Business Rules and BPMN

Modeling Scenario

Let's say we want to model a process in BPMN and the process induces some business rules. We will use the example of creating a bill. In order to create the bill, a discount needs to be computed. The sum of the order and the customer type are the relevant criteria to compute the discount.

This is a very simple example which will show us where to apply BPMN and where not to.

The Solution as BPMN 2.0 Diagram

Rule EngineCreate BillBillrequestedComputediscountCreate billBillcreated

Explanation

During modeling, we focus on the process flow. In this example, the process has two steps. A discount is computed before the bill is created. The result is a very simple process.

It does not make sense to model the calculation of the discount itself in the BPMN model (see the example below). For the rules decision tree, for every additional criteria, the cardinalities will grow exponentially. That is not what we want in a BPMN model.

Therefore, it makes sense to separate process and business rules.

The Wrong Way to Model It

Create BillCompute 2%discountadd an extra 1%discountcustomertype?BillrequestedSum of order?customertype?Create billBillcreatedCompute 3%discountCompute 4%discountcustomertype?add an extra 1%discountadd an extra 1%discount1000 –1500500 – 999>2000< 500Type AType AType Aordinaryordinaryordinary

Dependent Instances

Modeling Scenario

Let's say we want to model a process with concurring instances. We are using a simple example. If one credit check of a customer is running, we do not want another credit check for the same customer to be performed at the same time.

The reason could be that the total number of credit checks performed influences the result of the check.

Let's assume that we are running a credit check for a customer and we get a second request for the same customer at the same time.

What all solutions have in common is that every new instance needs to check for concurring instances on the data level before starting the actual credit check.

Solution with Signal Event

Creditworthiness CheckCheckrequestedcheck forrunninginstances(of samecustomer)running instancesof same customer?perform creditcheckcredit checkperformedcredit checkperformedEngineDatabasenoyes

Explanation

The signal event is the easiest and most compact way to model the interaction between different instances. The problem of the signal is that it functions as a broadcast and does not address any specific instance. So, strictly speaking, the customer is ignored and all waiting instances catch it.

Solution with Message Event

Creditworthiness CheckCheckrequestedcheck forrunninginstances(of samecustomer)running instancesof same customer?perform creditcheckcredit checkperformedwaiting instancesof same customer?check for waitinginstances (ofsame customer)running instancefinishedinformwaiting instanceEngineDatabaseEngineDatabasenonoyesyes

Explanation

This solution is a bit more complex, since you need to determine the recipient (a single instance) of the message. That induces a second data request before the end of the instance. However, this is the correct way to solve the problem that occurs in the signal event solution.

Solution with Timer and Loop

Creditworthiness CheckCheckrequestedcheck forrunninginstances(of samecustomer)running instancesof same customer?perform creditcheckwait sometimecredit checkperformedEngineDatabasenoyes

Explanation

In this example we do not need any communication between instances. The instance itself checks periodicity if it can proceed to the credit check. The downside is that this might cause delays and overhead due to the loop.

Four Eyes Principle

Modeling Scenario

We want to model the following situation using BPMN 2.0. For a request (e.g., a payment) two approvals of two different people are needed. A Process Engine should ensure that both approvals are fulfilled before the request is approved. The manual steps that are performed by the two approvers should also be modeled in the BPMN diagram. The approval decision is performed using a portal with a Tasklist.

The Use Cases

The use cases for this pattern are numerous. Here are some examples:

- Payment Approval

- Invoice Approval
- Contract Approval
- …

The Solution as BPMN 2.0 Diagram

1st ApproverApprovalrequestedevaluate requestdocument andsubmit decisiontaskcompletedProcess EngineApprovalrequesteddecide onapproval(1st stage)approved?requestrejected(1st stage)decide onapproval(2nd stage)approved?requestrejected(2nd stage)requestapproved2nd ApproverApprovalrequestedevaluate requestdocument andsubmit decisiontaskcompletednoyesyesno

Explanation

We use separate pools for the Process Engine, for the 1st Approver and for the 2nd Approver. This way, we can clearly define who is in control of which process.

In the engine pool, user tasks are used. These user tasks correspond to the tasks which are shown in the Tasklist of the 1st and the 2nd approver.

The interaction between the user tasks in the engine and between the manual process of the approvers is modeled using message flows. These message flows encapsulate the manual steps which the approver needs to perform in order to complete the user task.

The Tasklist itself is not modeled, in order to reduce complexity.

Variations

Approver as Collapsed Pools

1st Approver2nd ApproverProcess EngineApprovalrequesteddecide onapproval(1st stage)approved?requestrejected(1st stage)decide onapproval(2nd stage)approved?requestrejected(2nd stage)requestapprovednoyesyesno

Approver Determination with LDAP

1st ApproverApprovalrequestedevaluate requestdocument andsubmit decisiontaskcompletedLDAPProcess EngineApprovalrequesteddecide onapproval(1st stage)approved?requestrejected(1st stage)decide onapproval(2nd stage)approved?requestrejected(2nd stage)requestapproveddetermine 1stand 2ndapprover2nd ApproverApprovalrequestedevaluate requestdocument andsubmit decisiontaskcompletednoyesyesno

Monthly Invoicing

Modeling Scenario

This example explains a very common struggle with structuring BPMN 2.0 diagrams. Let’s say there is a lawyer who offers legal advice to his customers. The service works as follows: The customers can ask for legal advice whenever they need it. The lawyer provides the requested advice and puts the billable hours on the customer’s

time sheet. When the month is over, the lawyer's accountant determines the billable hours based on the time sheet and creates the invoice.

