Case study: Comparison of different modeling languages

Ing. Matej Minarik
Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 1/2, 612 66 Brno, Czech Republic
iminarikma@fit.vutbr.cz

Abstract

1. Introduction

We would like to compare three different modeling languages in this article. We will introduce a non-trivial business process example from real world and try to represent this process in three chosen languages.

These modeling languages are: Petri Nets, UML Activity diagrams and BPMN (Business Process Modeling Notation). Petri Nets were first introduced almost 80 years ago, they have strong mathematical and formal background. They are used for business process modeling because they have some beneficial properties, which are used for process validation and analysis. UML was standardized two decades ago, but it has been used for many decades before. UML 2.0 standard, which we use in this article was standardized in 2005. UML was developed in order to model software architectures, both statically and dynamically. UML activity diagrams are common for business process modeling. BPMN is a notation developed specifically for modeling business processes. BPMN 2.0 was standardized in 2005.

1.1 Process example

An example process is a variation of pizza ordering. This process is not trivial, nor too complicated and in our opinion many people are quite familiar with this process.

The process starts when a customer is hungry. The first step would be to open pizza delivery website or find some handout and select a pizza. Then the customer orders the selected pizza either online or via a phone. After approximately 1 hour, he or she will pick up a phone and ask about that order, if it did not arrive. When the pizza arrives, the customer will pay for it and eat it.

Pizza delivery is a local small family company with one delivery guy, some stuff in their kitchen and one shop clerk which is responsible for communication with customers and other things, not related to this process. The shop clerk will receive an order via email or phone. He will tell the chef to bake the ordered pizzas. If a customer will be asking about the order later on, the clerk will calm him down. The kitchen chef is responsible for baking. In this process we will not go into details of baking, but it could easily be represented by a sub-process or another business process. After the pizza is ready, it is handed out to the delivery guy, who is responsible for the delivery and for receiving the payment from the customer.

2. Petri Nets[5]

Petri Nets are a great process modeling technique, which was developed by Carl Adam Petri in the sixties. Petri Nets have found their use in many different domains. Recently it has been extended with color, time and hierarchy, which facilitates the need to model complex business processes. The most important features of Petri Nets are: *formal semantics*, meaning that it has been defined formally; *graphical nature*, ensuring easier communication; *expressiveness*, meaning possibility to model simple and complex processes; *properties*, based on strong mathematical foundation; *analysis*, which is able to prove or disprove the properties; *vendor independent*, it can be used with different software packages.

2.1 Classical Petri Nets

The Petri Net is a directed bipartite graph with two types of nodes, *places*, represented by circles and *transitions*, represented by rectangles. *Arcs* are used to connect these nodes. Nodes of same type cannot be connected.

Definition 1. A Petri Net is a triple (P, T, F):

- P is a finite set of places,
- T is a finite set of transitions, such that $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs

If there exists a directed arc from place p to transition t, p is called an input place of the transition t. Similarly, if there exists a directed arc from transition t to place p, p is called an output place of the transition t. A set of input places of a transition t can be denoted by $\bullet t$. A set of output places of a transition t can be denoted by $t\bullet$. Similarly, we could define $p\bullet$ and $\bullet p$.

Places are special types of nodes, which can hold a so called *tokens*. Tokens move throughout the net and define a state, known as *marking*, by current distribution. The state can be represented as follows: $1p_1+3p_2+4p_3$, which means that the place p_1 holds one token, place p_2 holds three tokens and place p_3 holds four tokens. The states can be compared by a partial ordering defined as follows: $M_1 \leq M_2$, if $\forall p \in P : M_1(p) \leq M_2(p)$.

The state can be modified by changing the amount and positions of tokens. Tokens have to obey a so called *firing rule*:

1. transition t is enabled, if each input place of t contains at least one token
2. if transition t is enabled, it may fire, which means, that t consumes one token from each input place and produce one token to each output place

Given a Petri Net (P, T, F), we can define a following notations for state transitions:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing it results in state M_2, this can be also written as $M_1 \rightarrow M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1t_2t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n
- M_n state is called *reachable* from state M_1 ($M_1 \rightarrow *M_n$), if there is a firing sequence σ, such that $M_1 \xrightarrow{\sigma} M_n$. Empty firing sequence is also allowed.

(PN, M) denotes a Petri Net PN, with an initial state M.

Definition 2. A Petri Net (PN, M) is *live*, if for every reachable state M' and every transition t, there is a state M'', reachable from M' which enables t.

Definition 3. A Petri Net (PN, M) is bounded, if for every place p, there is a natural number $n \in N$, such that for every reachable state, the number of tokens in place p is less than n. If n is for every place at most 1, the net is considered *safe*.

Definition 4. A path C from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \ldots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$. $\alpha(C) = \{n_1, n_2, \ldots, n_k\}$ is the alphabet of C. C is elementary, if for any two nodes n_i and n_j on C, $i \neq j \Rightarrow n_i \neq n_j$

Definition 5. A Petri Net is strongly connected if for every pair of nodes x and y, there is a path leading from x to y.

2.2 Extended Petri Nets

Petri Nets are useful when modeling business processes. However, real world processes tend to be complex and large. Moreover, classical Petri Nets lack support for time and data modeling. To overcome these issues, some extensions were proposed and formally defined. One of the extensions is (1) color, to model data. In classical Petri Nets, tokens represent objects, which are transformed during the process. In order to model state of these objects, *Colored Petri Nets* define a *color*, which represents the current state of the token. Token can change its *color* during the process and it is possible to define a 'precondition' for each node, to take token color into consideration before its processing.

Another possible extension of Petri Nets is time (2). Real world tasks take usually a noticeable amount of time. In order to model and simulate this behavior, time can be associated with tokens, places or transitions.

Real world processes tend to be complex, thus corresponding Petri Nets are usually large and lack the ability to give a different level of process abstraction. Hierarchy (3) construct, called *subnet* was presented to overcome this. *Subnet* is an aggregate of places, transitions and other subnets, which can represent some parts of complex processes as a single element.

