Artifact-Centric Business Process Models in UML

Montserrat Estañol[1], Anna Queralt[2],
Maria Ribera Sancho[1], and Ernest Teniente[1]

[1] Universitat Politècnica de Catalunya
Departament d'Enginyeria de Serveis i Sistemes d'Informació
Jordi Girona 1-3, 08034 Barcelona
{estanyol,ribera,teniente}@essi.upc.edu
[2] Barcelona Supercomputing Center
Jordi Girona 31, 08034 Barcelona
anna.queralt@bsc.es

Abstract. Business process modeling using an artifact-centric approach has raised a significant interest over the last few years. This approach is usually stated in terms of the BALSA framework which defines the four "dimensions" of an artifact-centric business process model: Business Artifacts, Lifecycles, Services and Associations. One of the research challenges in this area is looking for different diagrams to represent these dimensions. Bearing this in mind, the present paper shows how all the elements in BALSA can be represented by using the UML language. The advantages of using UML are many. First of all, it is a formal language with a precise semantics. Secondly, it is widely used and understandable by both business people and software developers. And, last but not least, UML allows us to provide an artifact-centric specification for BALSA which incorporates also some aspects of process-awareness.

Keywords: business artifacts, BALSA framework, business process modeling, UML.

1 Introduction

Business process design is one of the most critical tasks in current organizations since they rely on the services they offer, i.e. on the business they perform. Business process models have been traditionally based on an activity-centric perspective and thus specified by means of diagrams which define how a business process or workflow is supposed to operate, but giving little importance (or none at all) to the information produced as a consequence of the process execution. Therefore, this approach under-specifies the data underlying the service and the way it is manipulated by the process tasks [1].

Nearly a decade ago, a new information-centric approach to business process modeling emerged [2] and it is still used today. It relies on the assumption that any business needs to record details of what it produces in terms of concrete

M. La Rosa and P. Soffer (Eds.): BPM 2012 Workshops, LNBIP 132, pp. 292–303, 2013.

information. Business artifacts, or simply artifacts, are proposed as a means to record this information. They model key business-relevant entities which are updated by a set of services (specified by pre and postconditions) that implement business process tasks. This approach has been successfully applied in practice [3] and it provides a simple and robust structure for workflow modeling.

The artifact-centric approach to business process specification has been shown to have a great intuitive appeal to business managers. However, further research is needed with regards to the "best" artifact-centric model since none of the existing models can adequately handle the broad requirements of business process modeling [4]. The chosen formalization should be based on a formal structure suitable for use in rigorous development and design analysis [2]. Moreover, it should support flexibility both at the level of the individual enactment of the workflow and by enabling rich evolution of the workflow schema.

Our work in this paper represents a step forward in this direction since we propose to specify artifact-centric business process models by means of well-known UML diagrams, from a high-level, technology-independent, perspective. The advantages of using UML are many. First of all, it is an OMG and ISO/IEC standard [5]. Secondly, it is used to represent both the static structure and the dynamic behavior of the elements that are part of a system using a graphical notation; thus it is possible to use diagrams to represent most of BALSA's components. In addition, these diagrams are understandable by people involved in the business process, both from the business and from the system development perspectives. Finally, UML provides extensibility mechanisms that permit more flexibility in its use without losing its formality.

The diagrams we have chosen to use for business process specification allow recording what information is produced by the business and how it is produced, thus achieving the advantages of artifact-centric modeling. Moreover, these diagrams and the way we specify them make our proposal artifact-centric but incorporating also some notions of process-awareness. In this way, we may also explicitly capture the control flow of the business process, aspect which is usually lacking in previous artifact-centric proposals.

The rest of the paper is structured in the following way. Section 2 provides the details of our proposal for artifact-centric business process models in UML and shows its application to an example. Section 3 compares our proposal with related work. Finally, section 4 summarizes our conclusions and points out further work.

2 Artifact-Centric Business Process Models in UML

Traditional process-centric business process models are essentially unidimensional in the sense that they focus almost entirely on the process model, its constructs and its patterns, and provide little or no support for understanding the structure or the lifecycle of the data that underlies and tracks the history of most workflows [4].

In contrast, the artifact-centric approach provides four explicit, inter-related but "separable" dimensions in the specification of the business process [4,6].

