Kopernik: Modeling Business Processes for Digital Customers

Montserrat Estañol[1,3], Manuel Castro[2], Sylvia Díaz-Montenegro[2], and Ernest Teniente[1]

[1] Universitat Politècnica de Catalunya, Barcelona
estanyol | teniente @essi.upc.edu
[2] Léelo - Process as a service, Madrid
manuel.castro | sylvia.diazm @leelo.es
[3] SIRIS Lab, Research Division of SIRIS Academic, Barcelona

Abstract. This paper presents the Kopernik approach for modeling business processes for digital customers. These processes require a high degree of flexibility in the execution of their tasks or actions. We achieve this by using an artifact-centric approach to process modeling and the use of condition-action rules. The processes modeled following Kopernik can then be implemented in an existing commercial tool, Balandra.

1 Introduction

Business processes are a key element in organizations, as they are directly involved in the achievement of an organization's goals. Representing those business processes using models can facilitate the communication between the parties involved in the process, they provide a basis for the process improvement and can help to perform process management.

Traditionally, business processes have been represented or visualized as a workflow which contains a set of tasks or actions that should be performed in a certain order in order to achieve the process's goals. However, these representations are not appropriate for processes which deal with digital customers. These customers wish to perform the different actions on their own, through different channels (e.g. telephone, e-mail), and a bad experience may mean that they severe all ties with the company. An example of such processes is contracting an insurance policy or the process for on-line debt collection [12].

These processes are very flexible: they require a set of actions to achieve their goals, but it does not matter the order in which these actions are performed or, more precisely, this order is driven by the data collected from the customer and not by the flow of process execution itself. Considering this, traditional approaches for process modeling are not appropriate, as they force the execution of the actions in a certain order. In contrast, modeling business processes for digital customers requires a representation which is able to reflect their flexibility.

We believe that the artifact-centric approach for process modeling is appropriate for this purpose. According to this approach, the key elements of a

business process are the data and the definition of the actions in the process and it does not necessarily force the actions to execute in any particular order. We will take advantage of this capability to allow for the required flexibility.

There are some tools in the market, such as Balandra [12], which are capable of modeling these flexible processes for digital customers and then automatically generating and customizing software systems for digital customers from these models. However, as we will see, current (artifact-centric) methodologies do not fit well with the features required by the models supported in these tools.

For this reason, in this paper we outline a generic approach, called Kopernik, for modeling these flexible processes from an artifact-centric perspective. It builds upon the BALSA framework [11] to structure the artifact-centric process and the resulting approach is applicable to real-world flexible processes.

2 The Kopernik approach

The BALSA framework for artifact-centric process modeling defines four dimensions which should be present in any such model [11]. The **business artifacts** represent the key data for the business. These artifacts evolve during their life, and this evolution is represented by means of a **lifecycle**. The **services**, or actions, represent atomic units of work that create, update and delete the business artifacts. Finally, the **associations** establish restrictions in the manner in which the services may make changes to the artifacts.

To illustrate the Kopernik approach, we will use an example based on the process of on-line debt collection.The goal of an on-line debt collection process is to be able to amicably recover the debt incurred by a client. The client's debt file may be, in fact, related to various debts acquired over time and the payment of the whole debt may be divisible in installments. We will call the client's debt file *OnlineDebtPayment*.

There are two possible ways of making the payment(s): either by credit card or by bank transfer. We assume that the client will have to choose at least one of these two methods, and he will have to respect that choice. However, he can change the payment method at any time.

An online debt payment may be in three different states: *being processed*, *canceled* or *completed*. *Being processed* means that the debt has not been repaid yet. The *OnlineDebtPayment* will be *canceled* if the client's debt is canceled. Finally, it will be in state *completed* when the whole debt is repaid, that is, when the sum of the amount of all payments equals the sum of the amount of the debts linked to that online debt payment.

2.1 Business Artifacts

Business artifacts represent the relevant data for the business. We are interested in defining the details of the data and the relationship between them. In this context, we will distinguish between a *lead* and *business entities*. The *lead* represents the key business artifact in the process, and it is generally related to other complementary data, the *entities*.

We represent the lead and its related entities using a special type of UML class diagram. Both the lead and the entities will correspond to classes, and the relationships between them will be represented as associations. However, we establish some restrictions in the way they are related to each other.

To begin with, the classes in the diagram will be placed in a star shape, with the lead at the center of the diagram and the remaining entities surrounding it. A relationship between a lead `L` and an entity `E` will have a multiplicity of `*` and `1`. That is, an instance of an entity `E` will be related to exactly one instance of `L` and an instance of the lead `L` may be related to `N` instances of `E`.

