MERgE SAFETY & SECURITY

MERgE

ITEA2 – Project #11011

Multi-Concerns Interactions System Engineering

01.12.2012 to 30.11.2015

Process enactment, deviations & recovery concepts and their application in use cases

Project Deliverable D3.3.2

Task 3.3 – Advanced concepts in engineering process modelling and enactment (UPMC)
WP3 –Advanced multi-concern engineering concepts (Sam Michiels)

Status	[ ] Draft	**Document created**	:12.01.2015
	[ ] To be reviewed	**Last edited**	:19.05.2015
	[X] Final	**Due date**	:28.02.2015
Confidentiality	[X] Public (for public distribution)	**Date finalised:**	:19.05.2015
	[ ] Restricted (only MERgE internal use)	**Document version**	:0.4
	[ ] Confidential (only for individual partner(s))	**Pages**	:27

Executive summary

This deliverable presents the core concepts regarding software process modelling with a special focus on process enactment. It initially explains the concept of a process and presents the key notions used in this domain, according to the state of the art. Then the concepts regarding process modelling are explained. It further describes the advanced concepts of process modelling and explains process enactment. It continues on to explain the notions of process planning, deviation and recovery. Then we present the architecture of the provided tool, PRODAN, for dealing with process enactment, deviation & recovery. This describes the concepts behind the implementation of tool and explains the technical choices made for its realization. Finally this deliverable presents the application of these concepts in the processes acquired from the use cases of the project. This analysis is explained using a single process from one of our industrial partner. Application of our methodology on the processes from the use cases allows us to gather feedback on our approach. This feedback will be taken into account, so that the next deliverable can overcome shortcomings of the current version of the prototype.

Contributing authors: Fahad R. Golra, Jacques Robin, Regina Hebig, Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie.

If overdue, provide reason here.

Version.	Content	Resp. Partner	Date
0.1	Process enactment, deviations and recovery concepts	UPMC-LIP6	15-01-2015
0.2	Process enactment, deviations and recovery concepts and their application in use cases	UPMC-LIP6	06-03-2015
0.3	Process enactment, deviations and recovery concepts and their application in use cases. (Update to overall document structure)	UPMC-LIP6	19-03-2015
0.4	Process enactment, deviations and recovery concepts and their application in use cases. (post quality review updates)	UPMC-LIP6	11-04-2015
0.5			
0.6			

Reviewed & Accepted	Name	Partner
Independent Reviewer (outside WP)	Kati Kittilä	Codenomicon
Validation from Management Board	C. Robinson	Thales R&T

Contents

List of Figures

1 List of abbreviations

- BPMN: Business Process Management Notation
- BMP: Business Process Management
- BPI: Business Process Improvement
- COP: Constraint Optimization Problem
- CSP: Constraint Satisfaction Problem
- FUML: Foundational subset for executable UML, sometimes referred to as Formal UML
- MDE: Model-Driven Engineering
- OCL: Object Process Language
- OMG: Object Management Group
- PDL: Process Design Language
- PIL: Process Implementation Language
- PML: Process Modelling Language
- PSL: Process Specification Language
- PSEE: Process-driven Software Engineering Environment
- SPEM: Software Process Engineering Metamodel
- UML: Unified Modeling Language
- WfMS: Workflow Management System
- WS-BPEL: Web Service Business Process Executable Language
- XPDL: XML Process Definition Language

2 Process Overview

2.1 Process Modelling

The term 'engineering' in software engineering focuses on the *systematic* and *organized* procedures to carry out the activities for software development. In contrast to ad-hoc methods, the target of a procedure in *engineering* is not only to achieve goals, but also to accomplish it by following precise and well-ordered tasks. The greater goal of following such methodology is to ensure quality in these practices. These well-ordered tasks need to be specified before their actual execution. Well-documented procedures also allow standardization and possibilities of improvement. Process models are used to specify these tasks and the order in which these tasks need to be performed in a process. These processes can be modelled using various languages in different contexts. We will not be focusing on the specifics of each process modelling approach in this deliverable; instead we are going to discuss the main concepts of the domain.

2.2 What is a Process?

Process is a generic term that has been used in many fields like **Business Process Management (BPM), Workflow Management (WfM), Business Process Improvement (BPI),** etc. These domains present process as the specification of the core methodology and then the implementations of the business domains are developed around it. They normally call this process, a 'business process'. It is defined by Davenport as:

> "*A structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers*" (Davenport, 1993)

This definition of process focuses on a general structure and motivation of a business process. A *'software process'* in our view is also a business process that is targeted towards the development of software systems. Specifically, software process is defined in the literature as:

> "*A set of partially ordered process steps, with sets of related artefacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables*" (Lonchamp, 1993)

So we can interchangeably use ‘process’ and ‘software processes’ in this deliverable for two reasons. First, ‘process’ is a more general term that can be used to explain the core concepts. Second, to define software process, the software industry uses BPM technologies, where business processes represent the software processes.

Longchamps's definition can be viewed as an extension to the Davenport's definition of process, where he focuses on a clear process boundary, well-defined inputs and outputs and a structure of action, which transforms the inputs to outputs. Numerous other definitions of process can also be found in the literature, but they all focus on related groups of activities, common goals, and the use of

people, information and resources (Lindsay, Downs, & Lunn, 2003). Level of granularity in the definition of process may vary, but the key concepts are fundamental for the completeness of a process. Processes are defined in detail because adhering to them may be critical for a project's success, especially for the large-scale projects (Lehman, 1991).

Initially, the software engineering community had put a lot of stress on the linear structure of a process, which does not fit well with the software development practices. There has been an argument that workflow view of processes with definable inputs and outputs of discrete tasks, having dependencies on one another in a clear succession is limiting. So a more flexible definition of a process is

> "*Any work that meets the following four criteria: it is recurrent; it affects some aspect of organizational capabilities; it can be accomplished in different ways that make a difference to the contribution it generates in terms of cost, value, service, or quality; and it involves coordination*" (Keen, 1997)

This definition does not explain the structure of a process; neither does it constrain the ordering of activities, it rather focuses on the significant characteristics of a process. We tend to follow this approach definition in complement with the earlier definitions. This allows us to take processes both in forms of imperative and declarative representations. Thus we have the flexibility to specify the processes without focusing on their sequence of execution. In such cases, the processes can be specified as a collection of activities having multiple constraints on them that guide their execution to achieve the objectives.

2.3 Process Modelling Languages and Notations

Software process programming started to evolve as soon as the software community started to give software processes the same importance as that of software programs (Osterweil, 1987). Gradually software enterprises realized the need to develop processes for each software project. This created a need for a well-defined approach to describe their processes. Once specified, these processes could be later on re-used for multiple projects and tailored according to the specific needs of the projects. They were expected to capture all the details of the product and the organization for developing that product. To respond to this need, Osterweil suggested a notion of 'process program', which would describe the work routines of a software enterprise relating to a specific project by taking all the needed process elements into account (Osterweil, 1987). Gradually, these process programs evolved into full-fledged languages for formal specification of processes, called **Process Modelling Languages (PML)**.