This four-dimensional framework is referred to as "BALSA" - Business Artifacts, Lifecycles, Services and Associations. By varying the model and constructs used in each of the four dimensions one can obtain different artifact-centric business process models with different characteristics [4]. By showing the UML diagram which is more appropriate to define each one of these four dimensions we will be able to construct our proposal for the specification of artifact-centric business process models in this language.

Usually, UML diagrams make use of some textual notation to precisely specify those aspects that cannot be graphically represented. Currently, OCL (Object Constraint Language) [7] is probably the most popular of these notations and an ISO/IEC standard. OCL supplements UML by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. It was initially developed by IBM and now it is part of the UML standard. Therefore, we will also use it in our proposal.

In the following subsections, we give a brief explanation of the four BALSA dimensions and we detail how we propose to specify them in UML. We also illustrate our proposal by showing some examples drawn from its application to a well-known and widely used case study: EU-Rent, which summarizes a generic process for renting a car within a car rental company. The whole specification of EU-Rent as an artifact-centric business process model in UML can be found in [8].

2.1 Business Artifacts as a Class Diagram

The conceptual schema of business artifacts is intended to hold all of the information needed to complete business process execution. A business artifact has an *identity*, which makes it distinguishable from any other artifact, and can be tracked as it progresses through the workflow of the business process execution. It will usually also have a set of *attributes* to store the data needed for the workflow execution. The relationship of a business artifact with other artifacts must also be shown when this information is relevant for the business being defined. In business terms, an artifact represents the explicit knowledge concerning progress toward a business operational goal at any instant. Therefore, at any time of the execution, the information contained in the set of artifacts records all the information about the business operation.

There is a strong parallelism between the notion of business artifact and that of "domain concept" in conceptual modeling [9]. Domain concepts are represented in UML by means of class diagrams. A UML class diagram shows the business entities and how they are related to each other, represented as classes and associations respectively. Each class (or business artifact) may have a series of attributes that represent relevant information for the business. Moreover, they can be externally identified by specific attributes or by the relationships they take part in. A class diagram may also require a list of integrity constraints that, as their name implies, establish a series of restrictions over the class diagram. Constraints can be specified either graphically in the UML class diagram or textually by means of the OCL language.

Furthermore, UML allows representing class hierarchies graphically. We will benefit from this by representing the different states in an artifact's lifecycle as subclasses of a superclass, as long as these subclasses hold relevant information or are in relevant relationships. The advantage of having different subclasses for a particular artifact is that it allows having exactly those attributes and relationships that are needed according to its state, preserving at the same time the artifact's original ID and the characteristics that are independent of the artifact's state which are represented in the superclass.

In our example, the diagram in Figure 1 shows the relevant EU-Rent business artifacts and how they relate to each other. Integrity constraints are defined in natural language instead of OCL for the sake of readability.