This example illustrates a very common modeling scenario. It's not the steps of the processes that are difficult, it's the structure of the diagram.

The Solution as BPMN 2.0 Diagram

LawyerProvide Legal AdviceLegal Advicerequestedprovide legaladviceregister timeRequesthandledCustomerTime SheetCustomerAccountingMonthly Invoicing1st dayof monthdeterminebillable hourscreate and sendinvoicemoneyreceivedInvoicesettled14 dayssend reminderjust oneinstance permonthmany instancesper month

Explanation

The most important aspect of the diagram is its structure.

The Provide Legal Advice process is performed many times per month. The Monthly Invoicing process is only performed once a month. Therefore, these two processes should be modeled as separate pools.

Of course these two pools are not completely independent from each other. Why? They work on the same data – the customer's time sheet. Our ability to model such a data-related connection is very limited in BPMN. This is due to the fact that BPMN is focused on control flow rather than on data flow.

However, we can use the data store element to model this connection on the data level.

The Wrong Way to Model It

LawyerProvide Legal AdviceLegal Advicerequestedprovide legaladviceregister time1st of nextmonthdeterminebillable hourscreate and sendinvoicemoneyreceivedInvoicesettled14 dayssend reminderCustomer

Explanation why this is wrong

In this example, both processes are mixed into one. This is – at best – a very implicit way to model it. It would mean that for every provided legal advice an invoice is sent once the month is over. This way of modeling is wrong in most cases.

Additional Information Required after User Task

Modeling Scenario

Let's assume we want to model the following scenario: we want to execute a user task which is performed by a user in a portal. After the user task is completed, additional information might be required. If that is the case, the process engine sends an information request either to another user (solution 1) or to a technical service (solution 2).

Solution 1: Request information from another User

User in PortalUser in PortalProcess Enginesome task forthe useradditional informationrequired?requestinformation(from user)…noyes

Solution 2: Request information from a technical service

User in PortalSome Technical ServiceProcess Enginesome task forthe useradditional informationrequired?send informationrequest (technical)informationreceived…noyes

Processing a Batch of Orders from a Marketplace

The Situation

We want to model the following scenario using BPMN 2.0: let's assume a company receives orders from different distribution channels. One of these channels is a marketplace. Within certain intervals of time, the orders from the marketplace are fetched as a batch. Every order in this batch needs to be validated before being imported into the ERP System.

The Solution as BPMN 2.0 Diagram

ERP SystemSome MarketplaceImport Orders from Marketplace to ERPEvery 10minutesCollect allorders frommarketplaceProcess OrderNew singleorderCheck orderdatadatacorrect?Import order toERP systemSingle orderprocessedOrder dataincorrectAll ordersprocessedfor each singleordernoyes

Explanation

This example shows a very common modeling scenario. We sometimes call it a 1-to-n problem. One process instance (Import of Orders) results in many single process instances of another process (ERP System). Typical constructs are multi instance or loops that start other processes using messages (message flows).

Reassigning User Tasks

Modeling Scenario

This example shows a very common modeling scenario. We sometimes call it a 1-to-n problem. One process instance (Import of Orders) results in many single process instances of another process (ERP System). Typical constructs are multi instance or loops that start other processes using messages (message flows).

Solution 1: Message boundary event and reassignment service

User in PortalProcess Enginedetermineassigneesome user task…assigneeunavailable

Note

This makes sense if the engine calls a service to determine the new assignee.

Solution 2: Message boundary event and reassignment rules

User in PortalProcess Enginedetermineassigneesome user task…assigneeunavailable

Note

This makes sense if the engine calls a rule engine to determine the new assignee.

Solution 3: Message boundary event and implicit reassignment

User in PortalProcess Enginesome user task…assigneeunavailable

Note

This makes sense if the engine determines the new assignee itself, e.g., by using an expression.

Two Step Escalation

Modeling Scenario

We will use the following example to illustrate how to model a two step escalation using BPMN 2.0. When we want a pizza, we order one. Sometimes the pizza delivery screws up and the delivery takes longer than 30 minutes. Then we complain to the delivery service. After that, we give them another 20 minutes to deliver the pizza. If they do not make it in time, we give up and cancel our order.

Solution 1: Two Event-Based Gateways

PizzawantedOrder PizzaPizza receivedEat PizzaPizzaeaten30 minutesComplain toDelivery ServicePizza received20 minutesCancel OrderOrdercancelled

Advantages of this solution

This solution very explicitly shows how the two step escalation is performed. Timers are modeled separately, followed by their corresponding escalation activities.

