Flexible Artifact-Driven Automation of Product Design Processes

Ole Eckermann and Matthias Weidlich

Hasso Plattner Institute at the University of Potsdam, Germany
`Ole.Eckermann@student.hpi.uni-potsdam.de`
`Matthias.Weidlich@hpi.uni-potsdam.de`

Abstract. Automated support of business processes by information systems can be seen as state-of-the-art for many domains, such as production planning or customer relationship management. A myriad of approaches to the automation of business processes in these domains has been proposed. However, these approaches are not suited for highly creative processes, as they are observed in the field of innovative product design. These processes require a high degree of flexibility of the process implementation. In this paper, we focus on product design processes and propose a methodology for the implementation of supporting workflows. In order to cope with the imposed flexibility requirements, we follow an artifact-centric approach. Based on high-level process models, object life-cycle models are derived. Those are manually enriched and used for automatic generation of an executable workflow model. We also present an implementation of our approach.

Keywords: process automation, flexibility, artifact-centric, object life-cycles, methodology

1 Introduction

Since process orientation has been brought forward as a paradigm for structuring enterprises, process awareness emerged not only as an organizational principle, but had an impact on the design of information systems, cf., [1,2]. Process-aware information systems and workflow technology gained importance for the support of business processes in various domains [3], such as production planning or customer relationship management. This trend was manifested in a myriad of approaches to the automation of business processes and various standardization efforts as, for instance, WS-BPEL [4] and the WfMC reference model [5]. However, business processes in certain domains turn out to impose requirements on their implementation that are hard to address using common imperative workflow technology. An example for such a setting are clinical pathways that describe the different steps during interdisciplinary diagnosis and clinical treatment of patients in a hospital [6]. Information systems supporting this kind of processes, for instance, should allow for ad-hoc deviations of workflow instances. As such a feature is not supported by common workflow technology, specialized systems like the ADEPT system [7] have been developed.

In this paper, we focus on another example for processes that require a high-degree of flexibility of their implementation. That is, we consider the workflow support for highly creative processes as observed in the field of innovative product design. Although the actual design of consumer products is a manual task, the internal treatment of design proposals follows on predefined processes. Further, a high degree of repetition of these processes along with the need to track design decisions suggests to support such processes with according workflow technology. Unfortunately, processes for the treatment of product design proposals impose particular requirements on the underlying implementation, among them state-driven execution of activities and the possibility to react to externally triggered state changes of the proposals accordingly. These requirements suggest to approach the implementation of product design processes with the case-handling paradigm [8,9] or an artifact-driven approach [10,11]. We take up these ideas and show how they are adapted and extended for the concrete use case of product design processes.

Our contribution is a methodology for flexible artifact-driven automation of product design processes. With respect to the different involved stakeholders in product design processes, we propose two modeling perspectives to provide easy understandable process descriptions on the one hand and detailed technical specifications on the other hand. As design processes are artifact-centric we suggest object life-cycles for the technical specification of artifacts, while the high-level description is defined in a process modeling notation. Furthermore, we outline how both perspectives can be interrelated to apply consistency verification. While we rely on existing formalisms for the description of object life-cycles, we elaborate on object life-cycle composition and inheritance in detail. Finally, our approach comprises the generation of an executable workflow model with a structure that addresses the special needs of product design processes at runtime. Hence, our methodology covers all steps from the initial design of the overall processing to the specification of the supporting workflow. As an evaluation, we present an implementation of our approach.

The remainder of this paper is structured as follows. Section 2 outlines the characteristics of product design processes and defines requirements for process automation. We describe the levels of our methodology in Section 3 and elaborate on relations between models in Section 4. Our implementation is presented in Section 5. Finally, we review related work in Section 6 and conclude in Section 7.

2 Processes for Innovative Product-Design

Innovation is one of the key values for the success of almost every enterprise. Major potential for innovation lies in the process of designing new products. To get a maximum amount of promising product proposals, as many design processes as possible are performed. Thus, product design is a costly process that must be supported accordingly by information systems. Dealing with these processes involves some major challenges.

Artifact-centric process. A design process is characterized by intensive interdisciplinary collaboration, which requires artifact-centric views on running process instances. Various experts must be able to access the whole data available for certain proposals to be able to make decisions.

Many objects. To design a product, many ideas have to be evaluated and compared. Rather bad ones are rejected and resources are concentrated on promising ones. Furthermore, a product may be a composition of many different parts that are developed by different experts, but still influence each other.