WorkFlow net (WF-net) is a special Petri Net, which models a workflow process definition. A WF-net has only one input place (i) and one output place (o). Input and output place in a WF-net is similar to initial and final states in a traditional state machine. If there is a token present in the input place, case represented by the WF-net needs to be *handled*. If there is a token present in the output place, case represented by the WF-net is *done*. Every place and transition has to be on a path from the input place i to the output place o. It means, that every place and transition should contribute to the processing.

Definition 6. A Petri Net $PN = (P, T, F)$ is a *WF-net* if and only if:

- PN has two special places i and o, where place $i(\bullet i = \emptyset)$ is the input place and $o(o\bullet = \emptyset)$ is the output place,
- if we add a transition t' to PN which connects o with i, then the resulting net is *strongly connected*

2.3 Modeling with Petri Nets

Sequential routing is easy to express. Causally dependent tasks, represented by transitions are connected to interleaving places by arcs. Sequential routing in figure 1 represents causal dependency between tasks A and B, meaning that B have to be processed after A.

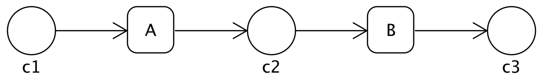

Figure 1: Simple sequential routing

Parallel routing is used for tasks, that can be processed in parallel, or which order of execution is less strict. This kind of routing is usually modeled by AND-split in the beginning and AND-join in the end. AND-split and AND-join are modeled with transitions. In figure 2, transition A represents AND-split and transition D represents AND-join. Transition A will produce token to both *c2* and *c3*

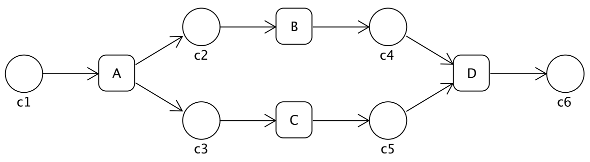

Figure 2: Simple parallel routing

places. Transition *D* will fire, once tokens from *c4* and *c5* are consumed, meaning that tasks *B* and *C* are done.

Conditional routing allows that the same process acts differently for each run. In order to model conditional routing, OR-split and OR-join, represented with places, are used. In figure 3 place *c2* represents OR-split and place *c3* OR-join. The token produced by transition *A*, have to be consumed by either transition *B* or *C*, which is ensured by place *c3*. Places *c2* and *c3* make sure, that either transition *B* or transition *C* will be processed, but *not* both.

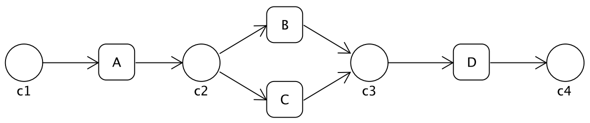

Figure 3: Simple conditional routing

This behavior is generally *non-deterministic*, to make it *deterministic*, we may either use Colored Petri Nets (CPN) with a precondition in both transitions *B* and *C*, or we may use a so called *explicit OR-split*, which explicitly sets a condition, based on which a subsequent behavior is determined. Figure 4 shows a simple explicit-OR routing. The main difference between explicit-OR and implicit-OR is in the *moment of choice*. In explicit-OR example, the choice is made right after transition *A* is completed, while implicit-OR choice is made as late as possible.

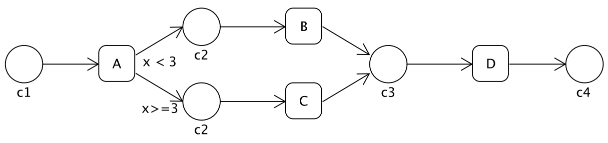

Figure 4: Explicit-OR example

Iteration The last stepping stone of modeling dynamic behavior is *iteration*. Iteration itself may lead to undesired repetitive behavior, during which the process itself does not make any progress. However, there are cases, where iteration is necessary, such as the need to repeatedly ask for information, until it is correct and complete. In figure 5, transition *B* may be repeated several times, before transition *C*.

2.4 Asynchronous triggers

Until now, we assumed that once the task, represented by the specific transition is enabled, it will automatically fire. However, in real world, we cannot process all the tasks,

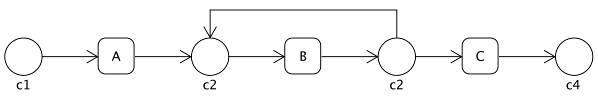

Figure 5: Simple iteration

that are enabled, immediately. The specific task may, for example require user interaction. Users are mostly humans and humans are unpredictable. The process may require approval by manager, but the manager could be out of office for a couple of days. In order to align workflow management processes with a real world situations, we need to introduce some kind of asynchronous task *triggering*.

A trigger is an external stimulus, which leads to the execution of enabled task. The task is able to be triggered, *only if* it is enabled. There are several types of asynchronous triggers: (1) a task is triggered by *user*. Users can have their own 'to-do lists', which contain all tasks, that are enabled and require a user interaction in order to be executed. (2) an external event (message), such as a phone call or email can trigger an enabled task. (3) a task may be enabled by an internal application clock. The time can be specified as a relative duration (e.g. 15 minutes), or it could be specified as an absolute time stamp (e.g. 10-10-2016 14:30).

2.5 Soundness property

Although, Petri Nets have not been developed for business process modeling and they may seem a little harsh at first, they come with some properties, which comes in handy when it comes to verification. One of these properties is *soundness*. This property ensures, that a Petri Net, which is sound will terminate eventually and when it does, there is just one token in the sink place *o*.

Definition 7. A process modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:

- For every state M, reachable from state i, there exists a firing sequence leading from state M to state o. Formally:

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

- State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

- There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M'$$

In figure 6 is an example of a Petri Net, which is not sound. After transition *C* fires, there are two tokens in *c3* and *c4*. Then, transitions *B*, *D* and *E* fires, which will eventually end up with tokens in places *c2* and *c5*. This state violates the definition of soundness, because place *c5* is a sink place and when a token reaches it, there should be no other token.

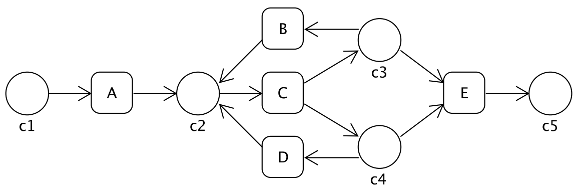

Figure 6: Petri Net, which is not sound

2.6 Resources

Tasks in business processes are executed by *resources*. Resources are durable objects (they cannot be created or destroyed), which can be claimed and released during the execution. The most common types of resources are people (human resources) and machines (servers, printers, ...).

