The Role of BPMN in a Modeling Methodology for Dynamic Process Solutions

Jana Koehler

IBM Research - Zurich, 8803 Rüschlikon, Switzerland

Abstract. This paper introduces a design method for dynamic business process management solutions in which the well-known modeling elements of *business object life cycles*, *business rules*, and *business activities* are integrated in a distributed system as equal communicating components. Using the EURENT car rental domain originally developed by the business rules community, it is demonstrated how this method can be used to enable adhoc and rule-driven activities integrated with the life cycle management of business objects. A modeling methodology based on BPMN collaboration diagrams is proposed to describe component interactions and behavior. Agile principles are applicable to incrementally build the solution in which scenarios play a major role to validate and further evolve the solution's behavior. A clear separation between components, their interaction, and details of the internal component behavior facilitates change and the implementation of business patterns.

1 Towards Dynamic Process Solutions

Over the past years, the Business Process Management (BPM) community has been increasingly discussing more flexible BPM solutions that have a stronger focus on data and offer simple, yet powerful ways to integrate business rules. For example, the authors of [1] discuss case handling as a paradigm for dynamic BPM solutions and identify the following requirements:

- Avoid context tunneling and provide all information relevant to a case as needed, not just the information subset required to execute an activity within a predefined process flow.
- Decide which activities are enabled on the basis of the available information rather than enforcing a specific activity flow.
- Separate work distribution from authorization instead of using activity-based routing as a single mechanism to simultaneously address both requirements.

In addition, a dynamic BPM solution must support actor-initiated activities, which means that the human participants in a business process should be able to decide when to perform a business function. However, business conduct as captured in business rules should always be followed and monitored. Business rules should thus not only be used as automated decisions as it is common today, but they should primarily serve to establish obligations for actors to perform activities within the boundaries of appropriate business conduct.

This paper proposes a modeling methodology that addresses the above-mentioned requirements. It proposes a shift from the explicit modeling of predefined end-to-end processes to an agile design approach where the business capabilities (*activities*) of actors, the *business rules* that initiate and govern actor behavior, and the *life cycles* of the main *business objects* move into the primary focus of attention.

Activities, rules, and life cycles are considered as three types of equal communicating components in a distributed system. None of the three component types is subordinate to the other; they interact via message exchanges with each component type having different capabilities. Whereas activities, business rules, and object life cycles are well-known modeling elements, their equal interaction and uniform representation has not been investigated so far. The paper explores the consequences of such an *integrated approach* for a BPM solution.

Instead of proposing a new modeling language, the approach relies on BPMN 2.0 [2] and introduces a modeling methodology using BPMN collaboration diagrams to specify a dynamic process solution. By using BPMN, a uniform representation of activity, rule, and object life cycle components is achieved, which integrates all functional aspects of a BPM solution in a seamless manner. As BPMN 2.0 strives for a new level of integrating business-user-friendly modeling with direct model execution, the immediate exploration and simulation of process scenarios during modeling will help to build dynamic process solutions of high quality.

The paper uses a subset of the EURENT domain that models the typical operations of the branch office of a car rental company containing many adhoc activities and more than 100 business rules [3, 4]. To validate the proposed methodology, parts of the domain were implemented using the process engine ePVM [5]. As the methodology is independent of a particular runtime engine, the discussion in this paper focuses on the conceptual pillars and abstracts from implementation details.

The paper is organized as follows: Section 2 introduces the main idea and shows examples of the three component types business rules, object life cycles, and actors in the EURENT domain. In Section 3, a 3-layer architecture as the foundation of the dynamic BPM solution is discussed, whereas Section 4 introduces principles of component interaction and BPMN collaboration diagrams as a means to specify component communication and behavior. Section 5 identifies next steps in research to further mature the proposed approach. Section 6 summarizes related work and Section 7 concludes.

2 Actors, Rules, and Object Life Cycles as Equal Communicating Components

The basic idea of the proposed modeling methodology is simple. Instead of modeling a process by capturing the flow of activities, refining this flow with data, adding roles to activities, and eventually refining decisions with business rules, the focus shifts to actors and their business capabilities, business objects capturing the main data and possessing a life cycle, as well as business rules establishing obligations for actors. Actors, object life cycles, and business rules form three types of communicating components where none has control over the other. The interaction of the components using messages creates the end-to-end process.

Actors have certain capabilities and perform business functions by acting in a particular role. These capabilities are modeled as BPMN activities. For example, in the EURENT domain, three main roles possessing numerous activities can be distinguished:

– The **branch manager**, who oversees a particular branch office of the rental car company, is responsible to *buy*, *sell*, or *transfer cars*, *keep the contact to the police* in case of missed cars, and *bar customers* who misbehaved when renting a car.
– The **car park assistant** oversees all technical operations on the car fleet, including activities to *service cars* and *receive returned cars*.
– The **front desk clerk** interacts with the customer. For example, she will *hand over the keys and the contract* when the customer arrives to pick up the car, she might *grant a contract extension* upon a call of a customer, which may well interrupt the *registration of a new customer* who waits at the counter.

Figure 1 shows some of the activities of the front desk clerk in a BPMN adhoc process. It is obvious that it would not make sense to establish a predefined control-flow between these activities as it could never capture all reasonable process instances.