State-driven enactment. Product design is a very creative process and must not be restricted by the underlying information system. Only the current state of a proposal determines the next activities, as it is very likely that certain activities have to be performed multiple times until their outcome is a suitable result. Besides, domain experts must be able to directly change properties of proposals, which might trigger state changes that cannot be anticipated.

Flexibility. Even if a proposal was accepted, it is still in progress. Certain tasks have to be performed again to incorporate new ideas and optimize the overall result. Therefore, redoing and skipping of tasks must be under the control of domain experts.

Impact of process environment. Product design processes may have long durations and take place in a constantly changing business environment. That makes it important to adapt processes fast. Additionally, changes might influence the goals of design processes and, therefore, should have direct impact on the execution of running process instances.

As the design process is driven by states of proposals, the actual process execution should be specified by object descriptions. Nevertheless, process models are helpful for the communication between stakeholders and for the specification of how business goals should be achieved, e.g. which design methods should be applied. Thus, a methodology for the automation of design processes should involve process specifications and detailed object descriptions. To maintain these models, which cover different perspectives and abstraction levels, methods for consistency verification must be provided. Design processes may become very complex as they comprise many different artifacts. A common approach to deal with such complexity is to create various models describing different artifacts and their behavior in different contexts. Consequently, there is the need for model composition to be able to derive one complete process specification. Finally, process execution must be state-based and flexible. Unforeseen external events must have direct impact on running process instances.

3 Methodology Levels for Artifact-Driven Process Automation

This section presents our methodology for the automation of product design processes. The three different levels are illustrated in Fig. 1. At the highest level, business experts create high-level models (HLMs) to define how business goals of the company have be achieved w.r.t. the design of new products. At the second

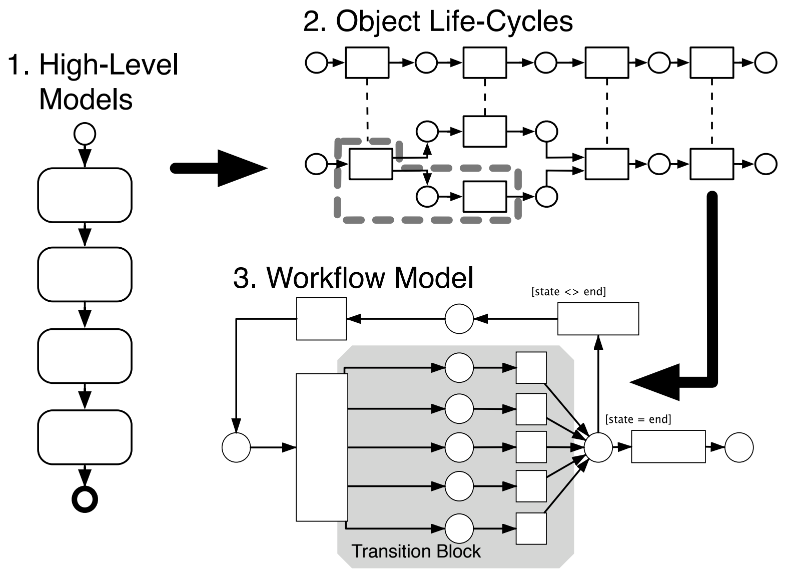


Fig. 1. Methodology for the automation of product design processes

level, domain and IT experts model OLCs for all business entities involved in the process. In addition, they create relations between HLMs and OLCs and among OLCs, that describe similar behavior, specialization of behavior and dependencies between objects. Based on such relations we provide consistency verification for appropriate handling of frequently changing processes. The third level describes an executable workflow model that can be automatically derived from an OLC. The workflow model has a structure that enables purely state-based process execution and rich flexibility at runtime.

3.1 High-Level Models

Notations for business process modeling such as BPMN, EPCs, or UML activity diagrams have become state of the art for business process discovery and design [12,13]. They provide rich tools for the definition of an ordering of activities to achieve business goals on the one hand and role models to define responsibilities for execution on the other hand. Even though product design processes are data-driven, process models are very helpful. May it be discovery of processes, process documentation, specification of responsibilities or adaptation to and verification of compliance rules, it turned out that process models are the most suitable language for the communication between all involved stakeholders [2]. Therefore, we propose to start modeling product design processes with a common process description language to get an overview of the general procedure. For illustration purposes, we use a BPMN subset containing activities, control nodes and edges to define such high-level models (HLMs). The restriction to a subset is necessary to be able to reason on the relation between HLMs and OLCs. For this subset, execution semantics are defined unambiguously by a translation into Petri nets models, cf., [14].