2.6.1 Resource-Constrained Workflow Nets

Workflow nets do not take into account resources available for the execution. In [6] they introduced *Recource-Constrained Workflow Nets* in order to include information about resources into the model. Every resource is of a specific type. Each type has its own place in the model, where all free resources are located. These resources are claimed and released during the process execution. *Production net* is a net, which is abstracted from the resource places.

Definition 8. WF-net $N = (P_p \cup P_r, T, F_p^+ \cup F_r^+, F_p^- \cup F_r^-)$, where:

- P_p is a set of places
- P_r is a set of resources
- F_p^+ and F_p^- are mapping functions from transitions to places and from places to transitions respectively
- F_r^+ and F_r^- are mapping functions from transitions to resources and from resources to transitions respectively,

with initial place $i \in P_p$ and final place $f \in P_p$ is a Resource-Constrained Workflow net (RCWF-net) with the set P_p of production places and the set P_r of resource places if and only if:

- $P_p \cap P_r = \emptyset$
- F_p^+ and F_p^- are mappings $(P_p \times T) \to \mathbb{N}$
- F_r^+ and F_r^- are mappings $(P_r \times T) \to \mathbb{N}$
- $N_p = (P_p, T, F_p^+, F_p^-)$ is a WF-net, which we call a *production* net of N

In figure 7 is a simple example of a Petri Net with resources. Resources are modeled by places r_1 and r_2. Transitions B and C are enabled if and only if there is a free resource, which will execute them. Recall, that these resources can be people, so if a particular person is available to execute some task, the black dot will appear in one of these places.

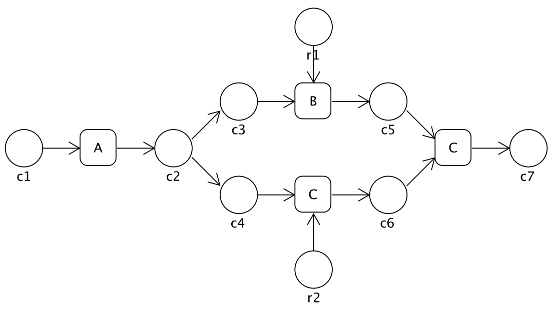

Figure 7: Petri net with resources

2.6.2 Resource-Constrained Colored Petri Nets

Another way to represent resources in Petri Nets is with *Resource-Constrained Colored Petri Nets* as described in [2]. Resources may be included in the *color* of the tokens. Please note, that token color does not really has to be a color, represented for example in RGB. Token's color may in fact be any value, or set of values. If one would insist on representing these values as colors, there may be generated a mapping table. This mapping table would have all possible values mapped onto RGB (or another) color space. These resources may be managed by a central resource manager, which would serve as a centralized source of truth.

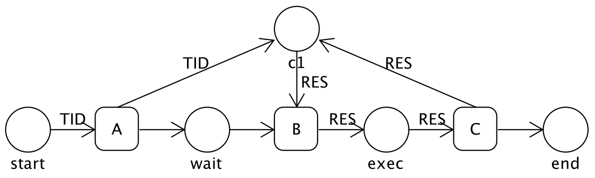

Figure 8: Example of Colored Petri Net with a resource manager

In figure 8, the resource manager is represented by c_1 place. Task id (TID) is sent to the resource manager, which then responds with a requested resource (RES). After resource *RES* is claimed, transition *B* is enabled and fires, which represents executing a task by a person or a machine. After the task is done, transition *C* is enabled, the resource *RES* is freed and resource manager is informed.

2.7 Process model

Model in figure 9 is a sound, workflow net. It was modeled using WoPeD[1] (Workflow Petri Net Designer), which is an open source petri net modeler developed at DHBW university in Germany. This modeler is a great tool for Petri Net modeling, analysis and simulation. At first, It seemed like a daunting task, to model a non-trivial business process using Petri Nets, however this tool made it much easier. Petri Nets support just two types of nodes, which makes them simple and easy to use. On the other hand, it degrades their readability, because one has to be familiar with them in order to understand this model properly. It would be easy to extend this model with human resources, there would be three additional places,

