Conceptual Process Modeling Language: Regulative Approach

Oleg Svatoš
Department of Information Technologies,
University of Economics,
W.Churchill sqa. 4, 130 67 Prague 3

Abstract:
In this work I try to introduce little bit different perspective incorporated into Process Modeling (PM) than contemporary approaches provide. I try to look at processes through tool used by people for centuries - rules and regulations. From this perspective I challenge contemporary technical approaches by an approach that, from my perspective, better captures the substance of a business process as relations among people and not machines. I suggest a new type of modeling notation that incorporates this perspective and also incorporates other features that I miss in contemporary approaches.

Keywords: analysis and design, process modeling languages, conceptual modeling, business process modeling.

1. Introduction
Process modeling and especially Business Process Modeling is a very active filed these days. The businesses are trying to find a way how to keep their control over their dynamically expanding organizations and IT departments are trying to help them to achieve this goal. It is no wonder that many Information System development methods and techniques are spreading into managerial levels. There are two main directions process modeling is heading. One represented by the Business Process Reengineering (BPR) that draws the IS design methods up to the managerial levels and second represented by IS design and especially Service Oriented Architecture (SOA) and service integration (web-services). 
Contemporary methodologies and methods of BPR are well presented in [1] including overview of BPM (Business Process Modeling) techniques/ languages. The goal is to capture the business processes, analyze them and introduce changes that make processes more effective, measurable and kept under control. Overview specifically focused on business process modeling languages with a good background can be found in and [5]. There are many sources on modeling languages itself, but there are few on their comparison. For one of the few good comparisons of conceptual languages see [6].
The SOA approach [22] to business process modeling is different. The goal is to withdraw the business logic out of the code and keep it in different easily manageable/ changeable form – models that can be seen as configuration files for web services orchestration or choreography. Well described in [4]. If there is not only service to service cooperation, but also people are involved, models are called workflows [3]. These models extend the process executing engines by human interface so that the IS users are part of a process.
Interesting thing is that both these branches use the same modeling languages. The question is whether IS modeling tools are appropriate for managerial high-level task or they introduce more problems one has to deal with.

In this paper I will focus on languages used for formal description of processes in BPR. I will discuss their appropriateness for high-level (conceptual) modeling and eventually introduce a new process modeling language for conceptual process modeling.

2. Contemporary Process Modeling Languages

The purpose of this chapter is not to provide complete list and characteristics of all existing process modeling languages but instead it tries to capture their variety encountered in contemporary methods. It should provide the reader with a basic overview of existing modeling languages, showing that they all share the same concept of event and activity flow, which is criticized in the Critique of Available Process Modeling Languages chapter. The main focus, in the listed characteristics, is on the way how the modeling language captures the dynamics of a system (activity/work flow) rather than on what all concepts the modeling language has since the basic problem of using contemporary modeling languages for conceptual modeling does not lie in the number of concepts a modeling language is supporting but in the way how activities and their flows are treated.

2.1 UML Activity Diagram

The Activity diagram is defined by OMG [8] as:

Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.

The core concepts in their notation in fig 1.

![Activity Diagram Core Concepts](image)

**Figure 1:** Activity Diagram Core Concepts [8]

Following example should give a clear idea, how an Activity diagram depicts a process (fig 2).
Figure 2: Example of an Activity Diagram [8]

All represented flows are control flows except the one related to the “Request Order” object and those related to “Invoice” object – those are object flows.

The Activity diagram is very well readable notation coming from a large family of modeling notations focused on modeling different aspects of an information system, including Class Diagrams [8]. It is not pure formal language. It contains concepts like complex decisions, postconditions and preconditions that cannot be subject to machine interpretation.

2.2 BPMN/ BPDM

BPMI /OMG [7] specifies BPMN as:

The Business Process Modeling Notation (BPMN) specification provides a graphical notation for expressing business processes in a Business Process Diagram (BPD). The objective of BPMN is to support business process management by both technical users and business users by providing a notation that is intuitive to business users yet able to represent complex process semantics. The BPMN specification also provides a mapping between the graphics of the notation to the underlying constructs of execution languages, particularly BPEL4WS.

It should be noted that the BPMN is not originally created as a notation for BPEL (it was BPML [15]), the relation is more of a one way type [11] – BPMN to BPEL. More appropriate model, but still in development is BPDM [16] by OMG.