Entities may be related to other entities but only one of these entities may be linked to the lead. That is, the lead and the business entities have a tree-like structure (with the lead as the root) with no cycles. In this way, the lead acts as an aggregated summary of the process. This contrasts with other approaches, such as [8, 9], which model the artifacts using a standard UML class diagram.

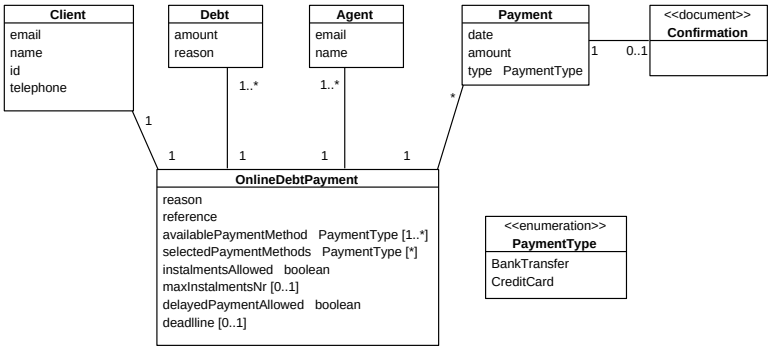

Fig. 1. Class diagram for the *online debt collection* example.

The class diagram for our example can be seen in Figure 1. Notice its star-like shape, with artifact *OnlineDebtPayment* at the center of the diagram. *OnlineDebtPayment* is the lead in this example. It represents the total debt that a certain *Client* owes, and it will correspond to the sum of several *Debt*s (notice that *OnlineDebtPayment* may be related to several *Debt*s). It is also related to one or more *Agent*s who follow up on the evolution of the *OnlineDebtPayment* and the *Payment*s themselves. Note that an *OnlineDebtPayment* is related to zero or more *Payment*s as the global debt may be paid in several installments, and at the very start no payment has been made.

2.2 Lifecycles

Lifecycles represent the evolution of artifacts. In general, existing artifact-centric approaches such as [3, 6, 9] represent states using a single model, which may be, for instance, a state machine diagram or a variable.

The Kopernik approach presents two different, complementary ways of representing the lifecycles of the business artifacts: internal states and tagsets. Intuitively, internal states correspond to the "traditional" way of representing the lifecycle of the business artifacts. In contrast, tagsets are views over the artifacts which provide a customized overview of the evolution of the process, as they can be defined for each user/role type. This section presents both of them.

Internal States: Both leads and entities may have internal states. In order to represent these states and how the artifact evolves through them, we will use a state machine diagram. We refer to these states as internal because they are not necessarily perceived by the user, but they are relevant from the point of view of the business process evolution.

Each of these states may have several incoming and/or outgoing transitions, indicating the conditions which will trigger the change from one state to the other. These transitions reference exactly one service or action: when the task is performed, if the artifact or object is in the appropriate state (the source state), then the artifact will change to the target state. Note that many actions do not have an impact on the internal state.

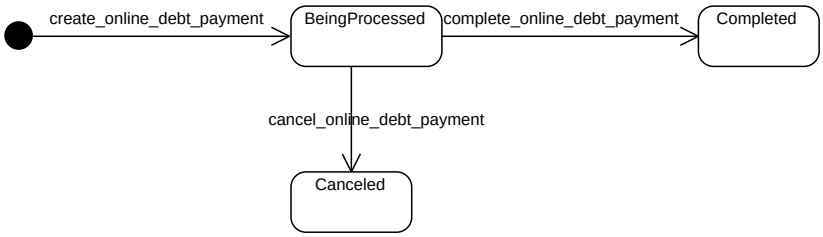

Fig. 2. State machine diagram showing the internal states for *OnlineDebtPayment*.

As shown in Figure 2, the lead in our example, *OnlineDebtPayment*, has three internal states: *BeingProcessed*, *Completed* or *Canceled*. When it is first created it has state *BeingProcessed*. If, while it is in state processed, *cancel_online_debt_payment* executes, it changes its state to *Canceled*. On the other hand, if *complete_online_debt_payment* takes place, it changes its state to *Completed*.

Due to space limitations, we only show the state machine diagram for a lead. If an entity also had internal states, it would be represented in the same way.

Tagsets: Apart from the internal states, it is possible to define views over the business artifact which are used to provide the user with information of process evolution. We call them tagsets. Each tagset can be configured for every user profile. A user profile represents the role of a user within the process. Examples could be the client, an agent or the manager. Note, however, that the definition of user profiles is out of the scope of this paper.