Curtis et al. presented four distinct views to describe the elements modelled by the process programs/models (Curtis, Kellner, & Over, 1992). These views are: 1) *Functional View*, that covers the functional dependencies between the processes. These functional dependencies can be input and output dependency, where the output of one process is an input to the other. 2) *Dynamic View*, that covers the control sequencing of the process elements. The control flow and the sequence of processes describe the overall behaviour. 3) *Informational View*, that provides the description of work products used or produced by the process. 4) *Organizational view*, which includes the description of the performer of processes and the organizational hierarchy regarding the responsibilities.

The problem with PMLs is the level of detail and formal specification that makes it quite difficult to use in the industry. For this reason, PMLs are mainly used by academia to formally prove various assumptions and characteristics of process modelling. However the research carried out on PMLs gives a formal foundation for high-level process modelling notations. The term 'high level' is to demonstrate that other languages use a higher level of abstraction, thus hiding the fine details from the end user. These high level languages can be divided in two categories. First, the Business

process modelling languages, which provide the possibility to graphically draw the process flows (**Object Management Group**, 2011). These process flows are used for discussions between stakeholders and for keeping the documentations. Originally these languages were not meant to be executable, but now with growing influence of IT in business, a need to execute them has been growing. To handle this need, some executable languages have been presented, to which the business processes can be transformed (OASIS, 2007). The second category is the workflow models, which also allowed drawing the process flow graphically. They were intended to be directly executable on a workflow management system. Workflow notations are developed for enactment, so they need well-defined execution semantics. For the development of information systems using workflows at the core, the target of the system analysis phase is to understand the process in which the intended system would be deployed. In some recent endeavours, process models are used to describe these processes, which are embedded in the information systems and control their execution (Weigold, Aldinucci, Danelutto, & Getov, 2012).

2.4 Process-Centered Software Engineering Environments

Process Modelling Languages became one of the key research areas of software engineering research and since then new dimensions on process modelling approaches are being explored. The development of ***Process-Centered Software Engineering Environments (PSEE)*** is based around the concepts of process modelling. PSEEs are the information systems that provide the notations and mechanisms for the development of process models. These systems also foster the possibility to maintain and enact a process model. PSEE offers support for process management in one or more phases of process lifecycle ranging from requirements specification, assessment and problem elicitation, (re)-design, implementation to monitoring and data collection (Ambriola, Conradi, & Fuggetta, 1997). The PSEE is designed to guide/enforce the user in the development process. The role of PSEEs in guiding a user is classified into four levels from least active to most active as: 1) *Passive role*, that operates on user requests 2) *Active guidance*, where PSEE guides the user 3) *Enforcement*, where user is forced to act as per the direction of PSEE 4) *Automation*, where system does not require user intervention (Dowson & Fernström, 1994).

A PSEE offers a PML to support the definition of process models, which are then analysed and enacted by the environment (Türetken, 2007). The analysis of these process models is based on different properties like consistency, redundancy and circularity. The enactment of the process model is handled by the environment according to the degree of guidance provided by the PSEE, where it can demand the user to execute some processes or perform them itself by invoking the related application and IT tools. The focus of PSEE remains on the analysis and enactment of the processes, so they rely on formal languages (PMLs) that are very close to software programs (Bandinelli, Braga, Fuggetta, & Lavazza, 1994; Sutton & Osterweil, 1997). Some recent research endeavours targeted the use of process models in PSEE by exploiting **MDE (Model-Driven Engineering)** (Montoni, et al., 2006; Maciel, Gomes, Magalhaes, Silva, & Queiroz, 2013). Ambriola et al. provide a classification of PMLs based on the support that they provide for a specific phase of process lifecycle and the level of abstraction (Ambriola, Conradi, & Fuggetta, 1997). This classification identifies:

- ***Process Specification Languages (PSL):*** they are used for the requirement specification and assessment of processes.
- ***Process Design Languages (PDL):*** they support the design phase of the process development.
- ***Process Implementation Languages (PIL):*** they are used for the implementation and monitoring of the processes.

3 Executing the Processes

3.1 Process Enactment

With growing trend of reliance on standardized systems, more and more enterprises are adopting process driven methodologies, especially for software development. Ideally, all the different activities performed in a software enterprise should be explicitly defined to promote standardization and improve control, flexibility and effectiveness of standard practices. In order to deliver quality customer value, these processes are often supported or at times fully implemented by software systems (Rossi & Turrini, 2007). A process execution typically involves various applications, services and humans. Specialized process management systems are developed to integrate and control these processes to achieve the desired business goal.

3.2 Process Management Systems

These process management systems rely on the inherent behaviour of the process modelling languages. For industrial use, typically two types of process management systems are in use: **Workflow Management Systems (WfMS)** and Business Process Management i.e. Workflow Management Systems and Business Process Management. In WfMS, a workflow is a process model that is used to describe process definitions by modelling the contained activities, procedural rules and associated control data to manage its execution (Hollingsworth, 2004). Each process instance in a workflow has its own specific set of data associated to that individual process instance. In order to execute these processes, a workflow engine is required. Workflow engine interprets the (graphical) workflow representations using a computations form, known as workflow description language. On the other hand, BPM processes are implemented as web services. For their execution, Web Services Business Process Execution Language (WS-BPEL) has played the most significant role (OASIS, 2007). WS-BPEL is a language for defining and executing business processes. It is based on web services and it exploits the web services composition, orchestration, and coordination for realizing SOA.

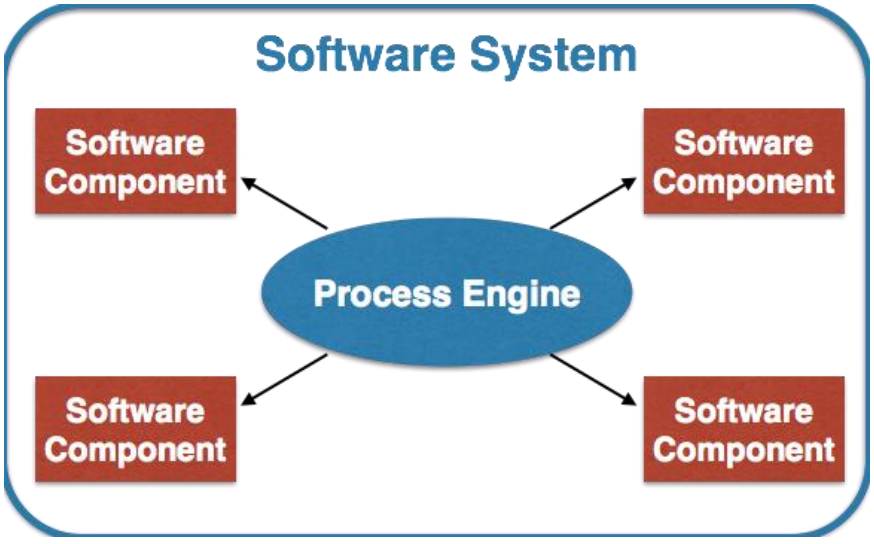

Figure 1: Process Driven Application

3.3 Process-driven Applications

Processes are developed and executed to automate the software development methodologies. In order to do so, the process engines are connected with the application tools and services. This way, the complete Information system can be integrated and controlled around the process engine. In such situations, there are two methods to empower the process engine to control/help in controlling the rest of the development environment. One of the ways to automate software development is to embed the process engine within the software application as a component. This component is then bound to other components of the system that provide the actual functional code, as shown in Figure 1. This way process definition represents the main control logic of the application and process engine is responsible for triggering other components. Applications developed with this architecture are called process-driven applications (Weigold, 2010).