Unlike other modeling languages, there are three types of Events, based on when they affect the flow: Start, Intermediate, and End.
There are commonly used gateways: logical (XOR, OR, AND) or decisions points pictured as diamonds with a sign or text within.

There are several types of flows:
- Normal flow, that represents flow without any condition – object or control (regular arrow)
- Conditional flow – flow that is valid if the condition is true (arrow with a diamond)
- Default flow – for Data-based decisions – used when all other conditional flows is not true
- Message flow
- Exception flow
- Compensation Association
Following example should give an idea, how BPMN depicts a process and its sub-process (Discussion cycle):

BPMN, similarly to Activity Diagrams, is not pure formal language. It contains not only concepts like complex decisions, postconditions and preconditions that cannot be subject to machine interpretation, but also a concept unique to BPMN – immediate events. The fortunateness of this approach I will discuss later on.
2.3 EPC

Definition [20]: The Event-Driven Process Chain (EPC) method was developed at the Institute for Information Systems (IWi) of the University of Saarland, Germany, in collaboration with SAP AG. It is the key component of SAP R/3's modeling concepts for business engineering and customizing. It is based on the concepts of stochastic networks and Petri nets. The EPC method is focused on the business modeling executed by an operator. It is geared to modeling the macro behavior of a process chain.

EPC or eEPC is its nature simple but very powerful modeling language that is part of ARIS\(^1\) methodology. Its expression power is well suited for SAP R/3 process configuration/programming.

\(^1\) http://www.aris.com
2.4 BPEL
Definition [13]: WS-BPEL (Web Services Business Process Execution Language) provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

![BPEL Meta-model](image)

BPEL is widely accepted process language for web-services orchestration. Its concepts are very low-level, they have no standardized graphic notation and for non-technical users they are unreadable. Since BPEL servers as configuration language for web-services, it reminds more of a programming language than modeling.

2.5 Workflow Management Coalition (WfMC)/ XPDL
Definition [14]: (XML Process Definition Language) An XML-based language from the Workflow Management Coalition (WfMC) for defining business processes. Whereas BPML and other business process languages are geared to Web services, the foundation of XPDL was based around a common set of functions for work distribution found in most workflow products.

Goal of XPDL is “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rule.” [14] Even though it is focused on
human-computer interaction its concepts are low-level due the fact it serves as configuration language for execution engine. XPDL has no standardized graphic notation and again for non-technical users it is unreadable.

![Figure 12: XPDL Meta-model [14]](image1.png)

### 2.6 IDEF3 PROCESS DESCRIPTION LANGUAGE

Definition [21]: Process schematics provide a visualization mechanism for process-centered descriptions of a scenario. The graphical elements that comprise process schematics include Unit of Behavior (UOB) boxes, precedence links, junctions, referents, and notes.

![Figure 13: IDEF3 Core Concepts and an Example diagram [21]](image2.png)

IDEF3 is part of very well designed methodology IDEF. Its power relies on its interconnection with other diagrams from IDEF specification, especially Object...
State Transition Network Diagram. IDEF3 concepts are simple a clear but miss the expression power of business modeling notations like EPC.

2.7 PSL

Definition [17]: PSL (Process Specification Language) has a rigorously-developed semantics using first order logic, and is based on a three-step methodology: identifying intuitions, refining them in mathematical structures, and then defining a logical language for the intuitions.

PSL is an interesting approach based on FOL (First Order Logic) that is standardized as ISO 18629. Its goal to be unambiguous makes it very strict and formally strong, but the lack of implementation, standardized graphic notation and software supporting it, makes it hard to use.

2.8 Graham Detail Process Charts

Definition [19]: Information processes include flows of numerous documents, forms, email, systems, parts, people… that are all tied together to accomplish the process objective. Understanding the steps along EACH flow and the relationships between the items is crucial to understanding the process - detail process charts show the steps and the relationships.
Another case is the Graham Process Chart. This is the only one not connected to the IT world. It is purely of managerial origin. At closer look one can see a problem here. Even though it is called process chart, it is obviously an object state diagram and that rules it inappropriate for process mapping done the way as the other listed modeling languages do. This notation will not be discussed in this paper.