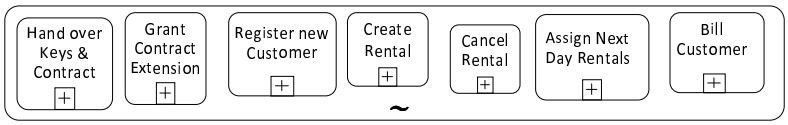

Fig. 1. Capabilities of the front desk clerk actor as activities in a BPMN adhoc process.

Business objects represent the main information objects of a business process. Identifying business objects is an established modeling step when designing a BPM solution. The EURENT domain contains very detailed data models from which the main business objects can be extracted, such as the *rental contract*, the *customer record*, and the *rental car*. Business rules also provide important information about business objects. For example, the structural rule *"a Customer has at least one of the following: a Rental Reservation, an in-progress Rental, a Rental completed in the past 5 years"* provides further insight into the concept of the customer record and suggests possible subtypes or states of rental contracts that should be distinguished.

The methodology discussed in this paper associates a state machine with each main business object that represents its life cycle. The methodology proposes that the object life cycle controls any create, read, delete, and update (CRUD) operations on the business objects as it is common in object-oriented programming encapsulation. The encapsulation is extended by making the life cycle explicit as a communicating component, which implies that business objects cannot be claimed as resources under possession by an actor, who would then be sure that executing an activity changes the state of the object. Instead, an actor can only send a message to an object requesting the object to change state and has to implement its own error-handling behavior in case the request is not successful. Business logic for the manipulation of business objects is decomposed along valid state transitions of the life cycle and represented in BPMN collaboration diagrams, see Section 4.

State information can be extracted from the EURENT domain models, but it is also worth noting that business rules are an important source for further detailing a life cycle. For example, the rules *"a car from another branch may be allocated, if there is a suitable car available and there is time to transfer it to the pick-up branch"* and *"if a car is three days overdue and the customer has not arranged an extension, insurance cover lapses and the police must be informed"* reveal three important states of the car business object. A possible life cycle for the car business object is shown in Figure 2(a).

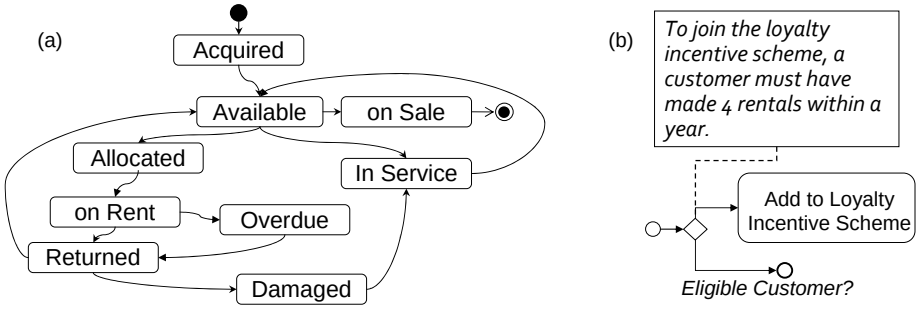

Fig. 2. Lifecycle of a rental car and example of a business rule automating a decision in a process.

More than 100 business rules were extracted from the EURENT domain documents that can be grouped into *structural* and *operative* rules. Structural rules capture necessary characteristics of business objects and are primarily used to ensure data integrity throughout the business processes. An example is shown in Figure 2(b) where the rule defines the necessary condition for a customer to be eligible for a bonus program. Operative rules formulate obligations and govern the conduct of business activity. They focus directly on the propriety of conduct in circumstances where willful or uninformed business activity can fall outside the boundaries of behavior deemed acceptable [3] and are thus of particular importance to the methodology proposed in this paper. The proposed methodology is independent of a particular rules language, but encapsulates rules into communicating components, which allows a rule to receive events of interest and initiate activities or changes on business objects by sending messages to actors and business object life cycles. Consequently, business rules are no longer passive structures evaluated or manipulated by business processes, see Section 4 for details.

3 Architecture and Influence of the Communication Infrastructure

The modeling methodology is associated with a 3-layer architecture that reuses proven principles from service-oriented architectures (SOA). This architecture was developed for the implementation of the EURENT dynamic BPM solution in ePVM. It is shown in Figure 3. In this architecture, the first layer comprises the presentation layer, which is clearly separated from the business logic captured in the other layers. Although the view of the actors on the business objects and other related information (for example business

rules and other actors) is very important, it should not be mixed with the business logic. In contrast, forms appeared as an almost integral part of the definition of business activities in [1]. The design and integration of the presentation layer can exploit common technology and is outside the scope of this paper as are also technical details that would address non-functional requirements such as scalability and performance.

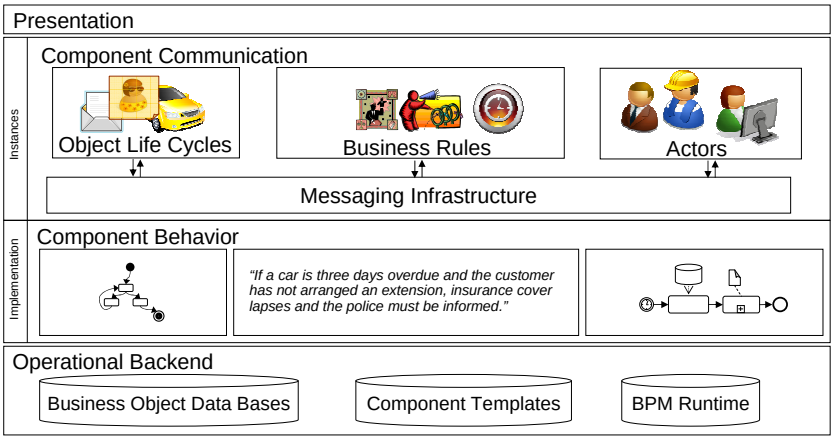

Fig. 3. Layered architecture with actors, object life cycles, and business rules as equal communicating components.