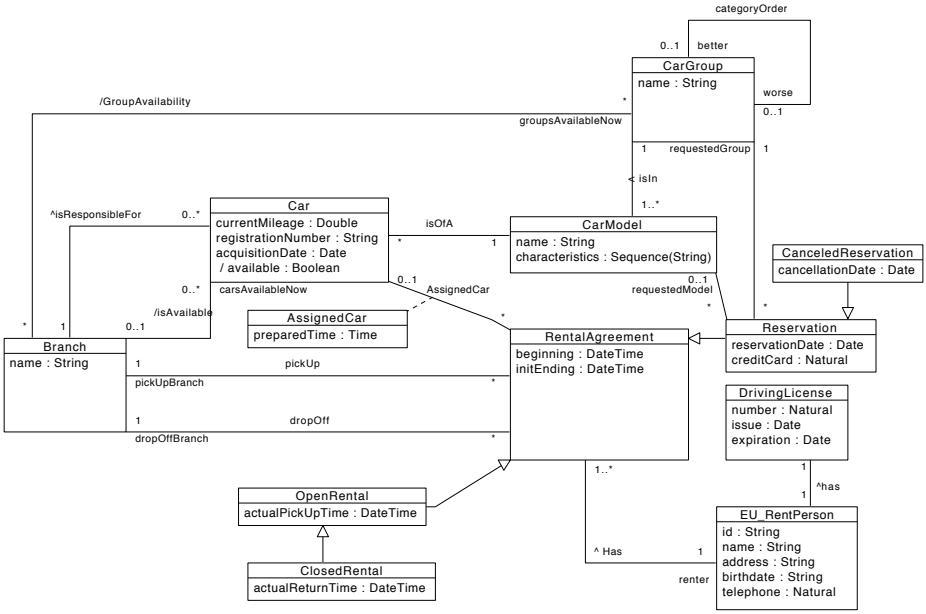

Fig. 1. Class diagram showing the business artifacts as classes

The integrity constraints for Figure 1 are the following:

- *EU_RentPerson* is identified by its *id*.
- *DrivingLicense* is identified by *number*.
- *Branch*, *CarModel* and *CarGroup* are identified by their *name*.
- *Car* is identified by *registrationNumber*.
- *RentalAgreement* is identified by *beginning* and its *renter*.
- An *EU_RentPerson* cannot have overlapping *RentalAgreements*.
- An *EU_RentPerson* must have a valid (i.e not expired) *DrivingLicense* for the *RentalAgreement* duration.

- The *initEnding* of a *RentalAgreement* must be later than its *beginning*. The *actualPickUpTime* of an *OpenRental* must be later or equal to its *beginning*. The *actualReturnTime* of a *ClosedRental* must also be later than its *actualPickUpTime*.
- Relationship *categoryOrder* of *CarGroup* cannot have any cycles.
- A *Car* is available if all the rentals that it is assigned to are closed or canceled. Otherwise it is not available.
- Relationship *IsAvailable*: *Cars* that a *Branch* is responsible for and that are available.
- Relationship *GroupAvailability*: *CarGroups* available for a particular *Branch*, obtained through the available *Cars*.

In Figure 1, *RentalAgreement* is the key business artifact in the car rental process. A *RentalAgreement* is the result of an *EU-RentPerson* wanting to rent a car for a particular period of time. It is identified by its attribute *beginning* and a business artifact it is related to, *EU_RentPerson*. Notice that it is related to many of the other classes in the diagram. It has a pick-up and drop-off *Branch*, and may have a car assigned. A *RentalAgreement* may also be of several subtypes. It will be of the *Reservation* subtype if the client has made a reservation in advance. A *Reservation* is linked to a *CarGroup* for the rental, and may also be linked to a particular *CarModel* if the client has expressed his/her preferences for a particular brand and model of car. *Reservations* can be canceled, so there is also a *CanceledReservation* subtype. A *RentalAgreement* will become of the *OpenRental* subtype when a car is successfully handed over to the customer, and will be of the *ClosedRental* subtype when the client returns the car to the branch.

2.2 Business Artifacts Lifecycles as State Machine Diagrams

The lifecycle of a business artifact states the key, business-relevant, *stages* in the possible evolution of the artifact, from inception to final disposal and archiving. It is natural to represent it by using a variant of state machines, where each state of the machine corresponds to a possible stage in the lifecycle of an artifact from the class diagram [4].

We propose representing the states an artifact may go through in a UML state machine diagram, in a similar way to the one proposed in [6]. However, in contrast to this work, our state machine diagram includes a representation of the events and the conditions about them that trigger the transitions between consecutive states of the business artifact.

We distinguish two different kinds of events: *external events* (named call or signal events in [9]) and *internal events* (named temporal or condition events in [9]). External events are explicitly requested by the customer of the business process and their behavior is specified by means of a set of associated services. Internal events correspond to conditions stated over the content of the business artifacts and cause the execution of services without requiring the customer intervention. Services will be defined in the next section.