2.9 YAWL

Definition [10]: An acronym for Yet Another Workflow Language. Unlike other efforts in the BPM area, YAWL sought to provide a comprehensive modeling language for business processes based on a formal foundation (Petri nets).

![Diagram of newYAWL Concepts](image-url)
YAWL notation is very hard to read and understand since it has closer to Petri-nets than to reality (no events - just input and output conditions and tasks). It is a nice tool for process patterns analysis, but its focus only on activity flow makes it hard to use for business or even IS modeling.

3. Critique of Available Process Modeling Languages

3.1 Two Types of Users of Process Modeling Languages

There are two types of users that use process modeling. In this paper I will differentiate them as process analysts and process designers. These terms became nowadays so vague, they have to be defined first.

- The goal of the process analyst is to identify and capture the process that is already being performed in the reality with or without knowledge of its existence.
- The goal of process designer is to take already identified and captured processes and make them formal – ready for optimization and automatization.

Here we have two groups of users with different interests, way of analysis, way of working with reality. The most visible difference in meanings we can find in the idea of what a process as such is.

Process designers see the process as “A collection of related, structured activities--a chain of events--that produce a specific service or product for a particular customer or customers.” The important word here is structured. That is what designers do. They take relatively well structured process, turn it in some formal model that can be later on used for automatization through some workflow.

---

2 www.gao.gov/policy/itguide/glossary.htm
3 WfMC
or process execution engine. Modeling language for such use requires well founded (logically, mathematically) semantics, that can be converted into machine language.

Looking through the languages listed above, one can see that except the Graham Charts all modeling languages are well founded for this type of use. No wonder, since they all, except PSL, are founded on Petri-nets bases.

Unfortunately the practice had taught us that in reality there are very few well structured processes. Growing complexity of business puts us in different position. In this complex world, we have to first understand the part of the reality we are analyzing, than identify the core elements and model an abstraction in relevant semantics, knowing the limits of the language we use. That is what the process analysts do. The model they produce is a conceptual model and it is a base for structuring down the captured process.

Guizzardi [18] cites John Mylopoulos who defines the discipline of conceptual modeling as

"the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication. Conceptual modelling supports structuring and inferential facilities that are psychologically grounded. After all, the descriptions that arise from conceptual modelling activities are intended to be used by humans, not machines... The adequacy of a conceptual modelling notation rests on its contribution to the construction of models of reality that promote a common understanding of that reality among their human users."

Understanding the conceptual process modeling this way brings several problems into our consideration that will be discussed in following chapters.

3.2 Focus on capturing reality

The first problem of listed modeling languages is their convertibility into machine language. This feature is not for free. In different words, they are derived from machine language and modeling a process in them means programming a process. It is fine when thinking how to create a BPEL code for execution, but it is a disaster when trying to understand the reality and model a part of it. When creating conceptual model one should focus on the right understanding of the reality and not to spend the time by thinking, how to capture the reality in the modeling language used. The use of the conceptual language should be natural and independent on later usage (implementation).

3.3 Simple Things Done Hard Way

Another problem is how abstraction in listed languages works. All these languages are low level ones that assume well structured processes. It is very hard to use

---

4 BPEL
them to model something that is incomplete and where several abstraction layers meet at one place (and that is a case when trying to capture the reality).

Let us consider an activity where a mechanic is repairing a car. There is an event that executes this activity and after the repair is finished, the car goes to another mechanic for inspection and so on.

The trouble comes when capturing the details of the repair activity. Process analyst wants to capture only relevant and essential events, that affect the activity (he is not interested in “implementation details”) – when there is launch break the activity is paused for some time, or when comes another repair job with higher priority, work is paused again etc. Analyst knows that this is a complex activity and if one wants to capture those details in above listed languages one has to draw the more detailed model of the activity. But one’s job is to capture the whole process. If the analyst would model everything to the last detail, analyst would never finish his job – it would not be modeling, it would be writing the whole process in algorithms. All this means that modeling languages listed above are capable of capturing the essential parts of a model completely only when done in all levels of abstraction.

The only language that seems to have something to offer is BPMN. The concept of immediate events allows us to break an activity, without necessity of drawing more detailed diagram, at least. But that is all.

3.4 Humans vs Machines

Convertibility into machine language has another drawback. That is readability of a modeling language for humans.