The second layer focuses on the communication of the component instances separated from the definition/implementation of the internal component behavior. Business rules, object life cycles, and actors are modeled as communicating components that rely on a common messaging infrastructure. The proposed methodology studies a setup in which all components act as equal partners, i.e., none has control over the other and they only interact through messages. The reaction to a particular message is under the control of a component. At this layer, the message interfaces of each component are defined and principles of interaction are established, clearly separated from details of the internal component behavior. In addition, it is assumed that monitoring components (or notification broadcasters) are provided by the messaging infrastructure to help the components communicate with each other. BPMN collaboration models are used to describe the component communication as well as the internal component behavior, see Section 4 for further details.

The third layer represents the operative backend systems: the runtime engine used, the data bases in which business objects and templates for the object life cycles, business rules, and activities are stored. For example, the runtime provides functionality to create new instances of business object life cycles, activities, and rules based on the available model templates. Data bases hold information about actors, business objects, and rules, and can be queried by the components.

The messaging infrastructure influences the concrete design of the components. The subsequent sections assume that components can directly send messages to each other

and also use a publish/subscribe mechanism, possibly moderated by monitors, to receive information of interest published by other components. Such a setup is common in today's business infrastructures. However, the proposed methodology can also be tailored to a different communication infrastructure, with the component communication changing accordingly.

4 Component Communication and Interaction Principles

This section establishes basic principles for the interaction behavior of the components and introduces BPMN collaboration diagrams for each component type. A component follows a processing cycle where it

1. receives an incoming message that acts as the primary trigger to initiate some behavior of the component,
2. sends and receives further secondary messages during the course of the behavior to determine its reaction to the primary trigger,
3. decides on the reaction, which can result in a reply to the originator of the primary trigger,
4. terminates its life or waits for the next primary trigger message.

Figure 4 summarizes the processing cycle that is inspired and natively supported by the ePVM runtime. It also corresponds to the conventional activation model of business processes that are instantiated by some start event.

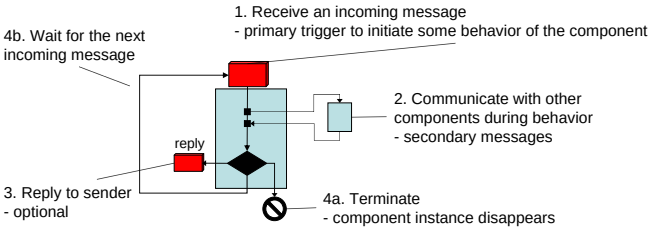

Fig. 4. Processing cycle of a communicating component.

Business rules and business object life cycles always require an explicit primary trigger message from another component. Business activities can be directly executed by an actor without any foregoing primary trigger message, because actors often self-initiate a processing cycle as a reaction to an externally observed event that happens in their environment. For example, the manager of an airport branch decides to buy additional cars for his branch before the airport opens a new runway that will increase air traffic and passenger numbers. Information about the runway opening will very likely not occur as a message within the dynamic process solution supporting the branch manager's work.

Consequently, without going into details of the internal behavior of a component, specifying the component communication already defines important aspects of the component behavior.

– **Primary Trigger**: What types of incoming messages do the component trigger?
– **Reply**: What reply will a component provide?
– **Secondary Incoming messages**: What types of additional incoming messages can the component handle during a processing cycle?
– **Secondary Outgoing messages**: What types of additional outgoing messages can the component send to obtain further information to compute its reaction to the primary trigger?
– **Visibility scope**: Which other components exchange messages with the component?
– **Published information**: What information does the component publish once a processing cycle is completed?
– **Internal memory**: Does the component maintain an internal memory? In the case of the object life cycle, the internal memory would capture the object state; however in the case of an actor, the memory can be much more comprehensive, storing message and activity histories for example.

This information can be specified using a BPMN collaboration diagram. In a concrete design, one detailed BPMN model would be drawn for each business rule, for each activity of an actor, and for each state transition of an object life cycle. Keeping a separate component for each activity and state transition leads to simpler collaboration diagrams. Furthermore, it will facilitate the configuration and modification of dynamic BPM solutions built with the proposed approach.

The next subsections specify the information from the preceding list for each component type using an abstract form of a collaboration diagram with disconnected model elements within a pool, which does not define details of the component behavior.

4.1 Business Objects

A business object consists of the data object stored in the data base and the associated life cycle, which is deployed as a communicating component instance in the runtime. In the ePVM implementation, life cycle instance and data base object are linked by using the component instance id as the primary key of the data object in the data base. Remember that the method proposes that the object life cycle controls all manipulation of a business object in the corresponding data base to ensure consistency between the life cycle instance and the object.