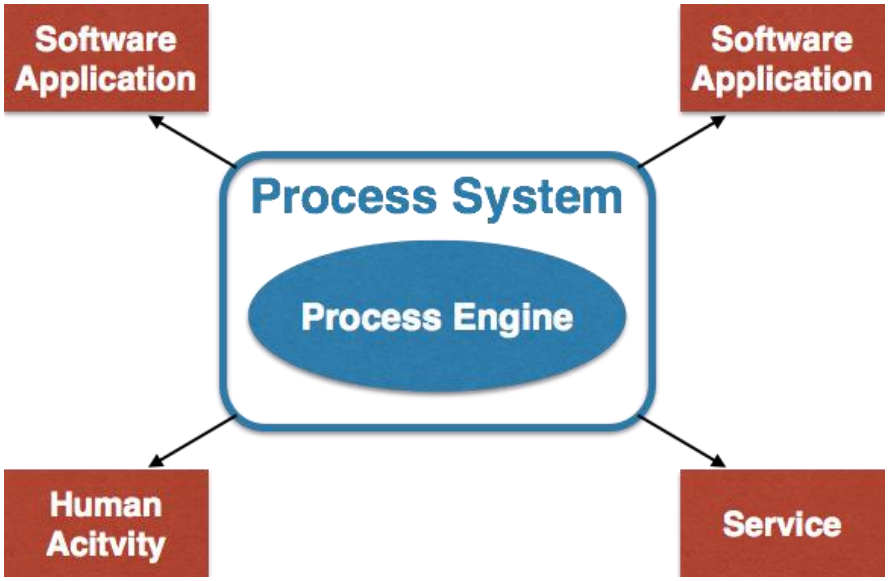

Figure 2: Dedicated Process Management System

Figure 2 depicts the architecture followed by the process management systems (BPMS/WfMS), where the business process engine is implemented as a standalone software system that interacts with other software applications, services and humans to achieve the business goal. This architecture is not very domain specific and allows a generic process engine that can be used with different types of applications in multiple domains.

3.4 Process Enactment issues

For industrial development, software processes can potentially be modeled using languages coming from four communities:

- The software process engineering community, which specified its special purpose **SPEM (Software Process Engineering Metamodel)** standard (Object Management Group, 2008);
- The model-driven engineering community, which specified the general purpose **UML (Unified Modeling Language**) standard (Object Management Group, 2011) to model many different aspects of a software project, together with its companion standards **OCL (Object Constraint Language**) (Object Management Group, 2014) and **FUML (Formal UML)** (Object Management Group, 2013);
- The business process and web service modelling community, which specified the **BPMN (Business Process Management Notation**) (Object Management Group, 2011) and the **WS-BPEL (Web Service Business Process Execution Language**) (OASIS, 2007) standards

that are dedicated to business processes in general, but not specifically to software processes.
- The workflow modelling community, which specified the **XPDL (XML Process Definition Language)** standard (Workflow Management Coalition, 2012) to support interoperability between workflow engines capable of enacting business processes represented as workflows.

Several syntactic and/or semantic bridges among these various industry standards have been included in their latest versions. A complete mapping of SPEM to a UML profile has been published as part of the latest 2.0 version of SPEM. This profile defines stereotypes specializing the main UML metaclasses in the classes, common behaviours, activities and actions packages. Two formal semantics have been published in the latest version of the FUML standard. The first of these semantics is an operational execution semantics defined by a virtual machine implemented in Java. The second is a declarative axiomatic semantics defined in first-order logic. The first semantics provides a standard to enact software process models specified as FUML classes, activities and actions. The second semantics provides a standard to support formal verification of properties of such models. A SPEM model can thus be enacted and verified by first being mapped into a UML model decorated with stereotypes from the UML profile for SPEM onto which to run enactment and verification engines based on the FUML semantics. However, the SPEM profile does *not* provide additional semantics that are proper to the stereotyped UML metaclasses as compared to their base UML metaclasses. Thus today, modelling a software process directly as UML activity diagram is more convenient for enactment and verification purposes than in SPEM.

Syntactically, WS-BPEL is defined as an XML-schema. Its specification includes an operational semantics for process enactment. In contrast to UML, it neither provides a concrete graphical syntax, nor a denotational or axiomatic semantics to support process verification.

BMPN defines such a concrete graphical syntax, as an alternative to those of UML and SPEM. However, it neither provides a direct operational semantics supporting standard enactment nor a direct denotational or axiomatic semantics supporting verification. The latest version of BMPN however includes a mapping to WS-BPEL. The main motivation for this mapping is to support reusing the operational semantics of WS-BPEL and the available enactment engines implementing it to enact BMPN process models. However, this mapping is both partial and ambiguous: some BMPN constructs have no possible correct translation into WS-BPEL while others have several different possible translations.

XPDL defines an XML serialization of BPMN that can be enacted by many workflow tools, which accept a process model in XPDL as input and translate it into their internal executable proprietary workflow language. XPDL can thus be used to enact BPMN processes.

Thus, in order to model software processes using an intuitive graphical notation but nevertheless enactable following a standard semantics, one has the following choices:
- Modelling using BMPN and enacting it using a workflow engine accepting XPDL inputs;
- Modelling using UML activities and enacting using an activity engine implementing the FUML operational semantics.

However, one can only use UML activities in order to be able to easily verify properties of the process model prior or during its enactment. The lack of a standard graphical or axiomatic semantics for BPMN makes it inappropriate to use it with such usage in mind.