First of all, people usually do not need all the context that is relevant to a process. A lot of it is for us natural, self-explanatory. Not for machines - they need all the relevant context. This requirement causes that the models are too detailed and for a human user is hard to find the essential elements since they are drowned in their context. In practice, when listed modeling languages are used for conceptual modeling, they have to be crooked – formal rules are being broken, not all is being modeled in detail. The problem is that it is not a solution – models created this way are ambiguous and hard to interpret since leaving out, in any of the listed modeling languages, some “obvious” context goes usually hand in hand with leaving out some essential element of a model.

3.5 Function vs Activity

Another core problem is the vision of the world. The computer world sees every change as a product of a function. No matter what are these changes called (transitions, activities, tasks) they are still seen as functions. The only differentiation we can find here is simple (atomic) function and complex function. Unsurprisingly a complex function is consisting from atomic functions. The idea is that there are atomic functions and the process of modeling is to disassemble the complex functions (activities) into the atomic ones. Again this is maybe fine with the design but not with the analysis.

When analyzing reality, looking for atomic functions means getting somewhere at subatomic level. That is not lucky idea to get that low. Atomic functions are applicable only in computer programming. In conceptual modeling one has to bear in mind that there are only activities and all are complex. They are no functions.
There is another point that makes functions and activities unlike. The problem is the nature of atomic functions.

Atomic functions are created at design time. The designer upon understanding the whole system, the designer is to implement, decides what the atomic functions are and how the complex functions will be assembled from them. The designer creates functions and must understand completely the system, which is modeling, otherwise the created function structure is wrong.

Activities, in conceptual sense, are captured at analysis time. They represent part of reality one is trying to understand. There are no atomic activities since when modeling one does not understand the system completely yet. The analyst captures activities he sees as essential to the model. The analyst is making his way to understand the reality and tries to create model that would allow others to understand the system in the reality.

3.6 Function and Activity States

From programmers point, function can be executed, it can finish ok or produce an error or an exception. It is simple, it is obvious, but one can find it only in very few modeling languages – BPEL and BPMN (BPDM). Yes, functions have states and these states are caused by events. In case of functions all events, except the one starting the execution, are of internal origin. We do not know what has caused an error, exception or why has the function finished successfully. In case of functions it is irrelevant to capture the causes since it is depended on execution environment and other factors beyond our control.

BPMN has introduced one enhancement in function states – immediate events – function can be also aborted due external event – problem of this feature is that it goes against atomicity of atomic functions. BPMN therefore extends the machine language, but it tries to mix two different approaches (conceptual and design) and it does no good.

From process analyst point (or at least from my point), activity can be started, finished, stopped, continued, cancelled or failed. Since at conceptual level all the activities can be considered complex, it makes sense to let the activity states to be caused by external events. There is no execution environment – we are trying to capture a part of reality and we should be interested what events cause activity state changes.

Functions having states allow us to capture them not only in state form (box that represents running function), but also in a event form – start, finish, error, exception – four events that can well capture the run of a function in time. Same way an activity could be represented by a chain of events related to certain activity. In case of a function it has no sense, since we would end up with the same “representation box” contemporary modeling languages provide us. In the case of activities this is useful since the changes of activity state are caused by external events.

If there were few languages aware of function states, there are none aware of activity states. BPMN is stuck somewhere in the middle. Its immediate event feature signals awareness of this problem but its focus on executable code renders this feature useless.
3.7 Events and Timings

Another problem is: what does it mean that some activity has been finished? What is the condition of finishing – what has to happen in order to call an activity finished? And is there only one event/condition for finishing an activity or there are more? This is easily solved when we work with atomic functions (well not that easily – one has to model all levels of abstraction), but in conceptual modeling one has to deal with complex activities. Since the listed modeling languages are based on atomic functions, none of them is capable of capturing this detail. It requires different approach outlined in the Function and Activity States chapter.

3.8 Plans and Duties

Looking at process model, one can see a plan. A plan, who should do what and when. These plans also contain different scenarios that are either used or not, depending on whether a specific event occurs or a specific state of affairs has become true.

This brings an interesting question up. Is a plan consisting only from duties (that is what activities are) are not there any rights for involved parties?

