Synthesizing Object-Centric Models from Business Process Models

Rik Eshuis(✉) and Pieter van Gorp

Eindhoven University of Technology, P.O. Box 513,
5600 MB Eindhoven, The Netherlands
{h.eshuis,p.m.e.v.gorp}@tue.nl

Abstract. Business process models expressed in UML activity diagrams can specify the flow of multiple stateful business objects among activities. Such business process models implicitly specify not only the life cycles of those objects, but also their communication. This paper presents a semi-automated approach that synthesizes an object-centric system design from a business process model referencing multiple objects. The object-centric design can be used to perform the process in a flexible way.

1 Introduction

The classic way to model business processes is to specify atomic activities and their ordering in a flowchart-like process model. In recent years, data-centric modeling paradigms have increasingly grown popular in research and industry as alternative to the classic, process-centric paradigm. Data-centric modeling approaches aim to have a more holistic perspective on business process [1,2] and support semi-structured, knowledge-intensive business processes [3].

These two paradigms are often positioned as alternatives, each having their own modeling techniques and implementation technologies. Data-centric approaches use for instance state machines [1,2,4,5] or business rules [6] as modeling techniques, while process-centric approaches use process flow models such as UML activity diagrams [7] or BPMN [8], where each modeling technique is supported by dedicated engines.

In practice, however, the strengths of both approaches should be combined. Process-centric models show clearly the behavior of the process, while in data-centric models the expected behavior is difficult to predict, either since the global process is distributed over different data elements or since the behavior is specified in a declarative, non-operational way such as with the Guard-Stage-Milestone approach [6]. Whereas data-centric approaches support more flexible ways of performing business processes than process-centric approaches [9].

We envision that process modeling techniques will be used to specify the main "default" scenarios of a process whereas a data-centric approach is actually used to realize the approach, adding additional business rules for exceptional circumstances. This allows actors to perform the process in the prescribed way for a default scenario, but respond in a flexible way to exceptional circumstances

N. Lohmann et al. (Eds.): BPM 2013 Workshops, LNBIP 171, pp. 155–166, 2014.
DOI: 10.1007/978-3-319-06257-0_13,

not covered by the default scenarios, which is one of the strengths of object-centric process management [9].

This paper outlines a semi-automated approach for creating an object-centric design from a business process model that specifies the main scenarios in which these objects interact. The approach uses synthesis patterns that relate process model constructs to object life cycle constructs. The resulting object-centric design can be refined with additional "non-default" behavior such as exceptions.

The approach uses UML, since UML offers a coherent notation for specifying both process-centric and data-centric models. We specify business process models that reference objects in UML activity diagrams with object flows. We use UML statecharts (state machines) to model communicating object life cycles, which specify an object-centric design.

The remainder of this paper is structured as follows. Section 2 summarizes a previously developed approach for synthesizing one object life cycle from a business process model that references the object. This paper extends that approach to the case of multiple objects that interact with each other. Section 3 presents coordination patterns between multiple object life cycles that realize object-flow constraints from the activity diagram. Section 4 presents execution patterns between multiple object life cycles that realize control-flow constraints from the activity diagram. Section 5 discusses various aspects of the approach. Section 6 presents related work and Sect. 7 ends the paper with conclusions.

2 Preliminaries

We assume readers are familiar with UML activity diagrams [7] and UML statecharts [7]. Figure 1 shows an activity diagram of an ordering process that we refer to in the sequel.

2.1 Synthesizing Statecharts from Activity Diagrams

In previous work [10], we outlined an approach to synthesize a single object life cycle, expressed as a hierarchial UML statechart, from a business process model, expressed in a UML activity diagram. This paper builds upon that approach. To make the paper self-contained, we briefly summarize the main results here.

Input to the synthesis approach is an activity diagram with object nodes. Each object node references the same object but in a distinct state. An activity can have object nodes as input or output. An activity can only start if all its input object nodes are filled, and upon completion it fills all output object nodes [7]. If an activity has multiple input or output object nodes that reference the same object, the object is in multiple states at the same time; then the object life cycle contains parallelism. Object nodes are exclusive: if multiple activities require the same object node as input, only one activity can read (consume) the object node.