In addition, BMPN only supports representing processes *imperatively* by way of a model backbone consisting of strict activity sequences, linked together predominantly by *AND*, *OR*, *XOR*, *split* and *join* nodes. UML activities are also generally used to represent processes in such imperative style. However, the integration of OCL with UML activities also allows representing looser processes *declaratively* by using OCL constraints to define a minimal partial order among activities, instead forcing them into a somewhat arbitrary strict order sequence using control flows. This is an essential

property to model highly iterative and agile software processes that have become prominently adopted over the last decade for non-critical software development in application domains not subjected to certification obligations

4 Handling Process Deviations

4.1 Process deviations

We already described that process management systems are responsible for enacting the process models that are specified in the planning phase. A process management system is responsible for; 1) taking process model as input and allowing the process agents to realize the software development activities, 2) making sure that the activities in the process enactment are being realized in the same way as specified in the process model, 3) the specified artefacts are being produced by the execution of these activities. In order to take on the above stated responsibilities, a process management system has to take into account all the 'abnormal' situations as well. Real life processes in software development projects may not follow the exact plan. The reasons for not following the exact plan may vary; the situational demand, incapacity to execute the process under given circumstances, past experiences or the volatility/evolution of requirements. In such cases, the Process-centered Software Engineering Environment (PSEE) should be able to observe the inconsistency between the software process model and its actual execution.

Any action of the process agent that is not compatible with the process model is called a deviation. A precise definition of process deviation found in the state of the art is:

> "*Deviations are actions that violate the constraints present on the process model over the sequence of actions or over the artefact states these actions produce*" (Silva, 2012)

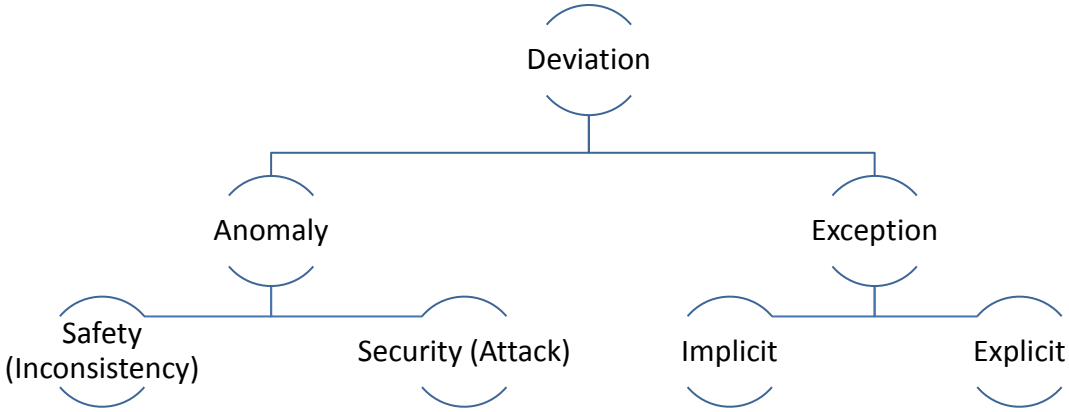

Figure 3: Process Deviation Classification

A deviation denotes the fact that the process enactment is not conforming to the specified process. Process deviation itself does not give any reason for such behaviour. A classification of process deviations can help in understanding the causes that bring the process enactment engine to a non-conformant state. As shown in Figure 3, we classify deviations into two sub-groups i.e. anomaly and exception. Anomalies refer to the abnormal enactment of the process model such that it reaches a '*state'* that was not specified at process design phase. The first main category of deviation is anomaly. Anomalies can either be inconsistencies or security attacks. An inconsistency is an operational error that may be caused by information system or human mistake that takes the process to a state, which was not specified by the process. An inconsistency is a safety threat to the information system. A security attack is a deliberate action by some human agent to work around the

system to achieve a harmful goal. Exceptions are the second main category of Deviations. They refers to the '*actions'* performed by the process agents in violation to what was already planned. Exceptions can be divided into implicit and explicit exceptions. Explicit exceptions are the ones that are described in the specified process model. Not all process-modelling languages allow specifying explicit exceptions in the process models. Little-JIL is one of the languages that has developed a very concrete mechanism to handle explicit deviations in the process models (Cass, Lerner, McCall, Osterweil, Sutton, & Wise, 2000). Implicit exceptions in a process model are the unexpected deviations of a process model. These deviations may occur due to unforeseen circumstances, employee's over-confidence, etc.

4.2 Deviation detection as a constraint satisfaction problem

Our approach to deviation detection is to view it as a **Constraint Satisfaction Problem (CSP)** Prior to enactment. The process modeler-oriented representation of the process model is translated into a set of constraints expressed as a formula M in a sorted first-order logic. During enactment, the trace of the actions executed so far is also expressed into a formula T in the same logic. We then use a constraint solver to check the satisfiability of the formula $M \wedge T$. If the solver answers that this formula is unsatisfiable, we conclude that the enactment trace is violating a subset of constraints of the process model and we return these constraints to the process modeler expressed in the modeler oriented representation. If the constraint solver finds a variable assignment that satisfies $M \wedge T$, we conclude that the enactment trace is so far respecting the constraints of the process model.

Depending on the expressive richness of the constraints needed in a particular process modelling domain, different sorted logics and different associated solvers are usable in practice. Therefore, the translation scheme between the process modeler-oriented representation and the solver-oriented representation may need to differ in different applicative domains. With respect to variations in process domain size, one must note that process models with more than 50 elements rapidly become visually overwhelming for a human user. Therefore, modelling very large process always involves in practice decomposing it into sub-processes encapsulated into compound activities of a higher-level process. Then deviation detection can be carried out recursively on small models at each abstraction level. Consequently, the size of the input formula passed to underlying constraint solver remains within restricted bounds that ensure that it can return an answer in reasonable time for each call.

4.3 Deviation Recovery

Once the process is process is specified through the development of a process model, it defines the standard execution paths through the activities. The involvement of human actors in the processes, at times makes it hard to continue with the specified process and thus we come across process deviations. Once the process enactment deviates from the standard paths of the specified process model, the execution of the remaining standard path might not make any sense. Some of the activities that were not executed or skipped due to a deviation might be mandatory for the process. Thus a re-planning of the process enactment is required, that guides the user about the new sequence of execution for the activities in that process.