Disadvantages of this solution

The event-based gateway is not an intuitive BPMN symbol of the BPMN standard, experience is required.

Using two event-based gateways makes the model larger and leads to a duplication of the “Pizza received” message event.

Solution 2: Receive Task with timers attached

PizzawantedOrder PizzaEat PizzaPizzaeatenComplain toDelivery ServiceCancel OrderOrdercancelledWait for PizzaOrdercomplained50 minutes30 minutes

Advantages of this solution

This model is smaller than the first solution and probably the way most developers would solve the problem on the engine. Since we use a non-interrupting attached timer event, this solution is more flexible when it comes to multiple complaints (e.g., we want to complain every 5 minutes until 50 minutes are over).

Disadvantages of this solution

The receive task is usually not intuitive for the “business guys”, who would rather use message receive events for that kind of wait state.

The way that the interrupting and non-interrupting timer collaborate requires profound understanding of attached events.

Solution 3: One Event-Based Gateway with a generic timer

PizzawantedOrder PizzaPizza receivedEat PizzaPizzaeatentime’s up!Complain toDelivery ServiceCancel OrderOrdercancelledalreadycomplained?timer is more“generic” in thisversionyesno

Advantages of this solution

This model is the compact and generic solution to the problem. If it comes to n-step escalation then you will need this generic approach to avoid huge diagrams.

Disadvantages of this solution

The generic solution is less explicit than the other solutions. We do not see the actual duration of the timers, as a single timer is used for both durations.

For a fast understanding of the two step escalation, this method of modeling is not suitable.

Avoid Crossing Flows

Recommendation

This BPMN example is about creating a good layout of process models. The better the layout, the higher the degree of understanding. That is what we want to achieve when we create process models.

Try to avoid crossing flows as much as possible. This will increase understanding of BPMN process models – for both experienced and inexperienced BPMN users.

Of course it is not always possible to entirely avoid this problem. Bear in mind that it always makes sense to invest some extra time in optimizing the layout in a way that most crossing flows are eliminated.

The examples below illustrate the problem with an abstract example.

Good Example of Handling Flows

processstartedperform taskonerequiredaction?perform task twoprocessfinishedperform taskthreeok?yesplan Aplan Bno

Counter-example

processstartedperform taskonerequiredaction?perform task twoprocessfinishedperform taskthreeok?yesplan Aplan Bno

Naming Conventions

Recommendation

Most important: every BPMN symbol should have a label.

Events should be labeled using object + past participle. Start events should always be labeled with an indication of the trigger of the process. End events should be labeled with the end state of the process.

The process (pool) itself should also always be labeled. This label should indicate the name of the process and the role that is performing it.

Tasks should be labeled using object + verb. This forces the modeling person to focus on what is really done during the task.

X-OR Gateways should be labeled with a question. The outgoing sequence flows should be labeled with the possible answers to these questions (conditions).

Good Example of Naming

Check Order DataCustomer ServicesOrderreceivedCheck orderOrdercheckedOrder datacorrectOrder datacorrect?Order datanot correctyesno

The Generic Version

Process NameRole Performing the ProcessTriggerof processObject + VerbObject + Past ParticipleFirst end stateafter processis finishedQuestion?Second end stateafter processis finishedanswer 1answer 2

Counter Example

Order ProcessStartCheckingset status indatabaseEndFailureok

Symmetric Modeling

Recommendation

This BPMN example is about creating a good layout of process models. The better the layout, the higher the degree of understanding. That is what we want to achieve when we create process models.

We have determined that symmetric structures increase understanding of BPMN process models – for both experienced and inexperienced BPMN users.

Good Example of a Symmetric Model

prepare saladhungernoticedchoose recipedesired dish?cook pastaeat mealhungersatisfiedcook steakdesiredcomponent?Choice:– salad– pasta– steaksteakpastasaladwarmfood

Counter-example

prepare saladhungernoticedchoose recipedesired dish?cook pastaeat mealhungersatisfiedcook steakdesiredcomponent?Choice:– salad– pasta– steaksteakpastasaladwarmfood

Good Example of a Symmetric Model 2

produce freshproductOrderreceiveduse old productfrom stockOrder valueabove 25.000 €?Process orderOrganizeShipmentPackage goodsShip orderOrderprocessedyesno

Counter-example 2

produce freshproductOrderreceiveduse old productfrom stockOrder valueabove 25.000 €?Process orderOrganizeShipmentPackage goodsShip orderOrderprocessedyesno

Use Equal Task Sizes

Recommendation

We recommend to always use equal task sizes.

The reason is simple. People tend to interpret task sizes although they do not have any semantics in the BPMN standard.

Some think that bigger tasks are more important than smaller tasks – according to BPMN that is wrong.

Some think that bigger tasks take more time than smaller tasks – according to BPMN that is wrong.

You can easily avoid that confusion by using equal task sizes.

Good Example of Equal Task Sizes

1st ApproverApprovalrequestedevaluate requestdocument andsubmit decisiontaskcompleted

Counter-example

1st ApproverApprovalrequestedevaluaterequestdocument and submitdecisiontaskcompleted