Before there were project management and computer program modeling there had been a need for planning anyway. Since it was cooperation among people, there has evolved a tool that formally specified who has what duties and rights, what is the goal of cooperating people and what are the penalties for not following the agreed plan. They are called contracts [2].

Processes are very similar to contracts. They define the parties involved, their duties, time/activity schedule that has to be followed in order to fulfill the object of the contract/ process. What makes these two different is that contracts define not only duties of involved parties but also rights. At designers’ level there are only commands for the machines what should they do and how the people should cooperate but at a conceptual level we can leave the machine language behind and introduce duties and rights into conceptual process modeling just like we do in contracts or agreements..

4. Regulative Approach to Conceptual Process Modeling

In this chapter I will outline the conceptual modeling language that tries to implement into modeling critique done in previous chapter. This outline only deals with the event/ activity flow modeling. The business object, inputs, outputs, organization units, etc. are not subject to this outline.

Requested Features Summary:
- Conceptual language is focused on reality analysis.
- Understandable and usable by non-programmers.
- All activities are complex.
- Activities occur in time and there are states an activity can be in.
- There are events that cause transition to specified activity state.
- There are duties and rights.
- There are complex decisions.
Following the requests, the notation could look like in Figure 19.

**Figure 19: Basic Concepts of Regulative Approach**

An Event is either related to a duty, right or a fact. If we group events by their content, we can get a Grouped Events element that is very similar to the element Activity, we know from other modeling languages.

In this approach term “Activity” has a different meaning than in presented modeling languages. It does not represent a function or what should one do, but it represents state of affairs. In the case of a “fact” event it represents occurring state of affairs (a fact that is a case), in case of a “duty” event it represents desired state of affairs and in case of a “right” it represents claimable state of affairs. The “Activity”, in sense of meaning in other modeling languages, is in Regulative approach a Duty. The “fact” event represents concept of “regular” event. The Right is something one cannot find in other modeling languages. It expands the means of expression of this approach. It allows to incorporate information that had to be in different model/paper when modeling in other languages. The dynamics of gaining and loosing rights in case of business customer rights, IS customers and their rights, system roles and their rights, etc. was captured separately, even though rights do dynamically change as the process proceeds. The Regulative approach allows capturing dynamics of duty assignments as well as of rights assignments.

Every event has a mark that specifies what change of state of duty or right or fact (reality) the event signifies. The six recognized states of an activity allow expressing more than regular approaches. All states are caused by external events - we specify events that change activity states. This is in contrast with states caused by internal events, where an error or an exception is caused by an unknown force behind the function execution (BPMN). The “fact” event is in a special position here. Model cannot change the reality. In that sense there is no reason to specify events that cause “fact” events. On the other hand its declarative nature may be useful in some other cases – judge’s decision whether someone has or has not done some activity, etc.

The state sign in a “fact” event represents:

The Start and Finish state events allowing us to better specify when the transition should come – right after the activity begins or when it is completed. Other modeling languages are either ambiguous (do not capture this detail) or there has
to be regular language used to draw the difference (past tense, adding the term
start into activity description, etc.). **Stop** state event specifies that the activity is
interrupted and **Continue** that the activity is continued. **Cancel** state event
specifies that the activity has been permanently stopped. **Fail** state event specifies
that the activity has finished unsuccessfully.

For the “Duty” and “Right” events is the meaning bit different since the states refer
to either desired or claimable states of affairs.

The **Start** and **Finish** state events allow us to specify when and why a duty or right
begins and ends. **Stop** and **Continue** state events allow us to specify when a duty
or right is temporarily suspended and when it is back in force. They allow us to
describe, without a necessity to draw more detailed model of the duty or right, what
can interrupt a duty or right and under which condition their enforcement is to
continue. Since there can exist more than one event causing stopping or
continuation, there has to be captured which “Continue Events” continue which
“Stop Events”. This is done by a dashed line between related events (Figure 23).

**Cancel** state event allows us to specify when and why a duty or right is suspend
permanently while in force. Again without a necessity to draw more detailed model
of the duty or right. **Fail** state event allows us to specify when and why an actor
was unable to fulfill a duty or right. Unlike BPMN concept of the Compensation
Flow it allows us to specify what causes the duty or right to fail.