3.2 Object Life-Cycles

OLCs describe all states and state transitions of an object during its life time. States correspond to certain property sets but abstract from concrete data values, while state transitions represent business activities that have impact on the described object. State chart based notions like UML state machines and notions based on Petri nets are the most popular notions to model OLCs. The main difference between both notions is, that in a state machine an object state is represented by a node and, following on existing work [15,16], in a Petri net an object state is given by the *marking* of the net. In design processes, a product likely consists of various objects, which are developed in parallel. This has major impact on the respective OLCs, as it is necessary to define dependencies between those objects for a correct and complete system specification. Therefore, we us Petri net to define OLCs.

We interpret transitions as business activities and markings as states of an object handled by these activities. Accordingly, the Petri net must be safe as there is no reasonable interpretation in terms of an object state for a place of the Petri net that is marked with more than one token. Furthermore, we adopt the definition of OLCs introduced in [16]. An OLC has exactly one initial place corresponding to object creation and exactly one final place corresponding to object destruction. Each transition and each place is located on a path between these dedicated places.

OLCs describe the behavior of single object types. We assume that decisions at places, where alternative continuation is possible, are made based on the whole object state. This includes decisions made earlier in concurrent paths. We assume that potential deadlocks are avoided by the use of this information. Therefore, we consider OLCs to be correct, if they are relaxed sound [17]. That is, every transition has to be part of some firing sequence from the marking containing the initial place only to the marking containing the final place only.

3.3 Workflow Model

The third level of the methodology is a workflow model that enables rich flexibility during runtime and eases adaptation. The main idea behind the workflow model is to separate activities from the restrictiveness of control-flow and to determine execution orders purely state-based. That is, process execution is in general determined by the given OLCs and the defined state transitions, but the workflow model allows for deviations in terms of externally triggered state changes. Thus, the OLCs keep maintainable as they do not have to contain all deviations possible at runtime and business users are enabled to influence the actual process execution by manipulating the processed business objects and their states. Furthermore, unforeseen events in the business environment can be reflected by state changes and have direct impact on the process execution. The workflow model can be derived in an automatic transformation of an (composite) OLC.

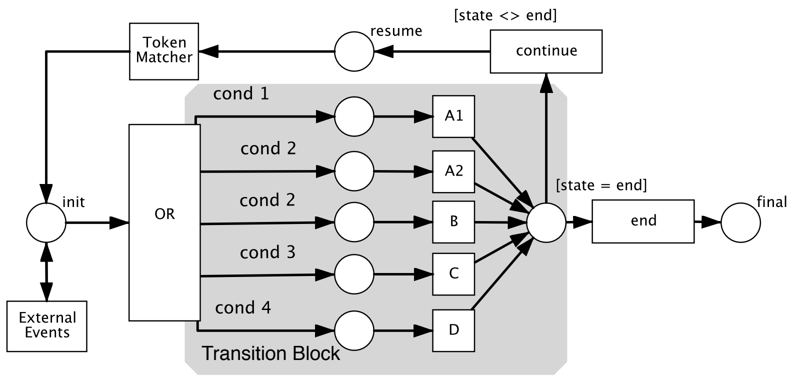


Fig. 2. Overview of the workflow model

We define the workflow model using the Coloured Petri nets formalism as it is illustrated in Fig. 2. The *Transition Block* contains all implementations of the activities performed on an object, i.e., transitions of the OLC. These transitions can be executed completely independent from each other. Tokens running through the net carry a variable that represents a current substate of the process, e.g. a process starts with one token in the place *init* carrying the initial state. According to the substate of a token, the transition OR enables transitions in the *Transition Block*. Each outgoing arc of OR has an expression, that specifies the substates where the respective transition becomes enabled. If multiple arc expressions are true, all respective transitions are enabled and executed concurrently. Basically, substates correspond to places in the OLC. A place in an OLC is either marked with zero or one token. Thus, a token in the workflow model carrying a value x indicates that in the current marking of the OLC there would be a token in place x. A major intention of the workflow model is to separate control-flow restrictions from the actual execution of transitions. Therefore, we have to mind concurrency and exclusive decisions in the transformation from an OLC to a workflow model.

Concurrency The concurrent execution of transitions in the workflow model is realized by enabling these transitions at the same substate carried by a token. The synchronization of tokens is also realized independent from the actual execution of transitions in order to keep the model flexible. Synchronization is realized by the *Token Matcher*. It is a subnet, that is entered by all tokens carrying a substate that requires synchronization. Tokens are matched pairwise. Synchronization of three or more tokens is realized in multiple steps. To be able to keep track of such complex synchronizations, additional substates for tokens are introduced during the transformation.