In Figure 2 we can see the corresponding state machine diagram for the business artifact *RentalAgreement*. Initially, the *RentalAgreement* can be a *Reservation* or an *OpenRental*, depending on whether the client makes a reservation to pick up the car on a later date (*Make Reservation* event) or he wishes to rent a car immediately (*Make Walk-In Rental* event). If the user picks up the car after having made a reservation (*Pick Up Car* event), the *RentalAgreement* becomes an *OpenRental*. On the other hand, if the customer does not pick up the car on the scheduled day or he decides to cancel the *Reservation* (*Cancel Reservation by Customer Demand* event), the *RentalAgreement* becomes a *CanceledReservation*. Finally, from *OpenRental* we will reach state *ClosedRental* after the user returns the car (*Return Car* event). All the events are external, except the one that takes place when the customer does not pick up the car. In this case, the transition is triggered by comparing the current date with the beginning of the rental agreement, when the beginning of the rental agreement has already gone by. In this case, *Cancel Reservation* is executed. Finally, we should note that there is a condition between square brackets in some of the events, *success*, that indicates that the event should finish successfully in order for the business artifact to change state.

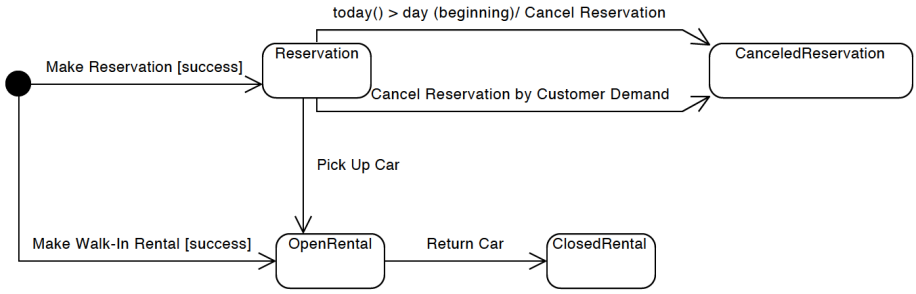

Fig. 2. State machine diagram that represents the lifecycle of RentalAgreement

It is worth noting that this diagram does not follow exactly the standard described in [5]. For instance, we have more than one outgoing transition from the start node. This is necessary because the *RentalAgreement* can be created in different ways (e.g. by making a walk-in rental or a reservation).

2.3 Services as Operation Contracts in OCL

A service (or "task") in a business process encapsulates a unit of work meaningful to the whole business process. The action of services makes business artifacts evolve, e.g. they may cause modifications on the information stored by the artifacts or they may make artifacts to evolve to a new stage, relevant from the business perspective.

Existing approaches usually specify services by means of *preconditions* and *postconditions*. We follow in this line by using OCL operation contracts. As

we have mentioned before, OCL is a formal language that avoids ambiguities. Moreover, it is declarative, which means that it does not indicate *how* things should be done, but rather *what* should be done. The advantage of using OCL is that it is a part of the UML specification, which is an OMG standard and also a ISO/IEC standard [5]. Moreover, it can be automatically translated into first-order logic [10] or description logics [11], for instance, thus permitting to perform automatic reasoning on the UML/OCL specification.

Operation contracts consist of a set of input parameters and output parameters, a precondition and a postcondition. Both input and output parameters can be classes (i.e. business artifacts) or simple types (e.g. integers, strings, etc.). A precondition states the conditions that must be true before invoking the operation and refers to the values of artifact attributes at the time when the service is called. The postcondition indicates the state of the business artifacts after the execution of the operation. It may refer to the values of artifact attributes at the time when the service is called (appending operator *@pre*) *and* to their values after the service has finished execution (no operator or appending operator *@post*). Those artifacts that do not appear in the postcondition keep their state from before the execution of the operation.

Returning to the example, we will focus on the process of making a walk-in rental. A walk-in rental can be defined as the process whereby a client goes to a branch of the company and rents a car on the spot without any previous reservation, as long as there are available cars. Therefore, we need a service or operation that obtains all the data necessary for the rental. We have named it *ObtainRentalData* and its code can be seen in Listing 1. Given an *EU_RentPerson*, the pick-up and drop-off *Branches* of the car, a *CarGroup* and/or a *CarModel*, and the end date, the operation creates a new *RentalAgreement*. Note that we do not check, for example, that this new *RentalAgremeent* does not overlap with other *RentalAgreements* that the *EU_RentPerson* may have. This is because we want to avoid redundancy and therefore we do not check conditions which are guaranteed somewhere else in the artifacts' specification, such as the class diagram, as described in [12]. The non-overlapping rental condition is already guaranteed by the class diagram and its integrity constraints.