Every tagset has, as the name implies, a set of tags. Each tag represents a relevant state from the *point of view* of the *user* and it has an associated condition. The conditions may refer to an internal state of the artifact, as defined in its state machine diagram, to the values of its attributes or the classes related to it. All the conditions of the tags in a same tagset must be mutually exclusive and collectively exhaustive; that is, at every point in time, exactly one tag should be active.

We propose representing these tagsets in the class diagram using derived subclasses, for easier readability, and a table to define their conditions. Each tagset will make up a (multi-level) hierarchy, in which each level fulfills the disjointness and covering constraint. That is, an instance of an artifact must have exactly one of the subtypes (or *tag*). The derivation conditions should be defined for each of the leaves in the hierarchy. Classes in the tagset are only views over the artifacts in the system.

The conditions which determine the derivation of the classes can be defined using a table. Figure 3 shows an example tagset for a client that owes money. The top row will contain the relevant class (or artifact) names and the last column the tag. The conditions in each column must be mutually exclusive, e.g. an *OnlinePaymentDebt* cannot be *canceled* and *completed* simultaneously. And the tags in the last column must be mutually exclusive and commonly exhaustive.

OnlinePaymentDebt	Payment	Tag
Canceled		Debt canceled
Completed		Completed payment
In process	No payment yet	No payment
	At least one payment	Payment begun

Fig. 3. Table showing a simple tagset for our running example.

Note that the four different tags in the table depend on the states or values of classes *OnlineDebtPayment* and *Payment* (Figure 3). If the state of *OnlineDebtPayment* corresponds to *Cancelled* or *Completed*, as defined in its internal state machine diagram, the client will see tags *Debt canceled* or *Completed payment*, respectively.

On the other hand, if the *OnlineDebtPayment* is in state *InProcess*, the tag will be determined by looking at *Payment*. If there is no payment related to the online debt payment, the tag will be *No payment*. On the other hand, if at least one payment has been made, the tag will correspond to *Payment begun*.

Figure 4 shows the hierarchy that corresponds to the tagset defined in Figure 3 in the form of a class diagram for easier readability. Each class in the leaves corresponds to one of the tags in the table.

Note that tags do not correspond to any kind of class in the system, as they are "calculated" given the available information. As such, they are a mere view over the existing system. These views are meant to be flexible and easily

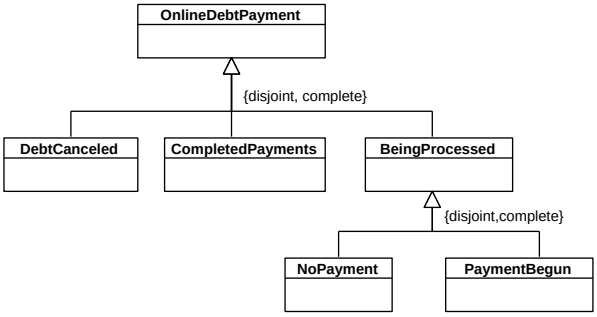

Fig. 4. Tagset for the *OnlineDebtPayment*. The definition of the conditions can be seen in Figure 3.

changeable. Therefore, they cannot be referenced by the actions or any of the system's specification, as they are prone to change.

Note that we do not specifically define business goals in the Kopernik approach. They could be represented by a special marking on internal states or tagsets. The latter would be useful to represent the different points of view from various users, as it would allow us to define different goal states for each user type. However, we leave this for further work.

2.3 Services

Services correspond to actions, that is, atomic units of work that create, update or delete the business lead and the entities. We will represent them by means of operation contracts.

Each service or task has a set of input parameters, a precondition, a postcondition and may have an output parameter. The precondition states the conditions that must be true before the operation can execute. The postconditions indicate the state of the system after the operation's execution. Both preconditions and postconditions may refer to the input parameters, but the output parameter can only be modified by the postcondition. We assume that all the elements that do not appear in the postcondition keep their state previous to the execution.

In terms of the language to specify the operation contracts, the best option would be to use a formal or semi-formal language such as OCL. This would avoid ambiguities and errors in the definition of the contracts, and would facilitate their automation. However, as OCL is not as readable as natural language, we opt for the latter in this approach. This specification is used as a starting point for their final implementation, which should be done in a programming language.