Exclusive decisions Exclusive decisions need special handling as the structure of the workflow does not allow for such decisions. Transitions, that are enabled at same substate are executed in parallel. Therefore, exclusive decisions become explicit transitions during the transformation. They occur in the *Transition Block* and produce a token with a substate that indicates which of the exclusive transitions has to be executed. Accordingly, each of the exclusive transitions has a unique substate as its precondition (i.e. arc expression).

External Events One important requirement for the execution of design processes is the ability to handle unforeseen events. We propose the strict separation of control-flow restrictions and implemented transitions as the solution. An external event causes an arbitrary change of data of an object involved in the process. To be able to continue the process execution, the object must be in a consistent state after the event happened. Therefore, the new object state must occur in the respective OLC and can be translated into valid substates for tokens. Consequently, an external event can be reflected by continuing process execution with a set of tokens carrying the substates that represent the new object state. Note that process continuation after an external event is not equal to a complete rollback. Likely, such an event changes certain properties of an object, but it will not have impact on all data that was created during former process execution. Some transitions might be skipped when continuing process execution, as the data they would produce is already present and still valid. Additionally, an external event during the execution of two concurrent branches might trigger redoing of an activity preceding the parallel split. In this case, either the split and all succeeding activities are performed again or the execution of one branch continues and activities of the other one are executed again, because only data relevant for this branch has changed. To ensure the correct number of tokens in the net at any time, the latter option is traced back to the first one by skipping all activities that have been executed before. We propose that such skipping or redoing of transitions is under the control of domain experts, as an execution engine is in general not able to decide whether existing values are still valid.

4 Relations between Models

The business environment of product design processes is constantly changing. Hence, the development of sufficient support with information systems is an iterative process where changes are very likely. Changes induce the possibility of inconsistencies. Therefore, we propose to relate similar parts among OLCs and and between OLCs and HLMs and present a notion for consistency verification based on these relations. Besides, we investigate OLC composition in this section, which is necessary to express dependencies between different objects.

4.1 Inheritance of Object Life-Cycles

Products are often related to others, they belong to product categories. Members of the same category differ from each other in certain specializations. A methodology for product design processes must reflect such specialization, to express that similar goals are achieved in similar manner. In order to allow for effective handling, specialization dependencies should be reflected when modeling OLCs. Thus, the life-cycle may be specified for a category of products first, while it is later refined for specific product subtypes. We address this demand by introducing a novel notion of inheritance for OLCs. Fig. 3 depicts a parent and a child model, which we will use for illustration purposes. We assume a refinement relation between transitions as indicated by dotted lines, i.e. the transition set $\{A1', A2'\}$ in the child model refines the transition set $\{A\}$ in the parent model. A pair of related transition sets is called correspondence.

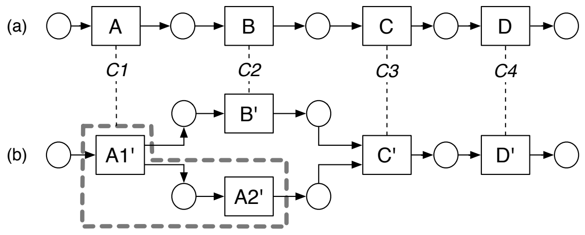


Fig. 3. Specialization of an Object Life-Cycle

Evidently, we do not require such refinements of OLCs to be *hierarchical.* This is motivated by the observation that most sequential orderings of transitions are due to some data dependencies. That is, the preceding transition will provide some data that the succeeding transition requests. During specializing of OLCs, the single transitions might be refined into several transitions representing smaller units of functionality. Now, it is very likely that in the process of refining transitions one will discover potential for optimization in terms of parallelization as the data dependencies hold solely between some of the refined transitions. Fig. 3 illustrates such a scenario. The transition B' depends on the data provided by $A1'$ but is independent from $A2'$. Therefore, it is not necessary to wait until $A2'$ is finished to start B'.

