Variability in Artifact-Centric Process Modeling: The Hetero-Homogeneous Approach

Christoph Schütz **Michael Schrefl**

Department of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz, Austria
e-mail: {schuetz,schrefl}@dke.uni-linz.ac.at

Abstract

Today's dynamic business environment demands from companies variable and flexible processes. Rather than imposing a single fixed process, process models must account for the variability of real-world business problems. Many companies are hierarchically organized with top-down decision making processes. On the one hand, company policies and legal regulations often require compliance with standard process models prescribed by higher-level management. On the other hand, lower-level employees should be flexible within the prescribed boundaries. In this paper, we propose a hetero-homogeneous approach to modeling process variability. We employ the multilevel business artifact (MBA) in order to represent within a single object the homogeneous schema of an abstraction hierarchy of processes. We employ multilevel concretization for the introduction of heterogeneities into sub-hierarchies which comply with the homogeneous global schema.

Keywords: Business Artifact, Multilevel Modeling, Process Variability, Process Flexibility

1 Introduction

Business process models should reflect the variability of real-world business problems which are rarely solved by a single fixed process. Rather, different variations exist for tackling the same problem, depending on the exact situation. Thus, in order to accurately represent reality, a process model incorporates several variants for handling different situations. For example, a car rental company handles walk-in rentals differently from advance rentals, both being variants of a car rental.

An artifact-centric process model represents data along with the business processes that work with these data (Nigam & Caswell 2003, Kappel & Schrefl 1991). These business processes are represented as life cycle models of classes of data objects. It is common to model object life cycles with variants of finite state machines (Hull 2008). During its life cycle, a data object assumes several states which are defined in the life cycle model. Depending on the state, different operations may be performed on a data object.

In artifact-centric process modeling, behavior-consistent specialization of life cycle models allows for the representation of process variability. The specialization of a life cycle model may be considered a variant of the more general life cycle model. This variant must follow specific, well-defined rules in order to ensure consistency with the more general life cycle model (Stumptner & Schrefl 2000, Schrefl & Stumptner 2002, van der Aalst et al. 2002).

Many, if not most, companies are hierarchical organizations. From higher-level management to lower-level operatives, the different levels of the organization have their own business processes which are interconnected with each other. For example, top management decides what businesses a company operates in, area managers are concerned with shaping the businesses, and low-level operatives handle the specific business events. Multilevel process models represent processes at multiple levels of an organization together with the interactions between the processes at the different levels.

While higher-level management sets out general business policies, the exact processes may differ between the various subparts of the organization. Lower-level operatives must comply with the general policies but are flexible in adapting their respective processes within the limits specified by higher-level management. For example, the top management of a car rental company defines general policies for handling car rentals. The area managers for private and corporate renters may extend and refine these policies according to the particularities of each segment.

The hetero-homogeneous modeling approach provides modelers with increased flexibility for the representation of variability in multilevel process models while preserving the advantages of homogeneous schemas. Previously, the hetero-homogeneous modeling approach has been successfully employed in data warehouse modeling (Neumayr et al. 2010). A hetero-homogeneous business process model consists of a generally homogeneous schema but allows for the introduction of heterogeneities in well-defined sub-hierarchies. The extended and refined process models of the sub-hierarchies comply with the more general models and are themselves the homogeneous schema of their respective sub-hierarchy. The process models of sub-hierarchies of sub-hierarchies may again be extended and refined, and so on.

Figure 1 illustrates the hetero-homogeneous approach to modeling business process variability. Several multilevel artifact-centric process models (Rental, Private, Corporate, Rental2175) describe data and life cycle models at various hierarchically-ordered levels of abstraction (business, renterType, rentalAgreement, rental). Each process model consists of several boxes connected by dotted lines. Each box consists of several compartments, the top compartment containing, in angle brackets, the name of the level. The remain-

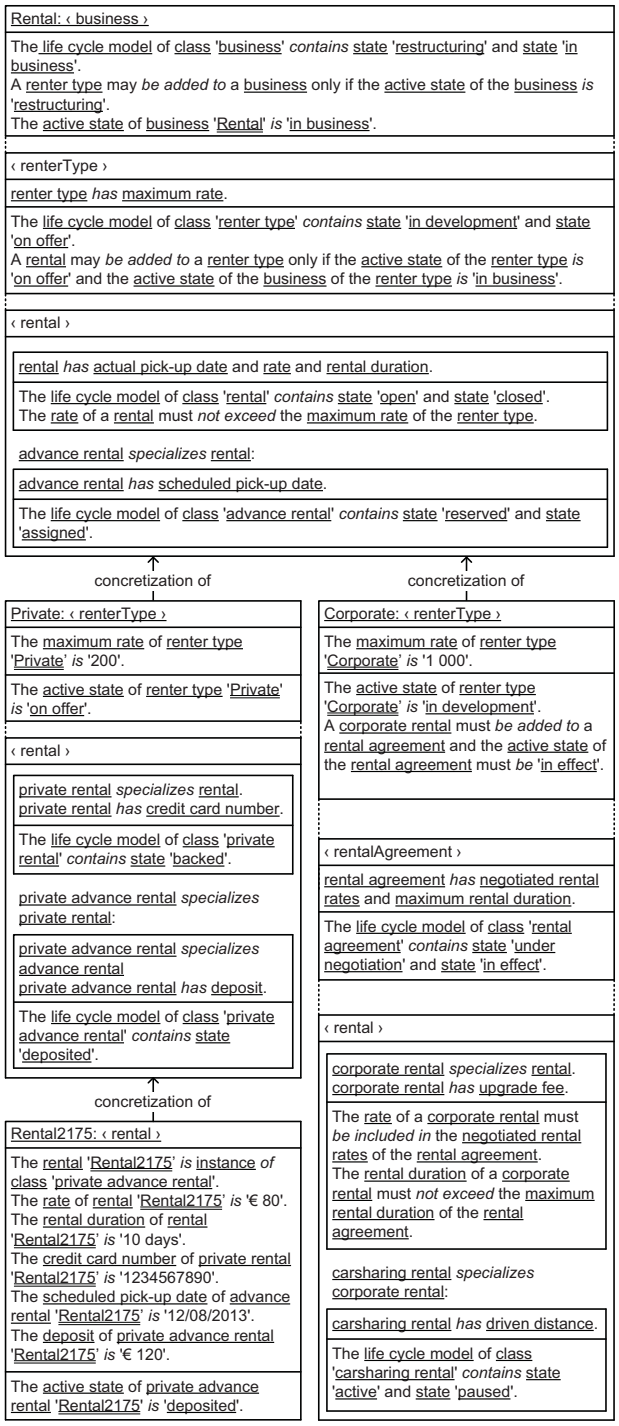


Figure 1: Modeling variability in business processes using the hetero-homogeneous approach

ing two compartments specify the data and life cycle model, respectively, at the particular level. The top compartment of the top box of each process model contains, underlined, the name of the process model together with the name of the top level, separated by colon. The name of a process model should be read in conjunction with the top level. For illustration purposes only, the data and life cycle models are represented in the style of the Semantics of Business Vocabulary and Rules (SBVR) standard; the example scenario is based on the EU-rent use case from the SBVR standard (OMG 2008, p. 267 et seq.).