The synthesis approach consists of two steps. First, irrelevant nodes are filtered from the activity diagram. Object nodes are relevant, but activities are

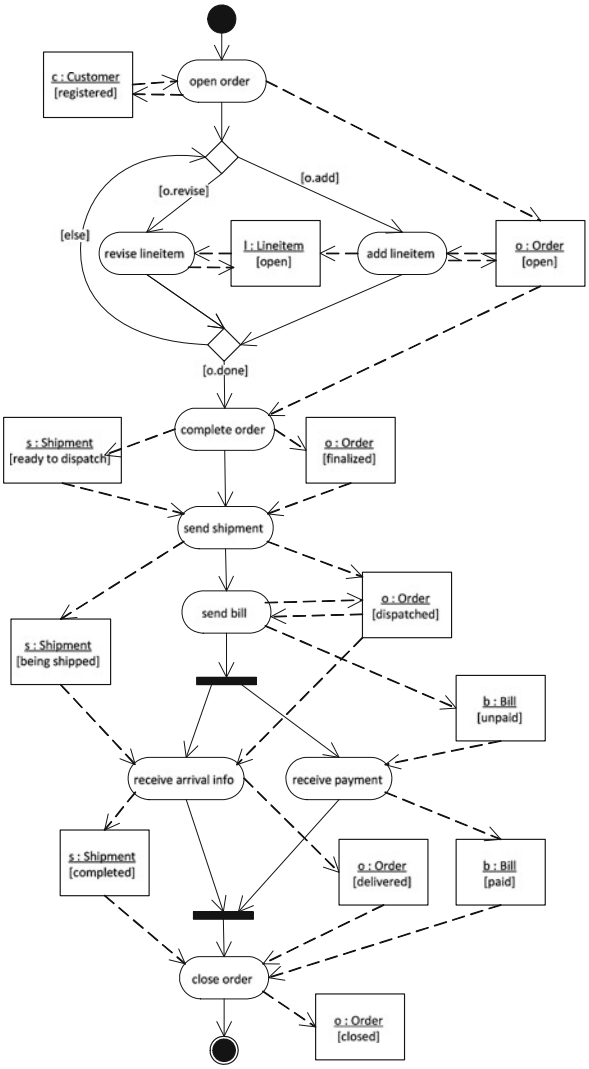

Fig. 1. Activity diagram of ordering process

not. Control nodes are only relevant if they influence the object nodes. Second, the filtered activity diagram is converted into a statechart by constructing a state hierarchy that ensures that the behavior of the statechart induced by the state hierarchy is equivalent to the behavior of the filtered activity diagram.

A limitation of that approach is that it assumes that an activity diagram references exactly one object. If an activity references multiple different objects, for each object a different version of the activity diagram specific to that object can be created. While this ensures that for each object a statechart skeleton can be created, the generated statechart skeletons are not yet executable, lacking