[1]http://woped.dhbw-karlsruhe.de/woped/

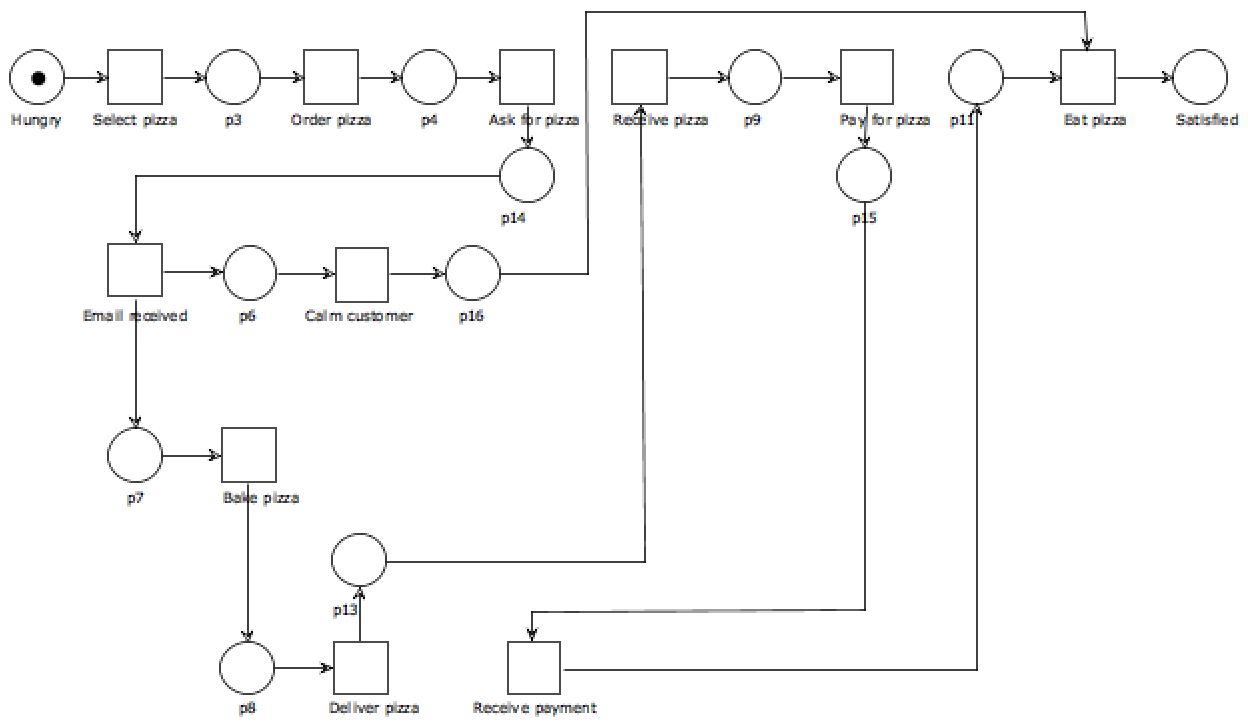

Figure 9: Pizza delivery process modeled with Petri Nets

for shop clerk, kitchen chef and delivery guy, with a single token.

2.8 Pros and cons of Petri Nets

Pros:

- strong mathematical background, formal definition
- simple syntax
- simple validation and analysis
- straightforward token flow

Cons:

- readability, harder to understand
- some modeling background needed
- overwhelming formal definitions
- not straightforward modeling process
- tricky event modeling
- more complex processes would be daunting to model

3. UML[4]

The *Unified Modeling Language* (UML) is a standardized way of expressing and communicating static and dynamic aspects of software. The most recent version is 2.5, however articles, companies and modeling software are in many cases implementing version 2.0, which we will focus on in this article. Models have many benefits. They are relatively cheap, so they can be thrown away later. They are able to express different aspects of software, static connections and dependencies of modules, but also dynamic details of communication. Software designer may choose the right level of abstraction and model type according to audience and purpose of model.

In the context of *Business Process Modeling*, we will examine *Activity Diagrams* (AD), which are commonly used for process modeling. Activity diagrams are quite similar to Petri Nets. The key element of them is *Activity*. *Activity* represents an operation of the system. It may also represent a task, which has to be made by a person. There are several node types, which serve as a split/merge, fork/join. Each diagram has to start in a special node, called *initial node* and end in a *final node.*

3.1 Modeling with Activity Diagrams

Sequential routing is very similar to sequential routing using Petri Nets, except that the imaginary *token* do not have to go through places. Notice, that every diagram have to start in initial node, which is represented by black circle and end in final node, represented by double black circle. These nodes and nodes, representing activities are shown in figure 10.

Parallel routing is represented in figure 11. In order to model parallel activities, we use a special AND-split node. Parallel activities are synchronized with AND-join node. Notice, that split and join nodes are both represented by a thick black horizontal line.

Conditional routing is represented in figure 12. Similar to parallel routing, there are used special nodes, OR-split and OR-join. There are both represented by a diamond. There is also possible to represent *iteration*, an example is in figure 13.

3.2 Events [3]

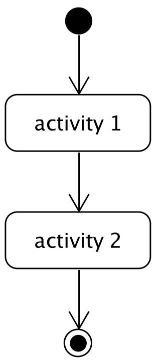

Figure 10: Simple sequential routing

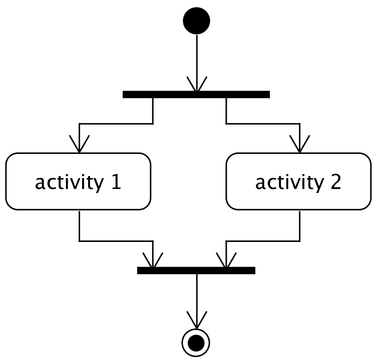

Figure 11: Simple parallel routing

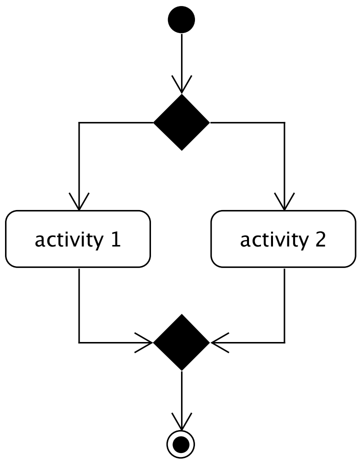

Figure 12: Simple conditional routing

Petri Nets have *asynchronous triggers*, which simulates external events. Activity diagrams are also able to simulate external or time based events.

Accept event action is used to receive external events of specific type. It usually have no incoming edges. After the event is received, the node remains enabled and waits for other events. If accept event has an incoming edge, it is enabled only after the process reaches it. No external

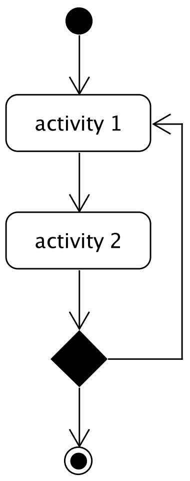

Figure 13: Simple iteration

events are handled by an accept event, until it is enabled. *Emit event action* is used to generate events of specific type. It usually have an incoming edge, which is used to specify when the event should be generated. It might also have an outgoing edge connected to accept event node, which is supposed to catch the event. *Accept time events* are used to generate events based on time, e. g. at the end of month or after two weeks. Event nodes are shown in figure 14.

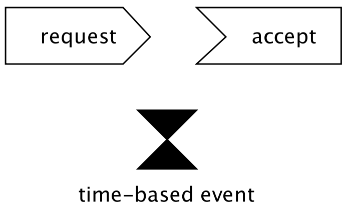

Figure 14: Event nodes in Activity diagrams

3.3 Representing data

Real world processes work with data, e.g. orders, customers, items, etc. Petri Nets are extensible (with color) in order to represent these data. Activity diagrams do not work with tokens, which may represent these data. However, they have different mechanism of dealing with data, which is described in [3].