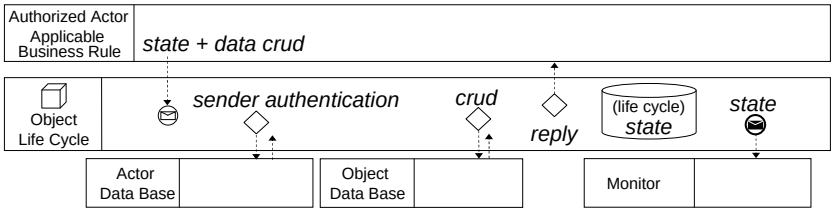

Fig. 5. Object life cycle component interaction.

Figure 5 shows an abstract BPMN collaboration diagram for object life cycles. The primary trigger is represented as the incoming message of the start event, wheras any published information is captured as the outgoing message of the end event of the pool representing the component. The reply of a component to the primary trigger is an outgoing message to the component that is the source of the trigger. The secondary messages occur between the start and end event. They originate from a decision gateway if they are optional. The data store symbol represents the internal memory of the component and shows the information that the component stores internally. The visibility scope is represented by other components as the originators or receivers of messages. They are represented by additional empty pools.

The primary trigger of a life cycle component is a message from an authorized actor or an applicable business rule. A life cycle component is not aware of other components. It can return a reply to the sender of the trigger message, but not actively contact other components. The life cycle component keeps the current state in its internal memory. While transiting to the next state, it can send a message to the actor data base to request the authentication of the sender of the primary trigger. Sender authentication turned out to be an important capability of object life cycles in a dynamic BPM solution. A life cycle receives messages from other components about which it does not know any details nor is it even aware that these components exist. As these messages can arrive in arbitrary order from arbitrary senders, a life cycle should have the capability to verify that a sender is indeed authorized to send a state/data change. For this purpose, the life cycle is aware of data sources (or services), where it can query information about the sender. Accepted changes are persisted in the corresponding business object in the data base. Upon finishing the processing cycle, the object life cycle will publish the state to a monitor. The monitor can record the state change history of the object and optionally inform other components that are interested in a state change of the object.

Figure 6 shows a concrete example of a BPMN collaboration diagram for the state transition from *acquired* to *available* of the car object life cycle shown in Figure 2.

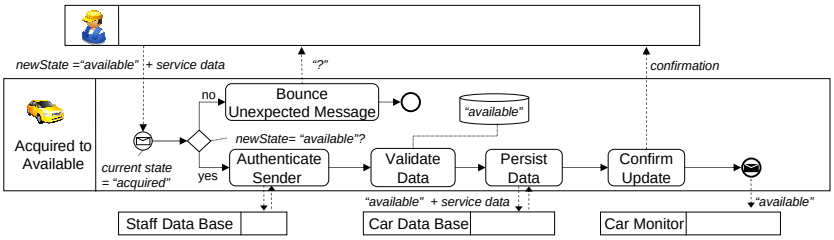

Fig. 6. Collaboration model for the state transition from *acquired* to *available* of the car object life cycle.

The car life cycle expects certain state update messages in certain internal states, e.g., only the new state *available* is expected in state *acquired*. These messages can only originate from the car park assistant authorized to do the initial inspection on a newly acquired car. Consequently, when the *available* message arrives, the car queries the actor data base to confirm that the responsible car park assistant has sent this message.

Only upon authentication and if the data passes other internal validations, will it transit to the new state and persist the state and data change in the car data base. If the update was successfully confirmed by the data base, the car life cycle sends a confirmation to the sender of the primary message and publishes the new state to the car monitor. If the component receives another message than the one expected, it interprets it as an error. Two reactions were implemented in the case study: swallow the message and do nothing or bounce it back to the sender as a reply. The latter is shown in the example model. Both give the sender a chance to realize that its communication might not have achieved the desired behavior at the receiving side: In the first case, no confirmation or notification of a state change will eventually happen. In the second case, the object directly notifies the sender through the reply.

4.2 Business Rules

A business rule component is triggered by a timer event, a query from another business rule, a request from an actor, or a state of an object life cycle, see Figure 7.

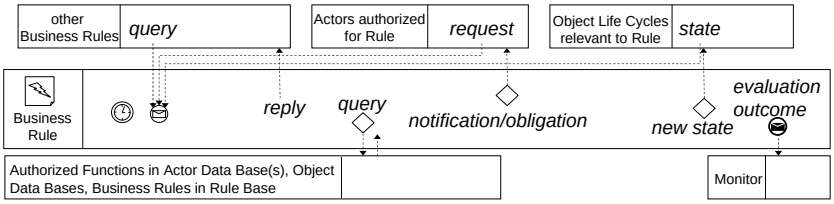

Fig. 7. Business rule component interaction.

To evaluate its outcome, the rule can query information about business objects from the corresponding data bases for which it is authorized and use authorized functions from the actor data base(s) as well as other business rules. It responds with a reply to a query or request, can send a notification or obligation to an actor, or can send a state update to a business object life cycle component. Upon completion of its processing cycle, it publishes its evaluation outcome to allow the monitor to maintain a history of executed business rules.

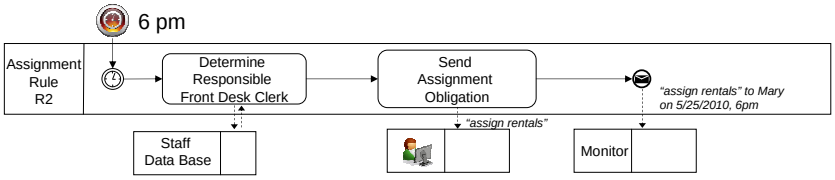

Fig. 8. Collaboration model for the rental assignment rule.