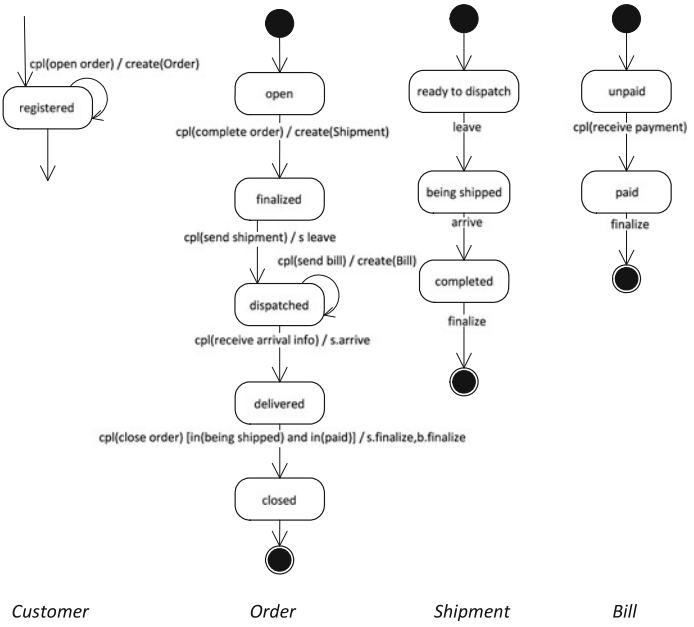

Fig. 2. Statecharts for a few ordering process artifacts

coordination and execution logic. This paper removes that limitation by defining coordination and execution patterns that materialize statechart skeletons synthesized using the earlier defined approach [10]. Figure 2 shows for some objects part of the object life cycles that are generated using the approach proposed in this paper. The earlier defined synthesis approach [10] is used to construct the skeletons of the life cycles, so the nodes and edges without labels. The approach defined in this paper adds annotation to the skeletons. Also, it refines some atomic states into compound states (not shown due to space limitations).

2.2 Terminology

To simplify the exposition, we introduce the following terminology for UML activity diagrams with objects. Let A be an activity (action node) and let $o{:}O[s]$ be an object node representing object o of class O where o is in state s. Node $:O$ represents an anonymous object of class O.

A *finalizes* $:O$ if there is an object flow from $:O$ to A but not from A to $:O$.
A *creates* $:O$ if there is an object flow from A to $:O$ but not from $:O$ to A.
A *accesses* $:O$ if there is an object flow from $:O$ to A and from A to $:O$. There are two kinds of *access*-relation:
  A *reads* $:O$ if there is an object flow from $:O[s]$ to A and from A to $O{:}[s]$.
  A *updates* $:O$ if there is an object flow from $:O[s]$ to A and from A to $O{:}[s']$ for $s \neq s'$.

3 Coordination Patterns

Coordination Patterns specify how different objects interact with each other. They derive from the object flows of an activity diagram. Coordination Patterns are not executable, since the patterns do not consider external event triggers. Section 4 presents executable patterns.

3.1 Roles

For each pattern, we identify two different roles for objects: coordinator and participant. Each activity that accesses an object has exactly one coordinator. Performing the activity typically causes a state change with the coordinator. This implies that the activity updates the coordinator. Any object that the activity updates can play the role of coordinator.

If A does not access any object, there is no coordinator. We consider this as a design error that can be detected automatically and fixed via additional user input. If A accesses multiple objects, the user has to decide which object is responsible for coordinating $:P$. Alternatively, from a process model with object flows a priority scheme on object classes can be automatically derived, for instance based on the dominance criterion [4]. The object class with the highest priority can be made the default coordinator of $:P$.

In the sequel, we present patterns for which we assume each activity has one coordinator, which is either derived automatically from the activity diagram or designated by the user.

3.2 Creation

Activity A creates an object of type P under coordination of object $:O$. Figure 3 specifies the creation pattern: Since $:O$ is coordinator, A changes the state of $:O$ from $S1$ to $S2$. When coordinator $:O$ moves from $S1$ to $S2$, an object of type P is created with action $create(P)$. Coordinator $:O$ moves state in task A, but task execution details are not considered for coordination patterns, only for the execution patterns.