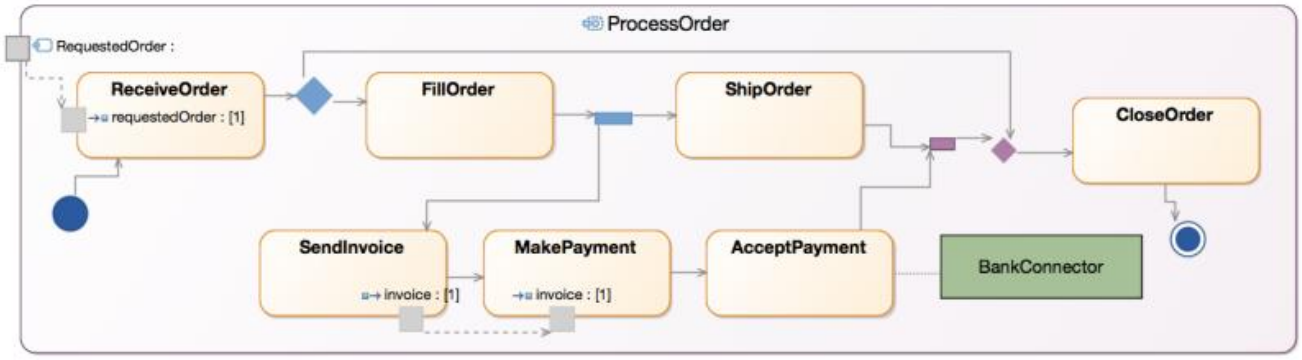

Figure 4: Example Process

Let's take the example of a process to explain the concept of process recovery, as shown in Figure 4. This process has seven activities starting from *ReceiveOrder* activity to *CloseOrder* activity. Let us suppose that during the executing of this process *ReceiveOrder* activity was the first activity to be executed. After the execution of this activity, there were two possibilities to continue the execution according the specified process; executing *FillOrder* or *CloseOrder* activity. However, due to some internal settlement, *ShipOrder* activity was selected for execution. This lead to a deviation situation and the tool detected this deviation. However the user insisted on executing *ShipOrder* activity. After the execution of this activity, the tool cannot propose the execution of *CloseOrder* (or any other activity in this case). A re-planning of activities is carried out here and the user is guided to execute the *FillOrder* activity. It allows the user to continue the execution with *SendInvoice* activity and continue till the completion of this process. This re-planning of sequence of execution for the activities is called process recovery. Process recovery helps the user to come back to the normal execution of the process, as specified initially. The current implementation of PRODAN can handle all types of deviations explained in Section 4.1. This ensures support for dealing with security and safety concerns in process model enactment.

4.4 Deviation recovery as a planning problem

Our approach to deviation recovery is to view it as an artificial intelligence planning problem. The initial state of the planning problem is the last state of the current process enactment trace. The goal state of the planning problem is the final state of the process model. The actions of the planning problem and their associated preconditions and postconditions are those of the process models. Just as for deviation detection, we thus translate the process-modeler oriented representation into a AI planner oriented representation. We then run the planner which returns a recovery plan, i.e., an ordered sequence of actions, or path, that leads from the initial state (i.e., the current enactment state) to the goal state (i.e., the final process model state).

Note that in the general case, there can be many different goal states and many different paths from the initial state to any of these goal states. Finding all paths might be computationally expensive and wasteful in practice. What we need to return to the process modeler is either one or a small set of plans to choose from based on domain-specific expertise. This indicates that the underlying planner must not merely find *any* plan but *the best* plan(s). In order to do so, the planner must thus be given some criteria to choose among alternative paths during planning. Common criteria involve minimizing resource usage such as enactment time or cost.

Note also that in some cases, there may also be no path leading to any goal state from the initial state. When this happens we alert the process modeler that the process deviations detected can no longer be fully corrected by any forward action sequences and we offer two possible courses of action. The first is to jump back to the last point in the deviation path and from there a full recovery plan is generated. The other is to propose *partial forward* recovery plans that minimize the deviation impact at final state. This impact is computed from the process model constraints that will remain violated in the plan.. In practice, this requires distinguishing between hard constraints that must not be violated, from soft constraints that can be relaxed, but their violation must be minimized. It may also

involve attributing weights to the soft constraints. This can be done, in part automatically, by exploiting the semantics of the user process model representation. However, in the general case, it may require the process modeler to manually tag or rank the various process model constraints for tie-breaking purposes.

Finally, it should be noted that the last two decades has witnessed a resurgence of reformulating the AI planning problem as either CSP or a **Constraint Optimization Problem (COP).** Many of the most efficient and scalable planners in recent planning competitions use this approach. This suggests that the process deviation recovery problem can be translated into a CSP or COP just as the deviation detection problem. Therefore, a common constraint solver can be re-used for both tasks and most of the steps to translate from deviation detection problem into a CSP problem accepted as input by such solver can also be reused to translate the deviation recovery problem into a CSP or COP problem.

5 Process Modelling Tool - PRODAN

In MERgE, we want to be able to enact software processes. We also want to use verification engines to automatically detect when the enactment deviated from the process model and search for possible deviation recovery plans that minimize the number of process model constraints violated at the end of enactment. We thus need a process modelling language with a standard operational semantics and axiomatic semantics. In the current state of the published standard, only the UML activity provides both, which is why it serves as the basis for the PRODAN tool.

PRODAN allows modelling the processes in the process editor view. Once the process model is specified using the UML activity diagram, it can be enacted from within the tool. The enactment interface of the tool allows enacting the process by executing the activities on the process model. It lists all the activities of the process model and gives user the possibility to execute any of the activities in the model at a given time. Process planning and recovery mechanisms guide the user through the enactment of the process by suggesting the next candidate activities for execution.

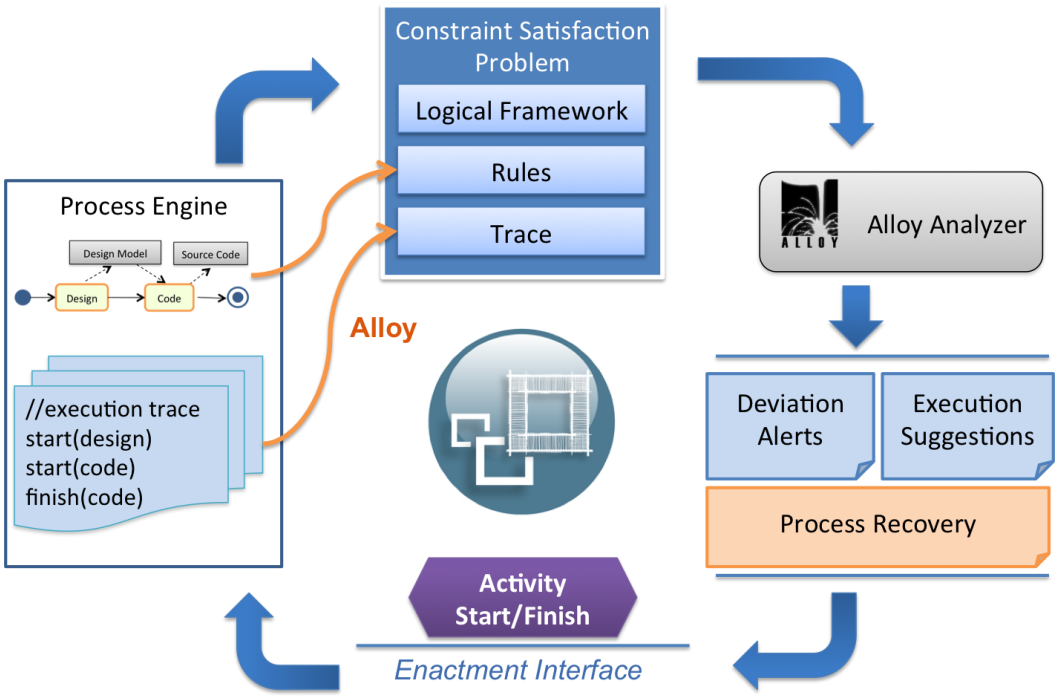

Figure 5: PRODAN Architecture

The process specified using the process editor in UML activity diagram is an imperative model. In our methodology we do not execute the activities of the process model directly. We transform this imperative process model to a declarative process model. This is carried out by the process engine implemented in PRODAN, as shown in Figure 4. The process engine takes the UML model as an input and transforms it into Alloy model. A metamodel for declarative process modelling is developed in Alloy, which serves as the output metamodel for the transformation definition. This transformation results in a declarative process model, which is a set of nodes (representing each activity of the process model) and a set of constraints (representing the control flow between the activities). These set of constraints are called process rule-set in our methodology. The process engine is also

responsible for managing the execution trace of the process. Besides managing the execution trace, it also keeps track of the state of currently executing activities.