Multilevel artifact-centric process models are hierarchically ordered through concretization relationships which allow for the introduction of variability in well-defined partitions of the hierarchical model. Each of the multilevel process models in Figure 1 is the homogeneous schema of an entire (sub-)hierarchy. For example, Rental represents the company's rental business itself, defining the homogeneous schema of the entire hierarchy, describing the data and life cycle models of the rental business as a whole, the different renter types, and the individual rentals. The multilevel process models Private and Corporate, both representing renter types, define a homogeneous schema of their respective sub-hierarchies while introducing heterogeneities with respect to Rental. The renter type Private has credit card information attached to the rental level and introduces the possibility of advance rentals. An individual private rental must be backed by a credit card; a private advance rental requires a deposit. The renter type Corporate has an additional level, rentalAgreement, which is only applicable to corporate renters. An individual corporate rental for the employee of a company is opened under the company's corporate rental agreement which governs the maximum rental duration and defines a set of pre-negotiated rental rates. At the rental level, Corporate introduces an upgrade fee and the possibility of carsharing rentals. The introduction of heterogeneities in the Corporate sub-hierarchy does not affect the schema of the Private sub-hierarchy, and vice versa.

Besides having variants of entire process hierarchies through concretization of multilevel process models, a single multilevel process model may define different process variants within an abstraction level. For example, in Figure 1, the Rental multilevel process model defines advance rentals as a variant of rentals. Different sub-hierarchies may specialize these variants or introduce additional variants. For example, the Private multilevel process model specializes the schema of rentals and advance rentals. The Corporate multilevel process model specializes the schema of rentals and introduces carsharing rentals as a variant of corporate rentals.

In this paper, we adopt and extend the multilevel business artifact (Schütz et al. 2013) for the hetero-homogeneous modeling of artifact-centric process models. The encapsulation of information about an entire hierarchy of artifact-centric process models within a single object together with a concretization mechanism allows for a flexible introduction of heterogeneities while preserving the advantages of homogeneous process models.

The remainder of this paper is organized as follows. In Section 2, we present the modeling and incremental evolution of hierarchies of multilevel process models. In Section 3, we present the modeling and incremental evolution of hierarchies of process models within the individual levels of a multilevel process model. In Section 4, we review related work. We conclude with a summary and an outlook on future work.

2 Variability in the Large: Hierarchies of Multilevel Process Models

A multilevel process model represents a hierarchy of business processes at different levels of abstraction. These business processes are interdependent and interact with each other. The encapsulation of multiple processes at different levels of abstraction within a single model allows for a definition of variants of entire process hierarchies.

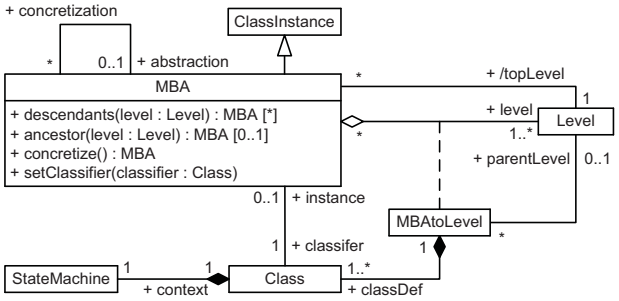


Figure 2: The MBA metamodel in UML

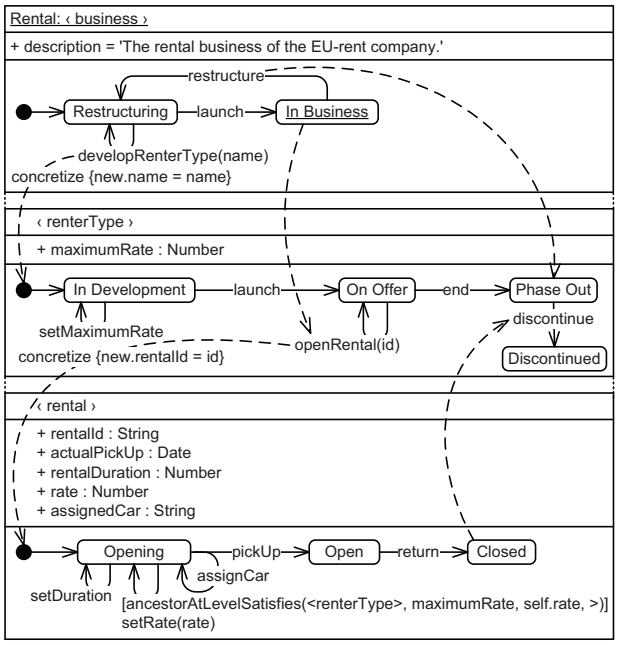


Figure 3: MBA Rental for the management of car rental data

2.1 Modeling

A *business artifact* (Nigam & Caswell 2003) encapsulates, within a single object, a data model as well as the corresponding life cycle model. Artifact-centric (or data-centric) process models focus on data and the operations that manipulate the data as well as their execution order. Many artifact-centric approaches to process modeling, for example, object/behavior diagrams (Kappel & Schrefl 1991), rely on variants of finite state machines for the representation of life cycle models, defining a set of states which an artifact runs through as the data change.

A *multilevel object* (m-object) encapsulates, within a single object, data models at various levels of abstraction (Neumayr et al. 2009). The abstraction levels are arranged in a hierarchy from most abstract to most concrete with a single top level which is the most abstract. To each abstraction level, an m-object links a class. The classes are related by aggregation relationships according to the level hierarchy. Modelers are free to choose the exact semantics of the aggregation relationships between the classes, the possibilities ranging from part-of relationships to aggregation with a materialization flavor (Dahchour et al. 2002). Besides defining classes at various levels of abstraction, an m-object instantiates its single top-level class, yielding a certain "class/object duality" (Atkinson & Kühne 2001) similar to clabjects.

The *multilevel business artifact* (MBA) is an extension of the m-object for artifact-centric process modeling (Schütz et al. 2013). An MBA encapsulates, within a single object, data and life cycle models at various levels of abstraction. To each abstraction level, an MBA links a class as well as the corresponding life cycle model. This life cycle model defines the legal execution order of the methods of the respective class. Being an instance of its top-level class, an MBA also has an active state (or several) from the top-level life cycle model.

Figure 2 defines the MBA metamodel using UML. Note, however, that MBAs are outside of traditional object-oriented thinking even though UML serves as the language for the definition of the MBA metamodel. The definition of the MBA metamodel in UML allows for the use of OCL constraints for the synchronization of life cycle models on different abstraction levels. Other modeling languages, for example, O-Telos (Jeusfeld et al. 2009), are equally well-suited for defining the MBA metamodel.

Class MBA is the metaclass of all MBAs. An MBA references several abstraction levels (Level) which are hierarchically ordered; an MBA has a single top level. For each level, an MBA defines a single class or a class hierarchy (see Section 3). Each class is only linked to a single MBA and level. Each class has a state machine which defines the life cycle model of the class. An MBA is also an instance of a class (ClassInstance), the classifier being a class that is linked to the MBA's top level. In order to ensure that each instance of an MBA's class is again an MBA, each of these classes is a specialization of the MBA metaclass. Each class has at most one direct instance, but may have several indirect instances via sub-classes which are defined by concretizations.

An MBA may be the concretization or, conversely, the abstraction of another MBA (see Section 2.2). An MBA inherits a set of levels from its abstraction. The concretization must maintain the relative order of inherited levels but may introduce additional levels between inherited levels. For simplicity of presentation, an MBA explicitly references the newly introduced as well as the inherited levels. The hierarchical order of levels is then defined locally for each MBA using an association class (MBAtoLevel). Each link between an MBA and a level references the parent level in the context of the MBA. In a similar way, class definitions are attached to these links.

