and consequently satisfiable solutions are generated by removing minimal sets of contradicting elements from *a.post*.

Example: In Fig. 2 the stages *GenerateInvoice, PayInvoice* and *Ship* have the same parent action $ProcessOrder$ in ToA. $ProcessOrder.pre = Completed$, $ProcessOrder.post = Processed$. The combined post condition of the previous steps is $ProductSelectionCompleted \sqcap Completed$ which is obviously a subclass of $Completed$. The post-conditions of the child stages are *Invoiced, Paid, Shipped*, and *Invoiced* $\sqcap$ *Paid* $\sqcap$ *Shipped* $\equiv$ *Processed* (see def. of $Processed$ in Fig. 3). Therefore, these stages can be nested into the stage $ProcessOrder$.

The nesting starts at the level of atomic stages / actions and subsequently also non atomic stages are nested based on the same principle. To ensure that as much of the structure of the ToA is reflected in the output process, the matching is done against the hierarchically nearest parents first and the procedure ends, when no nesting of two or more elements into a parent stage is possible.

Finally, the sketched translation approach allows to automatically generate the process shown in Fig. 2 c) based on the process shown in 2 a). The ontology and ToA should provide general descriptions of the domain of interest and they are not required to completely specify the input process. Therefore, the mapping of the activity-centric process to the ontology and ToA may only be partial. By starting with the syntactic translation we can still generate a complete and trace equivalent translation that takes advantage of the available knowledge.

5 Related Work

The translation of activity-centric processes to artifact centric models has been addressed in various works [15, 8, 21, 17, 20], where only [20, 21] and [8] generate GSM target processes. The approaches use different source models. While [8] takes UML activity diagrams as input (including data objects and state information), [21] is based on [20] and is used in combination with process mining and takes petri-nets as input. Our syntactic translation approach follows similar ideas as [20, 21]. However, since we use block-structured processes as input, a pattern based translation is directly possible for each block-type; references [21, 20] need to deal with more complex issues having petri-nets as input. The works in [15, 17] do not generate declarative GSM specifications but (synchronized) life cycle models. Ref. [15] focuses on the identification of relevant business artifacts based on the concept of domination of explicitly defined data objects. Although, the generation of life-cycle models is sketched, naming of states and state hierarchies are not addressed in the work. The work in [17] discusses the roundtrip transformation of activity-centric and artifact centric processes (non GSM). For generating artifact-centric representations, the activity-centric process is augmented with attribute definitions for defining pre- and post- conditions, which are derived from the business rules of the (existing) artifact centric target process. All approaches above generate syntactic translations of activity-centric to artifact-centric processes. None of them utilizes domain knowledge and no nesting of stages in the GSM target process is supported.

Abstractions of activity-centric processes have been studied in works such as [24, 10, 23, 16]. In particular knowledge-based approaches for the abstraction of activities such as [23] and [16] follow similar ideas, where a taxonomy of actions is used as input. However, these actions do not relate to business objects of the domain and they are defined by their label rather than by their pre- and post-conditions and an ontology.

Regarding efforts for the semantic enrichment of business processes, references [3, 22, 11] are closely related to our framework. In [11] business processes are annotated with semantic effects and cumulative effects are computed automatically, while [22] automatically derives post-conditions from log data. Ref. [3] assists the user in creating and augmenting process models by exploiting a domain ontology with a similar structure as our ontology framework but without a taxonomy of actions. Recently, an approach to represent GSM on the ontology level was presented in [5], leading to interesting future work such as the automatic translation to such models.

6 Conclusions

Syntactic translations cannot relate to relevant concepts and knowledge of the application domain and lead to poor results in comparison to translations provided by domain expert. In this paper we motivated and initiated the study towards a semantic translation approach to overcome this limitation. We developed a framework that allows to include the missing domain knowledge in the form of ontologies and semantic annotations. Based on this framework, we sketched a method for semantic translations that utilizes well agreed terms of the domain and that allows semantic nesting of stages in the target process. The resulting GSM models facilitate interoperation based on artifacts [2] syntactically and semantically. Current work includes the development of (semantic) quality metrics for GSM to further optimize and evaluate translation approaches.