The declarative process model generated from the specified process model can be treated as a CSP. It describes the logical framework for the execution of the process model, where a token is passed between each node to simulate its execution. This token is generated from the initial node and is then passed over to other candidate nodes without violating any constraints. The constraints in the system are termed as rules of the process rule-set. The first constraint specifies the first node to be executed in the process model, which by UML specification is the initial node. Other than the initial node constraint, there are three types of constraints: Response, Precedence and Existence. Let consider that two activities *a* and *b* are modelled in the process model such that the control flow passes the execution control to *b* after the execution of *a*. In this case, these constraints will be:

- *Response(a,b)* specifying that activity *b* must be executed anytime during the execution of the process, once activity *a* is executed. This does not constrain that activity *b* is to be executed directly after *a*.
- *Precedence(a,b)* specifying that activity *b* can be executed only if activity *a* has already been executed in the process. This does not constrain that activity *a* to be executed directly before the execution of activity *b*.
- *Existence(a)* specifying that activity *a* must be executed exactly once during the execution of the process.

The constraint satisfaction problem takes into account the logical framework, process rule-set and the execution trace from the process engine. This CSP is passed on to the solver of AlloyAnalyzer, which generates a solution for this problem. The solution of this problem is an execution path for the nodes such that it does not violate any of the given constraints. Generation of this execution path is termed as process planning in our methodology.

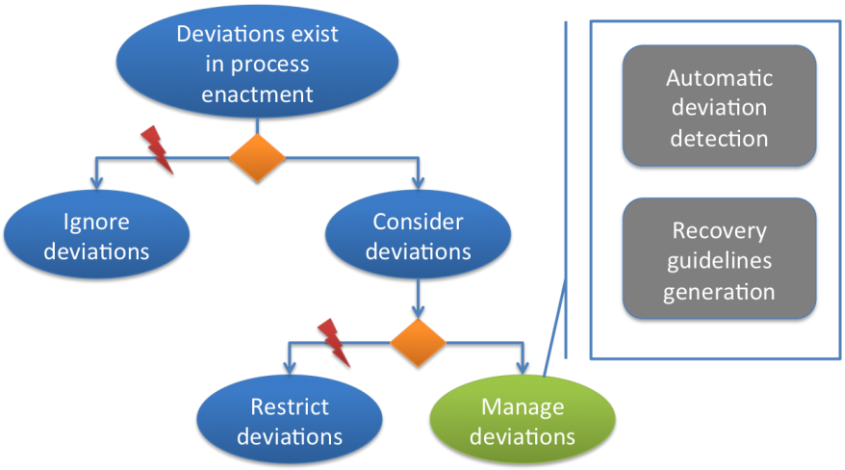

Figure 6: Process Deviations

Once the standard execution path is generated for the specified process model, it should execute all the activities in the given order. However, in the real life situations it is very hard to follow the exact specified process due to multiple reasons like volatility of the requirements, unforeseen situations that are not covered by the process, optimization of the process, etc. Figure 6 explains the situation of real life process enactments where deviations from the specified process are taking place. In these scenarios, one can choose of ignore the deviations, which results in situations where specified process is alienated from the real life process. In order to synchronise the real life processes with the process enactment, the tool has to consider the process deviations. Some of the tools consider the

occurrence of process deviations, but their enactment engines are not flexible enough to manage them. Thus, they restrict any deviation in process enactment, which results in situations where the users are restricted to the specified process, even if the situations demand to deviate. PRODAN considers the process deviation and gives the possibility to manage them effectively. For accomplishing it, our methodology offers the concepts of automatic deviation detection and process recovery.

Automatic deviation detection in the tool is implemented on the declarative process model. In case of a process deviation, one or more constraints from the process rule-set are violated. In case of violation of a constraint, the user is notified about the deviation, however the tool does not restrict the user from deviating. The warning to the user is generated alongside the details of the particular constraints that are being violated in case of a deviation. If the user chooses to deviate from the specified process, the re-planning of the process execution path is carried out. This re-planning of the execution path is called process recovery in our methodology. In this phase, the CSP takes into account the execution trace with the deviation. The activities that were skipped due to the deviation are considered as potential starting nodes for process recovery. Thus CSP is handed over to the solver of AlloyAnalyzer with last executed node and skipped nodes as starting nodes to generate multiple execution paths for the process enactment. These execution paths are suggested to the user in the process enactment interface. The final choice amongst multiple execution paths remains at the discretion of the user.

6 Application to Project Use Case Processes

One important reason of development of a process modelling methodology that can handle process deviations in project MERgE is to support its application across different industrial domains. There are four distinct use cases in the project that are focusing on the development of a demonstrator for each use case. These use cases are brought up by the industrial partners to show actual working constraints in the development of software systems, highlighting the security and safety issues.

6.1 Demonstrator processes

All of the use cases in the project focus on the multi-concern software modelling. None of the use case providers were following semi-automatically enactable process models in their routine software development practices. They thus could not deliver to us process models and enactment traces containing deviations on which to directly evaluate PRODAN However after extensive interactions with project partners, we were able to design some process models reflecting their practices. However, due to both intellectual property and privacy issues, these processes are sensitive information that cannot be fully disclosed in a public deliverable. An overview of these processes is given below.

6.1.1 Industrial Control Systems Process

A process from the Industrial Control Systems demonstrator was acquired through NSense. This is a process model that captures the service delivery mechanism used in that industry. This process was modelled using the process-modelling tool provided by UPMC. Further interviews were conducted to establish a knowledge base regarding the typical process deviations that occur in Industrial Control Systems demonstrator.

6.1.2 Aerospace Process

Space applications technical process (OAS-SA-PTD-001) was used to extract the software development processes used in the aerospace industry. An initial study on the flexibility of their process in terms of over and under-constrained processes was performed and an internal deliverable was sent to them. After this, their process was used as a case study for the demonstration of PRODAN enactment prototype.

6.1.3 Automotive Process

Melexis provided us with the Triaxis Software architecture. This state diagram of the software architecture was used to extract a basic process from this demonstrator. This allowed us to model this process. Its characteristics make it similar in nature that the processes extracted for the industrial control system and aerospace domains.