The MBA metaclass defines methods for the navigation along the concretization hierarchy. Method descendants takes a level as parameter and returns the set of the MBA's (transitive) concretizations with the argument top level. Similarly, method ancestor takes a level as parameter and returns the MBA's (transitive) abstraction with the argument top level. Method concretization creates a new concretization underneath the MBA. The schema of the new concretization may then be specialized using reflective methods which are not shown in the metamodel. Method setClassifier allows for a change of the MBA's classifier during run time (see Section 3). We refer to previous work (Schütz et al. 2013, Neumayr et al. 2009) for a more formal definition of the integrity constraints in the MBA metamodel.

In the graphical representation (Figure 3), each level of an MBA is a box with several compartments in analogy to class diagrams in UML. The top compartment of each box contains the name of the respective level in angle brackets. The other compartments contain definitions of attributes, methods, and the life cycle model. The boxes of an MBA are arranged according to their hierarchical order and linked by

dotted lines. The top compartment of the top level's box contains the name of the MBA in addition to the level name, underlined and separated by a colon, reflecting the instantiation of the top-level class by the MBA. Furthermore, attributes of the top level's box have values assigned.

Figure 3 illustrates an MBA model for the management of car rental data with MBAs Rental and Corporate; the example is based on the EU-rent use case (OMG 2008, p. 267 et seq.). MBA Rental has levels business, renterType, and rental. The business level represents the company's rental business, consisting of several renter types (renterType), each having several individual rentals (rental) associated. The rental business has a description. A renter type has a maximum rate (maximumRate). A rental has an identifier (rentalId), an actual pickup date (actualPickUp), a rental duration (rentalDuration), a rental rate which determines the total rental fee, and an assigned car (assignedCar). MBA Corporate is a concretization of Rental (see Section 2.2).

We use UML (protocol) state machines (OMG 2011, p. 535 et seq.) for the representation of life cycle models. We stress, though, that the employed process modeling language is substitutable. We use the UML state machine formalism since it is an industry standard. A state machine is defined in the context of a class and consists of states and transitions between theses states. A transition has a source state and a target state and is linked to a call event for a method of the context class. A method may be called for a particular object if in the object's life cycle model there exists a transition that is linked to the called method and originates in an active state of the object. Furthermore, possibly specified pre- and post-conditions must be satisfied. A valid method call triggers the transition of the object from source state to target state. Methods that are not linked to any transition may be called in any state. A state may have several sub-states which are also linked by transitions. Forks and parallel regions allow for an object to be in multiple states simultaneously.

In the graphical representation of UML state machines, a rounded box with a caption inside represents a state, a filled black circle represents the initial state, and an arrow with a method name represents a transition. Pre-conditions and post-conditions of transitions are placed in square brackets before the method name and after the method name, respectively. The name of an object's active state is underlined (non-standard notation).

For example, each level of MBA Rental (Figure 3) has a state machine as life cycle model. A business is either Restructuring or In Business and moves between these states. A renterType moves from In Development to On Offer, Phase Out, and Discontinued. A rental moves from Opening to Open and Closed. Since Rental is also an instance of its top-level class it has an active state, In Business, from the top-level (business) life cycle model.

The life cycle models at the various abstraction levels of an MBA constitute the model of a *multilevel business process*. The different life cycle models of a multilevel business process are interdependent; the MBAs that instantiate the corresponding classes interact with each other. For example, a new renter type may only be added to a business while it is Restructuring. When a business moves from In Business to Restructuring all associated renter types move to the Phase Out state. Similarly, a new individual rental may only be added to a renter type while it is On Offer and the business is In Business. A renter type may only be discontinued if all associated individual

Attribute synchronization

allDescendantsAtLevelSatisfy(*level*, *attrName*, *value*, *θ*)
```
self.descendants(level)->forAll( o | o.attrName θ value )
```

someDescendantAtLevelSatisfies(*level*, *attrName*, *value*, *θ*)
```
self.descendants(level)->exists( o | o.attrName θ value )
```

isDescendantAtLevelSatisfies(*obj*, *level*, *attrName*, *value*, *θ*)
```
self.descendants(level)->includes(obj) and
  obj.attrName θ value
```

ancestorAtLevelSatisfies(*level*, *attrName*, *value*, *θ*)
```
self.ancestor(level).attrName θ value
```

State synchronization

allDescendantsAtLevelInState(*level*, *state*)
```
self.descendants(level)->forAll( o | o.oclInState(state) )
```

someDescendantAtLevelInState(*level*, *state*)
```
self.descendants(level)->exists( o | o.oclInState(state) )
```

isDescendantAtLevelInState(*obj*, *level*, *state*)
```
self.descendants(level)->includes(obj) and
  obj.oclInState(state)
```

ancestorAtLevelInState(*level*, *state*)
```
self.ancestor(level).oclInState(state)
```

Concretization

newDescendantAtLevel(*level*)
```
self.descendants(level)->exists( o | o.oclIsNew() )
```

newDescendantAtLevelSatisfies(*level*, *attrName*, *value*, *θ*)
```
self.descendants(level)->exists( o | o.oclIsNew()
 and o.attrName θ value )
```

newDescendantAtLevelUnder(*level*, *obj*)
```
self.descendants(level)->exists( o | o.oclIsNew()
 and obj.descendants(level)->includes(o) )
```

newDescendantAtLevelUnderSatisfies
(*level*, *obj*, *attrName*, *value*, *θ*)
```
self.descendants(level)->exists( o | o.oclIsNew() and
  obj.descendants(level)->includes(o) and o.attrName θ value )
```

Auxiliary

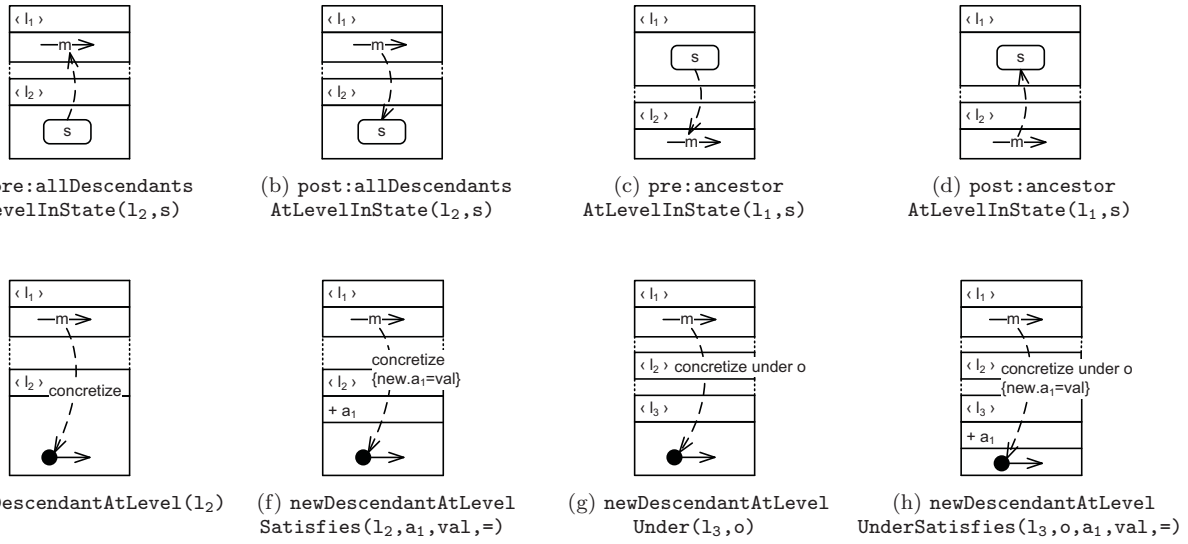
isDescendantAtLevel(*obj*, *level*)
```
self.descendants(level)->includes(obj)
```

Figure 4: Multilevel predicates for the definition of synchronization dependencies as macros for OCL

rentals are closed. A rental's rate must not exceed the maximum rental rate defined by the renter type. The life cycle models of an MBA are thus connected by *synchronization dependencies* which are pre- and post-conditions, expressed in OCL, for the transitions between states.