Figure 8 shows the model for the rule *"at the end of each day, cars are assigned to reservations for the following day"*. In this model, the concept of "end of day" has

been mapped to the specific timer event of 6 pm. When the timer fires, the rule first determines the actor for whom it has to establish the obligation. Then it sends a specific obligation message to this actor.

4.3 Actors

Actor components have much stronger communication capabilities than object life cycles and rules. They can spontaneously initiate some of their activities, while others might be triggered by an incoming message. Figure 9 illustrates both options, showing two types of start events.

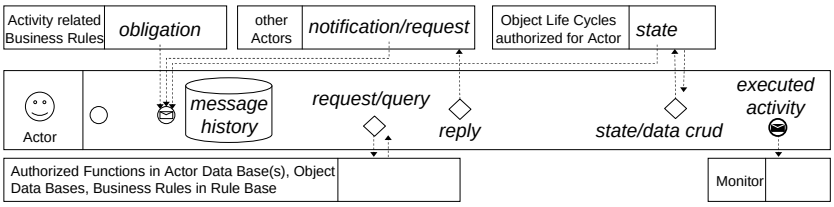

Fig. 9. Actor component interaction.

A triggering message can be an obligation established by a business rule, a notification or request from another actor, or a state information from a life cycle component. An actor keeps a message history in its internal memory, which allows the actor to correlate information to perform a complex behavioral activity. During the activity, actors can manipulate business objects by interacting with their life cycles or consult business rules. An actor can also send a request to other actors, which will trigger processing cycles for these actor components. An activity can result in the intention to change the state and data of a business object, which results in the corresponding message to the object life cycle component. The life cycle component may reply with a confirmation, which the actor receives as a secondary incoming message. An actor will usually return a reply to any request received. Upon finishing the processing cycle, the actor publishes information about the executed activity to the monitor, which allows the monitor to maintain the activity history of the actor. Business processes resulting from the message-coordinated activities of several actors can be mined based on such activity histories.

Activities usually work with object life cycles and business rules. For example, the branch manager initiates a new car life cycle when buying a car. A front desk clerk queries the actor data base to find out which car park assistants are currently on duty. To find a suitable upgrade for a customer, she consults business rules that represent the upgrade policies of the EURENT car rental company. Figure 10 illustrates how business rules, object life cycles, and activities play together in an end-to-end business process. The process starts with the branch manager buying a new car. A new car life cycle in the initial state *acquired* is instantiated through the *buy car* activity. The publication of this initial state state by the car triggers a technical inspection by the car park assistant taking the car to state *available*. Whenever a car enters this state, business rule (R1)

"each car must be serviced every three months or 10,000 kilometers, whichever occurs first" is evaluated, which may trigger a service. The car park assistant will execute the *schedule service* activity to respond to this obligation, but decides about the time frame himself, optionally monitored by another business rule not shown in Figure 10.

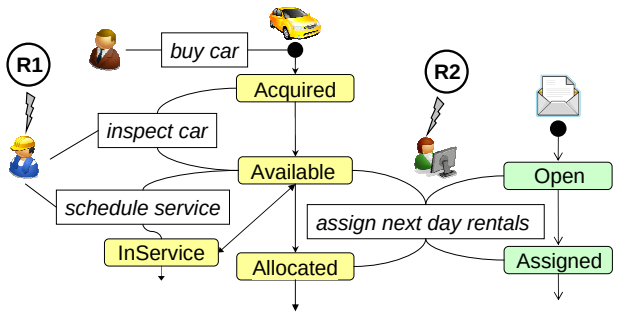

Fig. 10. Actors, business rules, and object life cycles in an end-to-end business process.

A car can also be rented to a customer when being in state *available*. The rental process begins when the front desk clerk chooses to assign this car to a rental contract by executing the *assign next day rentals* activity, which is initiated by rule (R2) *"at the end of each day, cars are assigned to reservations for the following day"*.

The *assign next day rentals* activity is of particular interest as it causes a synchronized state transition of two business objects, namely, the rental contract transiting from state *open* to state *assigned* and the rental car transiting from state *available* to state *allocated*. BPMN collaboration models for this activity are shown in Figures 11 and 12.

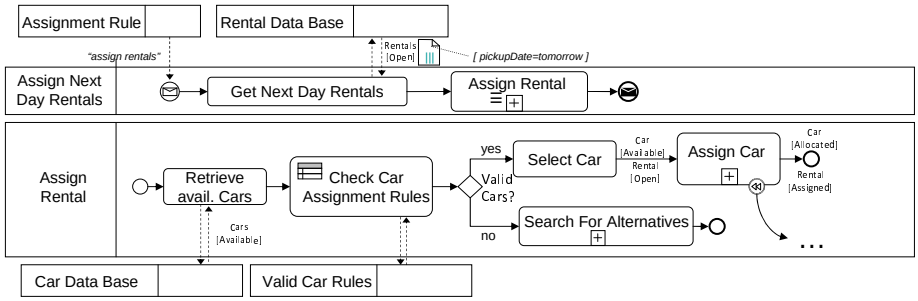

Fig. 11. Collaboration model of the *assign next day rentals* activity.