6.1.4 Radio Communication Process

Arcadia is a confidential proprietary process followed by some Thales division for model-driven systems engineering. It is generic process that specifies high-level cross-domain guidelines that can be followed in most Thales application domains, including the Radio Communication domain of Merge end-user partner and use case provider Thales Communication Systems (TCS). Melody Advanced is a proprietary modelling tool of Thales Global Systems (TGS) that Thales systems engineers can use to carry out the system modelling activities prescribed by Arcadia. Through TGS participation to the Eclipse foundation project Polarsys, the basic core modelling services provided by Melody Advanced have been released as an open-source tool called Capella. We thus studied Capella to analyse the small sub-process of Arcadia that it supports. We concluded that it does not present characteristics that make it very different in nature from the processes extracted from the other domains. Hence,

systems engineers using Capella, should be able to benefit from the process guidance services of the kind implemented in PRODAN.

6.2 Application to project use cases

In general, all the processes from different use cases represent the same sort of complexities. A software development process from any of the domains will constitute of some activities that might be sequenced in different fashions. These activities in real life might be performed differently in different domains. To explain the process modelling methodology, we are giving the results here from one of these processes.

Figure 7 shows a high level process by one of our industrial partners, NSense. This is a service delivery process from the industrial control systems demonstrator. For intellectual property and privacy protection reasons, we do not have the right to publish the complete process here. Thus, we have anonymised the process by changing the name of nodes with the letters of the alphabet and we do not give details on their descriptions. This process contains 19 *Actions*, 30 *ObjectNodes* with 28 *Pins* and 2 *ActivityParameterNodes*, 15 *ControlNodes* with 3 *MergeNodes*, 3 *DecisionNodes*, 4 *ForkNodes* and 5 *JoinNodes* ; 14 *ObjectFlows*, 40 *ControlFlows*, et 4 *Partitions.* Partitions are not part of the subset of fUML because they do not have execution semantics. They add information on the diagram, but do not affect its execution. Thus, the process includes (excluding partitions) *ActivityEdges* 54 and 64 *ActivityNodes* for a total of 118 elements in the UML model.

This process model was developed using the process editor of PRODAN tool. The tools allowed us to specify the process. The enactment of smaller parts of the process was possible from the enactment view, however the complete process was too big to be handled by the current process enactment mechanism. So if we carry out a *Single Entry Single Exit* (SESE) decomposition of the process, we can have multiple simpler processes from the same process model. We executed those SESE process models to demonstrate the capability of the tool.

Using the smaller decomposed process model chunks, we are able to enact the complete process model, where deviations from the process are automatically detected by PRODAN. In case of any deviation, the tool not only alerts the user about the deviation but also gives precise information about the constraints being violated. These constraints give insights about the exact location of the deviation. It helps the user to locate the exact problem area, in case the deviation was not really intended by the user and it is a safety or security loophole.

The tool also manages to give recovery guidelines to the user, by suggesting the activities to be performed next, after a deviation. The planning of the process is initially done according to the specified process. And the user is guided to follow the plan process initially. But once a deviation occurs, the normal course of execution is interrupted. In this case, the process engine goes for a re-planning of activities and suggests the activities to the user keeping in view the already executed activities through the execution trace.

6.3 Feedback from use cases

Application of our process modelling and enactment methodology on the processes from the four demonstrators of the project helped us collect the recommendations and feedback on our approach.

The feedback we received was generally positive concerning the practical usability of the tool for deviation detection on the use cases that they provided. It however pointed out that the assistance provided by the tool would be greatly enhanced in practice with the addition of a more comprehensive process recovery guideline functionality. They also laid down three desired requirements for this functionality. The first is that the guideline generated should not be limited to a set of possible single actions to execute immediately after deviation detection toward recovery, but should rather consist of genuine multi-step plans, i.e., action *sequences* leading from the current deviated point to one of the

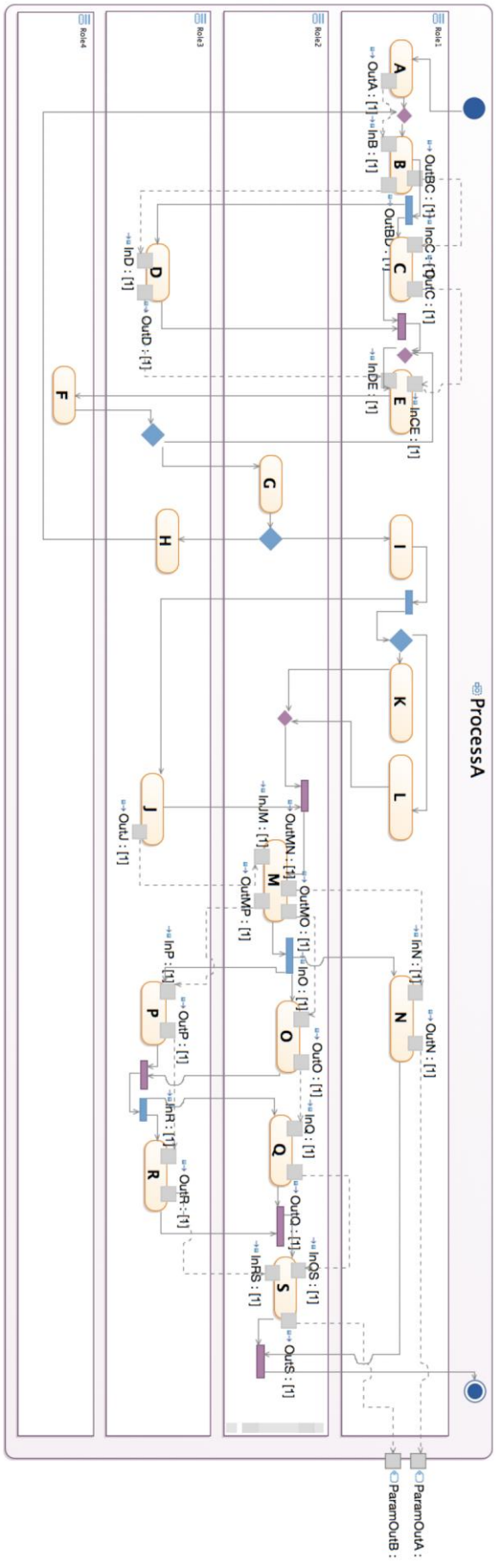

Figure 7: Anonymised process from the industrial partner

final goal states of the process. In addition, the feedback suggested a guideline consisting of multiple alternative recovery plans, ranked by increasing number of constraints violated by the final enactment trace resulting from appending to recovery plan to the executed actions from the initial state to the current deviated state. This would allow the project manager to quickly choose between recovery plans by making trade-offs based on domain expertise. Finally, it was suggested that the user should be able to label constraints in the process model by an importance weight to be taken into account for ranking the suggested recovery plans.

We will thus focus the next version of the prototype towards providing these functionalities and testing the efficiency of their execution using the process model test case that we presented earlier. In terms of underlying planning technology, we will consider using a Satisfiability Modulo Theory (SMT) solver instead of the pure SAT solvers provided by AlloyAnalyzer. This should improve the efficiency of both deviation detection and recovery plan generation with process models with non-Boolean, numerical constraints such as those dealing with resource allocation and deadlines.