We define a set of frequently used patterns of synchronization dependencies between abstraction levels, called *multilevel predicates*, as syntax macros (Leavenworth 1966) for OCL (Figure 4). Multilevel predicates are classified into attribute synchronization, state synchronization, and concretization predicates. Attribute synchronization refers to pre- and post-conditions demanding that the value of a given attribute of descendants or an ancestor satisfies some condition. State synchronization refers to pre- and post-conditions demanding that descendants are or an ancestor is in a particular state. Concretization predicates trigger the creation of new MBAs and can be used only in post-conditions. Multilevel predicates are translated into standard OCL constraints which use the methods descendants and ancestor of the MBA metaclass for navigation along the level hierarchy.

The multilevel predicates allDescendantsAtLevelSatisfy, someDescendantAtLevelSatisfies, isDescendantAtLevelSatisfies, and ancestorAtLevelSatisfies handle attribute synchronization. Predicate allDescendantsAtLevelSatisfy demands that all descendants at a given level satisfy some condition over an at-



(a) `pre:allDescendants AtLevelInState(l2,s)` (b) `post:allDescendants AtLevelInState(l2,s)` (c) `pre:ancestor AtLevelInState(l1,s)` (d) `post:ancestor AtLevelInState(l1,s)`

(e) `newDescendantAtLevel(l2)` (f) `newDescendantAtLevel Satisfies(l2,a1,val,=)` (g) `newDescendantAtLevel Under(l3,o)` (h) `newDescendantAtLevel UnderSatisfies(l3,o,a1,val,=)`

Figure 5: Graphical notations for some of the state synchronization and concretization predicates in Figure 4, the someDescendantAtLevelInState and isDescendantAtLevelInState predicates are not shown

tribute. Predicate someDescendantAtLevelSatisfies demands that at least one descendant at a given level satisfies some condition over an attribute. Predicate isDescendantAtLevelSatisfies checks whether a given object is a descendant at a given level and satisfies some condition over an attribute. Predicate ancestorAtLevelSatisfies demands that the ancestor at a given level satisfies some condition over an attribute.

The multilevel predicates allDescendantsAtLevelInState, someDescendantAtLevelInState, isDescendantAtLevelInState, and ancestorAtLevelInState handle state synchronization. Predicate allDescendantsAtLevelInState demands that all descendants at a given level are in a particular state. Predicate someDescendantAtLevelInState demands that at least one descendant at a given level is in a particular state. Predicate isDescendantAtLevelInState demands that a given MBA is a descendant at a given level and the MBA is in a particular state. Predicate ancestorAtLevelInState demands that the ancestor at a given level is in a particular state.

The multilevel predicates newDescendantAtLevel, newDescendantAtLevelSatisfies, newDescendantAtLevelUnder, and newDescendantAtLevelUnderSatisfies are concretization predicates. Predicate newDescendantAtLevel demands that a new descendant at a given level exists after the execution of the method. Predicate newDescendantAtLevelSatisfies demands that a new descendant at a given level exists after the execution of the method and that this new descendant satisfies some condition over an attribute. Predicate newDescendantAtLevelUnder demands that a new descendant at a given level exists after the execution of the method and that this new descendant is also the descendant of a given other MBA. Predicate newDescendantAtLevelUnderSatisfies combines predicates newDescendantAtLevelSatisfies and newDescendantAtLevelUnder.

We provide graphical notations for state synchronization and concretization predicates (Figure 5). These kinds of synchronization are visualized by dashed arrows between states and transitions of different levels. Depending on the direction of the arrow, the synchronization dependency is either a pre-condition (Figures 5a and 5c) or a post-condition (Figures 5b, 5d, and 5e-5h) for a method call. The annotation of a dashed arrow with the symbol for existential quantification (∃, not shown) denotes the someDescendantAtLevelInState predicate. The isDescendantAtLevelInState state synchronization predicate does not have a graphical notation.

For example, in Figure 3, synchronization dependencies between levels are defined using the graphical notations for multilevel predicates. At the business level of MBA Renter, method restructure has as post-condition an allDescendantsAtLevelInState predicate with the Phase Out state of the renterType level as argument. Method developRenterType has as post-condition a newDescendantAtLevelSatisfies predicate with the renterType level as argument and a condition over the name attribute which every MBA has implicitly defined. At the renterType level, method openRental has as pre-condition an ancestorAtLevelInState predicate with the In Business state of the business level as argument. Method openRental has as post-condition a newDescendantAtLevelSatisfies predicate with the rental level as argument and a condition over the rentalId attribute. Method discontinue has as pre-condition an allDescendantsAtLevelInState predicate with the Closed state of the rental level as argument. At the rental level, method setRate has as pre-condition an ancestorAtLevelSatisfies predicate defining that the rate that is to be set must not exceed the value of the maximumRate attribute of the ancestor at the renterType level. The ancestorAtLevelSatisfies predicate has no special graphical notation.

2.2 Incremental Evolution

An MBA defines a multilevel process model and *multilevel concretization* allows for the definition of abstraction hierarchies of multilevel process models. Each MBA represents an entire multilevel abstraction hierarchy of artifact-centric process models. For this hierarchy, an MBA defines a homogeneous schema. Through multilevel concretization, modelers may extend and refine the homogeneous schema for a particular sub-hierarchy. The extended and refined schema becomes the homogeneous schema of the sub-hierarchy. For a sub-hierarchy of this sub-hierarchy, in an incremental, iterative manner, modelers may again extend and refine the homogeneous schema.