Against this background, existing techniques for the verification of consistency of an inheritance relation between two behavioral models cannot be applied. Those either require refinements to be hierarchical in terms of single-entry-single-exit subgraphs [18] or require that the observable behavior does not change [19,16]. Consequently, the described scenario would be considered to be inconsistent. In order to cope with scenarios as introduced above, our notion of inheritance allows for sequentialization and parallelization. The core idea of the consistency notion is to check whether each ordering of transitions in the parent model, with respect to their correspondences, is reflected by some transitions in the child model. That is, if there is a transition t_1 in the parent model belonging to a correspondence C_1 that is always preceding a transition t_2 belonging to a correspondence C_2, there must be a transition $t_3 \in C_1$ that always precedes a transition $t_4 \in C_2$ in the child model.

Our inheritance notion is formally grounded on trace semantics. For the model for OLCs introduced above, a complete trace is a firing sequence of Petri net transitions leading from the initial marking to the final marking. In Fig. 3, for instance, the child model induces the traces $A1', B', A2', C', D'$ and $A1', A2', B', C', D'$. To decide consistency regarding to an inheritance relation between two models, consistency between all pairs of correspondences is checked. Consistency of a correspondency pair is decided by comparing all traces of both models. As a correspondence is possibly complex, and therefore, transition sets may have different cardinalities, we partition traces into sub-traces before comparing them. A partitioning induced by a pair of correspondences is based on a classification of transitions as either being *interleaving*, being part of one of the transition sets without being interleaving, or being not part of any transition set. A transition belonging to either of the transition sets is interleaving if there is another transition belonging to the other transition set and there is at least one reachable marking where these two transitions are enabled concurrently. For our aforementioned example, transitions B' and $A2'$ in the lower model are interleaving transitions, when comparing the correspondences C_1 and C_2.

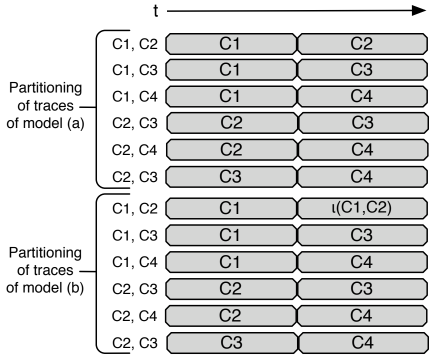


Fig. 4. Partitioning of traces of the models in Figure 3

Fig. 4 illustrates the partitioning of traces for all pairs of correspondences defined in Fig. 3. A sub-trace is labeled like a correspondence, if it contains non-interleaving transitions belonging to the transition set associated to the respective correspondence. Interleaving transitions are labeled by ι and the label of the two respective correspondences. Transitions that are not part of any of the transition sets are neglected. We conclude that both models show equal trace partitionings for the pairs of correspondences (C_1, C_3) and (C_2, C_3). They are sequentially ordered without any interleaving transitions. For the pair (C_1, C_2), however, both models have a different trace partitioning. While the parent model shows a sequential ordering, the child model shows a sub-trace of non-interleaving transitions belonging to C_1 followed by a sub-trace of interleaving transitions belonging to C_1 and C_2.

Based on such a trace partitioning we decide whether two correspondences are consistent. Therefore, the partitionings are transformed into a representation that reflects implied data dependencies only. That is, all interleaving parts are either neglected or replaced by an non-interleaving part belonging to one of the transition sets. An interleaving part is

- hidden if it succeeds a non-interleaving part of one transition set and precedes a non-interleaving part of the other transition set;

- ◦ replaced by a non-interleaving part C_2, if it is surrounded by non-interleaving parts of C_1 and vice versa;
- ◦ replaced with a non-interleaving part different to the one that precedes the interleaving part, if it is the last part of the trace;
- ◦ replaced with a non-interleaving part different to the one it precedes, if it is the first part of the trace.

Traces are considered to be compatible, if they have equal trace partitionings after this transformation. A pair of correspondences is consistent, if for each trace in the parent model, there is a compatible trace in the child model. Two OLCs are in a consistent inheritance relation if all pairs of correspondences are consistent. To ensure, that an object implements the complete behavior of its parent, all transitions of the respective parent model must be part of a correspondence with the child model.

4.2 Composition of Object Life-Cycles

Inheritance of OLCs copes with specialization relations between products. Products may also be related by composition relations. Many products are compositions of several parts with special functionality, which can be developed independently to a certain degree. Such compositions impact also on the respective OLCs and have to be considered when deriving one as a complete system specification for process automation. Furthermore, we propose to specify the behavior of a product in various OLCs that describe different aspects of the products behavior to reduce complexity and to ease maintainability. We distinguish three composition types.