For example, Listing 1 shows the specification of action *Create Online Debt Payment* (it also appears in the state machine diagram in Figure 2) in terms of an operation contract in natural language. The precondition of this operation ensures that there is not an existing *OnlineDebtPayment* with the given input *payment_ref*, which acts as its identifier. If this precondition is met, then the

operation creates an *OnlineDebtPayment* with the given input values, together with a *Debt*, *Client* and *Agent*. The *OnlineDebtPayment* changes its state to *BeingProcessed*.

Note the redundancy between the specification of this contract and the state machine diagram. That is, the contract specifies that the action changes the state of the lead, although this change is also reflected by the state machine diagram. Although in general it is best to avoid redundancy, in this case we use the state machine diagram to have a more intuitive representation of the evolution of leads and entities, but not as main component to determine the effects of the actions.

Listing 1. Specification of *create online debt payment*

```
create_online_debt_payment(p_reason: String, payment_ref: String, amount: real, reason:
      String, emailAg: String, nameAg: String, idCl: String, nameCl: String, emailCl:
      String, phoneCl: int, )

precondition:
- There is no OnlineDebtPayment whose payment_reference = payment_ref

postcondition:
- A new OnlineDebtPayment is created with the following values: reason = p_reason, reference
      = payment_ref, ticket = ticket, instalmentsAllowed = false, delayedPaymentAllowed =
      false and availablePaymentMethod = {BankTransfer,CreditCard}
- A new Debt is created with amount = amount and reason = reason
- A relationship is created between the new Debt and the new OnlineDebtPayment
- A new Client is created with id = idCl, name = nameCl, email = emailCl and telephone =
      phoneCl.
- A relationship is created between the new Client and the new OnlineDebtPayment
- A new Agent is created with email = emailAg and name = nameAg.
- A relationship is created between the new Agent and the new OnlineDebtPayment
- The created OnlineDebtPayment is in state "BeingProcessed"
```

2.4 Associations

Associations establish restrictions in the way the services, or actions, may make changes to the artifacts. In many cases they are represented using some kind of workflow diagram. However, as we mentioned in the Introduction, using workflows is not appropriate in this case, as we do not wish to force the execution of the actions in any particular order. On the contrary, we wish to restrict the actions as little as possible to allow for maximum flexibility.

Therefore, we will use condition-action rules to determine when an operation or task will be available to execute or will execute automatically. The conditions indicate the circumstances under which the action can execute. They may refer to content that is available on the information base but not to the actions that may have executed previously. They will have the following form:

`if <cond> then <allow execution | execute> opName`

By defining rules in this form, it is possible to automate them using rule management engines such as IBM ODM. We distinguish between two types of rules:

1. **Action enablers**: They allow the execution of actions, showing the conditions which must be true for an action to execute, but by themselves they do not trigger the action's execution. We use the keywords `allow execution` in their definition. The conditions may take into consideration the following

elements: the communication channel (e.g. website, phone), the user profile, the content in the information base, or time.
If an action is always available, we use the keyword *true* as a condition in the condition-action rules.

2. **Action triggers**: When certain conditions are met, these rules automatically execute an action. We use the keyword `execute` in their definition. The condition of these rules refers to the content of the information base or time.

The rule below is an example of a rule that enables an action. It states that *create online debt payment* may execute at any time (condition *true*):

```
if true then allow execution create_online_debt_payment
```

The second rule below shows that *remind client of payment deadline mail* executes automatically under very specific circumstances: the *OnlineDebtPayment* must be in state *BeingProcessed*, a delayed payment must be allowed and the current date is seven days before the payment is overdue.

```
if online_payment_debt is in state ‘‘BeingProcessed’’ and
(delayedPaymentAllowed = true) and (today() + day(7) = deadline)
then execute remind_client_of_payment_deadline_mail
```

3 The Tool: Balandra

There is already at least one tool, Balandra [12], which is available commercially and that it is able to provide the required degree of management and control over the processes which deal with digital customers.

The companies that use the tool can specify their requirements and use the Kopernik approach to define their models. Afterwards, these models can be easily incorporated into Balandra, which allows them to automatically generate and customize a software system to manage the behavior of digital customers according to the models.

Figure 5 shows the visualization of the business artifacts in the tool. Notice that in this case it has the form of a tree-like structure, that is equivalent to the class diagram we showed in Figure 1. New instances of the different leads and entities can be created using the graphical interface of the tool.

For the lifecycle dimension, there is no graphical representation in the form of a state machine diagram. The internal states of an artifact, be it a lead or a entity, are shown next to the artifact in the tree-like structure in Figure 5. In contrast, tagsets can be visualized using the tool in the form of a table, similar to Figure 3. Due to space limitations, we do not show them here.