Multilevel concretization is a relationship between a concretizing MBA, referred to as the concretization, and a concretized MBA, referred to as the abstraction. Multilevel concretization combines charac-

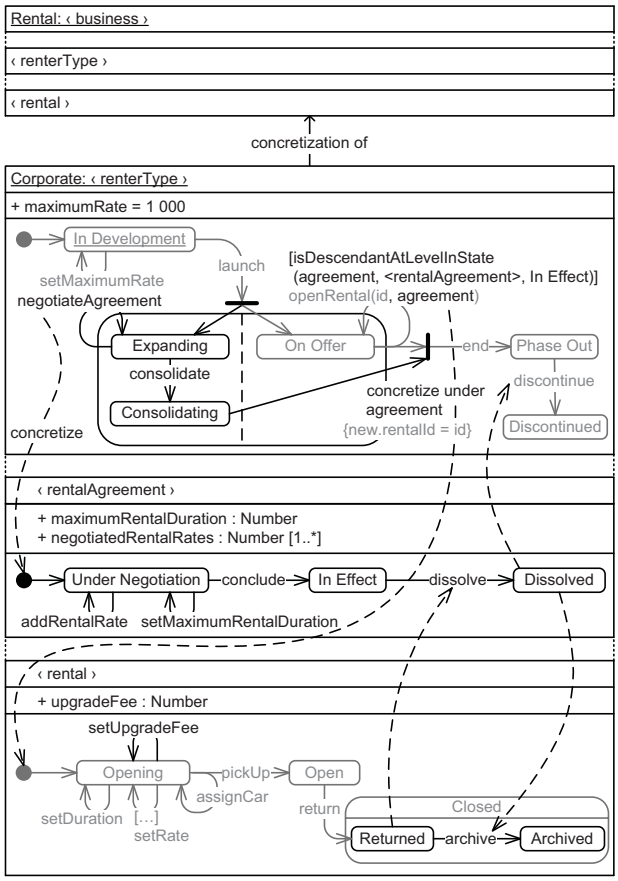


Figure 6: A concretization of MBA Rental in Figure 3

teristics of instantiation, aggregation, and specialization (Neumayr et al. 2009). This does not mean, however, that a concretization is instance of, part of, and specialization of the abstraction at the same time. Instantiation, aggregation, and specialization manifest in different aspects of concretization. Two MBAs that are in a concretization relationship are at different levels of abstraction. The concretization's top level is a second level of the abstraction. The concretization instantiates a class (and life cycle model) of the abstraction's top level. This instantiation relationship determines the concretization's membership in the aggregate represented by the abstraction. From the abstraction, the concretization inherits all levels from the concretization's top level downwards and including. The concretization preserves the relative order of the inherited levels. The concretization's classes and life cycle models that are linked to the inherited levels are specializations of the abstraction's classes and life cycle models at the respective levels. We refer to previous work (Schütz et al. 2013, Neumayr et al. 2009) for a more formal definition of concretization.

For example, in Figure 6, MBA Corporate at level renterType is a concretization of Rental. Renter type Corporate is part of the Rental business. MBA Corporate has top level renterType, the second level of Rental. MBA Corporate inherits from Rental all levels from renterType downwards and including, that is, levels renterType and rental. MBA Corporate specializes the class that is linked to the rental level of MBA Rental. A corporate rental has an upgrade fee (upgradeFee) which allows employees to upgrade the assigned car for a privately paid fee. The inherited attributes are not shown in the graphical representation. MBA Corporate is an instance of the class at the renterType level of MBA Rental, assigning a value of 1000 to attribute maximumRate. The active state of MBA Corporate is In Development. MBA Corporate also specializes the life cycle models of levels renterType and rental.

The concretization's life cycle models that are linked to inherited levels are observation-consistent specializations of the abstraction's life cycle models. Intuitively, observation consistency guarantees that if states and transitions added by the specialized life cycle model are ignored and refined states are considered unrefined, any processing of a data object according to the specialized life cycle model can be observed as correct processing from the point of view of the more general life cycle model (Schrefl & Stumptner 2002). An observation-consistent specialization of a life cycle model may extend the more general life cycle model with additional, parallel paths and refine inherited states with sub-states. Pre- and post-conditions in the specialized life cycle model must be at least as strong as in the more general model. The rules for observation-consistent specialization heavily depend on the employed modeling formalism. We refer to other work (Stumptner & Schrefl 2000, Schrefl & Stumptner 2002, van der Aalst et al. 2002) for a formal specification of these rules.

For example, MBA Corporate (Figure 6) extends and refines the inherited life cycle models of Rental. In the graphical representation, the inherited states and transitions are depicted in gray. At the renterType level, MBA Corporate extends the inherited life cycle model with a parallel region after the In Development state. Besides being On Offer, the Corporate renter type is, at the same time, also either in the Expanding or the Consolidating state. At the rental level, MBA Corporate extends and refines the inherited life cycle model. A corporate rental has an upgrade fee which may only be set when the corporate rental is in the Opening state. The Closed state is refined by sub-states Returned and Archived.

We permit the introduction of new transitions between inherited states in observation-consistent specializations. For example, at the rental level, MBA Corporate introduces a transition that is linked to the setUpgradeFee method, with the inherited Opening state as source and target. Depending on the modeling formalism, the introduction of transitions where either source or target state is inherited violates observation consistency (Schrefl & Stumptner 2002). For modeling formalisms that take into account the run time of methods, transitions may only be introduced between newly introduced states. Due to the run-to-completion assumption (OMG 2011, p. 574 et seq.) in UML, however, the introduction of transitions between inherited states may be considered observation-consistent. We stress, though, that the employed modeling formalism is not an important aspect of multilevel process models.

A concretization may also introduce additional levels with respect to the abstraction. A newly introduced level must be underneath the top level; the relative order of the inherited levels must be preserved. For example, MBA Corporate (Figure 6) introduces rentalAgreement between levels renterType and rental. A rental agreement is a contract which defines general conditions for individual rentals of a corporate client, specifying a maximum rental duration (maximumRentalDuration) and a set of rental rates (negotiatedRentalRates). While Under Negotiation, the maximum rental duration is determined and a set of rental rates is negotiated.

The observation-consistent specialization of synchronization dependencies is a particular case of specialization of pre- and post-conditions by strengthen-