Synchronous Some activities in a process have impact on multiple objects. For example, the design of intersection points between two independently developed components is one task that affects both components. Execution of such activities triggers state changes in both respective OLCs, and these activities likely have preconditions regarding to all involved objects. Thus, OLCs of these components must be composed not only to derive a complete system specification but also to be able to verify consistent behavior. The composition is realized by identifying and merging transitions of all involved OLCs that represent equal activities.

Asynchronous In contrast to activities that have impact on multiple objects, other activities require various objects to be in certain states without changing them. That is, certain properties of one component must already be defined because they have impact on the design of other components. In terms of OLCs this means a transition of one model is waiting for the completion of a transition of another model. Consequently, models are composed by connecting these transitions with an additional place.

Alternative Continuation The OLCs of single objects may become complex, for instance, if objects occur in various contexts where they show different behavior. A common approach to reduce complexity is to define several models that describe different aspects of the behavior of the same object. These models have certain states in common where a context change might happen. Hence, composition is realized by merging all similar places of models describing the same object. Composition by merging places makes sense for models describing the same object only. If one would compose OLCs describing different objects using a place, it would express that two different objects could reach the same state. This is a contradiction to the semantics of a state, as it says that two objects in equal states cannot be distinguished.

Consistent Compositions Composing OLCs as described before leads to a model that contradicts the definition of an object-life cycle, as it may contain multiple initial and final places. This must be resolved by creating a new initial and a new final place and connecting them with the initial and final places of the OLCs describing the singe components. After this transformation, composite OLCs are similar to standalone ones. Thus, the composition must be bounded and safe, as a marking still represents a state and there is no valid interpretation for a place with multiple tokens. Additionally, composite OLCs must always describe the aggregated behavior of all original OLCs and can optionally describe additional behavior in terms of interaction between objects. Consequently, the composite OLC must not contain dead transition, each transition must be part of a trace from the initial to the final marking. Otherwise, the described behavior has changed and the composite OLC is considered to be inconsistent. We have argued before, that syntactically possible deadlocks in OLCs are avoided, because the whole state of an object is considered in exclusive decisions. This argument also holds for OLC compositions. Thus, a consistent composition is relaxed sound.

In sum, OLC composition enables the structured definition of an artifact-centric system specification. A system can easily be extended by adding OLCs describing new aspects and adapted by editing or removing existing OLCs. Consistency verification of both, single models and composite ones, can be applied by checking for relaxed soundness. OLC compositions ease the maintainability of large, constantly changing sets of models and provide methods to derive one system specifications for process execution.

4.3 Relation between High-Level Models and Object Life-Cycles

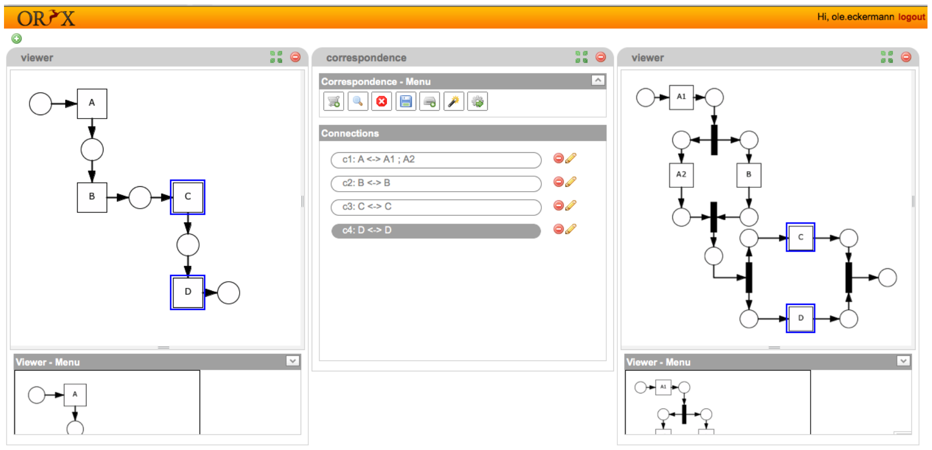
HLMs and OLCs have very different modeling purposes and, therefore, are different views on the same process. Furthermore, they are at different abstraction levels: while HLMs consist of the most important activities only, object-life cycles give detailed information about the single states and state changes of all involved business objects. Obviously, these models will have many differences, but there are also some similarities, since both views describe behavior of the same process. The similar parts of the models should not contradict in the described behavior. We propose to specify similar parts between HLMs and OLCs using correspondences. Hence, activities in HLMs correspond to transitions in OLCs. Again, correspondences might be complex, meaning they are defined between sets of activities and sets of transitions. A process likely involves several business objects. That raises the question how to correlate multiple OLCs with one HLM and whether it is allowed that a set of activities in a HLM corresponds to various sets of transitions in different OLCs. This is certainly true, but using the methods described in Section 4.2 these models can be composed to one OLC. Transition sets corresponding to the same activity are indicating synchronization points. Correspondences between the HLM and the composite OLC must be non-overlapping, as the semantics of overlapping correspondences are unclear.