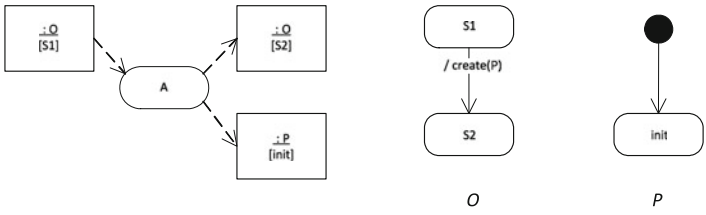

Fig. 3. Creation pattern

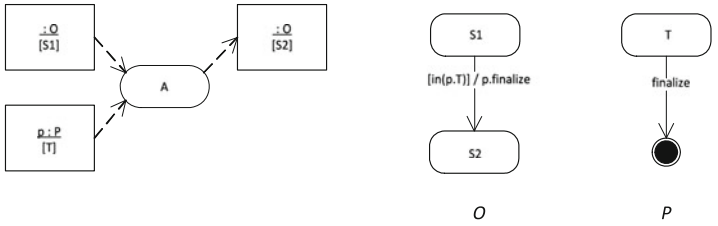

Fig. 4. Finalization pattern

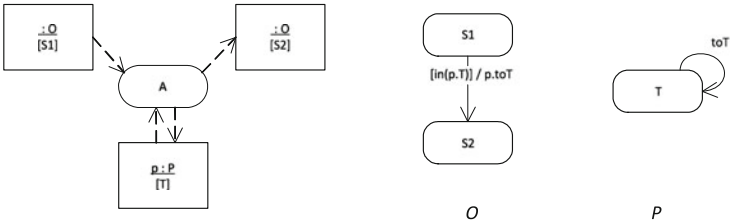

Fig. 5. Read-access pattern

3.3 Finalization

Activity A finalizes object $p{:}P$ by moving $:P$ into its end state; finalization does not mean that the object is destroyed, The finalization is realized (Fig. 4) by sending a special event $finalize$ to $p{:}P$ that moves the life cycle to the end state, provided $:P$ is indeed in the expected state $T1$. It might be that the life cycle has multiple end states; in that case, the other branches of the life cycle can still continue after this branch has been finalized.

3.4 Read-Access

If activity A reads object $p{:}P$, then $:P$ does not change state but is accessed. To model this, we use a self-loop from and to the state of $:P$ (Fig. 5) that is triggered an event from the coordinator, but only if the state of $:P$ is the precondition for A.

3.5 Update-Access

If activity A updates object $p{:}P$ under coordination of $:O$, then both $:O$ and $:P$ move to a new state. The state change of $:P$ is triggered by $:O$. Figure 6 shows that $:O$ generates an event that triggers $p{:}P$ to move to its next state, but only if $p{:}P$ is currently in the state that is precondition for A.

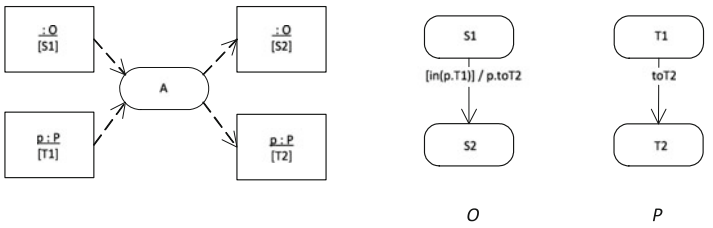

Fig. 6. Update-access pattern

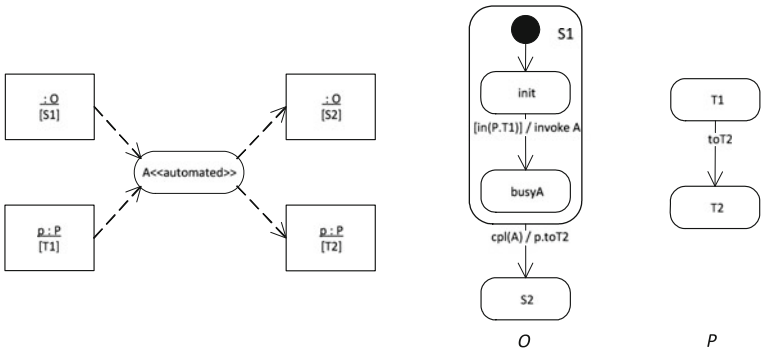

Fig. 7. Task pattern

4 Execution Patterns

Coordination Patterns only capture the object-flow constraints from activity diagrams. To capture the control-flow constraints, we use execution patterns based on control-flow constructs in activity diagrams.

4.1 Task

A task is invoked in an activity node. A typical distinction is between manual, automated, or semi-automated tasks. For this paper, we only consider automated tasks, but we plan to study other task types in future work. Figure 7 shows how a task invocation can be specified in object-centric design. The coordinator $:O$ is responsible for invoking task A; there its precondition state $S1$ is decomposed into two states, where *busy* A denotes that activity A is being executed. Upon completion, the coordinator moves to $S2$ and informs the other object $:P$ that it has to move to new state $T2$. In Fig. 7, the underlying coordination pattern is the update-access pattern, but the task pattern can be combined with any coordination pattern or none.