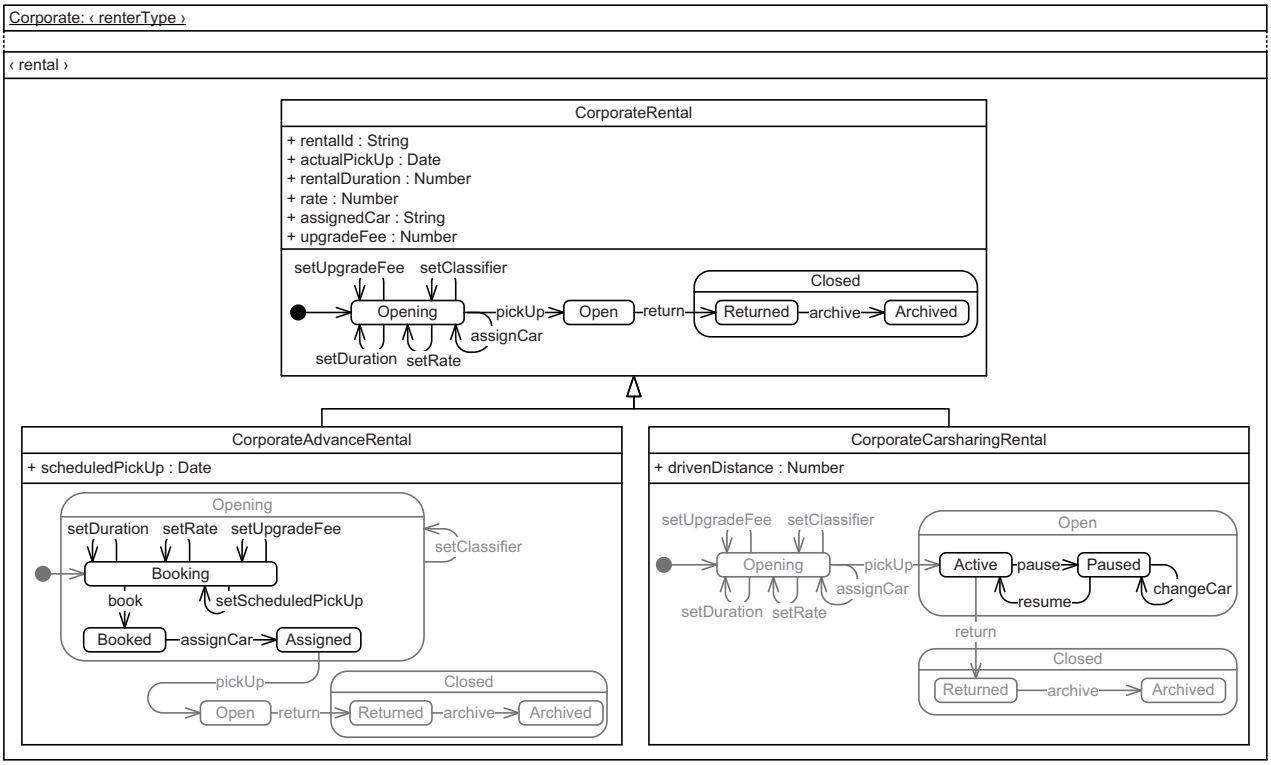


Figure 7: A class hierarchy within a single level of an MBA

ing. The inherited state synchronization must still be satisfied; the specialization may provide additional semantics. In particular, the concretization may refine a state synchronization by replacing the original state with a sub-state. In this case, however, the modeler must ensure that no deadlocks may occur due to this specialization. For example, at the renterType level, MBA Rental (Figure 3) defines a pre-condition for the discontinue method in the transition between Phase Out and Discontinued; a renter type may only be discontinued if all associated rentals are closed. This pre-condition is inherited by MBA Corporate which adds a transitive synchronization dependency to a sub-state of Closed at the rental level. In order to avoid possible deadlocks, the archive method has as pre-condition a state synchronization with the ancestor at the rental-Agreement level.

When introducing additional levels, modelers must specialize the concretization predicates. For example, at the renterType level of MBA Rental (Figure 3), the openRental method has a concretization predicate as post-condition, emphasizing the creation of a new concretization at the rental level by this method. MBA Corporate introduces an additional level and specializes this post-condition. A descendant of this renter type at the rental level must be added as a concretization underneath a rental agreement. The agreement must be a descendant of the renter type, which is emphasized by the additional pre-condition of the openRental method. As a modeler-defined constraint, the agreement must be In Effect.

3 Variability in the Small: Hierarchies of Process Models within Levels

For an abstraction level, a multilevel process model may define several variants. The selection of a variant for the process execution is deferred to a later point in time, thereby increasing the flexibility of employees.

3.1 Modeling

A multilevel business artifact (MBA) may link an entire specialization hierarchy of classes to an abstraction level, which allows for the definition of process variants within a single level. In this case, instead of a single class, an MBA defines a set of classes for the abstraction level. A single most general class serves as the superclass for an arbitrary number of specializations. The life cycle models of these classes follow the rules for behavior-consistent specialization. Thus, each class in such a specialization hierarchy together with the life cycle model is a variant of an artifact-centric process model.

For example, in Figure 7, MBA Corporate links an entire class hierarchy to the rental level. In this hierarchy, CorporateRental is the most general class, with CorporateAdvanceRental and Corporate-CarsharingRental being specializations. The life cycle model of class CorporateAdvanceRental refines the Opening state. An advance rental has a scheduled pick-up date (scheduledPickUp) and separates the recording of basic rental information (done in the Booking state) from the assignment of an actual car which is carried out when the rental is already Booked. The life cycle model of class CorporateCarsharingRental refines the Open state. A carsharing rental is billed by driven distance (drivenDistance) and may involve changes of the assigned car. The renter may pause an Active rental and choose another car from a car pool before resuming the rental.

After creation, an MBA is, by default, an instance of the single most general class that is linked to the top level. An MBA may then change its classifier during the life cycle. The setClassifier method of the MBA metaclass allows for the explicit consideration of classifier change in the life cycle model. This possibility of *incremental classification* defers a final instantiation decision, increasing the flexibility of employees.

Incremental classification allows for the dynamic specialization and generalization of an MBA's classifier. Instance specialization refers to a change of an MBA's classifier from more general to specialized. Instance generalization, in turn, refers to a change of an MBA's classifier from specialized to more general. Both types of incremental classification can be combined for *instance migration* which allows for a change of an MBA's classifier to another classifier that is a sub-class of a common super-class.

In order for instance specialization to be valid, certain conditions must be met by the MBA. A change of an MBA's classifier from more general to specialized is valid if the previous processing steps of the MBA in the more general life cycle model also represent a valid execution of the specialized life cycle model. In this case, the MBA can resume execution in the specialized life cycle model. For example, consider an MBA that is an instance of CorporateRental in the Opening state. A change of this MBA's classifier to CorporateAdvanceRental is possible and puts the MBA in the Booking state afterwards, a sub-state of Opening. Consider now an MBA that is an instance of CorporateRental and in the Open state. A change of this MBA's classifier to CorporateAdvanceRental is not allowed. The change of classifier would put the MBA in the Open state. As an instance of CorporateAdvanceRental the MBA would have had to run through the refined Opening state in order to present a valid life cycle. A change of classifier to CorporateCarsharingRental, however, is possible and puts the MBA in state Active, a sub-state of Open.

Instance generalization is always possible unless explicitly prohibited by the life cycle model. Values of attributes that are introduced by the specialized class are dropped. If in a refined state at first, after the change of classifier, the MBA is in the unrefined state of the general life cycle model. For example, consider an MBA that is an instance of CorporateAdvanceRental in the Assigned state. A change of this MBA's classifier to CorporateRental puts the MBA in the Opening state, the unrefined super-state of Assigned. The value of scheduledPickUp is dropped.

Instance migration refers to a change of an MBA's classifier to another sub-class of the MBA's current classifier's super-class; instance migration is realized as a sequence of instance generalization and specialization. For example, consider an MBA that is an instance of CorporateAdvanceRental. A change of this MBA's classifier from CorporateAdvanceRental to CorporateCarsharingRental is a two-step procedure. First, the classifier changes from CorporateAdvanceRental to the more general CorporateRental, the common superclass of CorporateAdvanceRental and CorporateCarsharingRental. Second, the classifier changes from CorporateRental to CorporateCarsharingRental.