The *assign next day rentals* activity is triggered by the *assignment rule R2*. The concrete timing for the triggering message was specified by the rule component model in Figure 8. The activity consists of two main tasks. First, the front desk clerk must obtain the next day rentals from the rentals data base. Note the use of a collection data object *rentals*, which comprises all *rental* objects of state *open* for which the pick-up date equals the date of the next day. As this task is only a look-up not changing data,

redirection through the object life cycle is omitted from the model. Second, a car is assigned to each open rental contract in the *assign rental* multi-instance subprocess. Details of this subprocess are further spelled out in the lower part of Figure 11.

The *assign rental* subprocess begins by retrieving available cars from the car data base for each *rental* object in the *rentals* collection. As the assignment should happen the evening before the rental car is picked up, cars on rent or cars in service are unlikely to be good candidates, i.e., only cars in state *available* are retrieved. Next, a conventional business rule task *check car assignment rules* is executed to query rules to determine whether an available car is also a valid choice for the rental reservation, e.g., is from the car group requested by the customer. If this is the case, one of the valid cars is selected and finally assigned to the rental contract. Otherwise, an alternative car assignment, e.g., an upgrade, has to be found by the *search for alternatives* subprocess, of which details have been omitted. The *assign car* subprocess encapsulates the synchronized state transition of the car and rental business objects.

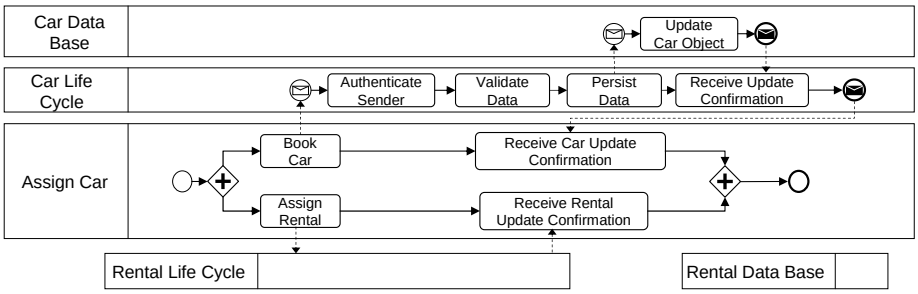

Fig. 12. Collaboration for the synchronized state transition of the *car* and *rental* business objects.

The collaboration model in Figure 12 specifies the message communication between the *assign car* subprocess, the car and rental life cycles, and the car and rental object data bases. It shows details of the car data base and car life cycle at the top, whereas the rental data base and rental life cycle are only shown collapsed at the bottom of the figure. To achieve the correct coordinated behavior of the car and rental life cycles, the *assign car* subprocess sends update messages to the car and rental life cycle components. The life cycle components in turn persist the updates in the corresponding car and rental data bases. Confirmations are sent from the data bases to the corresponding life cycles and from there to the *assign car* subprocess. If one of the confirmations does not arrive after a certain period of time, the front desk clerk (and probably also the car and rental life cycles) should initiate a compensating/error-handling activity, which is not further detailed in the example collaboration models.

5 Next Steps

The precise separation of activities, life cycles, and business rules helps users of this methodology to focus on the exact behavior of each element. However, they will benefit

from additional support to better understand the complexity of the resulting composite behaviors. This section identifies selected research problems that have to be tackled to take the proposed methodology from the current prototype status to a mature and industrial-strength solution approach.

Extensions of BPMN 2.0: The collaboration diagrams in Figures 5, 7, and 9 have been used in this paper to illustrate the communication behavior of each component type. In a specific dynamic BPM solution, business analysts will draw concrete collaboration diagrams like those in Figures 6, 8, and 11. In an initial modeling phase, such diagrams could be drawn as private processes showing only the incoming and outgoing messages. For this purpose, it would be desirable to visualize conditional message flow.

To make the human actors explicit who own the various activity components, lanes have been used. It needs to be investigated in greater depth whether the limited support for role modeling in BPMN 2.0 is sufficient for the proposed methodology. In addition, the relation to UML state machines must be further investigated. For example, Figure 6 shows a textual annotation that refers to the current state of the car life cycle, which influences how the incoming message is handled. The integration of the state attribute of a state machine model like the one in Figure 2 with its representation in the collaboration diagram representing a specific transition is currently an open question.

Refinement of the modeling methodology: When applying the integrated approach to the EURENT domain, the more than 100 business rules were grouped by object life cycle states. Structural business rules govern specific life cycle states, whereas operative business rules initiate, prevent, or govern state transitions. This helped to quickly bring structure into the large rule set and to detect inconsistencies. It would be desirable to evolve this aspect of the methodology into a systematic support for the structuring and consistent description of business rules, which helps business analysts to systematically transform textual business rules into operational communicating components.

Further investigations are required to confirm whether the methodology covers all aspects of case-handling solutions. It clearly provides information based on object states with business rules governing the activity flow. Within these boundaries, actors can self-initiate activities. Decoupling of authorization and distribution can also be achieved. The authorizations of a component are defined by other components with which it can interact. The work distribution in a business process can be organized by controlling the communication flow, possibly with the help of monitors.

An open question is the measurement of the quality of a dynamic process solution, for example, how the “right” granularity of components can be found. Further study also needs to be devoted to more advanced object interaction patterns, such as many-to-many synchronization relations between business objects, where, for example, one object can spawn an unknown number of “subordinate” object instances at runtime whose life cycles need to evolve to certain states before the spawning object can transit. In the spirit of the method proposed here, it would not be desirable that the object life cycles achieve such a coordination by communicating directly with each other. Instead, it seems that business rules would be the natural way to capture such complex forms of object coordination.