Section 4.1 introduced a consistency notion for OLC inheritance based on complex correspondences. This notion is motivated by data dependencies between activities and can be applied for relating HLMs and OLCs, even though modeling purpose and view on the process are different. In either model type control-flow restrictions for activities are specified. As correspondences identify similar activities, the restrictions defined for them must not contradict. To apply the consistency notion, HLMs have to be translated to Petri net. As previously mentioned, such a mapping for our BPMN subset is given in [14].

5 Implementation

We implemented our approach prototypically and integrated it into the Oryx Project[1]. The Oryx Mashup-Framework contains a Gadget that allows two define correspondences between models. Based on these correspondences, consistency regarding OLC inheritance can be verified. Fig. 5 depicts a parent (left) and a child model (right) and four defined correspondences. The correspondence pair $c3 = (\{C\}, \{C\})$ and $c4 = (\{D\}, \{D\})$ is inconsistent. Therefore, all respective transitions have been highlighted. Currently, the implemented algorithm is restricted to sound free-choice Petri net based models, i.e. HLMs that can be mapped to sound free-choice Petri nets and sound free-choice OLCs. This restriction is because of the strong relation between syntax and semantics of this Petri net class, that allows for more efficient calculation of consistency. Besides the consistency notion, we implemented the transformation from a (composite) OLC into a workflow model. To integrate the transformation into Oryx, a client and a server plugin have been realized. The client plugin offers the functionality to trigger the transformation for the OLC currently opened in the Editor. The server plugin realizes the transformation and responds with the generated workflow model in a JSON representation. The workflow model can be displayed in Oryx by creating a new Coloured Petri net and importing the JSON. Finally, the model can be exported to CPN Tools[2] for process simulation.

[1] `http://www.oryx-project.org`
[2] `http://cpntools.org/`



c1: A <-> A1 ; A2
c2: B <-> B
c3: C <-> C
c4: D <-> D

Fig. 5. Transitions regarding to inconsistent correspondences are highlighted

6 Related Work

In recent years, *artifact-centric* design of business processes gained increasing attention, see [10,11]. Following this methodology, the key driver of modeling and execution are business artifacts that are specified by both, an informational model and a life-cycle. Activities operate on artifacts and are responsible for state changes. In contrast to such an integrated view, we propose a methodology containing both process models and OLCs that are interrelated. Moreover, our approach goes beyond the existing work by defining OLC hierarchies and compositions. To this end, we introduced a notion of inheritance for OLCs that is related to work on behavior inheritance [19,16]. The latter builds on the notion of *branching bisimilarity* and adapts it to the setting of partially corresponding models. For two behavioral models, nodes that are without counterpart are either be *blocked* or *hidden* when assessing behavioral equivalence. The notion of inheritance introduced in this paper is much weaker than the notions of behavior inheritance. We argue that specialization of objects may be non-hierarchical, so that potential sequentialization or parallelization of activities calls for a more relaxed notion of inheritance.

Regarding the composition of OLCs, our notion for synchronous composition is related to work on the composition of UML state machines [20]. Asynchronous composition has been investigated in the work on *proclets* [21]. Finally, the composition of alternative continuations is related to scenario-based modeling using *oclets* [22]. Oclets specify intended behavior, while anti-oclets are used to express forbidden behavior under certain preconditions. A complete process description is derived by composing oclets at runtime. A different approach aiming for more flexibility during process execution is the concept of pockets of flexibility [23]. These approaches mainly concentrate on reducing complexity of modeling flexible (parts of) processes. We enable flexibility by allowing for not-specified execution sequences that are caused by unforeseen external events.

Finally, techniques for adaptive process management are also related to our work. Adaptive process management has been studied in the ADEPT project [7] – a process management system which is able to handle changes during runtime. The creation of consistent process structures by OLC composition is investigated in [24]. They further discuss changes in the process structure in terms of adding/deleting OLCs or dependencies between them. While these approaches aim for easy and consistent adaption during runtime, we rather focus on flexibility. Therefore, we consider them to be complementary.