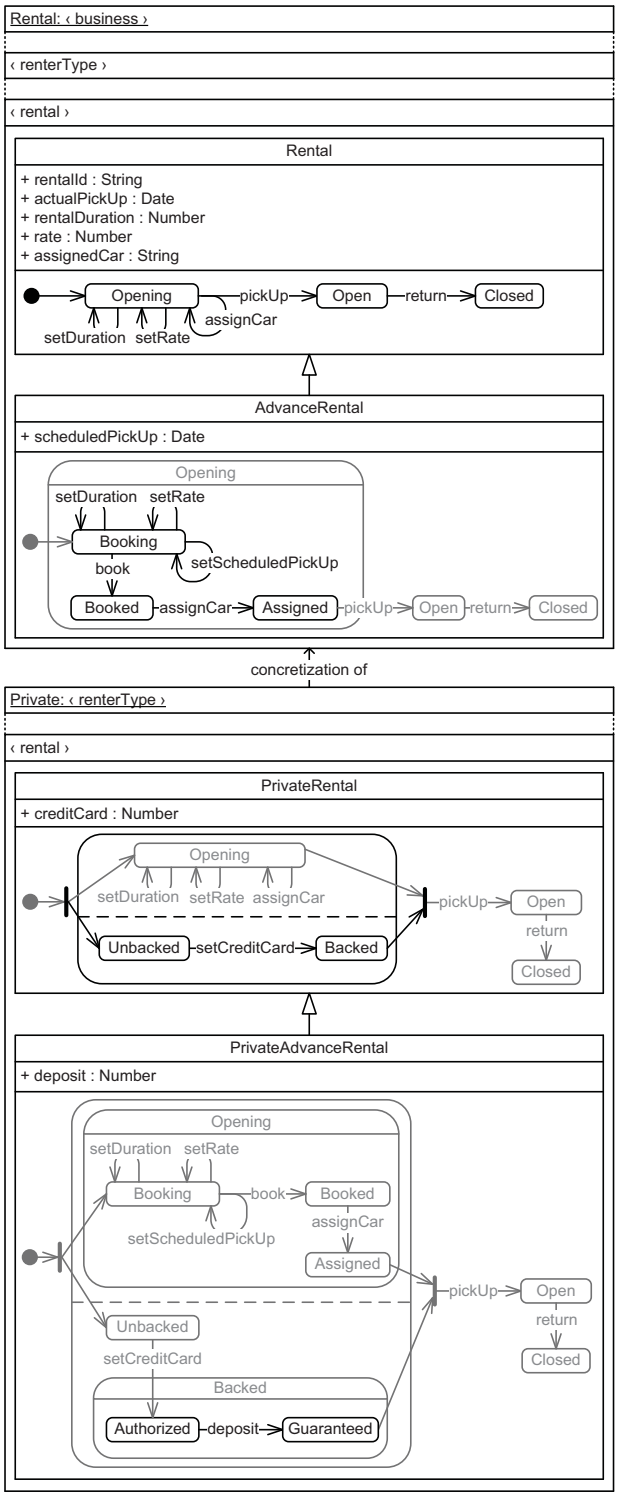


Figure 8: Specialization of an abstraction level's class hierarchy

3.2 Incremental Evolution

For each inherited level, a concretization inherits all of the linked classes and life cycle models. If an inherited level is linked to a class hierarchy, the concretization inherits the entire class hierarchy. This inherited class hierarchy may be specialized. On the one hand, the concretization may introduce additional sub-classes. On the other hand, the concretization may specialize only individual classes of the inherited class hierarchy.

With class hierarchies involved, multilevel concretization may lead to double specialization of classes and life cycle models. Each class of a concretization's inherited class hierarchy is a specialization of the abstraction's corresponding class. When the concretization adds additional features to one of the sub-classes of the inherited class hierarchy, this sub-class as well as its life cycle model must be consistent with both the abstraction's corresponding class and the superclass within the inherited class hierarchy. This superclass may also have additional features with respect to the abstraction's corresponding class. In this case, for the specialized sub-class in the concretization's inherited class hierarchy, double specialization occurs.

For example, in Figure 8, MBA Private is a concretization of MBA Rental which links a class hierarchy to the rental level. At the rental level, MBA Rental

defines classes Rental and AdvanceRental which are in a specialization/generalization relationship with each other. MBA Private, at the rental level, defines classes PrivateRental and PrivateAdvanceRental. Class PrivateRental is a specialization of class Rental which is defined by MBA Rental. A private rental must have credit card information and is either Unbacked or Backed, depending on the availability of credit card information. Class PrivateAdvanceRental is a specialization of class PrivateRental as well as class AdvanceRental which is defined by MBA Rental. A private advance rental requires the customer to deposit an amount of money in order to guarantee the reservation. Once deposited, a private advance rental turns from Authorized into Guaranteed, thereby refining the Backed state.

In this paper, we do not focus on the details of behavior consistency under double specialization. We provide, however, two modeling guidelines for the realization of observation-consistent double specialization. These guidelines simplify consistency checking under double specialization but restrict the freedom of the modeler. General consistency checking under double specialization is an open issue for future work. The issue of double specialization in process modeling is not the main issue in the hetero-homogeneous approach.

In order to avoid double specialization, a modeler may choose to specialize only the leaf nodes of a class hierarchy. In this case, behavior consistency must only be checked against the life cycle model of the super-class in the inherited class hierarchy. This simplification, however, limits the freedom of the modeler and reduces flexibility. Thus, it is desirable to allow double specialization of life cycle models.

Double specialization of life cycle models bears no conflict if the specializations occur in parallel regions or independent states of the life cycle model. For example, the specialization of the Rental class's life cycle model by the AdvanceRental class (Figure 8) is a refinement of the Opening state. The specialization of the Rental class's life cycle model by the PrivateRental class is an extension with a region that is parallel to the Opening state. These specializations are independent from each other. A combination of the two life cycle models in the PrivateAdvanceRental class's life cycle model is without problems.

4 Related Work

Just like a multilevel object (m-object), a multilevel business artifact (MBA) is very similar to a powertype. Powertypes present metamodeling capabilities (Odell 1998, p.28). The instances of a powertype are subtypes of another object type (Gonzalez-Perez & Henderson-Sellers 2006). Using the notion of "clabject", the instances of a powertype are class and object at the same time. Relating the MBA approach to powertype-based approaches (Eriksson et al. 2013), a level of an MBA may act both as partitioned type and powertype; in an MBA's level hierarchy, a parent level is the powertype of the child level.

MBAs (and m-objects) also present characteristics of deep instantiation and materialization. Deep instantiation (Atkinson & Kühne 2001) facilitates the modeling of arbitrary-depth instantiation hierarchies where data objects can instantiate (certain aspects of) other data objects which instantiate other data objects, and so on. Materialization (Dahchour et al. 2002), on the other hand, blurs the boundaries between aggregation and instantiation. In this respect, multilevel concretization is similar to materialization. Neumayr et al. (2011) provide a comprehensive comparison between m-objects and other multilevel modeling techniques.

In the context of business processes, model abstraction commonly refers to the hiding of unnecessary details from the user (Smirnov et al. 2012). The guard-stage-milestone (Hull et al. 2010) approach, for example, introduces this kind of abstraction for business artifacts. The issue of process model abstraction in the traditional sense is orthogonal to multilevel process modeling with MBAs. The MBA represents interdependent processes of objects at various levels of abstraction which are in some sort of aggregation relationship. A process space (Motahari-Nezhad et al. 2011), on the other hand, provides several views on the same business process, each view with the emphasis on a different aspect of the process. For example, the CEO of a company has a more high-level view on the sales process than the sales manager.

Many business process modeling approaches account for the variability of real-world processes. Configurable business process models (La Rosa et al. 2011, La Rosa 2009) incorporate several variants of a process which provide the process owners with different options. Business process families (Gröner et al. 2013) introduce the well-known principle of software product lines to business process modeling. A business process family comprises a reference model and a set of features which adhere to the core intended behavior specified by the reference model. Process owners may customize the business process by using different selections of features. Case handling (van der Aalst et al. 2005) provides the process owners with a choice of options, offering a great deal of flexibility.

The MBA approach employs behavior-consistent specialization of life cycle models for the representation of variability. A behavior-consistent specialization may be regarded as a variant of the more general life cycle model. Many frameworks for behavior-consistent specialization exist using various modeling languages, for example, Petri nets (van der Aalst et al. 2002), UML state machines (Stumptner & Schrefl 2000), or object/behavior diagrams (Schrefl & Stumptner 2002). In recent work, Yongchareon et al. (2012) investigate the observation-consistent specialization of synchronization dependencies.