Tooling capabilities: In a concrete design, one detailed BPMN model would be drawn for each business rule, for each activity of an actor, and for each state transition

of an object life cycle. Maintaining the consistency and linkage between the various BPMN models, e.g., organizing the set of collaboration diagrams representing all possible activities of one actor or linking a collaboration diagram to each transition of a state machine model representing the life cycle of a business object would be a main task for a BPMN editor supporting this modeling methodology. It also seems possible that BPMN *conversation* diagrams could be used to visualize component interactions and to generate different views, such as, for example, all components that can receive or send a particular message or all activities that affect a particular object state.

Static analysis and simulation: Simulation and static analysis capabilities become very important as it can be expected that solutions are incrementally build starting from a small set of components or are configured by selection from a set of pre-built components. In this context, scenario-driven agile development and testing as proposed in the context of embedded systems design [6] seems to be a very promising approach for the design of dynamic BPM solutions. A play engine that allows business analysts to initiate and observe the behaviors (scenarios) that result from a set of BPMN collaboration components and that automatically constructs the overall end-to-end process would facilitate the dynamic BPM solution creation. In addition, tools should possess analytical capabilities. These may range from detecting that there is no component in a solution that receives a message sent by another component to becoming aware of races between activities, for example, the service scheduling and the rental of the car.

Mapping to existing or novel runtime engines: Conventional BPM engines that support component communication or alternative approaches such as [7] can be used for implementation. Some commercial BPM suites already support business rules as communicating components based on the Service Component Architecture (SCA) standard.

Actor interfaces: The design of user-friendly interfaces for humans working in such a dynamic BPM solution is another interesting area of work. Users see the following types of activities: activities that are always possible, e.g., create rental, activities that are enabled or required by specific object life cycle states, e.g., contact the customer of a delayed car, activities that result from rule obligations, e.g., assign cars, and activities that reflect requests from human actors, e.g., reassignment of a rental contract.

6 Related Work

So far, business rules and business processes have been integrated in a static way whereby a process invokes a rule (or rule set) at a specific point in the flow. Gartner [8] recently described an entire spectrum of integration scenarios for business processes and rules, but also emphasizes how little is understood about these scenarios. The approach in [9] to embed rule-based control and compliance objectives in a business process design shows how complicated a tight integration can be, leading to a complicated process design and rather motivates the clear separation between rules and processes in the methodology discussed in this paper. The two most relevant standards, SBVR [3] and BPMN [2], emphasize the need for integration, but also declare it as out of scope. BPMN only provides the so-called business rules task, which allows a BPMN engine to synchronously invoke a set of business rules over the data of the process and use the result of the rule evaluation to decide on the further branching of the process. This

solution of using rules as automated decisions is widely accepted, but as the authors of [10] emphasize, it is important to make the reasons explicit upon which obligations arise and cease, and to allow actors to decide which actions to take as a consequence of the obligation instead of directly invoking a specific action based on a computed rule outcome.

The increasing need for a more explicit treatment of business information in business processes has led to a widespread agreement that business objects should play a more prominent role. For example, ARIS emphasized early on that an event represents the fact that an information object has taken on a business-relevant state which is controlling or influencing the further procedure of the business process [11]. However, the ARIS method has not paid explicit attention to the evolution of the business object states. The artifact approach [12] carried the idea on to a sophisticated approach of state machines that capture the life cycle of business objects. Instead of defining business processes explicitly, the approach assumes that a process results from the interaction of the object life cycles with each other. Recently, a proposal about how to combine classical processes with business object life cycles was published [13], but the life cycles lose much of their autonomy and operational power, because a very closely-coupled, transaction-oriented manipulation of life cycles by processes is envisioned.

Explicit dependencies between life cycles are investigated in [14], where they are used to synthesize coordination structures between the transitions of different life cycles. The consistency of object life cycles and business process models as well as methods to derive one from the other are discussed in [15].

The integration of business rules and object life cycles has only received little attention so far. The authors of [16] follow an event-condition-action paradigm and use rules to synchronously control the transition to the next state. The work in [17] studies a variant of the artifact approach where services, which correspond to activities in the methodology presented in this paper, are annotated with expressive pre- and postconditions and sets of rules are synthesized that dynamically generate a workflow from service invocations to achieve a given goal formulated over attributes of an artifact. The work is very interesting; however, the high computational complexity of the synthesis problem already under the limitation that no artifact interactions are permitted and services can only be invoked once by a workflow, leaves many open questions regarding the practical applicability and extensibility of this work.

An initial treatment of business processes and business objects as communicating components can also be found in [18, 19]. Constraint-based approaches [20, 21] have recently been proposed to capture dependencies between data, activities, and business rules using temporal constraint languages. Although promising, the proposals show a complete paradigm shift in how BPM solutions are modeled and pose new problems, such as the need for an efficient and complete constraint solver for the proposed languages.

In contrast to approaches that assume activity/service annotations with complex pre- and postconditions, the methodology as described here keeps the “precondition” of an activity limited to a single message and encodes the knowledge that guides activity execution in separate business rules. This makes the methodology more naturally aligned with business rules modeling as it is common today and avoids complicated declarative

specifications of activities, which has been identified as a major knowledge engineering bottleneck by the Artificial Intelligence (AI) community [22]. The AI community proposes a rather orthogonal approach based on negotiation and intelligent decision making between autonomous agents as the foundation of dynamic BPM [23, 24, 25] with business processes remaining essentially implicit. At this stage, it seems that negotiation patterns as well as process and activity patterns as, for example, described in [26, 27] can be adopted by the proposed methodology in the form of best practices and pre-defined component templates.