Listing 1. OCL code for *ObtainRentalData*

```
action ObtainRentalData(endDate: Date, dropOffBranch: String, carG:
    String, carM: String, p: EU_RentPerson): RentalAgreement

localPre availableCarModel: carM<>'' implies
    currentBranch().carsAvailableNow.carModel.name->includes(carM)
localPre availableCarGroup: carG<>'' implies
    currentBranch().groupsAvailableNow.name->includes(carG)

localPost:

-- Create Rental Agreement --
RentalAgreement.allInstances() -> exists(ra.oclIsNew() and ra.renter=p
    and ra.beginning=now() and ra.initEnding=endDate and
    ra.pickUpBranch=currentBranch() and
    ra.dropOffBranch=Branch.allInstances()->select(dob |
    dob.name=dropOffBranch) and
-- We assign the car model with the least mileage --
```

```
(if (carM <> '') then
    ra.car = currentBranch().carsAvailableNow -> select(c |
        c.carModel.name=carM)->sortedBy(currentMileage) -> first()
 else
    (if (carG = '') then
        ra.car = currentBranch().carsAvailableNow ->
            sortedBy(currentMileage) -> first()
     else
        ra.car = currentBranch().carsAvailableNow -> select(c |
            c.carModel.carGroup.name=carG) -> sortedBy(currentMileage)
            -> first()
    endif)
 endif)
and
-- We return the Rental Agreement --
result = ra)
```

As it can be seen in Listing 1, the service requires an *EU_RentPerson* as a parameter; therefore, we need to obtain this business artifact in some way before its invocation. To do so, we need to check if the person is already registered and, in case he/she is not, insert him/her into the system. As it will be seen in the next section, we have split this job into two different services. We do not show their details here due to space limitations.

2.4 Associations as Activity Diagrams

Having the services as detailed above is not enough. We also need a way to establish the conditions under which services can be executed since, in a business process, they make changes to artifacts in a manner that is restricted by a family of constraints. These conditions/constraints might either be defined through a procedural specification or through a declarative one. Most of the existing proposals follow the second approach and define associations by means of Condition-Action rules (as done in [13,14,15]) or by encoding them into the *service* definition itself (see for instance [1]).

We propose to follow a procedural specification and to use UML activity diagrams for specifying associations since they are aimed at defining the right sequencing of service execution. In particular, we will have an activity diagram for each external event in our state machine diagrams. In this way, each service in which the event is decomposed is represented as an action (a rounded rectangle) in the activity diagram. Arrows show the order in which actions (i.e. services) have to be executed. Swimlanes indicate the main business artifact involved in each action, and the notes stereotyped as *Participant* indicate who is responsible for carrying out that action.

By modeling associations in this way we achieve our purpose of incorporating some notions of process awareness, despite the intrinsic artifact-centric nature of our proposal. Therefore, our proposal shows a way to explicitly and graphically capture the control flow of the business process; in contrast to many proposals such as [15] or [13] where they are represented textually.

The corresponding activity diagram for the external event *Make Walk-In Rental* from our example is detailed in Figure 3.

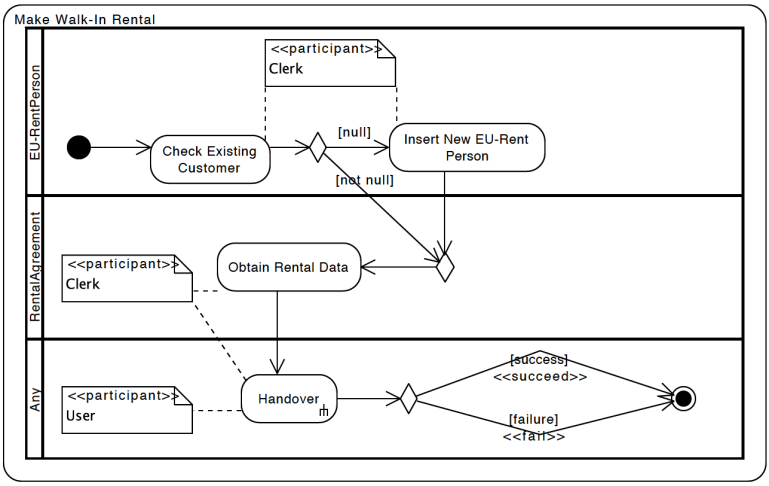

Fig. 3. Correspoding activity diagram for *Make Walk-In Rental*

As it can be seen in the diagram, we need to execute the service *CheckExistingCustomer* first. Depending on its output, which will indicate whether the person is registered in the system or not, we may insert a new customer (*InsertNewEU-RentPerson*). In the end, we obtain the data for the rental (*ObtainRentalData*) and we create a new *RentalAgreement*.