Related to the issue of variability are the notions of flexibility and agility in business process modeling. Reichert & Weber (2012) provide a comprehensive treatment of flexibility in business process modeling. Milanovic et al. (2011) identify a set of rule patterns which can be used for the modeling of agile business processes. Liu et al. (2012) propose an integration of reflective operations into the process model in order to explicitly account for the manipulation of a business artifact's schema. The MBA approach provides flexibility through incremental classification. MBAs could also incorporate reflective operations for schema manipulation. Rules for behavior-consistent specialization constrain the possibilities of on-the-fly schema manipulation.

5 Summary and Future Work

The hierarchical organization is arguably the predominant organizational structure among large companies. A hierarchical organization often has rigid top-down decision-making processes. Today's dynamic business environment, however, demands increased flexibility from companies. The hetero-homogeneous modeling approach overcomes the dichotomy between the rigidity imposed by a hierarchical organization

and the flexibility that is required in a dynamic business environment. Future work will integrate the hetero-homogeneous approach into existing modeling languages and tools, for example, BPMN or the guard-stage-milestone model for business artifacts.

References

Atkinson, C. & Kühne, T. (2001), The essence of multilevel metamodeling, *in* M. Gogolla & C. Kobryn, eds, 'UML 2001', Vol. 2185 of *LNCS*, Springer, Heidelberg, pp. 19–33.

Dahchour, M., Pirotte, A. & Zimányi, E. (2002), 'Materialization and its metaclass implementation', *IEEE Transactions on Knowledge and Data Engineering* **14**(5), 1078–1094.

Eriksson, O., Henderson-Sellers, B. & Ågerfalk, P. J. (2013), 'Ontological and linguistic metamodelling revisited: A language use approach', *Information and Software Technology* **55**(12), 2099 – 2124.

Gonzalez-Perez, C. & Henderson-Sellers, B. (2006), 'A powertype-based metamodelling framework', *Software & System Modeling* **5**(1), 72–90.

Gröner, G., Boskovic, M., Silva Parreiras, F. & Gasevic, D. (2013), 'Modeling and validation of business process families', *Information Systems* **38**(5), 709–726.

Hull, R. (2008), Artifact-centric business process models: Brief survey of research results and challenges, *in* R. Meersman & Z. Tari, eds, 'OTM 2008, Part II', Vol. 5332 of *LNCS*, Springer, Heidelberg, pp. 1152–1163.

Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F. T., Hobson, S., Linehan, M. H., Maradugu, S., Nigam, A., Sukaviriya, P. & Vaculín, R. (2010), Introducing the guard-stage-milestone approach for specifying business entity lifecycles, *in* M. Bravetti & T. Bultan, eds, 'WS-FM 2010', Vol. 6551 of *LNCS*, Springer, Heidelberg, pp. 1–24.

Jeusfeld, M., Jarke, M. & Mylopoulos, J. (2009), *Metamodeling for method engineering*, MIT Press.

Kappel, G. & Schrefl, M. (1991), Object/behavior diagrams, *in* '7th International Conference on Data Engineering', pp. 530–539.

La Rosa, M. (2009), Managing variability in process-aware information systems, PhD thesis, Queensland University of Technology, Brisbane, Australia.

La Rosa, M., Dumas, M., ter Hofstede, A. H. & Mendling, J. (2011), 'Configurable multi-perspective business process models', *Information Systems* **36**(2), 313–340.

Leavenworth, B. M. (1966), 'Syntax macros and extended translation', *Communications of the ACM* **9**(11), 790–793.

Liu, E., Wu, F. Y., Pinel, F. & Shan, Z. (2012), A two-tier data-centric framework for flexible business process management, *in* '18th Americas Conference on Information Systems', Association for Information Systems.

Milanovic, M., Gasevic, D. & Rocha, L. (2011), Modeling flexible business processes with business rule patterns, IEEE Computer Society, pp. 65–74.

Motahari-Nezhad, H. R., Benatallah, B., Casati, F. & Saint-Paul, R. (2011), 'From business processes to process spaces', *IEEE Internet Computing* **15**(1), 22–30.

Neumayr, B., Grün, K. & Schrefl, M. (2009), Multi-level domain modeling with m-objects and m-relationships, *in* '6th Asia-Pacific Conference on Conceptual Modelling', Australian Computer Society, Darlinghurst, pp. 107–116.

Neumayr, B., Schrefl, M. & Thalheim, B. (2010), Hetero-homogeneous hierarchies in data warehouses, *in* '7th Asia-Pacific Conference on Conceptual Modelling', Australian Computer Society, Darlinghurst, pp. 61–70.

Neumayr, B., Schrefl, M. & Thalheim, B. (2011), Modeling techniques for multi-level abstraction, *in* R. Kaschek & L. M. L. Delcambre, eds, 'The Evolution of Conceptual Modeling', Vol. 6520 of *LNCS*, Springer, Heidelberg, pp. 68–92.

Nigam, A. & Caswell, N. S. (2003), 'Business artifacts: An approach to operational specification', *IBM Systems Journal* **42**(3), 428–445.

Odell, J. (1998), *Advanced object-oriented analysis and design using UML*, Cambridge University Press, chapter Power types, pp. 23–32.

OMG (2008), *Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.0*. http://www.omg.org/spec/SBVR/1.0/.

OMG (2011), *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*. http://www.omg.org/spec/UML/2.4.1/.

Reichert, M. & Weber, B. (2012), *Enabling Flexibility in Process-Aware Information Systems – Challenges, Methods, Technologies*, Springer, Heidelberg.

Schrefl, M. & Stumptner, M. (2002), 'Behavior-consistent specialization of object life cycles', *ACM Transactions on Software Engineering and Methodology* **11**(1), 92–148.

Schütz, C., Delcambre, L. & Schrefl, M. (2013), Multilevel business artifacts, *in* M. La Rosa & P. Soffer, eds, 'BPM 2012 Workshops', Vol. 132 of *LNBIP*, Springer, Heidelberg, pp. 304–315.

Smirnov, S., Reijers, H. A., Weske, M. & Nugteren, T. (2012), 'Business process model abstraction: a definition, catalog, and survey', *Distributed and Parallel Databases* **30**(1), 63–99.

Stumptner, M. & Schrefl, M. (2000), Behavior consistent inheritance in UML, *in* A. Laender, S. Liddle & V. Storey, eds, 'ER 2000', Vol. 1920 of *LNCS*, Springer, Heidelberg, pp. 451–530.

van der Aalst, W. M. P., van Hee, K. M. & van der Toorn, R. A. (2002), 'Component-based software architectures: a framework based on inheritance of behavior', *Science of Computer Programming* **42**(2-3), 129–171.

van der Aalst, W., Weske, M. & Grünbauer, D. (2005), 'Case handling: A new paradigm for business process support', *Data & Knowledge Engineering* **53**(2), 129–162.

Yongchareon, S., Liu, C. & Zhao, X. (2012), A framework for behavior-consistent specialization of artifact-centric business processes, *in* A. Barros, A. Gal & E. Kindler, eds, 'BPM 2012', Vol. 7481 of *LNCS*, Springer, Heidelberg, pp. 285–301.