Activity Parameter Node is an object node for inputs and outputs to activities. They are at the beginning and end of activity diagrams, they accept inputs and provide outputs from processes. Input activity parameter nodes don't have input edges, but they must have output edges and vice versa for output activity parameter nodes. Activity parameter nodes are modeled as rectangles. Invoked activity inputs are placed as tokens on the activity parameter nodes with no incoming edges. After the activity is finished, outputs are placed to activity parameter nodes with no outgoing edges.

Object Node is an abstract activity node that is part of

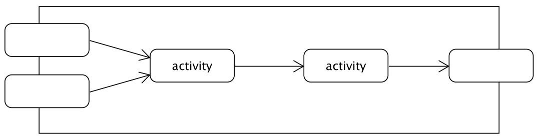

Figure 15: Activity parameter node example

defining object flow. Object nodes contain values at runtime, which correspond to type of object node. If no type is specified, then the values may have any type. There may be multiple instances of the same value present in the same object node. The ordering may be specified for each object node according to which will the values be offered to the outgoing edges. It can be for example FIFO (first in, first out), LIFO (last in, first out) or custom ordering defined by modeler. In the case of custom ordering, there has to be defined a selection behavior for the values. The selection behavior takes all values as an input and produces one particular value as an output. Object nodes are modeled as rectangles with labels in the format *"name:type"*. Selection behavior is specified with the keyword <<*selection*>> and attached to object node as a comment.

Object nodes are connected with *object flow*. Object flow is an ordinal edge, represented by a black arrow, which is able to pass and modify data tokens. Data sent by the source node are all passed to target node. More object flows may have the same source node. In that case, only one of the object flows have access to the data tokens of source node. Once the data is taken by one of the object flows, others do not have access to the data. In order to allow all object flows access to data tokens, they have to be forked.

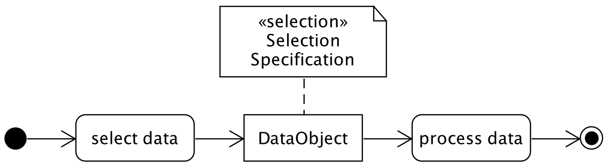

Figure 16: Activity object node example

Routing is generally dependent on process and external data values. There may be preconditions and postconditions specified, which decide the next step in activity flow. These conditions may be based on data existence or particular value. UML is able do model logical preconditions and postconditions based on logical expressions through *Object Constraint Language* (OCL). Preconditions and postconditions may be expressed using OCL. The data-based routing may also be modeled using decision OR-split node and guard conditions on edges.

3.4 Resources

Resources in Activity diagrams can be modeled, according to [3], by *Activity partitions*. Activity partition is an activity group for visual separation of activities, that have something in common. Activity partitions are very similar to *swimlanes*, used in BPMN. Modelers tend to create partitions according to organizational units in a company. Partitions usually have a label and they may be part of an activity partition hierarchy.

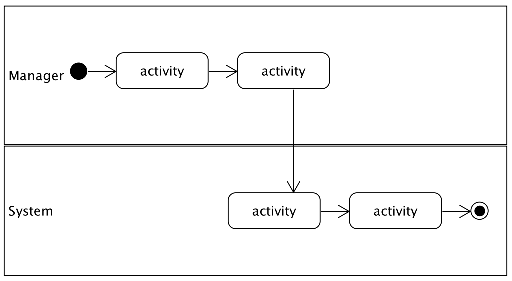

Figure 17: Activity partitions in Activity diagram

In figure 17 there is an example of activity partitions in Activity diagram. There are two partitions, called *S*, which stands for *System*, and *M*, which stands for *Manager*. Partitions do not affect the process itself. They serve as a visual clue to the reader. However, according to [3], partitions may be specified in distinct ways. *Classifier* and *Instance* are relevant in the context of business process modeling, which means that each invocation contained by the partition is executed in the context of the classifier represented by the partition.

3.5 Process model

The process model in figure 18 was modeled using Visual Paradigm[2], which is a professional modeling tool for UML and BPMN diagrams. Syntax of activity diagrams is more complex, than the syntax of petri nets, but it is not too complex to degrade the readability of the diagram. In my opinion, many UML diagrams are designed so that even a person with very little modeling background is able to read and understand the diagrams. Activity diagrams are very similar to Petri Nets, because there is an imaginary token present, which travels through the diagram.

3.6 Pros and Cons

Pros:

- readability
- simple syntax
- clear event modeling
- graphical resource modeling with swimlanes

Cons:

- no formal background
- harder to analyse and simulate

4. BPMN[1]

BPMN is the state-of-the-art in the field of business process and workflow modeling languages. For modeling languages, we can identify three different application domains: *pure description*, *simulation* and *execution*. BPMN can be used for all of these purposes. BPMN as-is, is probably the best option for pure description purposes. For