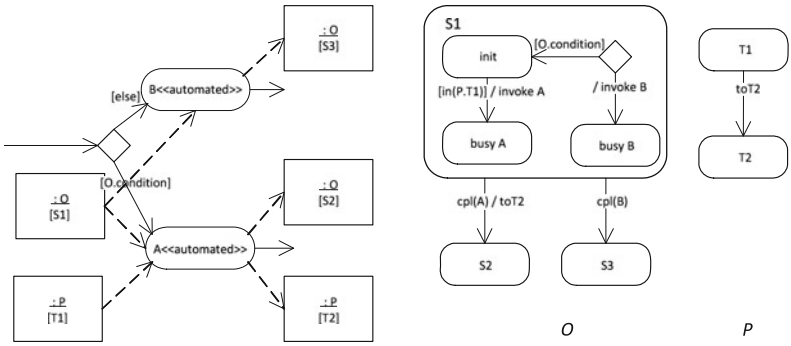

Fig. 8. Decision pattern

4.2 Decision

An object node can have multiple outgoing flows. This represents exclusive (choice) behavior: exactly one of the outgoing flows is taken if the object node is active. The actual decision is taken in the control flow, represented by a diamond.

Figure 8 shows the decision execution pattern that realizes a decision in an object-centric system. State $S1$ is precondition to both A and B; upon completion of either A or B object $:O$ moves to $S2$ or $S3$. As in the case of the task-pattern, the precondition state $S1$ is hierarchical. In this case, $S1$ contains the decision logic to decide between A or B; note that this decision logic comes from the control flow.

4.3 Merge

As in the previous case, an object node with multiple incoming edges represents exclusive behavior: if one of the edges is activated, the object node is entered. Again, the actual behavior is governed by control flow. The resulting merge pattern is symmetric to the decision pattern and omitted due to space limitations.

4.4 Fork

So far, we have seen only sequential state machines that do not contain any parallelism. However, an object can be in multiple states at the same time. Parallelism is created by an activity node that takes the object in a certain state as input and outputs the object in two distinct states, to the activity has two output object nodes that reference the same object. Since activity nodes activate all outgoing edges, both output object nodes are filled.

Figure 9 shows how the resulting fork pattern is specified. The state hierarchy is constructed using the approach we developed previously [10]. The two concurrent states model the two parallel branches started upon completion of A. The state hierarchy for $S2$ and $S3$ derives from the task pattern. The state hierarchy for $S1$ is not shown to simplify the exposition.

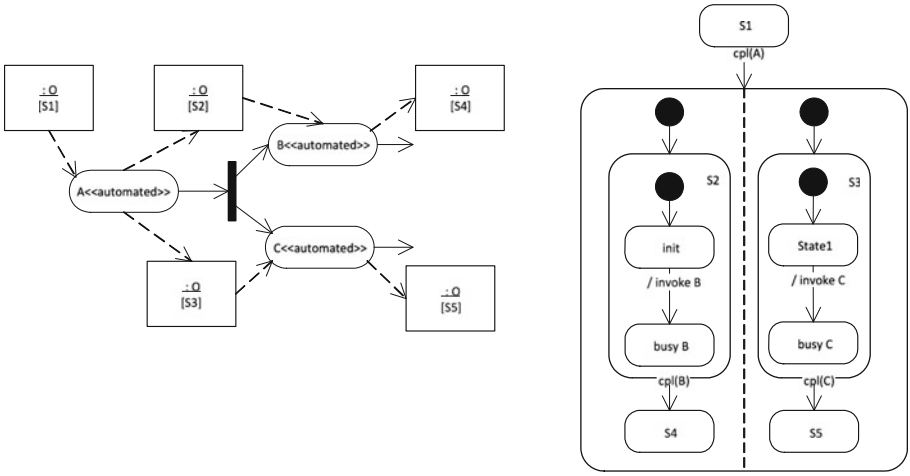

Fig. 9. Fork pattern

Note that if the object node with state $S1$ referred to another object, say $:P$, a create pattern would be present and no state hierarchy would be needed for the object life cycle of $:O$. In that case, the concurrency is expressed implicitly by having two object life cycles ($:O$ and $:P$) active at the same time.