Given a certain lead, the tool can also show the available actions that can be executed over that lead. See Figure 6 for an example. The attributes and entities related to lead appear on the left-hand side, whereas the list of actions appears on the right.

Finally, the condition-action rules can be easily defined in Balandra using structured natural language.

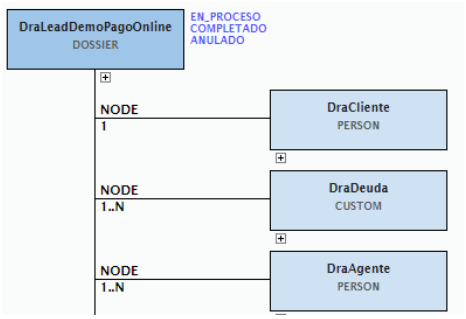

Fig. 5. Screenshot showing the lead and some of the entities related to it.

Fig. 6. Screenshot with the available actions to execute

4 Related Work

The approach we propose in this paper is related to the work presented in [9]. However, in [9] the execution order of the tasks or services is established by an activity diagram. Moreover, the authors do not distinguish between internal states and tagsets, and only represent states for the lead artifacts in this proposal.

On the other hand, the GSM approach [7] shows the stages in the evolution of an artifact and the guard conditions, but adding the concept of milestone, representing a condition that closes a state. It does not enforce the execution of actions or services in a certain order and, in fact, may be executed simultaneously. It does not distinguish, however, between internal states and tagsets, and artifacts are represented as a set of attributes. Based on GSM, CMMN [13] is used to represent flexible processes whose tasks may not require a specific execution order. However, it is mainly limited to the association dimension of the framework.

Other flexible approaches are presented in [1, 5, 4, 2, 6, 10]. This flexibility is achieved by using condition-action rules or preconditions to limit the execution of the services. However, they are based on logic, which makes them formal and unambiguous, but impractical from the point of view of the business. Moreover, they do not have a specific representation for states of an artifact's lifecycle.

Finally, [3] uses natural language to represent services and associations. The associations are also represented using condition-action rules and the services using pre and postconditions. Natural language makes the specifications easier to understand than those defined in logic. However they are more error-prone. This approach does not distinguish either between internal states and tagsets.

5 Conclusions

We have presented an approach for modeling business processes following an artifact-centric approach with the goal of modeling processes for digital customers which require a high degree of flexibility, to allow the client to execute the actions at any time and in any order. Unfortunately, current approaches do not fit well with modeling the flexibility required by this kind of processes.

As we have shown, our approach can be easily adopted to use in an existing tool, Balandra [12], which models processes for digital customers requiring a high degree of flexibility, and which has been used commercially and successfully in big companies such as ING direct or Zurich connect.

References

1. Bagheri Hariri, B., et al.: Verification of relational data-centric dynamic systems with external services. In: PODS. pp. 163–174. ACM (2013)
2. Belardinelli, F., et al.: Verification of deployed artifact systems via data abstraction. In: ICSOC 2011. LNCS, vol. 7084, pp. 142–156. Springer (2011)
3. Bhattacharya, K., et al.: A Data-Centric Design Methodology for Business Processes. In: Handbook of Research on Business Process Management, pp. 1–28 (2009)
4. Calvanese, D., et al.: Ontology-based governance of data-aware processes. In: Krötzsch, M., Straccia, U. (eds.) RR. LNCS, vol. 7497, pp. 25–41. Springer (2012)
5. Cangialosi, P., et al.: Conjunctive artifact-centric services. In: ICSOC 2010. LNCS, vol. 6470, pp. 318–333. Springer (2010)
6. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. ACM Trans. Database Syst. 37(3), 22 (2012)
7. Damaggio, E., Hull, R., Vaculín, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with Guard – Stage – Milestone lifecycles. Information Systems 38(4), 561 – 584 (2013)
8. Estañol, M., et al.: Artifact-centric Business Process Models in UML. In: BPM Workshops 2012. LNBIP, vol. 132, pp. 292–303. Springer (2013)
9. Estañol, M., et al.: Specifying artifact-centric business process models in UML. In: BMSD 2014, Revised Selected Papers. LNBIP, vol. 220, pp. 62–81. Springer (2015)
10. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Fagin, R. (ed.) ICDT. vol. 361, pp. 225–238. ACM (2009)
11. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: OTM 2008. LNCS, vol. 5332, pp. 1152–1163. Springer (2008)
12. Léelo: Balandra (2017), http://balandrasw.com
13. OMG: Case Management Model and Notation (CMMN) 1.1 (2016), http://www.omg.org/spec/CMMN/1.1/PDF