[2]https://www.visual-paradigm.com/

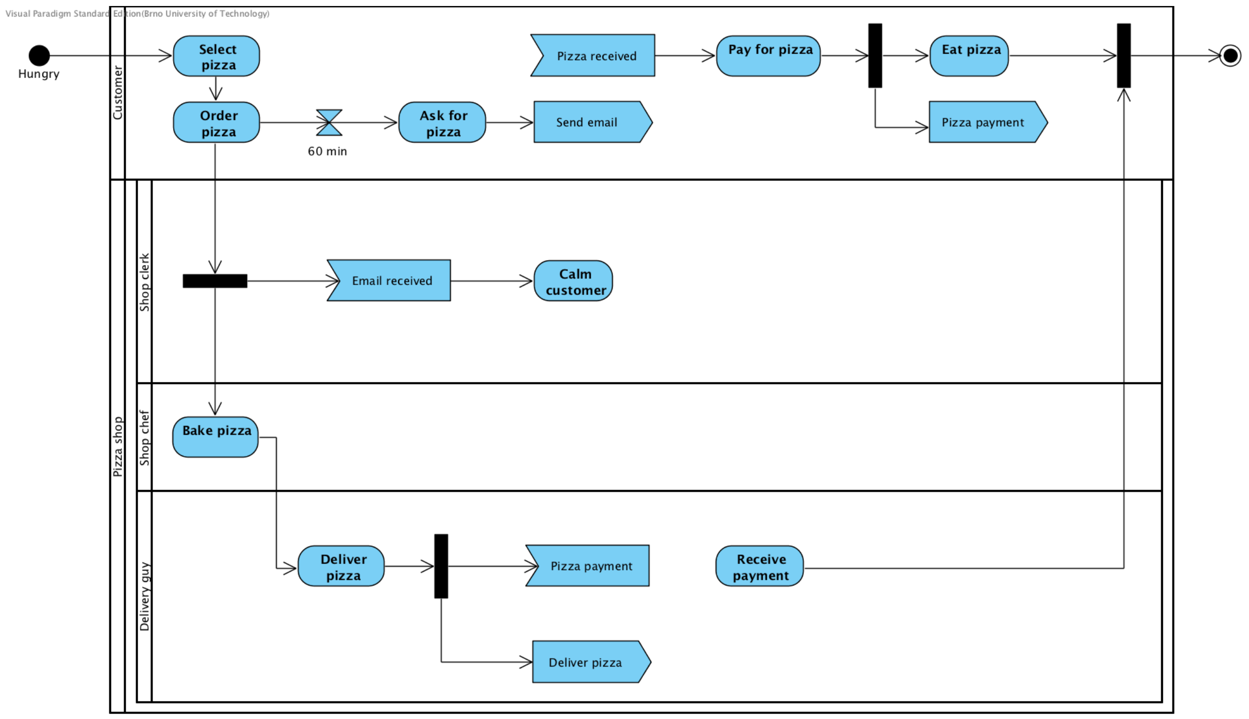

Figure 18: Pizza delivery process activity diagram

simulation purposes, we can use BPMN with XPDL language and WS-BPEL is used to transform BPMN into executable code.

WS-BPEL defines a model and a grammar for describing behavior of the business process. It was considered also for serialization purposes, but it lacks some graphical elements and properties linked to them. XPDL was standardized by *Workflow Management Coalition* (WfMC) and it is an XML-based language, which is aimed at interchange of business processes between different tools, e.g., modeling tools and management suites. XPDL defines an XML-schema for specifying all graphical elements and semantics of a model. It was also widely adopted as an exchange format for business processes.

In terms of validation, there are some tools available, which take advantage of the XPDL ability to represent all aspects of BPMN and provide some validation support. The validation is based on XPDL ability to serialize the model.

4.1 Modeling with BPMN

BPMN is very similar to Activity diagrams, to which we pay attention in section 3. Activities are points in process flow, where the work is done. Activities are represented with a rectangle. Process flow is represented with arrows. When a specific activity cannot be broken down into other sub-activities, it is called a *task*. These tasks may be annotated with additional properties, e.g., *loop*, *multi-instance* and *compensation*. Tasks may be of different type. One possible type is *service task*. Service tasks are tasks which use some kind of service, it may be a web service or an automated application. Other possible types are send task and receive task. Later, we will examine these types further.

Sequential routing is expressed in the very same way, as it is in activity diagrams, an example is in figure 10.

There are several types of gateways, which enable the modeler to express conditional and parallel routing. *Conditional routing* is expressed with the exclusive gateway, which is represented with a diamond and may include a letter 'X' in its center.

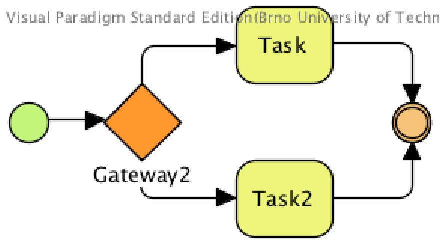

Figure 19: BPMN sequential routing example

Parallel routing is expressed through gateways with a '+' sign in their centers. Parallel split gateway is at the start of parallel tasks and after they are executed, they are merged with parallel join gateway.

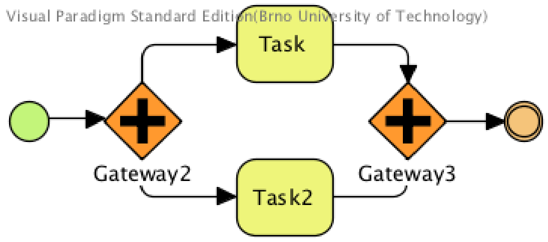
Visual Paradigm Standard Edition(Brno University of Technology)
Task
Gateway2
Task2
Gateway3

Figure 20: BPMN parallel routing example

There is also a special type of gateway, called *inclusive gateway*, which enables for parallel and/or conditional routing. All conditions are evaluated and for each condition, which evaluates to true, its respective path is traversed. This gateway may be also designed in a way that if all conditions evaluates to false, the default path is then traversed.

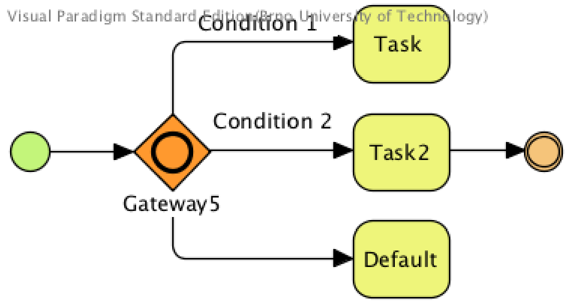
Visual Paradigm Standard Edition(Brno University of Technology)
Condition 1
Task
Condition 2
Task2
Gateway5
Default

Figure 21: BPMN inclusive gateway example

Another special type of gateway is called *complex gateway*. It can be used to model complex behavior. For example, it could specify, that three out of five incoming edges need to be activated in order to activate the gateway. If other two tokens arrive later, the gateway is reset to its initial state, which means that it will be waiting for three enabled incoming edges. The behavior of reset can be customized.

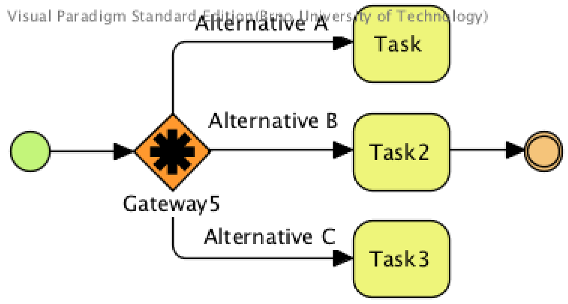
Visual Paradigm Standard Edition(Brno University of Technology)
Alternative A
Task
Alternative B
Task2
Gateway5
Alternative C
Task3

Figure 22: BPMN complex gateway example

The last special type of gateway is called *event-based gateway*. It represents a branching point, which result path is based on events that occur, rather than expressions associated with respective edges. These events are usually triggered upon receiving a specific message, or related to relative or absolute time. For example, a company may have a predefined set of activities specified, if a customer responds "Yes" and another set of activities if the response is "No". If that customer does not respond within a one week period, another set of activities is specified.