References

1. *Organizing Business Knowledge: The MIT Process Handbook*, volume 1. The MIT Press, 1 edition, 2003.

2. D. Boaz, T. Heath, M. Gupta et Al. The ACSI hub: A data-centric environment for service interoperation. In *BPM'2014 Demos, Sept., 2014.*, page 11, 2014.
3. M. Born, F. Dörr, and I. Weber. User-friendly semantic annotation in business process modeling. *WISE 2007 Workshops*, in *LNCS 4832*, pages 260–271. 2007.
4. E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard–stage–milestone lifecycles. *Information Systems*, 38(4):561–584, 2013.
5. R. De Masellis, D. Lembo, M. Montali, and D. Solomakhin. Semantic enrichment of gsm-based artifact-centric models. *Journal on Data Semantics*, 4(1):3–27, 2015.
6. J. Eder, N. Kerschbaumer, J. Köpke et Al. View-based interorganizational workflows. In *Proc. of CompSysTech 2011, Vienna*, pages 1–10, 2011.
7. R. Eshuis, A. Norta, O. Kopp, and E. Pitkanen. Service outsourcing with process views. *IEEE T. Services Computing*, 8(1):136–154, 2015.
8. R. Eshuis and P. Van Gorp. Synthesizing data-centric models from business process models. *Computing*, pages 1–29, 2015.
9. A. Gemino and Y. Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requirements Engineering*, 9(4):248–260, 2004.
10. C. W. Günther and W. M.P. van der Aalst. Fuzzy mining adaptive process simplification based on multi-perspective metrics. In *BPM*, pages 328–343. 2007.
11. K. Hinge, A. Ghose, G. Koliadis. Process SEER: A Tool for Semantic Effect Annotation of Business Process Models In *IEEE EDOC CONF 2009*, pages 55–63. 2009.
12. R. Hull, E. Damaggio, R. De Masellis, et Al. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proc. of DEBS 2011*, pages 51–62. ACM, 2011.
13. J. Köpke and J. Eder. Semantic annotation of xml-schema for document transformations. In *OTM 2010 Workshops*, *LNCS 6428*, pages 219–228. 2010.
14. J. Köpke, J. Eder, and M. Künstner. Top-down design of collaborating processes. In *Proc. of IIWAS'2014*, pages 336–345, 2014.
15. S. Kumaran, R. Liu, and F. Wu. On the duality of information-centric and activity-centric models of business processes. In *CAiSE'08*, *LNCS 5074*, pages 32–47. 2008.
16. S. Mafazi, G. Grossmann, W. Mayer, M. Schrefl, and M. Stumptner. Consistent abstraction of business processes based on constraints. *JoDS*, 4(1):59–78, 2015.
17. A. Meyer and M. Weske. Activity-centric and artifact-centric process model roundtrip. In *BPM Workshops*, *LNBIP 171*, pages 167–181. 2014.
18. D. L. Moody. Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *DATA KNOWL ENG*, 55(3):243 – 276, 2005.
19. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview.* W3C Recommendation, 27 October 2009.
20. V. Popova and M. Dumas. From petri nets to guard-stage-milestone models. In *BPM Workshops*, *LNBIP 132*, pages 340–351. 2013.
21. V. Popova, D. Fahland, and M. Dumas. Artifact lifecycle discovery. *International Journal of Cooperative Information Systems*, 24(01):1550001, 2015.
22. M. Santiputri, A.K. Ghose, H. Khanh Dam, and X. Wen Mining Process Task Post-Conditions. In *CAiSE'15*, *LNCS 9381*, pages 514–527. 2015.
23. S. Smirnov, R. Dijkman, J. Mendling, and M. Weske. Meronymy-based aggregation of activities in business process models. In *ER 2010*, *LNCS 6412*, pages 1–14. 2010.
24. S. Smirnov, H. A. Reijers, M. Weske et. Al. Business process model abstraction: a definition, catalog, and survey. *DISTRIB PARALLEL DAT*, 30(1):63–99, 2012.