4.5 Join

The join pattern is symmetric to the fork pattern (Fig. 10). Complicating factor is how to invoke the activity C that actually synchronizes the parallel branches. Using the task pattern, the parallel branches are only left if C completes. This implies that C needs to be invoked in one of the parallel branches, but only if the other branch is in the state that is precondition to C, i.e. $S4$. Which parallel branch is chosen to invoke C is arbitrary.

5 Discussion

Order example. To obtain the statecharts in Fig. 2 from the activity diagram in Fig. 1 all four coordination patterns are required. Next, the task, decision and merge patterns are used. Using these patterns introduces compound states which are not shown due to space limitations.

Multiple start states. If an object life cycle has multiple start states that are active in parallel, the synthesis approach will create multiple create actions. This results in multiple objects rather than a single object with parallelism.

We consider multiple start states as a design error: two parallel actions that create the same object $:O$ suggests that $:O$ is created synchronously while the two activities operate independently from one another. This error can be repaired

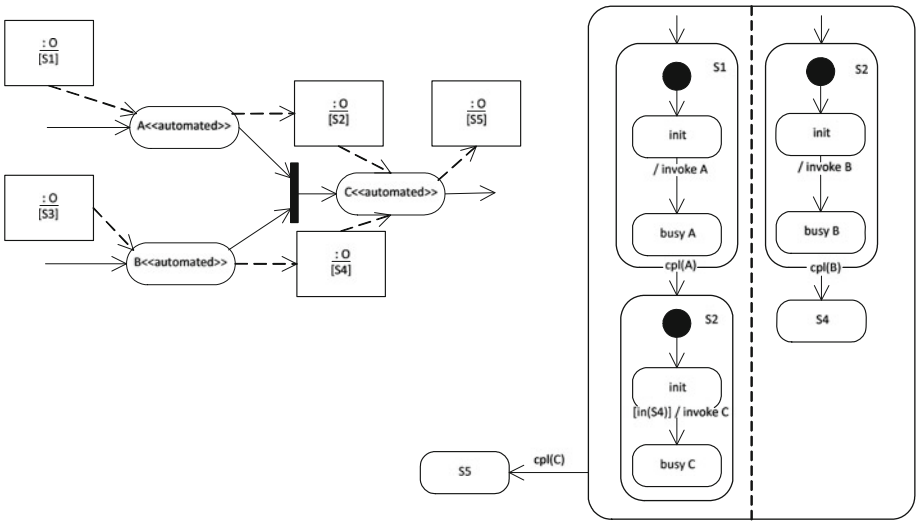

Fig. 10. Join pattern

by merging the activities that create the object. Another option is to insert an initial state that leads to the start states.

Refining. The object-centric design generated using the approach can be further refined, for instance to incorporate human-centric behavior. Suppose the company of the order process wishes to allow that a customer cancels a finalized order that has not yet been paid by rejecting bill. Extending the global process model of Fig. 1 results in a complex diagram with a lot of additional edges. In the object-centric design, only a few local changes are required: extending the life cycle of bill and order with additional cancelled states that can be reached if the cancel event occurs. Note the guard condition on the transition from delivered to closed in Fig. 2 prevents that a cancelled order is closed.

6 Related Work

As stated in the introduction, the last years a lot of research has been performed in the area of data-centric process modeling approaches such as business artifacts [1,4,5,11], case management [3,12], data-driven process models that are executable [2,9,13] and process models with data flow [14–16]. Sanz [17] surveys previous work on integrating the data and process perspective in the field of entity-relation modeling in connection to data-centric process modeling. This paper uses UML activity diagrams with object flows as data-centric process modeling notation.

More related to this paper are approaches that distinguish between process and data models and bridge the gap by deriving a process model that is coherent with a predefined data model [18,19] or object behavior model [20–22]. This

paper takes the opposite route: it considers a process model with data (object) flow and derives object behavior models that realize the process model.