7 Conclusion

A modeling methodology for dynamic BPM solutions is introduced that treats *business rules*, *actors*, and *business object life cycles* as equal partners in a loosely coupled system that interact through message exchanges, but have different capabilities. The clear distinction of the three component types makes it possible to separate component behavior from component interaction and facilitates the implementation of business patterns. BPMN collaboration diagrams capture the communication and behavior of each component. This methodology puts much more emphasis on messages and events in collaborations than conventional business process modeling does. The importance of scenario-driven development and testing is briefly discussed and identified as one of the open research challenges, not only for BPMN tool builders.

Acknowledgment I would like to thank Thomas Weigold for his support in using ePVM, Mark Linehan for his help in getting started with SBVR, the Zurich BIT team and Ksenia Wahler, ipt, for interesting discussions and valuable feedback, and the anonymous reviewers for their encouraging comments.

References

1. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data and Knowledge Engineering **53**(2) (2005) 129–162
2. Object Management Group: Business Process Model and Notation (BPMN) (2010) Version 2.0, OMG document number dtc/2010-05-03.
3. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR) (2008) Version 1.0, OMG document number formal/2008-01-02.
4. Business Rules Group: Defining business rules - what are they really? (2000) Final Report.
5. Weigold, T., Kramp, T., Buhler, P.: Flexible persistence support for state machine-based workflow engines. In: 4th Int. Conf. on Software Engineering Advances, IEEE (2009) 313–319
6. Harel, D., Marelly, R.: Specifying and executing behavioral requirements: The play-in/play-out approach. Software and System Modeling **2**(2) (2003) 82–107
7. Abiteboul, S., Bourhis, P., Marinoiu, B.: Efficient maintenance techniques for views over active documents. In: 12th Int. Conf. on Extending Database Technology, ACM (2009) 1076–1087
8. Sinur, J.: The art and science of rules vs. process flows. Research Report G00166408, Gartner (2009)

9. Sadiq, S., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: 5th Int. Conf. on Business Process Management (BPM). Volume 4714 of LNCS, Springer (2007) 149–164
10. Abrahams, A., Eyers, D., Bacon, J.: An asynchronous rule-based approach for business process automation using obligations. In: ACM SIGPLAN Workshop on Rule-Based Programming. (2002) 93–103
11. Davis, R.: Business Process Modeling with ARIS. Springer (2003)
12. Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3) (2003) 428–445
13. Nandi, P., et al.: Data4BPM: Introducing business entities and the business entity definition language (BEDL) (2010) IBM developerWorks.
14. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: OTM Conferences. Volume 4803 of LNCS., Springer (2007) 131–149
15. Küster, J., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: 5th Int. Conference on Business Process Management (BPM). Volume 4714 of LNCS, Springer (2007) 165–181
16. Linehan, M.: Ontologies and rules in business models. In: 11th Int. Conf. on Enterprise Distributed Object Computing (EDOC), IEEE (2007) 149–156
17. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: 12th Int. Conference on Database Theory (ICDT). Volume 361 of ACM Int. Conf. Proc. Series, ACM (2009) 225–238
18. van der Aalst, W., Barthelmess, P., Ellis, C., Wainer, J.: Workflow modeling using Proclets. In: 7th Int. Conference on Cooperative Information Systems (CoopIS). Volume 1901 of LNCS, Springer (2000) 198–209
19. Redding, G., Dumas, M., ter Hofstede, A., Iordachescu, A.: Modelling flexible processes with business objects. In: IEEE Conference on Commerce and Enterprise Computing (CEC), IEEE (2009) 41–48
20. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: The Role of Business Processes in Service Oriented Architectures. Volume 06291 of Dagstuhl Seminar Proceedings. (2006)
21. Wu, Q., Pu, C., Sahai, A., Barga, R.: Categorization and optimization of synchronization dependencies in business processes. In: 23rd Int. Conf. on Data Engineering (ICDE), IEEE (2007) 306–315
22. Barták, R., McCluskey, L.: Introduction to the special issue on knowledge engineering tools and techniques for automated planning and scheduling systems. Knowledge Enginering Review **22**(2) (2007) 115–116
23. Taveter, K., Wagner, G.: Agent-oriented enterprise modeling based on business rules. In: 20th Int. Conf. on Conceptual Modeling (ER). Volume 2224 of LNCS, Springer (2001) 527–540
24. Jennings, N., Norman, T., Faratin, P., O'Brien, P., Odgers, B.: Autonomous agents for business process management. Applied Artificial Intelligence **14**(2) (2000) 145–189
25. Weyns, D.: A pattern language for multi-agent systems. In: Joint Working IEEE/IFIP Conference on Software Architecture 2009 and European Conference on Software Architecture (WICSA/ECSA), IEEE (2009) 191–200
26. Hruby, P.: Model-Driven Design Using Business Patterns. Springer (2006)
27. Thom, L., Reichert, M., Iochpe, C.: Activity patterns in process-aware information systems: basic concepts and empirical evidence. Int. Journal of Business Process Integration and Management **4**(2) (2009) 93–110