4.2 Events

There are two types of events: events that *catch* a trigger and events that *throw* a result. Start events and some intermediate events are catching events. All end and some intermediate events are throwing events. In a typical process, throwing event carries some information outside of its scope to another scope, where is respective catching event. Some events are able to carry data.

Intermediate events happen somewhere between start and end of a process. These events can affect the flow of the process, but they cannot start or directly terminate the process. These events usually: show where messages are

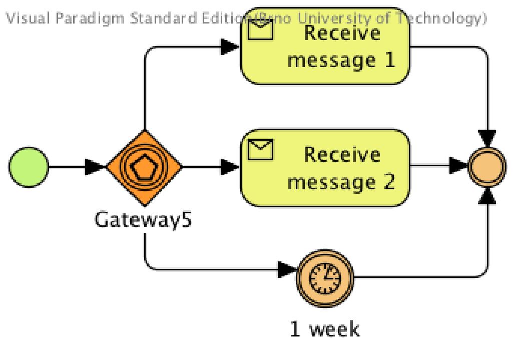
Visual Paradigm Standard Edition(Brno University of Technology)
Receive message 1
Receive message 2
Gateway5
1 week

Figure 23: BPMN event gateway example

expected or sent; show delays in the process; handle the process exceptions or show some extra work needed for compensation. Intermediate events are modeled with an open circle. In the middle of this circle can be placed some marker, which indicates the type of the event.

There are twelve types of intermediate events in BPMN: none, message, timer, escalation, error, cancel, compensation, conditional, link, signal, multiple or parallel multiple. Each type will have a different marker in its center placed. There is a convention, that intermediate events which are placed within the normal process flow can be used as catch or throw events, but events that are placed to the boundary of an activity can be only used as a catch events.

Now, we will describe a few event types:

- *Message* - can be used for sending or receiving a message; the element from which the message is received can be identified by connecting the event to that element
- *Timer* - acts as a delay mechanism based on absolute or relative time (or specific cycle)
- *Compensation* - is used for undoing previous steps, that were already completed, e.g., some cancellation of booked hotel, after we were unable to book the specific flight
- *Conditional* - is triggered, when a condition becomes true
- *Signal* - is used for sending and receiving signals, which are used for general communication within and across process levels, pools and between diagrams; signals do not have specific intended targets, but instead are used as general indicators that some particular event occurred
- *Multiple* - there are multiple triggers assigned to the event; when used as a catch event, only one of the assigned triggers is required in order to activate the event
- *Parallel multiple* - used as a catch event; there are multiple assigned triggers required in order to activate the event

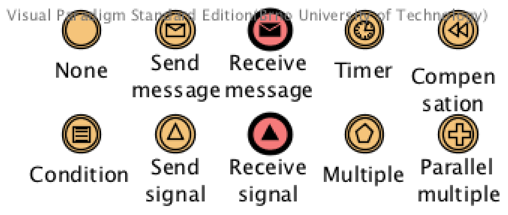

Figure 24: BPMN events example

4.3 Representing data

BPMN itself *does not* provide any means to model and query data items. Instead, it provides hooks that allow for externally defined data structures and query languages. BPMN also allows for coexistence of multiple languages in a single model. As a default data structure and query language, BPMN provides *XML Schema* and *XPath*.

Several BPMN elements are able to store items during process execution. These elements are called *"item-aware elements"*. They are similar to variable construct, which is common in many programming languages. These elements are associated to some kind of item definition, however it is not mandatory and it is up to modeler whether or not will this definition be provided.

The primary item-aware element is called *data object*. These elements must be displayed within the process or its sub-process. These elements may be reused via data object reference, which may specify a different state of the same data object. Data object references cannot contain item definitions and data objects cannot specify states. The name of data object references are constructed from the name of data object and its state. Data object may point to a collection of data, but it has to be visualized differently. See in figure 25.

Data store element provides a persistent storage, where the data will be available beyond the scope of the process. The same store can be visualized via its data store reference multiple times in the same process. The data store element can have a predefined capacity or may be unlimited.

Data input/output elements are visually displayed in the diagram to show the inputs/outputs of the process.

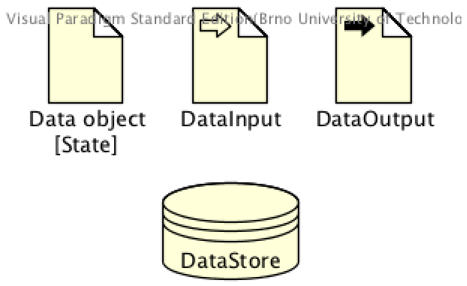
Visual Paradigm Standard Edition(Brno University of Technolo
Data object [State]
DataInput
DataOutput
DataStore

Figure 25: BPMN data example

4.4 Resources

In order to group elements or sub-partition the entire process, we can use *lanes*. Each lane has some text associated with it, which serves as its name of attribute. These text are usually placed on the left side for horizontal lanes and at the top for vertical ones. Lanes are modeled as square-cornered rectangles, drawn with solid single line. Lanes are used to organize and categorize elements inside the process. Their meaning is up to the modeler, however they are usually used to model task resources. In addition, we can use lanes to model internal departments (e.g., shipping, finance) and nest them. There could be an outer set of lanes to model company departments and inner set of lanes to model roles within those departments.

There is an example in figure 26. We can see one containing outer lane, which represents company itself. Then there are two inner lanes which represent departments of the company. Development departments further specifies roles manager and programmer.