7 Conclusion

In this paper, we presented an artifact-driven methodology to automate innovative product design processes. We propose two modeling perspectives: high-level models and object life-cycles. Our contribution here is a novel consistency notion that is insensitive to control-flow structures and relies on data dependencies. The notion can be applied to ensure consistency between both perspectives and among object life-cycles with non-hierarchical refinements. Furthermore, we presented methods for object life-cycle composition to derive a complete system specification. These methods ease adaption, extension and maintainability of complex and constantly changing processes. Finally, we introduced a workflow model for process execution. The novel structure of this model enables rich flexibility by determining control-flow purely state-based. The workflow model is derived in an automatic transformation from an object life-cycle into a Coloured Petri net.

In future research, we aim to extend our approach to instance correlation. In product design it is common, that starting from one proposal a large set of proposals with slightly different values for certain properties is created during the design process. As these proposals do have a lot in common, they are not handled isolated and run as a set through the process. Certain activities might handle them equal to a single instance, i.e. the manipulated data is equal for all instances, others will handle them as a list, for example approval tasks where certain proposals are rejected and others are accepted.

References

1. Dumas, M., van der Aalst, W.M., ter Hofstede, A.H.: Process-aware information systems: bridging people and software through process technology. John Wiley & Sons, Inc., New York, NY, USA (2005)
2. Weske, M.: Business Process Management – Concepts, Languages, Architectures. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
3. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall (1999)
4. Alexandre Alves et al.: Web Services Business Process Execution Language Version 2.0. Technical report, OASIS (January 2007)

5. Hollingsworth, D.: The Workflow Reference Model. Technical report, WFMC (January 1995)
6. Lenz, R., Blaser, R., Beyer, M., Heger, O., Biber, C., Bäumlein, M., Schnabel, M.: It support for clinical pathways - lessons learned. In: MIE. Volume 124 of Studies in Health Technology and Informatics., IOS Press (2006) 645–650
7. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science - R&D **23**(2) (2009) 81–97
8. Reijers, H.A., Rigter, J.H.M., van der Aalst, W.M.P.: The case handling case. Int. J. Cooperative Inf. Syst. **12**(3) (2003) 365–391
9. van der Aalst, W.M., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data and Knowledge Engineering **53** (2005) 2005
10. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling, chapter 23. (2009) 503–531
11. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3) (2009) 3–9
12. Yourdon, E.: Modern structured analysis. Yourdon Press Upper Saddle River, NJ, USA (1989)
13. Luftman, J., Papp, R., Brier, T.: Enablers and inhibitors of business-IT alignment. Communications of the AIS **1**(3) (1999)
14. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes - a survey. T. Petri Nets and Other Models of Concurrency **2** (2009) 46–63
15. Preuner, G., Schrefl, M.: Observation consistent integration of views of object life-cycles. In: BNCOD 16: Proceedings of the 16th British National Conferenc on Databases, London, UK, Springer-Verlag (1998) 32–48
16. Basten, T., van der Aalst, W.M.P.: Inheritance of behavior. Journal of Logic and Algebraic Programming **47**(2) (2001) 47 – 145
17. Dehnert, J., van der Aalst, W.M.P.: Bridging the gap between business models and workflow specifications. Int. J. Cooperative Inf. Syst. **13**(3) (2004) 289–332
18. Brauer, W., Gold, R., Vogler, W.: A survey of behaviour and equivalence preserving refinements of petri nets. In Rozenberg, G., ed.: Applications and Theory of Petri Nets. Volume 483 of Lecture Notes in Computer Science., Springer (1989) 1–46
19. Schrefl, M., Stumptner, M.: Behavior-consistent specialization of object life cycles. ACM Trans. Softw. Eng. Methodol. **11**(1) (2002) 92–148
20. Ryndina, K., Küster, J.M., Gall, H.: Consistency of business process models and object life cycles. In: Models in Software Engineering. Volume 4364 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2007) 80–90
21. van der Aalst, W.M., Barthelmess, P., Ellis, C., Wainer, J.: Workflow modeling using proclets. In: Cooperative Information Systems. Volume 1901 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2000) 198–209
22. Fahland, D.: Oclets - scenario-based modeling with petri nets. In: Petri Nets. (2009) 223–242
23. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specification. In: Conceptual Modeling – ER 2001. Volume 2224 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2001) 513–526
24. Müller, D., Reichert, M., Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In: Advanced Information Systems Engineering. Volume 5074 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 48–63