Notice that, apart from the services described above, there is an additional action, *Handover*, with a rake-like symbol. This indicates an action that is further defined in another activity diagram. The transition leading out of this action takes us to a decision node. Depending on the result of the Handover action, indicated between square brackets, *Make Walk-In Rental* will end successfully (stereotyped as *succeed*) or not (stereotyped as *fail*). Although the process ends in the same way regardless of its success or failure, it is important to make this distinction, as the next stage in the lifecycle of the artifact *RentalAgreement* depends on this outcome (see Figure 2).

Due to space limitations, we do not show the activity diagram of Handover nor the details of the services/actions that make it up. They can be found on [8].

3 Related Work

This section will look at the different alternatives used to represent the four elements in the BALSA framework, i.e. business artifacts, lifecycles, services and associations. Although most papers do not specifically state which form of representation is used for each element, it is not difficult to establish a correspondence between the representation in the different papers and the BALSA elements.

Business artifacts, sometimes referred to as business entities, are represented in different ways in the literature. Various authors use database schemas [13,14,16]. A similar representation is proposed in [1,15,17,2] where artifacts consist of a set of attributes or variables. [18] represents the business artifact and its lifecycle in one model that includes the artifact's attributes. Another possibility for representing them is using using an Entity-Relationship model as it is done in [6]. [19] represents the data model of their example by means of an UML class diagram. This differs from our proposal in that we propose using the UML class diagram for representing the business artifacts themselves. [20] chooses to represent artifacts as state machines defined by Petri nets.

Regarding the lifecycle of business artifacts, there are two main alternatives: they either offer an explicit representation of the evolution of the artifact or it is implicit. The explicit representations are normally based on a state machine diagram. Examples are [6,13]. A more formal approach is the one used in [19,20], where lifecycles are represented in variants of Petri nets. A similar alternative is proposed in [21], where ArtiNets (similar to Petri nets) and DecSerFlow, a declarative language, are used to represent the lifecycle of an artifact and its constraints. Sometimes there is a variable in the artifact which stores its state [1,15]. [18] uses GSM to represent the artifacts' lifecycles. The notation shares some characteristics with ours, such as the ability to represent graphically guards and stages. However, it adds the concept of milestone to represent conditions that determine the closing of a state. On the other hand, the sequencing of stages is determined by guard conditions instead of edges connecting the stages (although it is possible to use edges as a *macro*). In other cases, the lifecycle is implicitly represented by dynamic constraints expressed in logic [13] or the actions that act upon artifacts [16,14].

Services are also referred to as tasks or actions. Despite the different terminology, in general they are described by using pre and postconditions (also called effects). [13,1,15,17,16] use different variants of logic for this purpose. [14] follows the same idea but omitting the preconditions. [6] uses natural language to specify pre and postconditions.

Associations are represented in different ways depending on the approach of the paper. Some authors opt for using condition-action rules defined in logic [13,14]. [15] calls these conditions business rules; they should not be confused with business rules in [1], which are conditions that are superimposed in the already existing ones. In [1], preconditions determine the execution of the actions; therefore, they act as associations. [6] also uses event-condition-action rules, but they are defined in natural language. [19] uses what they call *channels* to define the connections between *proclets*. A proclet is a labeled Petri net with ports that describes the internal lifecycle of an artifact. Another alternative is DecSerFlow, that allows specifying restrictions on the sequencing of services, and it is used in [21]. It is a language grounded on temporal logic but also includes a graphical representation. On the other hand, [2,3] opt for a graphical representation using flowcharts. However, unlike our proposal, they do not use a particular language and little attention is given to the way of creating the flowchart.