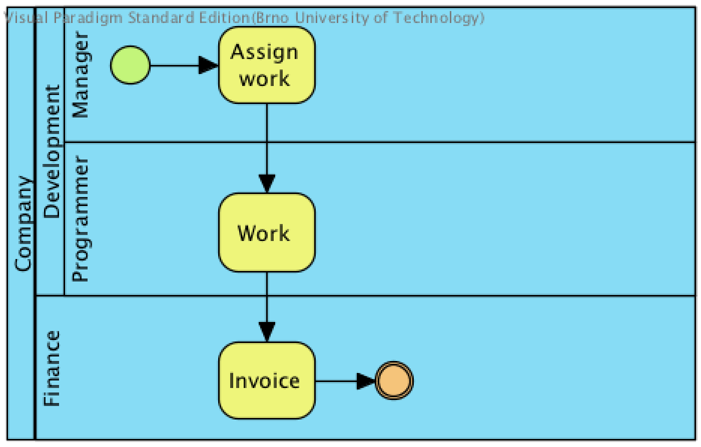

Figure 26: BPMN lanes example

4.5 Process model

In figure 27, there is our sample business process expressed in terms of BPMN. This model was also created using Visual Paradigm software. A good thing is, that BPMN was specifically designed for modeling of business processes, so the modeler has a wide variety of different graphical elements available. On the other hand, this could be also its flaw, because one can spend too much time thinking about the right graphical element for a specific task. I am absolutely aware, that this diagram could be modeled with much greater precision, but I think that the reader will be able to understand the basics of the underlying process.

4.6 Pros and Cons

Pros:

- great variety of graphical elements available
- especially created for business process modeling
- readability
- expressiveness
- great tool support

Cons:

- variety of graphical elements can be overwhelming at first

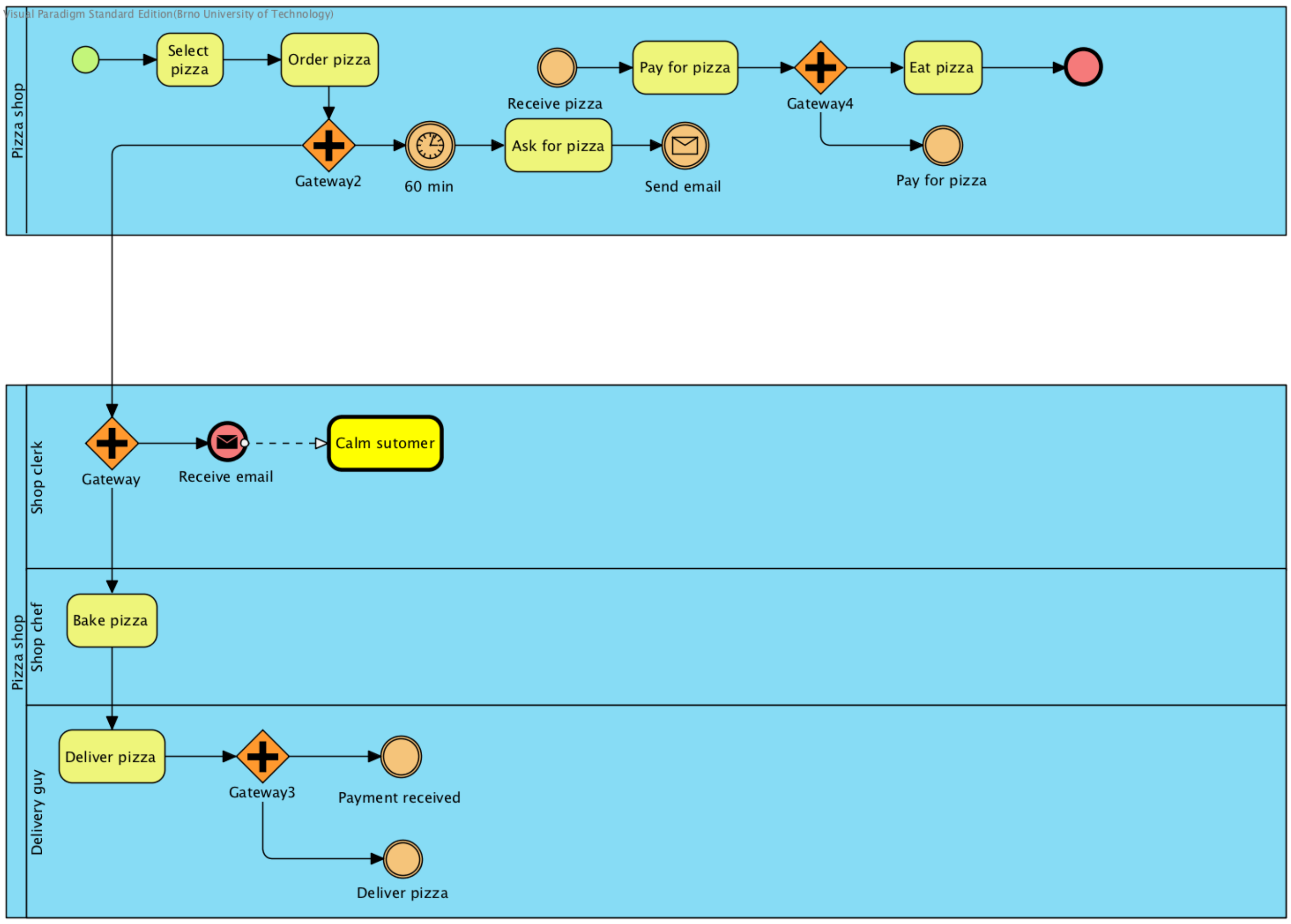

Figure 27: Pizza delivery process BPMN diagram

- no formal or mathematical background
- harder to analyse and simulate
- creating a valid BPMN model can be daunting

References

[1] M. Chinosi and A. Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[2] M. Netjes, W. M. van der Aalst, and H. A. Reijers. Analysis of resource-constrained processes with colored petri nets. In *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 576, pages 251–266. series DAIMI, 2005.

[3] O. M. G. (OMG). Meta object facility (MOF) 2.0 core specification, 2003. Version 2.

[4] N. Russell, W. M. van der Aalst, A. H. Ter Hofstede, and P. Wohed. On the suitability of uml 2.0 activity diagrams for business process modelling. In *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling-Volume 53*, pages 95–104. Australian Computer Society, Inc., 2006.

[5] W. M. Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[6] K. Van Hee, N. Sidorova, and M. Voorhoeve. Resource-constrained workflow nets. *Fundamenta Informaticae*, 20:1–15, 2005.