7 Conclusion

The first deliverable of our team, consisted in an initial deviation detection prototype with its accompanying user manual. It was focused on explaining the concepts of deviation detection and deviation recovery guideline generation from the perspective of the user in didactic fashion using illustrative process models. In this second deliverable, we look “under the hood” and present the research concepts underlying the implementation of the prototype and discussed alternative possibilities for its future improvement. We also discuss real world process models that we elaborated in collaboration with use case partners in several industrial domains to help them assess the practical usability of our prototype for deviation detection. We also used these industrial process models to gather further requirements for deviation recovery guideline generation. In the next deliverable, we will provide a final prototype addressing these requirements. We will also describe performance tests of this final prototype using the process models described in this second deliverable

8 Bibliography

Ambriola, V., Conradi, R., & Fuggetta, A. (1997). Assessing process-centered software engineering environments . *ACM Transactions on Software Engineering and Methodology (TOSEM) , 6*(3), 283-328.

Bandinelli, S., Braga, M., Fuggetta, A., & Lavazza, L. (1994). The architecture of the SPADE-1 Process-Centered SEE . In B. C. Warboys, *Software Process Technology, Lecture Notes in Computer Science,* (Vol. 772, pp. 15-30). Springer Berlin Heidelberg .

Bendraou, R., Combemale, B., Cregut, X., & Gervais, M. P. (2007). Definition of an Executable SPEM 2.0. *The 14th Asia-Pacific Software Engineering Conference, APSEC 2007* (pp. 390-397). IEEE.

Bendraou, R., Jézéquel, J.-M., Gervais, M.-P., & Blanc, X. (2010). A comparison of six uml-based languages for software process modeling. *IEEE Transactions on Software Engineering, 36*(5), 662-675.

Cass, A. G., Lerner, A. S., McCall, E. K., Osterweil, L. J., Sutton, S. M., & Wise, A. (2000). Little-JIL/Juliette: a process definition language and interpreter. *The 2000 International Conference on Software Engineering, ICSE'2000* (pp. 754-757). IEEE.

Curtis, B., Kellner, M. I., & Over, J. (1992). Process modeling. *Communications of ACM, 35*, 75-90.

Davenport, T. H. (1993). *Process innovation: reengineering work through information technology .* Boston, USA: Harvard Business School Press .

Dowson, M., & Fernström, C. (1994). Towards requirements for enactment mechanisms . In B. C. Warboys, *Software Process Technology, Lecture Notes in Computer Science* (Vol. 772, pp. 90-106). Springer Berlin Heidelberg.

Hollingsworth, D. (2004). *The Workflow Reference Model: 10 Years On .* Fujitsu Services, UK; Technical Committee Chair of WfMC .

Keen, P. G. (1997). *The process edge: creating value where it counts.* USA: Harvard Business Review Press.

Lehman, M. (1991). Software engineering, the software process and their support . *Software Engineering Journal , 6*, 243-258.

Lindsay, A., Downs, D., & Lunn, K. (2003). Business processes- attempts to find a definition. *Information and Software Techonology*, 1015-1019.

Lonchamp, J. (1993). A structured conceptual and terminological framework for software process engineering. *The Second International Conference on the Software Process* , (pp. 41-53).

Maciel, R. S., Gomes, R. A., Magalhaes, A. P., Silva, B. C., & Queiroz, J. P. (2013). Supporting model-driven development using a process- centered software engineering environment. *Automated Software Engineering, 20*(3), 427-461.

Montoni, M., Santos, G., Rocha, A. R., Figueiredo, S., Cabral, R., Barcellos, R., et al. (2006). Taba Workstation: Supporting Software Process Deployment Based on CMMI and MR-MPS.BR. In J. Münch, & M. Vierimaa, *Product-Focused Software Process Improvement, Lecture Notes in Computer Science* (Vol. 4034, pp. 249-262). Springer Berlin Heidelberg .

OASIS. (2007). *Web Services Business Process Execution Language (WS-BPEL), Version 2.0*. Retrieved from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Object Management Group. (2008, April). *Software & Systems Process Engineering Meta-Model Specification (SPEM) Version 2.0*. Retrieved from http://www.omg.org/spec/SPEM/2.0/

Object Management Group. (2011). *Documents Associated with Business Process Model and Notation (BPMN), Version 2.0*. Retrieved from http://www.omg.org/spec/BPMN/2.0/

Object Management Group. (2011). *Documents Associated With Unified Modeling Language (UML), V2.4.1*. Retrieved from http://www.omg.org/spec/UML/2.4.1/

Object Management Group. (2013). *Documents Associated with Semantics of a Foundational Subset for Executable UML Models (FUML), V1.1*. Retrieved from http://www.omg.org/spec/FUML/1.1/

Object Management Group. (2014). *Documents Associated with Object Constraint Language (OCL) Version 2.4*. Retrieved from http://www.omg.org/spec/OCL/2.4/

OMG. (2008). Software & Systems Process Engineering Metamodel Specification (SPEM). Version 2.0.

OMG. (2015). (BPMN Model Interchange working group) Retrieved from http://www.omgwiki.org/bpmn-miwg/doku.php

Osterweil, L. J. (1987). Software processes are software too. *The 9th international conference on Software Engineering, ICSE '87* (pp. 2-13). Los Alamitos: IEEE Computer Society Press .

Ouyang, C., Dumas, M., Hofstede, A. H., & Aalst, W. M. (2006). From BPMN Process Models to BPEL Web Services. *International Conference on Web Services, ICWS '06* (pp. 285-292). IEEE.

Rossi, D., & Turrini, E. (2007). Using a process modeling language for the design and implementation of process-driven applications. *International Conference on Software En- gineering Advances. ICSEA 2007* (p. 55). IEEE.

Silva, M. A. (2012). *Detection and Handling of Deviations in Process-Centered Software Engineering Environments.* PhD Thesis, LIP6/ Université Pierre et Marie Curie, Informatique.

Sutton, S. M., & Osterweil, L. J. (1997). The design of a next-generation process language. In M. Jazayeri, & H. Schauer, *Software Engineering – ESEC/FSE'97, Lecture Notes in Computer Science* (Vol. 1301, pp. 142-158). Springer Berlin Heidelberg.

Türetken, O. (2007). *A method for decentralized business process modeling.* The Middle East Technical University. PhD Thesis.

Weigold, T. (2010). A generic framework for process execution and secure multiparty transaction authorization, University of Westminster, PhD Thesis.

Weigold, T., Aldinucci, M., Danelutto, M., & Getov, V. (2012). Process-driven biometric identification by means of autonomic grid components. *International Journal of Autonomous and Adaptive Communications Systems, 5*(3), 274-291.

Workflow Management Coalition. (2012). *XML Process Definition Language (XPDL)*. Retrieved from http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20(2012-08-30).pdf