4 Conclusions

There is no consensus on the best way to represent an information-centric model and, in all probability, there will never be, due to the great variety of uses of data-centric models. However, as [4] points out, it is important to experiment with different models, in order to examine the different possibilities that each one offers. Therefore, one contribution of this paper is identifying the UML diagrams that can be used to represent a process from an artifact-centric perspective following the BALSA framework.

To our knowledge, ours is the first proposal that suggests the use of UML diagrams for representing all the elements in this framework. We have shown that business artifacts can be represented in a class diagram, each artifact's lifecycle can be shown in a UML state machine diagram, services can be represented by using OCL operation contracts, and a possible way of defining associations is by means of a UML activity diagram. The use of an activity diagram for representing the associations between services brings it closer to process-centric methodologies and, at the same time, makes it easier to understand than just having textual restrictions in the form of condition-action rules represented in logic.

The importance of our contribution lies in the fact that UML is a standard in the world of conceptual modeling, and OCL, as a complement to UML, is used to represent those elements that cannot be graphically specified in UML. Moreover, both languages can be automatically translated into logic and the translation can be used for reasoning purposes. Therefore, our proposal offers the advantages of a graphical representation, understandable by the users, without losing the capacity of being used for reasoning.

As further work, we intend to define a way to perform automatic reasoning on the definition of a process using our diagrams in order to be able to validate its correctness, appropriateness and its quality before it is implemented.

Acknowledgments. This work has been partially supported by the Ministerio de Ciencia e Innovación under projects TIN2011-24747 and TIN2008-00444, Grupo Consolidado, the FEDER funds and Universitat Politècnica de Catalunya.

References

1. Damaggio, E., Deutsch, A., Hull, R., Vianu, V.: Automatic Verification of Data-Centric Business Processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 3–16. Springer, Heidelberg (2011)
2. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. 42(3), 428–445 (2003)
3. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifact-centered operational modeling: lessons from customer engagements. IBM Syst. J. 46(4), 703–721 (2007)
4. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)

5. ISO: ISO/IEC 19505-2:2012 - OMG UML superstructure 2.4.1 (2012), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=52854
6. Bhattacharya, K., Hull, R., Su, J.: A Data-Centric Design Methodology for Business Processes. In: Handbook of Research on Business Process Management, pp. 1–28 (2009)
7. ISO: ISO/IEC 19507:2012 - OMG OCL version 2.3.1 (2012), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57306
8. Estañol, M., Queralt, A., Sancho, M.R., Teniente, E.: EU-Rent as an artifact-centric business process model (2012), http://www.essi.upc.edu/~estanyol/docs/artifacts_eu_rent.pdf
9. Olivé, A.: Conceptual Modeling of Information Systems. Springer (2007)
10. Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL constraints. ACM Trans. Softw. Eng. Methodol. 21(2), 13 (2012)
11. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. Data & Knowledge Engineering 73, 1–22 (2012)
12. Queralt, A., Teniente, E.: Specifying the Semantics of Operation Contracts in Conceptual Modeling. In: Spaccapietra, S. (ed.) Journal on Data Semantics VII. LNCS, vol. 4244, pp. 33–56. Springer, Heidelberg (2006)
13. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P.: Foundations of Relational Artifacts Verification. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 379–395. Springer, Heidelberg (2011)
14. Cangialosi, P., De Giacomo, G., De Masellis, R., Rosati, R.: Conjunctive Artifact-Centric Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 318–333. Springer, Heidelberg (2010)
15. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
16. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of Deployed Artifact Systems via Data Abstraction. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 142–156. Springer, Heidelberg (2011)
17. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Fagin, R. (ed.) ICDT 2009, vol. 361, pp. 225–238. ACM (2009)
18. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath III, F(T.), Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011)
19. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Behavioral Conformance of Artifact-Centric Process Models. In: Abramowicz, W. (ed.) BIS 2011. LNBIP, vol. 87, pp. 37–49. Springer, Heidelberg (2011)
20. Lohmann, N., Wolf, K.: Artifact-Centric Choreographies. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 32–46. Springer, Heidelberg (2010)
21. Kucukoguz, E., Su, J.: On Lifecycle Constraints of Artifact-Centric Workflows. In: Bravetti, M. (ed.) WS-FM 2010. LNCS, vol. 6551, pp. 71–85. Springer, Heidelberg (2011)