Wahler and Küster [16] define an approach that resembles this paper most closely, and we therefore discuss this work in more detail. They too consider process models that manipulate stateful business objects, where each step in a process model can lead to a change in one or more business objects. The setting is a static set of predefined business objects that need to be "wired" together, where the process model is used to derive the wiring relation. They study how to design the wiring in such a way, by changing the process model, that the resulting wired object design has a low coupling. In contrast, this paper studies the problem of deriving an object-centric design from a process model with object flows. The problem is then defining the set of business objects and their behavior, which are both given in the approach of Wahler and Küster.

7 Conclusion

We have presented a semi-automated approach that synthesizes an object-centric system design from a business process model that references multiple objects. The approach distinguishes between coordination patterns that realize object-flow constraints and execution patterns for control-flow constraints. The patterns heavily use the state hierarchy for the object life cycles to establish a clear link with the process model constructs. The resulting object-centric design can be used to perform the process in a flexible way [22].

The approach is defined in the context of UML [7], but we plan to define a similar approach for BPMN [8], which supports a similar object flow notation as UML activity diagrams, though the BPMN semantics appears to be different.

We are currently implementing the patterns in a graph-transformation tool [23]. We plan to apply the prototype to different examples from the literature [16] and from student projects.

References

1. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3), 428–445 (2003)
2. Künzle, V., Reichert, M.: Philharmonicflows: towards a framework for object-aware process management. J. Softw. Maintenance **23**(4), 205–244 (2011)
3. Swenson, K.D.: Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done. Meghan-Kiffer Press, Tampa (2010)
4. Kumaran, S., Liu, R., Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
5. Yongchareon, S., Liu, C., Zhao, X.: An artifact-centric view-based approach to modeling inter-organizational business processes. In: Bouguettaya, A., Hauswirth, M., Liu, L. (eds.) WISE 2011. LNCS, vol. 6997, pp. 273–281. Springer, Heidelberg (2011)

6. Damaggio, E., Hull, R., Vaculín, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. Inf. Syst. **38**(4), 561–584 (2013)
7. UML Revision Taskforce: UML 2.3 Superstructure Specification. Object Management Group, OMG Document Number formal/2010-05-05 (2010)
8. White, S., et al.: Business Process Modeling Notation (BPMN) Specification, Version 1.1. Object Management Group. http://www.bpmn.org (2008)
9. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: A flexible, object-centric approach for business process modelling. SOCA **4**(3), 191–201 (2010)
10. Eshuis, R., Van Gorp, P.: Synthesizing object life cycles from business process models. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 307–320. Springer, Heidelberg (2012)
11. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
12. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2), 129–162 (2005)
13. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
14. Meyer, A., Weske, M.: Data support in process model abstraction. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 292–306. Springer, Heidelberg (2012)
15. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Inf. Syst. Res. **17**(4), 374–391 (2006)
16. Wahler, K., Küster, J.M.: Predicting coupling of object-centric business process implementations. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 148–163. Springer, Heidelberg (2008)
17. Sanz, J.L.C.: Entity-centric operations modeling for business process management - a multidisciplinary review of the state-of-the-art. In: Gao, J.Z., Lu, X., Younas, M., Zhu, H. (eds.): SOSE, pp. 152–163. IEEE (2011)
18. van Hee, K.M., Hidders, J., Houben, G.J., Paredaens, J., Thiran, P.: On the relationship between workflow models and document types. Inf. Syst. **34**(1), 178–208 (2009)
19. Reijers, H.A., Limam, S., van der Aalst, W.M.P.: Product-based workflow design. J. Manag. Inf. Syst. **20**(1), 229–262 (2003)
20. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Fagin, R. (ed.): ICDT, ACM International Conference Proceeding Series, vol. 361, pp. 225–238. ACM (2009)
21. Küster, J.M., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
22. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: Generating business process models from object behavior models. IS Manag. **25**(4), 319–331 (2008)
23. Van Gorp, P., Eshuis, R.: Transforming process models: executable rewrite rules versus a formalized Java program. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part II. LNCS, vol. 6395, pp. 258–272. Springer, Heidelberg (2010)