We will consider, same as in BPMN, three types of events (Figure 20): Start,
Intermediate and End. This differentiation is necessary to make clear, from the
process analyst point of view, where the process starts, what events influence it
and where the process ends.

![Figure 20: Three Types of Events](image)

The Figure 21 shows the model in basic form. Events causing other events to rise.
Same way, as in the BPMN, events are differentiated so that it is clear what event
starts the process, what events are recognized during a run of the process and
what events are the final events of the process. If an event is a fact, type of event is
left blank.
For analytical use this is diagram too complicated since it is too hard to understand what these “groups of events” mean in the bigger picture. Now we try to group state events of the same activities.

The diagram (figure 22) looks far clearer than the figure 21. We have events and activities here same as in other modeling languages. However, activities here are in form of rights and duties. Grouping of events has not lost any detail, since differentiation among start, intermediate and end events is preserved.

Although, the model may look similar to regular ones, there are many differences. First of all it captures not only duties but also rights. Customer’s right to cancel the order is fully incorporated into the process. There is specified when this right starts, when and why it finishes or when and why it is canceled. This all is specified in context of the part of “Making Order Process” that is pictured here. Same as rights, the duties have specified what starts them and when and why they are considered finished. For instance the duty Verify the Order. It is clearly specified what starts and especially ends this “activity” without necessity to model this “activity” in greater detail. Another example is the duty Inform the Customer – without modeling how all that is being done, we just specify an event that specifies when this duty is successfully finished. There is no need to get bogged down in details.
Figure 22: Grouped Events Model

At the chapter Simple Things Done Hard Way I mentioned a case with a mechanic repairing a car. Figure 23 illustrates how it would be captured in Regulative modeling notation. The dashed line signifies related stopping and continuing events.

Figure 23: Repair a Car in Regulative Approach

For comparison I captured the same situation in Activity diagram (Figure 24). Since stopping and continuing is not possible in contemporary notations, the activity “Repair a car” has to be modeled in detail in order to capture stopping a running activity and what are the causing events. Even then, in many cases, “implementation” of stopping an activity is cumbersome (figure 24). First we have to disassemble the repair activity into concrete activities, even though the concrete
activities are not essential for our model. Second, after every activity occurrence, we have to check the possibility the process stopping conditions, which makes this model very confusing and algorithm look-like.

Figure 24: Repair a Car in Activity Diagram

Capturing continuation of an activity in contemporary modeling languages has to be done informally – new process starting with a complex decision asking whether the mechanic has some unfinished work. This cuts one process into two or more processes that certainly does not add on comprehensibility to the model.

5. Conclusions

In this work I have tried to show how contemporary process modeling languages are inappropriate for real conceptual process modeling. Their sensitivity on easiness of structurability of modeled reality makes them hard to use for capturing the reality. They push the analyst into thinking how he is going to model a process instead what he is going to model.
Following my critique I suggest a different approach, “Regulative Approach”, to conceptual process modeling based on thousands of years used technique – contracts. This approach recognizes that actors do not have only duties (activities) but also rights and that at cognition level all activities are complex. Activities are treated as entities that go through different states, which are caused/changed by events originating inside or outside of the modeled process. I suggest six possible activity states: Start, Finish, Stop, Continue, Cancel and Fail. Considering these six states of an activity, occurrence of activities in this approach is expressed by events that signify the change of an activity state. The activity flow itself then consists from chains of events as the states of activities are changed by explicit events over time.

This modeling language tries to be easily human-readable and understandable (events causing other events) without technology related concepts. It tries to save analyst’s time and let one focus on what is essential. For specifying essential events analyst doesn’t have to diagram a sub-process since all activities (Grouped Events) are treated as complex. Differentiation of duties and rights makes the diagram richer and more complete. It focuses on what is the purpose of the modeled system – Duties and rights define more of particular goals than list of actions what to do (depending on the level of detail, of course). Actors can be assigned rights and duties and this allows proper role permission definition through process models. This way the modeling language allows capturing how these permissions change dynamically as the actual process proceeds including worse case scenarios – what happens if an actor fails to complete his duty.

Regulative approach presented here is just at its start point. There is lot to improve and enhance. This paper should give the idea what makes it different from contemporary modeling process languages and why it has sense to work out this concept into more sophisticated method.

References:


