THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *26/01/2018* **par :**

Mojtaba Hajmoosaei

A Bottom-up Process Management Environment dedicated to Process Actors

JURY

Antoine Beugnard	Professeur IMT Atlantique Bretagne	Rapporteur
Eric Cariou	Maître de conférences Université de Pau	Examinateur
Marie-Pierre Gervais	Professeur Université Paris Nanterre	Rapporteur
Chihab Hanachi	Professeur Université Toulouse 1	Examinateur
Christian Percebois	Professeur Université Paul Sabatier	Directeur
Hanh Nhi Tran	Maître de conférences Université Paul Sabatier	Co-directeur

École doctorale et spécialité :

MITT : Domaine Mathématiques : Mathématiques appliquées

Unité de Recherche :

Toulouse Institute of Computer Science Research (IRIT)

Directeur(s) de Thèse :

Christian Percebois et *Hanh Nhi Tran*

Rapporteurs :

Antoine Beugnard et *Marie-Pierre Gervais*

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisers Prof. Christian Percebois and Dr. Hanh Nhi Tran for the continuous support of my Ph.D study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. They taught me how to ask questions and express my ideas. They showed me different ways to approach a research problem and the need to be persistent to accomplish any goal. In addition to our academic collaboration, I greatly value the close personal rapport that my advisers and I have forged over the years. I could not have imagined having a better advisers and mentors for my Ph.D study.

Besides my advisers, I would like to thank the rest of my thesis committee: Antoine Beugnard, Marie-Pierre Gervais, Eric Cariou and Chihab Hanachi for their encouragement, insightful comments, and hard questions.

I thank my team mates in IRIT MACAO: Herve Leblanc Amani Makhlouf, Robin Bussenot and Nicolas Hili for all the interesting discussion and fun we have had in the last four years. Moreover, I am grateful to our industrial partners who dedicated their precious time during this doctoral study and helped us on developing the case studies for our research and on validating our prototype.

Finally, and most importantly, I would like to thank my family. Their support, encouragement, quiet patience and unwavering love were undeniably the foundations upon which the past seven years of my life have been built. This last word of acknowledgement I have saved for my dear wife Larissa Haake, who has been with me all these years and has made them the best years of my life.

Abstract

Companies increasingly adopt process management environments, which offer promising perspectives for a more flexible and efficient process execution. Traditional process management environments embodies a top-down approach in which process modeling is performed by process designers and process enacting is performed by process actors. Due to this separation, there is often a gap between process models and their real enactments. As a consequence, the operational level of top down process environments has stayed low, especially in system and software industry, because they are not directly relevant to process actors' needs.

In order to facilitate the usage of process environments for process actors, this thesis presents a user-centric and bottom-up approach that enables integration of process actors into process management life cycle by allowing them to perform both the modeling and enacting of their real processes. To this end, first, a bottom-up approach based on the artifact-centric modeling paradigm was proposed to allow each process actor to easily describe the process fragment containing the activities carried out by his role.

The global process is thus decomposed into several fragments belonging to different roles. Each fragment can be modeled independently of other fragments and can be added progressively to the process model; therefore the process modeling becomes less complex and more partial. Moreover, a process fragment models only the structural aspect of a role's activities without anticipating the behavior of these activities; therefore the process model is less prescriptive.

Second, a data-driven process engine was developed to enact activities coming from different process fragments. Our process engine does not require predefined work-sequence relations among these activities to synchronize them, but deduces such dependencies from their enactment-time exchanged artifacts. We used a graph structure name Process Dependency Graph (PDG) to store enactment-time process information and establish the dependencies among process elements.

Third, we extend our process environment in order to handle unforeseen changes occurring during process enactment. This results in a Change-Aware Process Environment that allows process actors reporting emergent changes, analyzing possible impacts and notifying people affected by the changes.

In our bottom-up approach, a process is split into several fragments separately

modeled and enacted by process actors. Our data-driven process engine, which uses the availability of working artifacts to synchronize activities, enables enacting independently process fragments, and even a partially modeled process where some fragments are missing. The global process progressively emerges only at enactment time from the execution of process fragments. This new approach, with its simpler modeling and more flexible enactment, integrates better process actors into process management life cycle, and hence makes process management systems more attractive and useful for them.

Résumé

Les organisations adoptent de plus en plus les environnements de gestion des processus car ils offrent des perspectives prometteuses d'exécution en termes de flexibilité et d'efficacité. Les environnements traditionnels proposent cependant une approche descendante qui nécessite, de la part de concepteurs, l'élaboration d'un modèle avant sa mise en œuvre par les acteurs qui le déploient tout au long du cycle d'ingénierie. En raison de cette divergence, un différentiel important est souvent constaté entre les modèles de processus et leur mise en œuvre. De par l'absence de prise directe avec les acteurs de terrain, le niveau opérationnel des environnements de processus est trop faiblement exploité, en particulier en ingénierie des systèmes et des logiciels.

Afin de faciliter l'utilisation des environnements de processus, cette thèse présente une approche ascendante mettant les acteurs du processus au cœur de la problématique. L'approche proposée autorise conjointement la modélisation et la mise en œuvre de leurs activités quotidiennes. Dans cet objectif, notre approche s'appuie sur la description des artéfacts produits et consommés durant l'exécution d'une activité. Cette description permet à chaque acteur du processus de décrire le fragment de processus exprimant les activités dictées par son rôle.

Le processus global se décompose ainsi en plusieurs fragments appartenant à différents rôles. Chaque fragment est modélisé indépendamment des autres fragments ; il peut aussi être greffé progressivement au modèle de processus initial. La modélisation des processus devient ainsi moins complexe et plus parcellaire. En outre, un fragment de processus ne modélise que l'aspect structurel des activités d'un rôle sans anticiper sur le comportement des activités ; il est moins prescriptif qu'un ordonnancement des activités de l'acteur.

Un moteur de processus basé sur la production et la consommation d'artéfacts a été développé pour promulguer des activités provenant de différents fragments de processus. Ce moteur ne requiert pas de relations prédéfinies d'ordonnancement entre les activités pour les synchroniser, mais déduit leur dépendance à partir de leurs artéfacts échangés. Les dépendances sont représentées et actualisées au sein d'un graphe appelé Process Dependency Graph (PDG) qui reflète à tout instant l'état courant de l'exécution du processus.

Cet environnement a été étendu afin de gérer les changements imprévus qui se

produisent inévitablement lors de la mise en œuvre des processus. Ce dispositif permet aux acteurs de signaler des changements émergents, d'analyser les impacts possibles et de notifier les personnes affectées par les modifications.

En résumé, notre approche préconise de répartir les tâches d'un processus en plusieurs fragments, modélisés et adoptés séparément par les acteurs du processus. Le moteur de processus, qui s'appuie sur la disponibilité des artéfacts pour synchroniser les activités, permet d'exécuter indépendamment les fragments des processus. Il permet aussi l'exécution d'un processus partiellement défini pour lequel certains fragments seraient manquants. La vision globale de l'état d'avancement des différents acteurs concernés émerge au fur et à mesure de l'exécution des fragments. Cette nouvelle approche vise à intégrer au mieux les acteurs du processus dans le cycle de vie de la gestion des processus, ce qui rend ces systèmes plus attractifs et plus proches de leurs préoccupations.

Contents

mtcsettitleminitocList of Sections

Chapter 1

Problematic and Contributions

Contents

This chapter aims at providing a synthesis of our work. We present the context, the underlining problematic and outline our contributions.

1.1 Basic Concepts of Process Management

In practice, each organization uses some operational processes to accomplish their business goals in an application domain. These processes are enacted by process actors who are domain experts with specific skills in the given domain. Such operational processes are however not always documented in a clear and unambiguous way. Consequently, it is difficult to monitor the enactment of these processes and thus also difficult to improve them.

Process Management is the discipline that aims at identifying and formalizing processes in order to communicate, improve them and to control their execution [10, 48]. It has received considerable attention in recent years due to its potential for increasing productivity and saving costs. Process management comprises two main phases as *process modeling* and *process enactment*. Process management environments are software systems that support modeling and enacting processes.

The next sections give the basic concepts of process modeling and process enactment which are necessary to our work.

1.1.1 Process Modeling

Process modeling is the activity of representing processes, so that the examined process can be communicated, improved or automated. Process modeling is a critical component for a successful process management. Normally, process modeling is conducted by process designers who have process modeling techniques with the participation of process actors who are equipped with detailed domain knowledge about the process to be modeled. The objective of process modeling is to produce a process model, represented in a chosen notation, from the informal descriptions given by process actors. In general, a process model can be used to pursue two main objectives: (1) to improve process by getting a common basis which process actors can communicate and discuss about and result in increasing organization performance; (2) to control the enactment of the modeled process with the help of a process management environment.

Generally, process modeling is conducted in two steps: *process elicitation* and *process formalization*.

- *Process elicitation* In this step, process designer first gathers information from process actors who have intimate knowledge about how a process and its activities are performed in practice. The outcome of this process discovery is a process description reflecting the understanding of process actors in the organization about how their work is carried out.

  Once the process description is obtained, sometimes process designer need to refine the process to get enough details allowing making a fine-grained model, especially if the objective of process modeling is automatizing the control of process enactment by using a process management environment.

- *Process formalization* This step aims to provide a formal and unambiguous representation of the process. A process designer uses a process modeling language (PML) in order to create a process model corresponding to the process description obtained in the process elicitation [10]. A PML can be non-executable or executable. A non-executable PML is enough to model processes for communication purpose. If the purpose of modeling is controlling process enactment, an executable PML defining a precise operational semantics for process modeling concepts is needed so that the process model can be interpreted by the target process management environment.

Today, two most popular process modeling approaches are activity-centric and artifact-centric process modeling [30].

- Activity-centric process modeling describes a process based on the control-flow between its activities. It uses activities and work-sequence relations between them as first class modeling constructs. Most of the mainstream PMLs proposed over the last two decades are activity-centric as Software Process Engineering Metamodel (SPEM) [32] and Business Process Modeling Notation (BPMN) [33].

- Artifact-centric process modeling describes a process based on the data-flow passing through its activities. It regards data objects and their life cycles as first class modeling constructs [31, 26]. The life cycles describe the allowed data object manipulations and the interdependencies of multiple data objects in the course of process model execution. Languages such as Guard-Stage-Milestone (GSM) [22], Case Management Model and Notation (CMNN) [34] are proposed to define the life cycles of artifacts.

Each of process modeling approaches expresses a different point of view by considering either activity or artifact as their main modeling construct. This impacts the operational semantics of a PML at enactment.

1.1.2 Process Enactment

Process enactment is the activity that applies a process model to create and perform activities in a concrete project. Process enactment can be done with the help of a process management environment. In this context, we can see an enactable process model, i.e. process model represented in an executable PML, as a program and the process management environment supporting that PML is the process execution engine (process engine) of process models. Process enactment now means the process engine executes a process model by creating a corresponding process instance which is managed by the process environment, i.e. the process engine controls the creation, execution and termination of process's activities.

In order to enable a process engine to enact process models, the operational semantics of the modeling language must be defined. Operational semantics define precisely how the modeling language concepts will be instantiated and evolved during process enactment. The process engine implements the defined operational semantics to transform a valid process model into a running process.

A process engine needs to synchronize correctly all running activities, i.e., guaranteeing that the activities are performed according to the execution constraints specified in the process model. To do so, during the enactment of the process, the process engine uses the process model as a recipe to determine the sequence of activities to be executed. As said earlier, the process modeling approach determines the synchronization mechanism of process enactment. Activity-centric process modeling requires activity-driven enactment whereas artifact-centric process modeling requires data-driven enactment.

- Activity-driven enactment uses the event related to activity completion as the primary driver for the progress of a process. The process engine needs to knows the work-sequence betweens process's activities to synchronize them.

- Data-driven enactment uses the availability or the changes on states of data objects as the primary driver for the progress of a process. To start or stop an activity, the process engine needs to know if the preconditions or postconditions concerning the data objects used or produced by the activity are satisfied.

During process enactment, execution data regarding stateful process element instances is gathered. This execution data provides the basis for process engine to synchronize and monitor running processes.

1.2 Problem Statement

Generally, process management in most existing process environments adopts the activity-driven approach and is carried out in a top-down manner. As illustrated in Figure 1.1, process modeling is then performed by process designers with the indirect participation of process actors or people having domain knowledge. The result of modeling phase is a completely defined process model which can be enacted with support of an activity-driven process engine. That means that process actors perfom their activities as defined in the model under control of process engine. Based on this method, several process environments have been proposed during last decades to control and monitor the execution of processes related to System and Software Engineering (SSE) process or business process either in academic or commercial level, e.g., jBPM [23], AristaFlow [5], Bonita BPM Suite[7].

Notwithstanding the popularity of top-down process management, there are voices claiming that top-down process management relies too much on process designers, making it an activity performed by process actors but not really defined by them [49, 12, 35]. Some studies [37, 3, 35, 20] state that process actors are often cut from the development cycle of processes after their information on the process has been captured once. As a consequence, there is only little proactive feedback from process actors regarding modeling their processes.

Generally, for complex processes, it is usually very difficult for the process designer to define properly the entire process model before the actual activities are performed by process actors during project execution. Process models often do not reflect the real processes as perceived and supported during project execution by process actors. This fact has been led to limited adoption of process environments in industry, specially in Software and System Engineering (SSE) [16, 4, 28, 2, 9].

We present the reason of such rejection as a gap between what are described in a process model and what happen when enacting the process in a concrete project. We

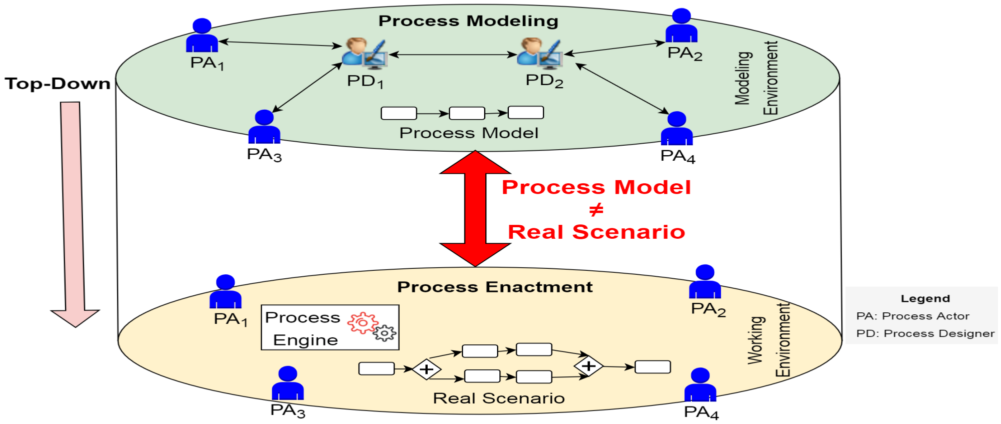


Figure 1.1: Activity-driven and top-down process management

identified the primary causes of the gap and classified them as following:

- *Dissimilar knowledge* As mentioned earlier, a process is modeled via collaboration of two roles as process designer and process actor. Process designers are experts in modeling and have through knowledge regarding modeling techniques and notations, i.e., process knowledge. However, they lack knowledge regarding process application, i.e., domain knowledge. On the other side, process actors who are operatively involved in work processes, are domain experts with extensive knowledge about their respective roles in a work process, but normally have little methodological knowledge about process modeling [38]. The role of process actors in top-down process modeling is limited to provide domain knowledge for process designers.

  In complex processes such as SSE, domain knowledge is vast in which a process comprises several expertise domains where each actor has its own local view of process, i.e., related to his expertise domain. Process designers require to acquire and combine knowledge of different expertise domains in order to model the global process. This issue can become a very difficult, time and effort consuming task for process designers.

- *Mixing temporal aspects* At process modeling-time, process designers and process actors try to describe not only the structural aspect of the process but also its behavior which occurs later at process enactment-time and thus is difficult to be fully anticipated.

  Envision all possible execution paths at modeling-time can result in an inaccurate process model [29] as there can be some situations which are not anticipated in the model. Moreover, it can bring rigidity as process designer must define the

> complete process model before enactment which the model enforces the prescribed procedure on process actors. Finally it can make the process models very complex as process designers require to apply different modeling constructs to describe all anticipated execution scenarios [29, 15].

To remedy the problem of top-down process management, many works either focus on facilitating process modeling by process actors with the objective of process improvement or providing flexibility to process actors. In the former, many researchers proposed end-user modeling approaches that better integrate process actors into process management, enabling them to proactively contribute to process development. These approaches consider process actors as the first-citizens in process modeling who are the ones that have knowledge and use the system at the end. However the degree and the way of actor's involvement are varied, i.e., in a centralized way [13, 43] or fragmented way [46, 44]. They state that when the process actors model their own contribution of process, a more detailed and accurate model representing the actual work is obtained [36, 11].

In the latter, many researchers proposed adaptive and flexible process environments. Flexibility can be achieved by enabling the enactment of loosely specified process models which include structural aspect of process and also rules and constraints to prohibit the undesired execution behavior, e.g., declarative approaches [47, 39, 1, 24, 40, 41]. Some other works provide flexibility by allowing process actors to adapt the enactment by only making alternations to the model or running instances, e.g., planned or ad-hoc process deviation [6, 50, 42, 27, 8, 5], refinement-based approach [14], etc.

The challenges addressed in both areas are of high relevance for the aim of improving process management. However, the proposed solutions are not really user-centric in order to be relevant to process actor's real works and needs.

Being aware of the benefits of process actors's participation in process management life cycle, the main challenge underlining this thesis is to how we can enable them actively contribute in process modeling and enactment. In other words, we aim at turning the procedure of process management from top-down to bottom-up where process actors have more autonomy in controlling their processes.

Currently there is a discussion in the process management community of how to properly integrate process actors, since they are the process performers having much more information about their process than process designers. The accuracy of the model that represents the real scenario directly increases proportional to the involvement of the process actors in process management. However, to properly move in the direction of user-centric process management, we require to provide some degree of autonomy and control to process actors regarding modeling and enactment of their processes. This autonomy can cause process actors to model and enact their processes as they occur in reality.

1.3 Objective and Contributions

We aim at a user-centric approach to seamlessly integrate process actors into the entire process management life cycle in order to bring down the gap happening in the transition from process modeling to process enacting. We hope that by reducing this gap, our solution can help promoting the use of process management environments.

In this thesis, the main objective of process modeling is process control than process improvement. While many results have been obtained on process improvement and provide good basis for process communication, as mentioned earlier the operational level of process control has been low. Thus, our primary objective is process control which a process modeling produces an executable process model that can be directly enacted by the support of a process engine. We assume that there is an already (informally) defined process which process actors perform their activities but without having a dedicated tool to control the process, i.e., systematic synchronization of their activities.

As shown in Figure 1.2, we propose to conduct process management in a bottom-up manner directly by process actors [17, 45]. Our objective is providing process actors a user-friendly process management environment that allows them to smoothly switch from modeling to enacting their working process. First, we aim at eliminating the need of process designer role in process modeling and empowering process actors to properly express their working processes. Second, we put the main emphasis on enactment-time data as the basis of synchronization to reflect better process actors' real behavior. The main propositions of our bottom-up process management are as follows:

- *Letting each process actor modeling his own fragment of process to eliminate the knowledge cause of gap* In order to model accurately an executable process, the understanding of both process technology domain and the application domain of the project are required. It is always challenging to make a process designer understand well the process's application domain to describe the whole process, especially when this later concerns several expertise domains. It is more feasible to make a process actor learning a process modeling solution to describe his working process, especially when the modeling solution is simple and does not require information outside his role's knowledge. Each process actor can model his contribution to the process which can be seen as a fragment of process belonging to his role.

- *Describing only structural aspect of the process at modeling-time to remove the temporal cause of the gap* In a top-down approach, process engine synchronizes activities based on their work-sequence relations which are described in the behavioral aspect of the process. Trying to model the complete behavior of a process before its applications in a project, however, is often source of complexity, rigidity and inaccuracy of process modeling. In fact, the dependencies among activities

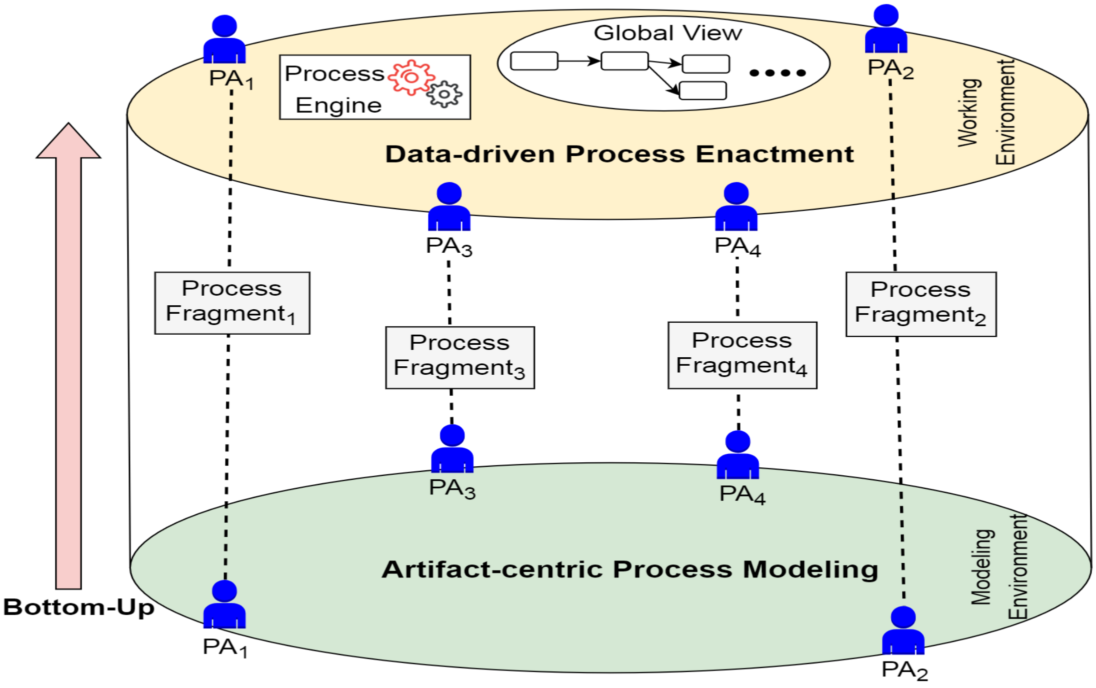


Figure 1.2: Bottom-up process management

can be deduced at enactment-time from the artifacts that they exchange, which are already described in structural aspect of the process. Such a synchronization is practically viable because in daily work process actors manage their artifacts [25, 26], i.e., specially in SSE processes. Thus, to reflect more exactly the reality, an efficient modeling approach would rather focus on describing the process's artifacts than process work-sequences relations.

We develop a *Bottom-up Artifact-centric Process Environment (BAPE)* for the support of process management. The BAPE environment has been developed in the ACOVAS (outil Agile pour la COnception et VAlidation Système) FUI project that was established in the context of this thesis. The core of the BAPE environment shall cover the modeling and enactment stages of process management life cycle as well as process monitoring. In contrast to conventional process environments which are typically built to suit the needs of process experts, our process environment must be easy to be used by process actors.

In the following we present the main points of the contributions of this thesis. The details of each contribution will be presented later in the separate chapters.

Artifact-centric Process Modeling

Enabling process actors to model themselves their operational process requires a modeling approach reflecting as exactly as possible how a process actor, playing a role, sees the process fragment that he performs. Moreover, to allow process actors performing the modeling with ease, a simple language using the concepts known by process actors in their daily work is advisable. In practice, generally a process actor often describes his process as a list of activities transforming working artifacts into some specific states, without giving explicit execution order of these activities.

Considering the above requirements and observations, we exclude the use of top-down modeling approach and activity-centric process modeling languages as BPMN [33] and SPEM [32]. In contrast, first we propose a bottom-up approach which assembles the global process model from several fragments; each fragment describes only the activities performed by a given role. Secondly, inspired by the artifact-centric modeling paradigm [21] which uses artifacts to combine the data and process logic, we define a simple process modeling language named *Structural Process Modeling Language (SPML)*.

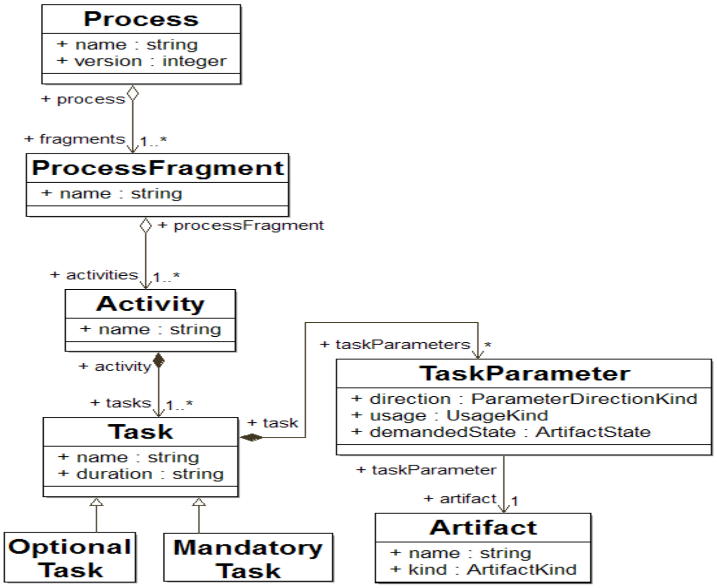
The core part of SPML meta-model is illustrated in Figure 1.3a. Note that Figure 1.3a is only shown to give an overview of the SPML whose complete meta-model is presented in the Chapter 2. A SPML process is comprised of several process fragments. Each fragment encapsulates the activities performed by a specific role. An activity describes a work composed of enactable tasks (mandatory or optional) corresponding to different sub-objectives. A task manipulates artifacts in some given states and puts them into required states. The relation between a task and an artifact allows specifying how the task uses the artifact.

Figures 1.3b and 1.3c show simple SPML process models regarding the fragments of role R_1 and R_2 in process P_1. For instance, in activity A_1 of R_1, mandatory task MT_1 consumes artifact Art_1 in state S_1 and produces artifact Art_2 in state S_1. Moreover, mandatory tasks MT_1 of R_1 and MT_2 of R_2 implicitly exchange the artifact Art_2 in state S_1.

SPML suppresses the work-sequence relations in process models but strengthens the description of data-flows in order to provide essential information for synchronizing activities. Concretely, SPML describes the preconditions and postconditions on artifacts used and produced by a task. Based on the changes on states of artifacts exchanged between tasks, the behavioral aspect of process emerges at enactment-time.

Our modeling approach allows fragmented process modeling in which each actor defines his fragment of process belonging to his role separately from other process fragments. To do so, we assume that process actors share common information about their global process, particularly about artifact definitions. Aligning different process information sources is out of scope of this thesis.

SPML provides a simple language containing only the process elements known by process actors but enough expressive to allow them defining real situations in their daily



(a) Core of SPML meta-model

```
Process $P_1$ options $O_1$, $O_2$
  ProcessFragment $R_1$
    Activity $A_1$ options $O_1$
      MandatoryTask $MT_1$
        Artifact $Art_1$(in, $S_1$)
        Artifact $Art_2$(out, $S_1$)
      OptionalTask $OT_1$ ifOption $O_1$
        Artifact $Art_2$(in, $S_2$)
        Artifact $Art_3$(out, $S_1$)
```

(b) SPML process fragment of R_1

```
Process $P_1$ options $O_1$, $O_2$
  ProcessFragment $R_2$
    Activity $A_2$ options $O_2$
      MandatoryTask $MT_2$
        Artifact $Art_2$(in, $S_1$)
        Artifact $Art_2$(out, $S_2$)
      OptionalTask $OT_2$ ifOption $O_2$
        Artifact $Art_3$(in, $S_1$)
        Artifact $Art_3$(out, $S_2$)
```

(c) SPML process fragment of R_2

Figure 1.3: Structure of SPML

work.

Data-driven Process Enactment

As the process modeling paradigm determines the process execution mechanism, to enact SPML process models, we proposed an enactment approach with the following features:

- Using a data-driven process execution mechanism where the main driver to progress an activity is the availability of artifacts in the demanded states.

- Offering a user-centric control where process actors have enough autonomy to decide the way they enact their processes as happen in reality.

In order to support aforementioned features we propose:

> *A Process Dependency Graph (PDG)* To define the operational semantics of a language, a structure keeping the information of process element instances at enactment-time is needed. For SPML, such a structure is Process Dependency Graph (PDG). PDG [45] is a graph structure that stores SPML process elements instances along with the dependencies among them.
> We define a direct mapping between the PDG concepts and the SPML concepts. Each element of PDG has a reference assigning them to their corresponding element in SPML as illustrated in Figure 1.4. PDG nodes represent the instance of SPML basic concepts and PDG edges represent the association among instances of these concepts. As the PDG represent the global view of system, we do not require a specific PDG node to represent SPML *ProcessFragment* concept.

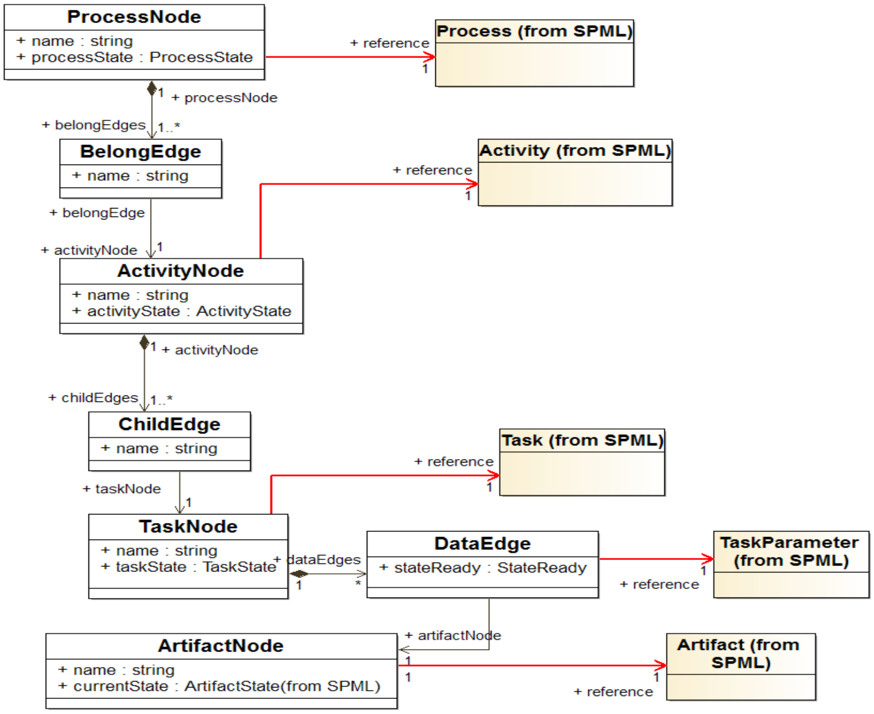


Figure 1.4: Core of PDG structure

> PDG's concepts are stateful to show the behavioral aspect of the running process. Thus each PDG element is described with its current state in the system.

Moreover, as process models are fragmented in our system, the role of PDG is not only storing running information but also establishing the global view of the system from separate process fragments. PDG unifies the process fragments at enactment-time based on shared artifacts and resources among activities instances of respective fragments.
Figure 1.5 gives a snapshot of PDG regarding the instances of process fragments R_1 and R_2 which their activities are enacted respectively by process actor $Actor_1$ playing role R_1 and $Actor_2$ playing role R_2. At the moment, task instance $MT_{1.1}$ consumed artifact instance $Art_{1.1}$ and produced $Art_{2.1}$ in state S_1. $Art_{2.1}$ is consuming by task instance $MT_{2.1}$. Task instance $OT_{1.1}$ is waiting for the artifact instance $Art_{2.1}$ in state S_2 which is under production by running task $MT_{2.1}$.

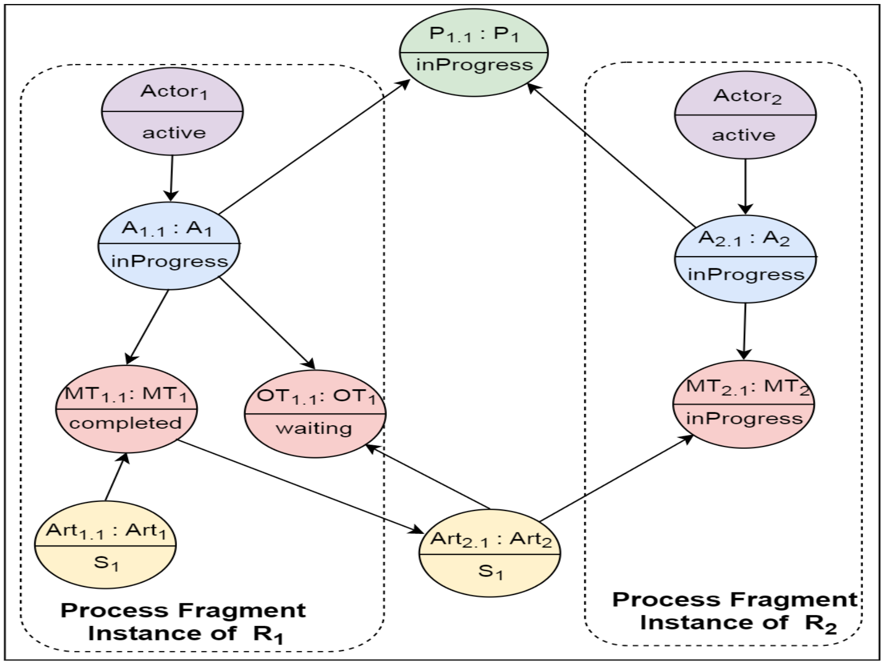


Figure 1.5: Snapshot of PDG

PDG is the source of information for process synchronization. When it comes to the process synchronization where process engine requires to know the dependencies among running process elements, having a mechanism implementing a graph structure is a promising solution which perfectly represents the connected information and provides highly effective query and traversal mechanisms.

A Data-driven Process Engine In SPML process models, there is no work-sequence relations defined among process activities or tasks. Thus, to enact SPML process models, we propose a process engine adopting a data-driven execution mechanism.

The progress of tasks, activities and processes are based on the states at enactment-time of the artifacts that they use or produce. The process engine analyzes the real data exchanged among the running tasks to deduce the work-sequence relations among process activities or tasks at enactment-time. Therefore, the global view of the system which is used for synchronizing all activities and for process monitoring is dynamically constructed based on the real situations occurring at enactment-time.
We present the SPML operational semantics by the state-machines associated to the SPML executable elements. SPML operational semantics define the possible state transitions of process, activity and task instances. By adopting the data-driven execution mechanism, we define the state transitions of these instances based on the states of their consumed or produced artifacts. The respective state machines pilot the process engine by enacting and evolving the process, activity and task instances in the PDG. For instance, Figure 1.6 gives our process state machine specifying the operational semantics of a process instance. The life cycle of a process is dependent on its activities, the life cycle of an activity itself depends on its tasks and finally the life cycle of a task depends on its consuming/producing artifacts.

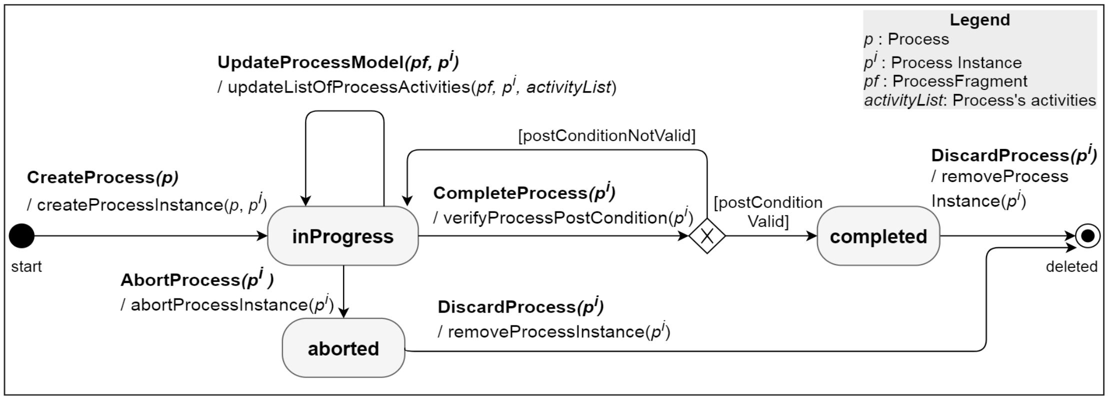

Legend
p : Process
p^i : Process Instance
pf : ProcessFragment
activityList: Process's activities
UpdateProcessModel(*pf*, p^i)
/ updateListOfProcessActivities(*pf*, p^i, *activityList*)
[postConditionNotValid]
CreateProcess(*p*)
/ createProcessInstance(*p*, p^i)
CompleteProcess(p^i)
/ verifyProcessPostCondition(p^i)
[postCondition Valid]
DiscardProcess(p^i)
/ removeProcess Instance(p^i)
start
inProgress
completed
deleted
AbortProcess(p^i)
/ abortProcessInstance(p^i)
aborted
DiscardProcess(p^i)
/ removeProcessInstance(p^i)

Figure 1.6: Process state machine

We emphasize the importance of providing flexibility to process actors in order to deal with unforeseen situations at enactment-time. Thus, the proposed process engine:

- Offers more control to process actors by allowing the definition of necessary transitions in the state machines of process, activity and task instances to allow capturing events emerging during enactment-time.
- Enables different configurations of the activities executions in response to different execution contexts (e.g., optional activities) or unforeseen situations

(e.g., rework) that may occur at enactment-time.

- Supports the enactment of partially-defined processes in which some fragments can be modeled and enacted where other process fragments are missing. This is achieved thanks to data-based synchronization mechanism.

By using the conditions based on the artifact states to decide the progress of task execution, our process engine better integrates the data into tasks life cycles and thus provides a finer control on the execution of tasks. While conventional process engines rely exclusively on the process actors to make progress a task execution, our process engine ensures that the execution of a task makes progress the concerned artifacts as required in the process models.

Bottom-up Artifact-centric Process Environment BAPE

To enable process actors to model and enact their executable process fragments, we propose our process environment BAPE which integrates our modeling and enactment solutions in a comprehensive environment. Figure 1.7 gives an overview of BAPE's architecture comprising its main components SPML, process engine, PDG and interactions among these components.

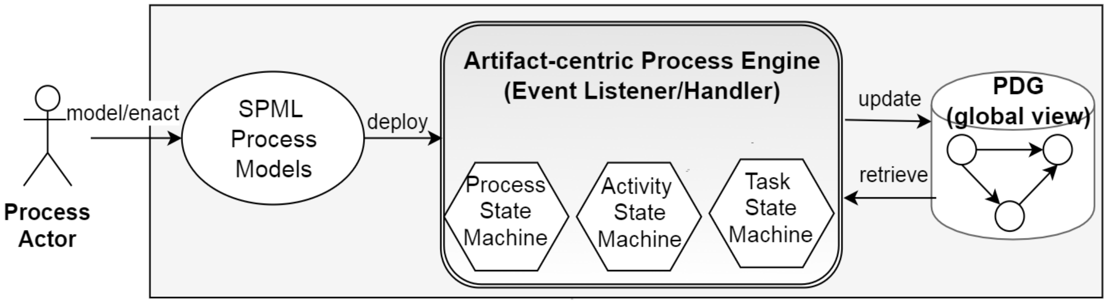


Figure 1.7: Overview of BAPE architecture

Process actors can model their process fragments in SPML and enact them by support of the process engine through user-friendly modeling and enactment interfaces. Process engine as the core of BAPE plays a role of event listener and handler by listening to different events as specified in the state machines. These events are regarding starting, termination, *etc.* of process element instances which result in updating PDG by creating and evolving respective process elements instances. As PDG establishes the global view of system, process engine uses the information of PDG in order to provide synchronization among activities instances via shared resources which are enacted from separate process fragments.

As an advantage of BAPE's architecture we can mention its extensible characteristic.

BAPE allows executions of different system processes (e.g., change management process, quality management process, risk management processes, etc.) without requiring any modifications on the core engine. Each of these processes adds a new functionality to BAPE and can be implemented and run in parallel with other processes.

Change-aware Process Environment

In many dynamic environments, unforeseen changes occurring during process enactment are almost inevitable but often poorly managed due to lack of efficient mechanism for spontaneously handling these enactment-time changes.

As process actors have a partial view on the global process, often they do not know how their work relates to other tasks and have difficulties, even impossibility, in identifying the right persons to communicate with for quickly resolving a problem. Lack of information on task connections can lead to unnecessary repetition or inconsistent results, which in turn will require rework or changes in other tasks. Insufficient communication usually discourages process actors to report all the changes they made or obliges them to follow hierarchical channels for propagating a change, where it can be delayed or misinterpreted.

We don't have the ambition to propose a complete solution for change management. We are primarily interested in the change notification issue. Our objective is to provide an effective assistance for coordinating process actors in order to keep their artifacts consistent.

We apply BAPE in the context of ACOVAS project to remedy the problem of unnoticed changes in order to permit concerned people to anticipate and respond to changes so that they can avoid obsolete works [45, 19, 18]. In addition to modeling and enacting supports of BAPE, we turned BAPE to a *Change-Aware Process Environment*. Our Change-Aware BAPE is reactive to change requests but proactive to change implementations with providing the following functionalities:

- Capturing in a centralized and continuous way all change requests sent asynchronously by various process actors.
- Analyzing the potential impacts when a change happens and notifying process actors affected by the change.

To achieve these objectives, we defined a change management process representing a possible policy to handle the change requests asynchronously sent by process actors. This process, executed by BAPE as a system process, comprises automated activities which implement the algorithms to analyze the impact of change and inform the affected people.

Proof-of-Concept Prototype

This thesis provides an extensive validation of the process environment BAPE. In particular, it introduces a proof-of-concept prototype for demonstrating fundamental concepts of the modeling and enactment environments of BAPE along with functionalities of change impact analysis. In addition, we apply this prototype to real-world cases within Acovas project to elaborate the benefits of our approach regarding modeling, enactment and change impact analysis as well as lessons and feedback learned from our industrial partners.

Figure 1.8 illustrates our overall solution which makes the process modeling simpler and more realistic by reflecting the partial point of view of roles. On the other hand, it makes the process enacting more flexible by synchronizing activities based on their real artifact exchanges at enactment-time and offering more autonomy to process actors to control their processes.

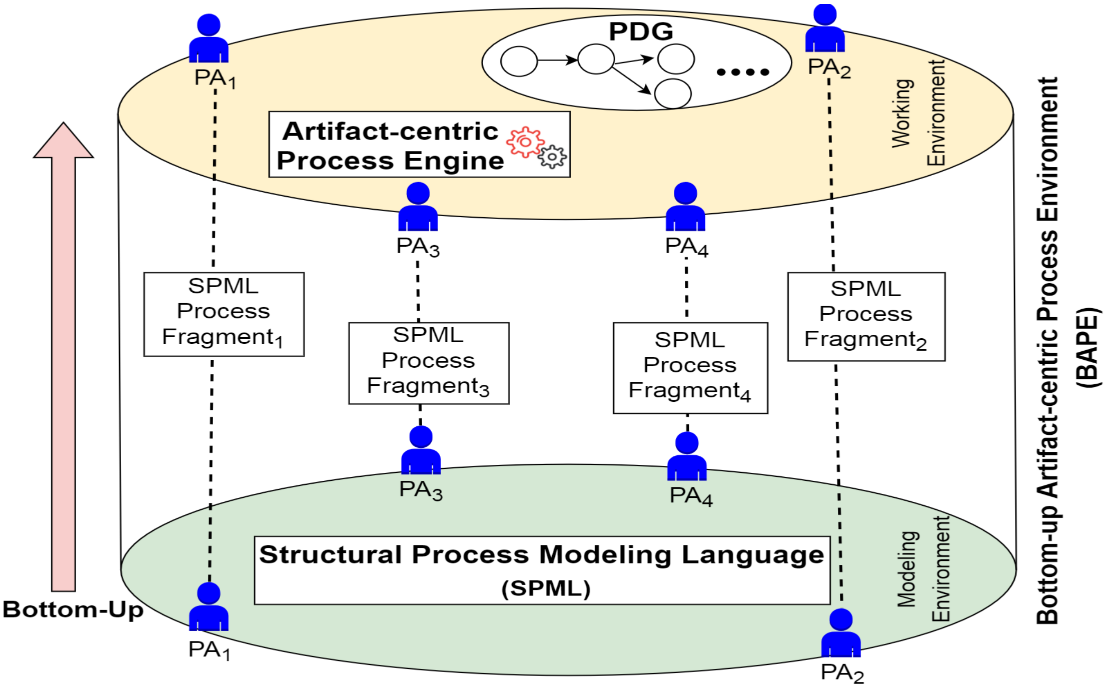


Figure 1.8: BAPE process environment

1.4 Outline

This thesis is organized in 6 chapters. As announced, this chapter provides the context, research problematic and outline our main contributions. Chapters 2, 3 and 4 present in details our proposed approaches for modeling, enacting and change analysis. Each chapter is written as an independent article comprising studying the problematic and the contributions as well as the related work on the topic. The evaluation and validation of proposed solutions are found in Chapter 5.

- *Chapter 2 - Artifact-centric Process Modeling* represents our process modeling approach which enables process actors to model themselves their operational process. Our objective is obtaining an enactable process model which is faithful to process actors' activities. To do so, a simple and user-oriented modeling language and the modeling process is introduced.

- *Chapter 3 - Data-driven Process Enactment* introduces our enactment approach which enables process actors to enact activities of their process fragments models encoded in our modeling language. We present the architecture of our proposed process environment, as well as its main components.

- *Chapter 4 - Change-aware Process Environment* presents an application of our process environment which is regarding handling the unofficial changes by allowing process actors to report emergent changes, analyze the possible impacts and inform the affected people by the changes.

- *Chapter 5 - Evaluation and Validation* illustrates first the proof-of concept implementation of the modeling, enactment and monitoring environment. Second, it provides an evaluation of our process environment on process modeling, process enactment and change management based on the experiments conducted with applying this prototype to real-world processes of our industrial partners.

- *Chapter 6 - Conclusion and Future Work* summarizes the contribution of this thesis, illustrates the benefit of the provided solution, and provides an outlook on future research directions.

Bibliography

[1] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. *Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows*, pages 291–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[2] A. Agostini and G. D. Michelis. Improving flexibility of workflow management systems. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 218–234, London, UK, UK, 2000. Springer-Verlag.

[3] Y. Alotaibi. Business process modelling challenges and solutions: a literature review. *Journal of Intelligent Manufacturing*, 27(4):701–723, Aug 2016.

[4] S. Arbaoui, J.-C. Derniame, F. Oquendo, and H. Verjus. A comparative review of process-centered software engineering environments. *Annals of Software Engineering*, 14(1-4):311–340, 2002.

[5] AristaFlow. http://www.aristaflow.com/.

[6] R. Bendraou, M. A. A. da Silva, M. Gervais, and X. Blanc. Support for deviation detections in the context of multi-viewpoint-based development processes. In *Proceedings of the CAiSE'12 Forum at the 24^{th} International Conference on Advanced Information Systems Engineering (CAiSE), Gdansk, Poland, June 28, 2012*, pages 23–31, 2012.

[7] BonitaSoftware. http://www.bonitasoft.com/.

[8] A. Borgida and T. Murata. Tolerating exceptions in workflows: A unified framework for data and processes. *SIGSOFT Softw. Eng. Notes*, 24(2):59–68, Mar. 1999.

[9] S. Brahe and K. Schmidt. The story of a working workflow management system. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*, GROUP '07, pages 249–258, New York, NY, USA, 2007. ACM.

[10] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management.* Springer Publishing Company, Incorporated, 2013.

[11] A. M. Ertugrul and O. Demirors. A method for modeling business processes in a role-based and decentralized way. In *Proceedings of the 8th International Conference on Subject-oriented Business Process Management*, S-BPM '16, pages 4:1–4:4, New York, NY, USA, 2016. ACM.

[12] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Brger. *Subject-Oriented Business Process Management.* Springer Publishing Company, Incorporated, 2014.

[13] A. Front, D. Rieu, M. Santorum, and F. Movahedian. A participative end-user method for multi-perspective business process elicitation and improvement. *Software & Systems Modeling*, pages 1–24, 2015.

[14] F. R. Golra. *A Refinement based methodology for software process modeling.* Theses, Télécom Bretagne, Université de Rennes 1, Jan. 2014.

[15] G. Grambow, R. Oberhauser, and M. Reichert. *User-Centric Abstraction of Workflow Logic Applied to Software Engineering Processes*, pages 307–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[16] V. Gruhn. Process-centered software engineering environments, a brief history and future challenges. *Ann. Softw. Eng.*, 14(1-4):363–382, Dec. 2002.

[17] M. Hajmoosaei, H. N. Tran, and C. Percebois. A user-centric process management for system and software engineering projects. In *presented at the 7th IESM Conference, October 11–13, 2017, Saarbrucken, Germany*, 2017.

[18] M. Hajmoosaei, H. N. Tran, C. Percebois, A. Front, and C. Roncancio. Impact analysis of process change at run-time. In *24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2015, Larnaca, Cyprus, June 15-17, 2015*, pages 156–161, 2015.

[19] M. Hajmoosaei, H.-N. Tran, C. Percebois, A. Front, and C. Roncancio. Towards a change-aware process environment for system and software process. In *Proceedings of the 2015 International Conference on Software and System Process*, ICSSP 2015, pages 32–41, New York, NY, USA, 2015. ACM.

[20] T. Herrmann, K. Loser, and I. Jahnke. Sociotechnical walkthrough: A means for knowledge integration. *The Learning Organization*, 14(5):450–464, 07 2007.

[21] R. Hull. *Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges*, pages 1152–1163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[22] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proceedings of the 7th International Conference on Web Services and Formal Methods*, WS-FM'10, pages 1–24, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] JBoss. jbpm, http://www.jbpm.org/.

[24] J. Klingemann. *Controlled Flexibility in Workflow Management*, pages 126–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[25] M. Kuhrmann and S. Beecham. Artifact-based software process improvement and management: A method proposal. In *Proceedings of the 2014 International Conference on Software and System Process*, ICSSP 2014, pages 119–123, New York, NY, USA, 2014. ACM.

[26] V. Künzle and M. Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance and Evolution: Research and Practice*, 23(4):205–244, June 2011.

[27] Z. Luo, A. P. Sheth, K. Kochut, and J. A. Miller. Exception handling in workflow systems. *Appl. Intell.*, 13(2):125–147, 2000.

[28] R. Matinnejad and R. Ramsin. An analytical review of process-centered software engineering environments. In *IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2012, Novi Sad, Serbia, April 11-13,2012*, pages 64–73, 2012.

[29] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven process modeling guidelines (7pmg). *Inf. Softw. Technol.*, 52(2):127–136, Feb. 2010.

[30] A. Meyer and M. Weske. *Activity-Centric and Artifact-Centric Process Model Roundtrip*, pages 167–181. Springer International Publishing, Cham, 2014.

[31] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42(3):428–445, July 2003.

[32] OMG. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0, Apr. 2008.

[33] OMG. Business Process Model and Notation (BPMN) Version 2.0.2, Dec. 2013.

[34] OMG. Case Management Model and Notation, Version 1.1, May 2016.

[35] S. Oppl. Articulation of work process models for organizational alignment and informed information system design. *Information & Management*, 53(5):591 – 608, 2016.

[36] S. Oppl and T. Rothschädl. *Separation of Concerns in Model Elicitation – Role-Based Actor-Driven Business Process Modeling*, pages 3–20. Springer International Publishing, Cham, 2014.

[37] M. Prilla and A. Nolte. *Integrating Ordinary Users into Process Management: Towards Implementing Bottom-Up, People-Centric BPM*, pages 182–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[38] J. Recker, N. Safrudin, and M. Rosemann. How novices design business processes. *Information Systems*, 37(6):557 – 573, 2012. BPM 2010.

[39] H. A. Reijers, T. Slaats, and C. Stahl. *Declarative Modeling–An Academic Dream or the Future for BPM?*, pages 307–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[40] S. W. Sadiq, M. E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, July 2005.

[41] H. Schonenberg, B. Weber, B. van Dongen, and W. van der Aalst. *Supporting Flexible Processes through Recommendations Based on History*, pages 51–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[42] B. Staudt lerner, S. Christov, L. Osterweil, R. Bendraou, U. Kannengiesser, and A. Wise. Exception Handling Patterns for Process Modeling. *IEEE Transactions on Software Engineering*, 36(2):162–183, Mar. 2010.

[43] J. Stirna, A. Persson, and K. Sandkuhl. Participative enterprise modeling: Experiences and recommendations. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, CAiSE'07, pages 546–560, Berlin, Heidelberg, 2007. Springer-Verlag.

[44] T. Stoitsev, S. Scheidl, F. Flentge, and M. Mühlhäuser. From personal task management to end-user driven business process modeling. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 84–99, Berlin, Heidelberg, 2008. Springer-Verlag.

[45] H. N. Tran, M. Hajmoosaei, C. Percebois, A. Front, and C. Roncancio. Integrating run-time changes into system and software process enactment. *Journal of Software: Evolution and Process*, 28(9):762–782, 2016.

[46] O. Turetken and O. Demirors. Plural: A decentralized business process modeling method. *Information and Management*, 48(6):235 – 247, 2011.

[47] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

[48] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[49] H. F. Witschel, B. Hu, U. V. Riss, B. Thönssen, R. Brun, A. Martin, and K. Hinkelmann. *A Collaborative Approach to Maturing Process-Related Knowledge*, pages 343–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[50] N. Zazworka, V. R. Basili, and F. Shull. Tool supported detection and judgment of nonconformance in process execution. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 312 – 323, 2009/10// 2009.

Chapter 2

Artifact-centric Process Modeling

Contents

This chapter introduces our process modeling approach which enables process actors to model themselves their operational process. Our objective is obtaining an enactable process model which is faithful to process actors' activities. First, in Section 2.1 we discuss the challenges of current top-down process modeling based on our observations from modeling the real processes of our industrial partners. Then Section 2.2 represents the fundamental characteristics of our artifact-centric process modeling which aims at overcoming the difficulties of top-down modeling by facilitating the participation of process actors to model themselves their operational process. To this aim, we define in Section 2.3 a simple process modeling language and propose in Section 2.4 a modeling process which, enable process actors doing the process modeling in order to produce their enactable process models. Section 2.5 positions and shows the strength of our modeling proposal compared to related work. Finally, Section 2.6 summarizes this chapter.

2.1 Challenges of Top-down Process Modeling

The process modeling in a top-down manner has been considered as a mostly adopted modeling approach. Notwithstanding the popularity of this approach, the procedure of process modeling can be very challenging when it comes to modeling complex processes, e.g., SSE processes. To better illustrate the difficulty of top-down process modeling, we model a process of our industrial partners in this manner along with presenting the observed challenges of such an approach.

2.1.1 Examined Process

We apply a top-down procedure in order to model the process of *Modify Testbench Wiring* given by our industrial partners in the project FUI[1] ACOVAS (Agile tools for COnception and VAlidation of Systems) which aims at proposing novel agile methods to manage complex avionic systems.

The top-down process modeling is conducted in two steps known as process elicitation and process formalization. Process elicitation itself comprises two steps as process discovery and process refinement. In process discovery, process designer gathers information from process actors who have intimate knowledge about how a process and its activities are performed through several interviews. Once the process description is obtained, in case the process description is incomplete [41], the process designer interviews process actors in order to refine the process and produce an enactable process model.

Figure 2.1 presents an extract of the outcome of the process discovery of the *Modify Testbench Wiring* process derived from the description of the process actors. This process is regarding the reconfiguring the wiring system of a testbench. In avionic systems, the global specification of an airplane is refined into system domains, systems, sub-systems and equipments. During the integration phase, the test center produces concrete test procedures to be executed on a testbench. In this context, ACOVAS partners require to reconfigure the wiring system of a testbench due to an evolution of the system under test. The real process concerns several other expertise domains than *wiring* such as *electrical, mechanical, instrumentation, simulation* and *network.* To keep the process description

[1] French academic industry joint program

less technical, we excluded the *network, mechanical* and *simulation* and focused on the rest aspects which are needed for the illustrating example.

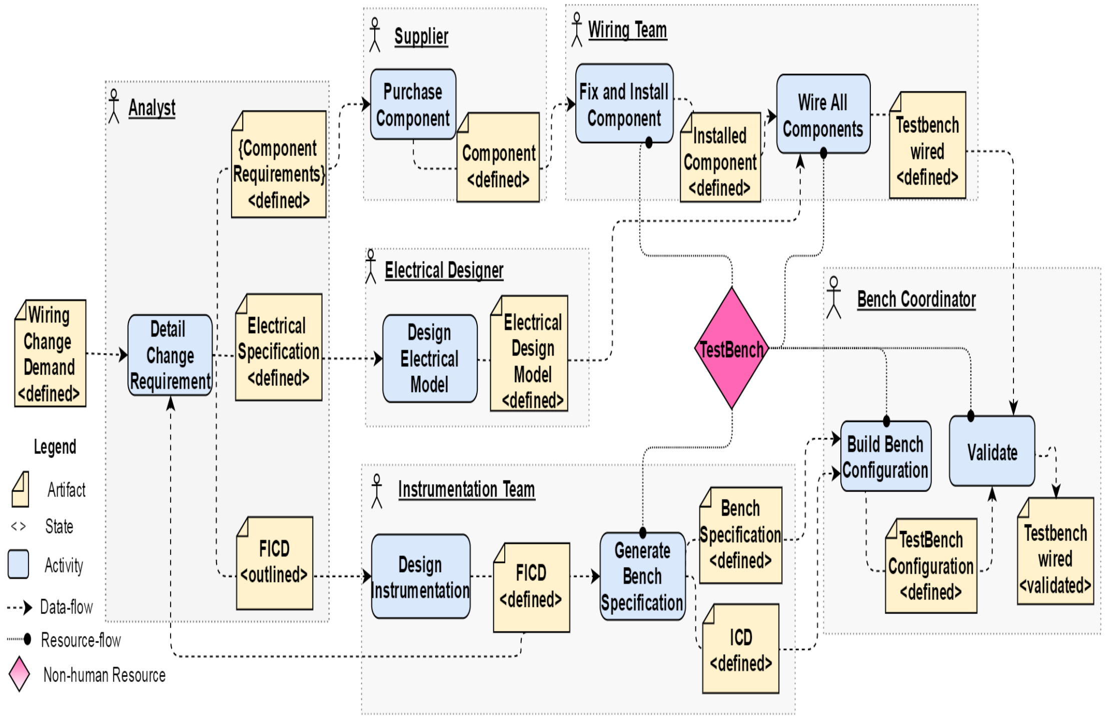

Supplier
Purchase Component
Component <defined>
Wiring Team
Fix and Install Component
Installed Component <defined>
Wire All Components
Testbench wired <defined>
Analyst
{Component Requirements} <defined>
Wiring Change Demand <defined>
Detail Change Requirement
Electrical Specification <defined>
FICD <outlined>
Electrical Designer
Design Electrical Model
Electrical Design Model <defined>
TestBench
Bench Coordinator
Build Bench Configuration
Validate
TestBench Configuration <defined>
Testbench wired <validated>
Instrumentation Team
Design Instrumentation
FICD <defined>
Generate Bench Specification
Bench Specification <defined>
ICD <defined>
Legend
Artifact
<> State
Activity
Data-flow
Resource-flow
Non-human Resource

Figure 2.1: Process of *Modify Testbench Wiring*

Description of the process *Modify Testbench Wiring*
This process has to produce a well-configured testbench, in terms of wiring, ready to run a test campaign for the evolved system. The process is performed by six roles as *Analyst, Electrical Designer, Instrumentation Team, Supplier, Wiring Team* and *Bench Coordinator* that are geographically dispersed. Each role performs his activities to produce the required artifacts in order to configure the test bench. The process starts when the *Analyst* receives the artifact *Wiring Change Demand* in state *defined* which contains the detailed description of the required change. Afterwards, he performs the activity *Detail Change Requirement* to specify the modifications required by the change on different components of the bench. This activity produces three output artifacts: a set of defined *Component Requirement* concerning the modification on the *wiring* components, a defined *Electrical Specification* for the support of *electrical* and a *Functional Interface Control Document (FICD)* in state *outlined* for the modification on *instrumentation*. The modifications on *wiring*

and *electrical* components are mandatory for any change whereas modification on *instrumentation* depends on the change requirements. In case of modification on *instrumentation*, the artifact *FICD* in state *defined* is needed by the *Analyst* in order to finalize the production of artifact *Electrical Specification*. For each required component specified in the artifact *Component Requirements*, the *Supplier* and *Wiring Team* respectively prepare and install the respective component into the testbench. The activity *Generate Bench Specification* of *Instrumentation Team* produces two artifacts in which the artifact *ICD* is produced if the change requires the modification of testbench specific interface. After installing and wiring all required components in the testbench, process finishes when the *Bench Coordinator* validates the configured testbench.

To illustrate an example of process refinement, we consider the activity *Detail Change Requirement* performed by the *Analyst*. The activity can be refined into three tasks which each task corresponds to producing a separate artifact as illustrated in the Figure 2.2. Then, the process designer requires to define the execution order of tasks by establishing work-sequence relations among them. The activity *Detail Change Requirement* can be enacted in two ways based on modification of the *instrumentation* component, i.e. on the bench's interface board. While the tasks that produce the mandatory artifacts *Component Requirements* and *Electrical Specification* are always performed in parallel, the task *Outline Instrumentation Model* producing the artifact *Functional Interface Control Document (FICD)* will be enacted only if the option *instrumentation* is chosen. Moreover, with the activation of instrumentation option, the task producing *Electrical Specification* now requires an extra artifact *FICD*.

Process modeling continues with formalization step where process designer formalizes the process with using a formal language. Figure 2.3 shows just one possible modeling of the examined process modeled with the BPMN [37] modeling language. Each fragment of the process is modeled in a separate swimlane. Each swimlane comprises the tasks of each role which are connected via work-sequence relations. The artifact exchanges among tasks are explicitly specified via message flows. Several modeling constructs as gateways and loops have been used in order to define different possible situations in the process model. As supplying and installation of components can be realized concurrently and independently, the sub-process *Component Purchase and Installation* is modeled as multi-instance task. The number of instances of the sub-process is determined at enactment-time by the number of elements in the collection artifact *Component Requirements*.

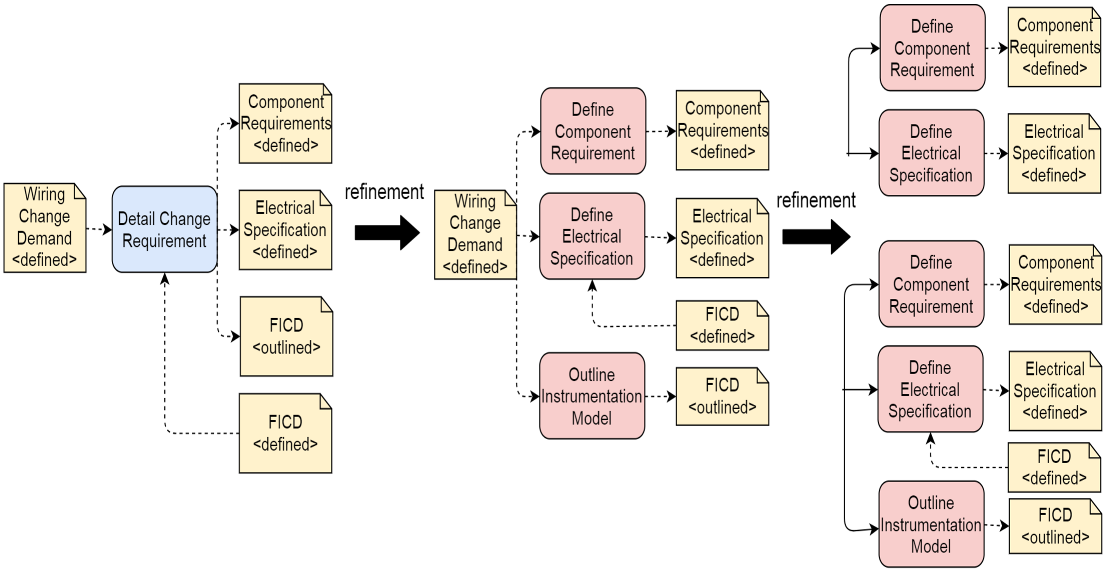

Wiring Change Demand <defined>
Detail Change Requirement
Component Requirements <defined>
Electrical Specification <defined>
FICD <outlined>
FICD <defined>
refinement
Wiring Change Demand <defined>
Define Component Requirement
Component Requirements <defined>
Define Electrical Specification
Electrical Specification <defined>
FICD <defined>
Outline Instrumentation Model
FICD <outlined>
refinement
Define Component Requirement
Component Requirements <defined>
Define Electrical Specification
Electrical Specification <defined>
Define Component Requirement
Component Requirements <defined>
Define Electrical Specification
Electrical Specification <defined>
FICD <defined>
Outline Instrumentation Model
FICD <outlined>

Figure 2.2: An example of process refinement

2.1.2 Observed Difficulties

We observed two main difficulties of top-down process modeling which are defined as follows:

- *Top-down process modeling requires to combine the knowledge of both process designer and process actors* This difficulty relates to the modeling process which is performed by two roles having different knowledge, i.e., process designer and process actor. Process designer has advance technical skills in process modeling and optimization (process knowledge) with lack of domain expertise (domain knowledge). On the contrary, process actor has domain knowledge but normally have little methodological knowledge about modeling [3, 44]. The separation of knowledge makes the communication and interaction between two roles very difficult. Moreover, SSE complex processes comprise often several expertise domains where each actor has a local view on the global process and only knows activities belonging to his domain. Therefore, process designer requires to first acquire each domain knowledge. In such processes, the process designer requires to acquire knowledge of all domains. As process knowledge is often spread out over the organization, this acquisition can take months especially for large organizations [54]. For instance, in our examined process, as a process designer, we had to spend several interviews with involved process actors to obtain the process knowledge comprising several expertise domains such as *electrical, mechanic, instrumentation, network, simulation*, etc.

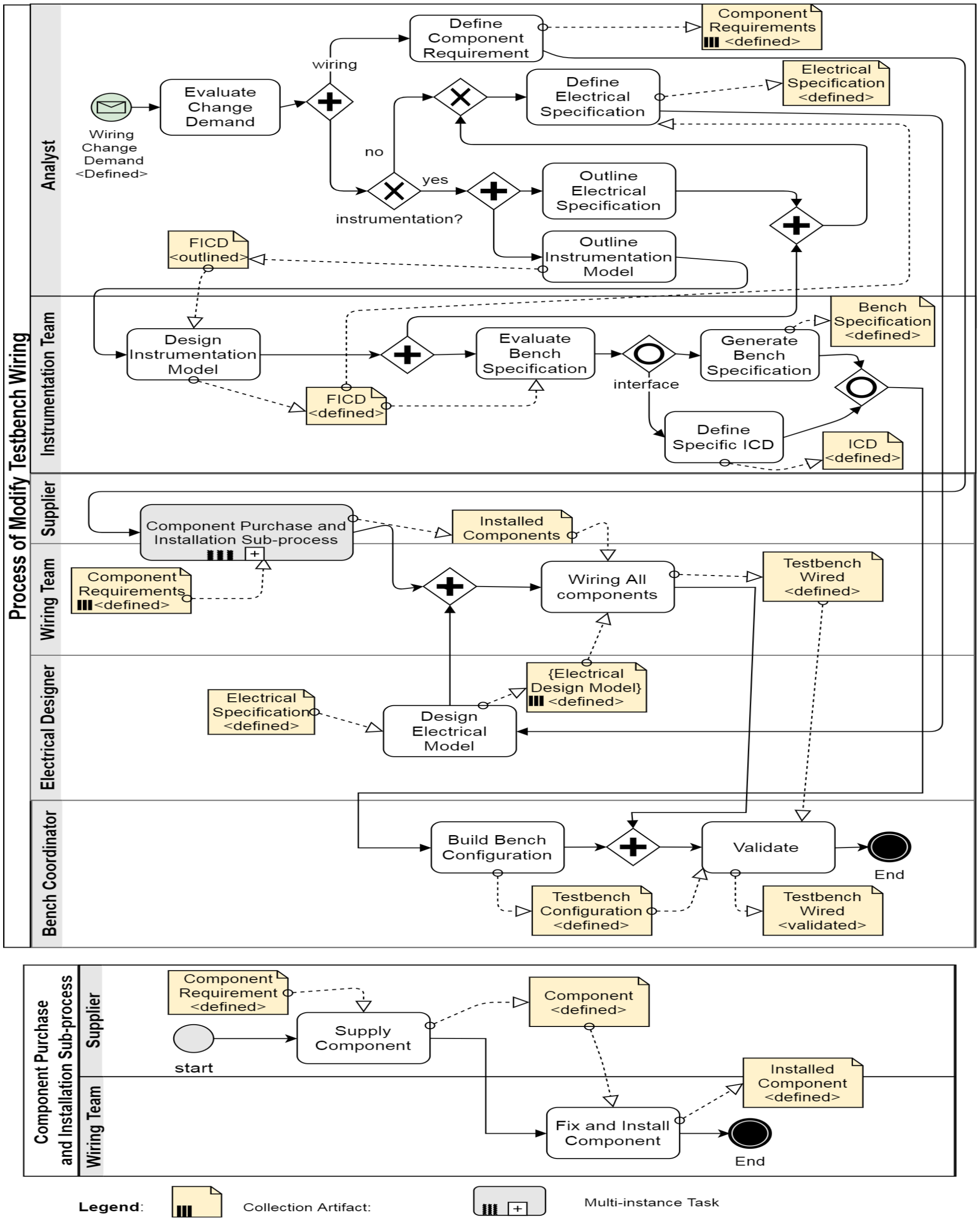

Process of Modify Testbench Wiring
Analyst
Instrumentation Team
Supplier
Wiring Team
Electrical Designer
Bench Coordinator
Wiring Change Demand <Defined>
Evaluate Change Demand
wiring
no
yes
instrumentation?
Define Component Requirement
Component Requirements <defined>
Define Electrical Specification
Electrical Specification <defined>
Outline Electrical Specification
Outline Instrumentation Model
FICD <outlined>
Design Instrumentation Model
FICD <defined>
Evaluate Bench Specification
interface
Generate Bench Specification
Bench Specification <defined>
Define Specific ICD
ICD <defined>
Component Purchase and Installation Sub-process
Installed Components
Component Requirements <defined>
Wiring All components
Testbench Wired <defined>
Electrical Specification <defined>
Design Electrical Model
{Electrical Design Model} <defined>
Build Bench Configuration
Testbench Configuration <defined>
Validate
Testbench Wired <validated>
End
Component Purchase and Installation Sub-process
Supplier
Wiring Team
Component Requirement <defined>
start
Supply Component
Component <defined>
Fix and Install Component
Installed Component <defined>
End
Legend:
Collection Artifact:
Multi-instance Task

Figure 2.3: Process of *Modify Testbench Wiring* modeled in BPMN

After acquiring domain knowledge, process designer needs to synthesize activities of all expertise domains by deducing and establishing the dependencies between them in order to build the global model. Very often, this is a very difficult, time and effort consuming task. For instance, we consider the fragments of *Analyst* and *Instrumentation Team* of our examine process as illustrated in the Figure 2.4. There are strong dependencies between two fragments in case the option *instrumentation* is activated, e.g., the *Analyst*'s task *Defined Electrical Specification* producing *Electrical Specification* now requires an extra artifact *FICD* provided by the *Instrumentation Team*'s task *Design Instrumentation Model*.

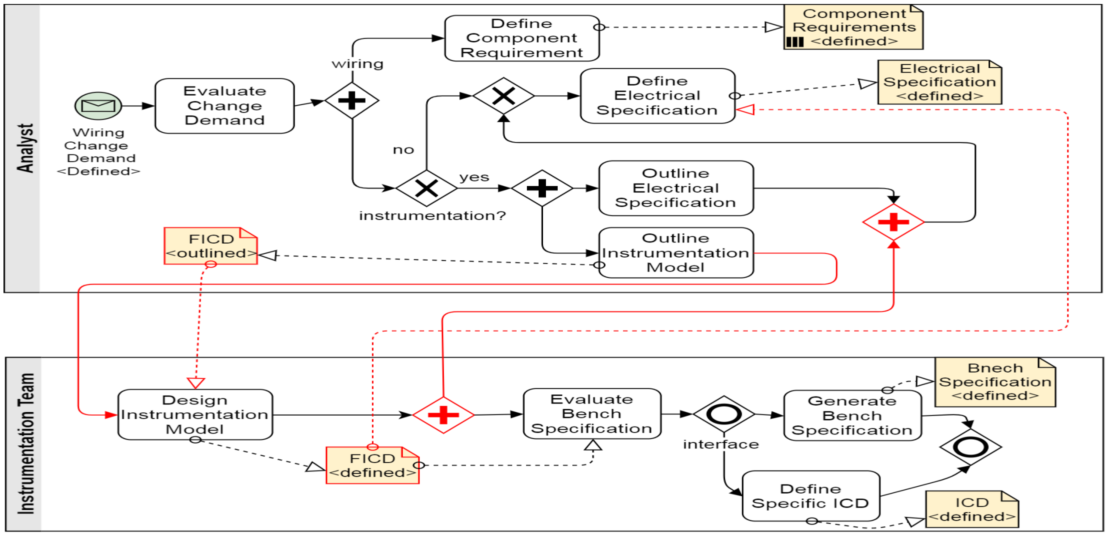

Analyst
Wiring Change Demand <Defined>
Evaluate Change Demand
wiring
Define Component Requirement
Component Requirements <defined>
Define Electrical Specification
Electrical Specification <defined>
no
yes
instrumentation?
Outline Electrical Specification
Outline Instrumentation Model
FICD <outlined>
Instrumentation Team
Design Instrumentation Model
FICD <defined>
Evaluate Bench Specification
interface
Generate Bench Specification
Bnech Specification <defined>
Define Specific ICD
ICD <defined>

Figure 2.4: Example of synthesizing local views (fragments) modeled in BPMN

- *Top-down process modeling requires to precisely describe the behavioral aspect of process at modeling-time* The behavioral aspect of process concerns defining all possible execution situations of process which occur at enactment-time. As the process has not been running yet at the modeling-time, it is not easy for process actors to describe for process designers all possible running scenarios. Because there are variety of situations that may occur at enactment-time which are not imagined at modeling-time.

  In SSE complex processes, the process modeling becomes a difficult task for process designer, particularly when there are many ways to enact the process depending on particular needs of the applying project. The causes of such difficulty can be the insufficient expressivity of the modeling language, or the complexity of using the language to correctly model the complex behavioral aspects.

To simply illustrate an example of difficulty of defining behavioral aspect of process in the model, assume the following situation that may happen in the process. Suppose that the *Analyst* finished the task *Evaluate Change Demand* the conclusion that the option *instrumentation* is no needed. This task is the main decision point which activates other branches in the process. At the moment, he performs the task *Define Electrical Specification* to specify the required modification on testbench's *electrical* component, i.e., illustrated in the Figure 2.5b. While performing his activity, the *Analyst* realizes lately that the change requires a modification on testbench's *instrumentation* component as well. Nevertheless, the pre-defined model does not define a work-sequence relation between *Define Electrical Specification* and *Evaluate Change Demand*; thus, the *Analyst* cannot come back to the preceding step to activate the option *instrumentation* and carry out the activity *Outline Instrumentation Model*.

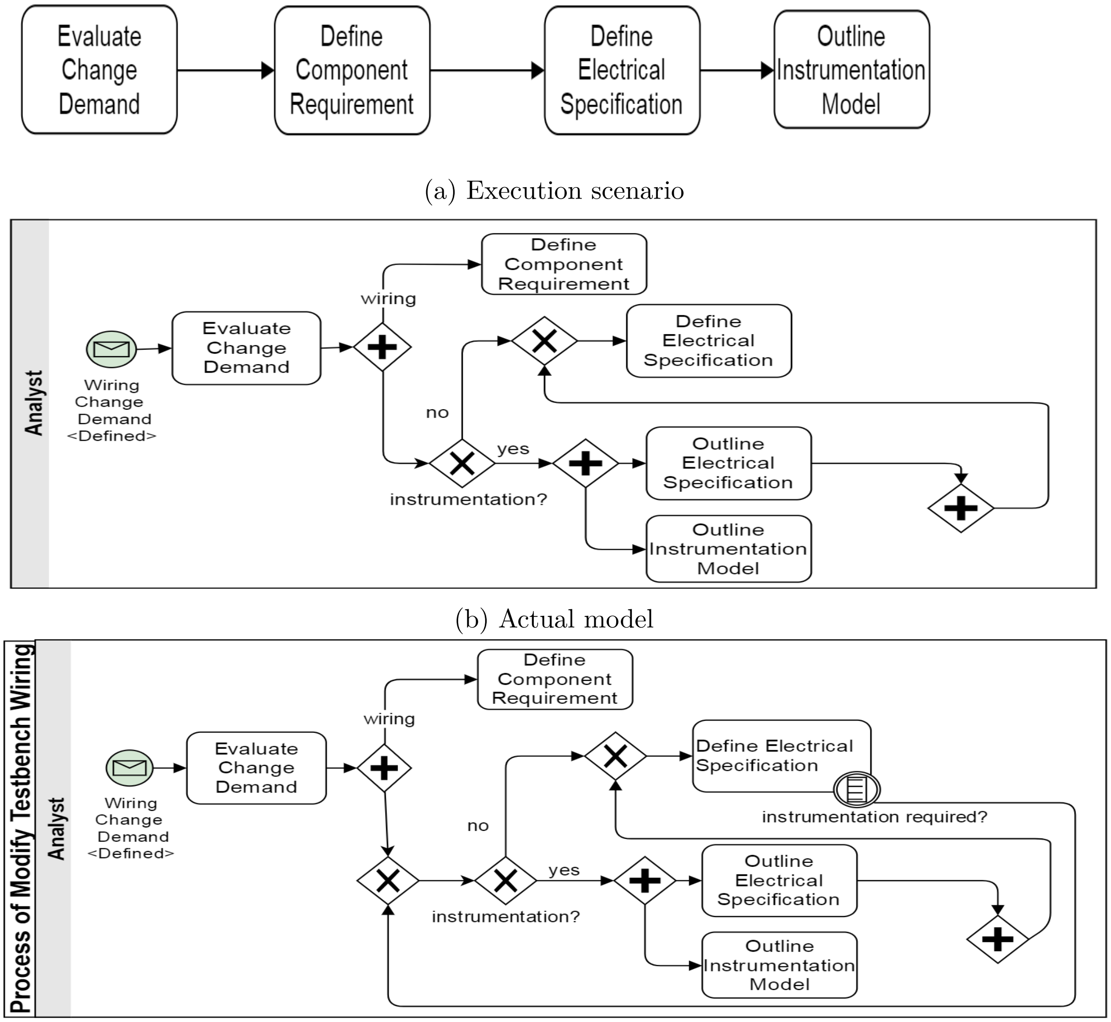
Figure 2.5c shows the desired model which support the corresponding situation. This is achieved by adding an intermediary conditional event to the task *Define Electrical Specification* and an exclusive gateway which enable *Analyst* to role back to the point where he can activate the option *instrumentation*. Notice that this model only support the situation where the realization of the option *instrumentation* is come during the execution of the task *Define Electrical Specification*.

In order to describe many situations in the model, process designer requires to use different modeling constructs such as gateways and events to express the non-linear sequences of activities' execution. As a result, the process model becomes super complex, unreadable and error-prone [32, 20, 38]. Process actors cannot easily validate the complex models as they do not have process knowledge regarding the modeling language.

In addition to the difficulty of pre-defining all situations in the model, if the expressiveness of the modeling language is weak, creating some scenarios become difficult. For instance, in case of *instrumentation* option is chosen, the task *Define Electrical specification* can not be completed as it requires the artifact *FICD* in state *defined*. This scenario cannot simply be defined in BPMN by associating the artifact as the post-condition of the task or associating intermediary events to the task. Therefore, as illustrated in Figure 2.3, we needed to split the task *Define Electrical Specification* into two tasks *Outline Electrical Specification* and *Define Electrical Specification* and define new gateways to describe such situation which can make the model complex and error-prone.

2.1.3 Discussion

Top-down process modeling requires two roles having different knowledge to collaborate together in order to produce an enactable process model. This combination of knowledge makes the procedure of top-down modeling complicated. We can resume three causes of


Evaluate Change Demand
Define Component Requirement
Define Electrical Specification
Outline Instrumentation Model
Analyst
Wiring Change Demand <Defined>
wiring
no
yes
instrumentation?
Outline Electrical Specification
Process of Modify Testbench Wiring
instrumentation required?

(a) Execution scenario

(b) Actual model

(c) Supporting model

Figure 2.5: Execution scenario and the model supporting respective scenario

such difficulty defined as follows:

- *Process modeling knowledge and domain knowledge are separate* process designer and process actors have totally different knowledge.
- *Domain knowledge is diverse* Process comprises several domain expertise which each one requires specific knowledge and skills.
- *Domain knowledge is partial* Each actor has a partial view of the process.

These causes make the acquisition and synthesis of domain knowledge for process designer a very hard, effort and time consuming task. To remedy such difficulty, we investigate the idea of making process actors to realize process modeling. Many researchers [40, 15, 54, 30, 23, 27] advocate the idea of process actor participation in process modeling procedure. They stated that the degree of actors involvement in process modeling directly affects both representation accuracy of actual work and total effort required for process modeling. In our opinion, as we observed in real modeling the process of our industrial partners as well, enabling full participation of process actors in modeling their processes can make the process models to be more faithful to its reality.

Process actors participation in process modeling has great advantages but solely it can not resolve the complexity of top-down process modeling. There is another important factor that must be considered when making the process actors as the first-citizen in process modeling. Process actors are the ones who know exactly their process. However, by enabling process actors to describe the behavioral aspect of their process in the model, we may face the same difficulties of top-down modeling. Mixing the structural and behavioral aspects of process in the model may result in:

- *Inaccurate model compared to the real process* The formal global model resulted from such modeling procedure might not suit process actors needs as it may not correspond to the real situations occurring at enactment-time. There exist some situations or exceptions that have not been predicted in the model and makes the process actors to disobey the model in which causes their activities go beyond the control of the process environment.

- *Rigid model* Process model imposes the enactment procedure where process actors should perform their activities step by step as defined in the model. By defining the exact behavior of process in the model, process model becomes rigid.

- *Complex model* Specification of behavioral aspect of process by using complex modeling constructs make the process model very complex, unreadable, error-prone and validation of model by process actors become a cumbersome task.

To remedy such difficulty, we investigate the idea of avoiding the specification of behevioral aspect of process in the model. In fact, flows between tasks are really imposed by exchanged data during the enacting-time. Therefore, defining only structural aspect of their process by emphasizing on the exchanged artifacts between their activities can lead to deduce the behavioral aspect of their process at enactment-time. Notice that the data-flow among tasks may define the implicit temporal dependencies among them but does not impose control constraints on process actors in executing their tasks. The process actors have more autonomy to enact their tasks based on real situations.

2.2 Artifact-centric Process Modeling Approach

Being aware of the overall difficulties of top-down process modeling we claim that by (1) separating the performers of process modeling and process enacting and (2) describing the behavioral aspect of the process in the model, a process model becomes often inadequately specified and therefore not faithful to its real enactment. In other words, the gap between process model and enactment occurs.

By using these arguments and our observations, we are interested in a process modeling where the main emphasis is on process actors who have sufficient knowledge about the actual work to model their operational working process [21]. Our main research challenge is how to propose a modeling solution which perfectly satisfies process actor's needs by enabling them to produce their enactable process models. The modeling objective is to give more liberty to process actors in order to react to situations in real execution. Thus, we expect the modeling which better mimics the real enactment of the process and thus can better deal with wrong or overcomplicated enactment.

To propose a satisfactory solution, we identified essential characteristics of modeling support of such an approach. These characteristics are identified based on our observations from modeling of our examined process which are reinforced in the literature as well [17, 54, 30, 31, 27]. In the next we describe each characteristic in details.

- *Fragmented process modeling* This characterisitic corresponds to the way that process modeling is conducted. The examined process, similarly to most of SSE processes, is complex and includes several roles which mostly relate to different domain expertises and can be geophysically located in different sites. In other words, each role only knows what are under his responsibility and only sees the part of process that he performs and is mostly unaware of other roles' activities. This issue makes each role to have a partial view of the global process [40, 54]. One advantage of such a fragmentation is that the process is less complex to be modeled, and more flexible to be instantiated or modified.

- *Declarative process modeling* This characteristic relates to avoid defining the behavioral aspect of the process in the model, i.e., by defining work-sequence relations among tasks. In fact, it is difficult for process actors to define the behavior of their working process in terms of exactly knowing on which order their activities will be executed. The reason behind such difficulty is the temporal difference between process modeling and process enactment. Fully anticipating the behavioral aspect of process at modeling-time which occurs at enactment-time is difficult and can lead to the potential gap between model and real enactment. In our approach, the focus is on structural aspect of the process so that process actors have the liberty to enact their tasks at enactment-time, including corrective steps for a bad implementation of the process.

- *Artifact-centric process modeling* This characteristic considers artifact as the first-citizen in process modeling. As we focus on structural aspect of process, specifying the artifact exchanges among activities in the model can provide information to deduce the behavioral aspect of process at enactment-time. Thus, the modeling language must specify the data-flow regarding the artifacts entering and leaving the activities.

- *Direct enactability* As our objective is to enable process actors to directly produce their enactable process models. In fact, the modeling language should be enough simple and expressive to enable the precise definition of its execution semantics [17, 31]. Moreover, simplicity of the language can result in an easier deployment where the effort required to deploy the process and bring it to the concrete project becomes minimum.

Additionally, we suggest a non-functional requirement and a hypothesis about the application of process modeling approach which are respectively defined as follows:

- *Language simplicity* An empirical evidence shows that the number of elements actually used during modeling is limited and highly dependent on the modeling objective [35]. When involving process actors, it seems to be appropriate to limit the number of available modeling elements. This issue has been the focus of the majority of the user-centric modeling approaches to guide inexperienced modelers through the process of creating a model without overwhelming them with syntactic formalism and complex modeling constructs [27, 9, 18, 11, 40]. Notice that the proposed language must be enough expressive and includes essential process elements to allow process actors define different situations in their process models. The structural definition of process models which avoids the needs of work-sequence relations results in reducing the number of modeling constructs and making modeling easier for process actors.

- *Alignment of concepts* As process modeling is conducted in a fragmented way where each fragment is modeled separately and can belong to different expertise domains, different notions can be used by different process actors to refer to the same concept. For instance, a same artifact can be known with two different names in two different fragments. Therefore, a common vocabulary used by all process actors to describe their process fragments elements is a prerequisite for reaching an alignment on the content level [48, 6].

In the next section, we present how our modeling approach supports the properties mentioned above. To do so, first we introduce the concepts of our modeling language. Then the main steps of process modeling are discussed to show how our language provides simplicity to process actors in order to directly create their enactable process fragment models.

2.3 Structural Process Modeling Language (SPML)

This section introduces our process modeling language, named SPML (Structural Process Modeling Language). SPML is developed as a DSL (Domain Specific Language) [16] which is dedicated to process actors to model their enactable process fragment models. SPML is a user-friendly modeling language aiming at suiting process actors needs which is simple but expressive enough to represent their operational process. The key features of SPML are as following:

1. *Removing the work-sequence relations among activities from the process model* In fact, SPML allows process actors to only describe the structural aspect of their process.

2. *Describing in detail the data-flow regarding the artifacts entering and leaving the activities* The respective data-flow provides sufficient information to deduce the behavioral aspect of process at enactment-time.

SPML includes only the process elements known by process actors in their daily work and avoids introducing concepts requiring process-special knowledge to understand.

A complete definition of a modeling language consists of the description of its syntax along with well-formedness rules, and its execution semantics [22]. In the next, we present the SPML abstract syntax and its well-formedness rules where the execution semantics are discussed in detail in the next chapter.

2.3.1 SPML Meta-model

The SPML abstract syntax is defined by a meta-model comprising two packages which each package deals with a specific aspect as illustrated in Figure 2.6. The package *CompanyAssets* defines the concepts representing the common information shared among process actors in the company. Company assets concern information that globally exists in the company including definitions of processes, primary roles participating in processes, definitions of artifacts together with their associated states, resource definitions and list of options. Company assets provide thus a common vocabulary about process elements distributed among process fragments. In practice, such company assets always exist even if the process is not formally defined, e.g., documentations, guidelines, etc.

The package *ProcessStructure* defines the concepts allowing to describe the activities and tasks inside each process fragments belonging to a role. The core concepts of SPML along with relations among them are illustrated in Figure 2.7.

The basic concepts of SPML along with their essential properties are defined as follows:

- *Process* defines a collection of activities that will be performed by different roles to achieve the process goal. An activity can use some resources during its execution and manipulate artifacts representing process's work products. A process is organised into several *Process Fragments*.

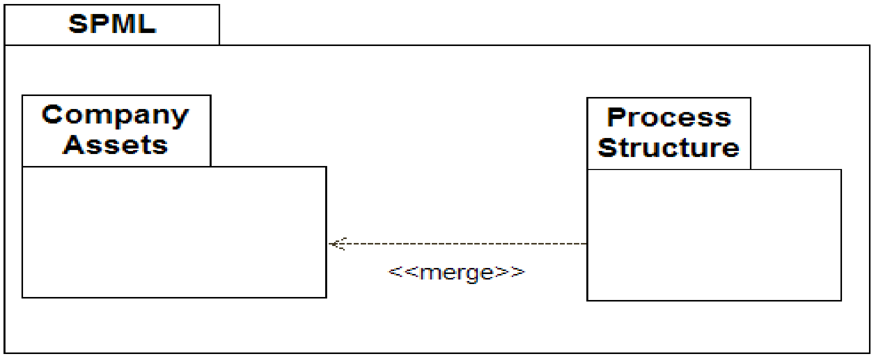


Figure 2.6: Structure of SPML

- *Process Fragment* defines a partition within a *Process* that encapsulates the activities belonging to the *Role* who performs the *Process Fragment*.

- *Role* represents the rights and the responsibilities to perform certain process's activities. Process actors play one or many roles when enacting a process.

- *Artifact* is a tangible work product consumed, produced, or modified by a *Process*. They may serve as a basis for defining reusable assets. The property *artifactKind* indicates that whether an *Artifact* is a *single* or a *collection*. The latter defines if the *Artifact* represents a collection of elements and does not assume any dependencies among the elements.

- *Activity* represents a unit of work within a *Process* that contributes to achieving process's goals. It is performed by a specific *Role* and is decomposed to a set of enactable *Task*s corresponding to activity's sub-objectives.

- *Task* represents an enactable and manageable action performed by one process actor taken to achieve one specific *Activity*'s sub-objective. A task can be *mandatory* or *optional*. The former indicates the *Task*s which are necessary for an *Activity* to be completed. The latter indicates the *Task*s which can be enacted at enactment-time based on process actors needs. The property *duration* indicates the temporal estimation of the performing *Task*.

- *Resource* is a non-human entity that provides capability for the task to be carried out, i.e., a resource that does not correspond to an actual person, e.g., machine environment, tools and equipment.

- *Option* allows process actors to specify alternative ways to realize an activity in different execution contexts. An option represents a specific execution context and is used to specify the condition to perform an *Optional Task* or to use an *Artifact*.

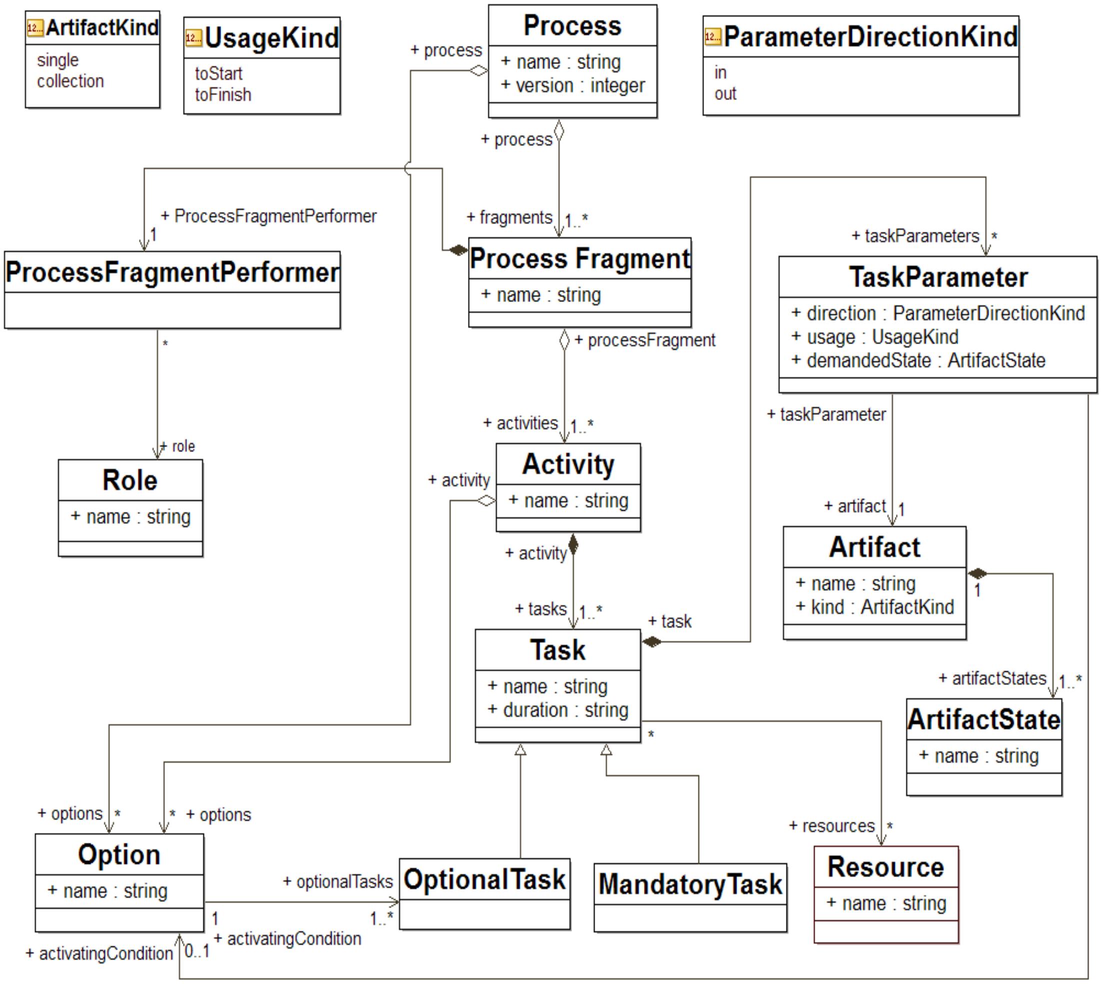


Figure 2.7: SPML meta-model

- *ProcessFragmentPerformer* represents the performing relation between a *Role* and the activities inside a *Process Fragment*.

- *TaskParameter* defines the relation between a *Task* and an *Artifact*. The property *direction* of the relation *TaskParameter* indicates if the concerned *Artifact* is necessary for a *Task* to be consumed (input) or produced (output). The property *usage* of the relation *TaskParameter* indicates if the concerned *Artifact* is necessary for a *Task*'s pre-condition (*toStart*) or for a *Task*'s post-condition (*toFinish*). The property *demandedState* defines a condition of existence of an *Artifact* at a particular moment in its life-cycle. An *Artifact* evolves during its life-cycle by passing through states. The property *demandedState* has a type *State* as defined in the package *CompanyAssets*. Finally, the *Option* is also included as a property of relation *TaskParameter* to define possible condition of using an *Artifact* in a *Task*.

Figures 2.8 and 2.9 illustrate the concrete syntax of SPML regarding two fragments of the process in Figure 2.1 which other fragments of the process are provided in the annex of this chapter. The fragment in Figure 2.8 is modeled by a process actor playing the role *Analyst* and the fragment in 2.9 is modeled by a member of *Instrumentation Team*. We assume that the information regarding the company assets (e.g., process definitions, role definitions, artifact definitions, etc.) is defined by the project manager in advance. The process contains two options as *instrumentation* and *interface* which are defined globally in the scope of the process. Each fragment however can use only some of these defined options. The process fragment of the role *Analyst* includes an activity *Detail Change Requirement* composed of two *mandatory tasks* and an *optional task*. The mandatory task *Define Component Requirements* requires an artifact *Wiring Change Demand* and produces a collection of artifacts *Component Requirements* as a result. The mandatory task *Define Electrical Specification* requires two input artifacts *Wiring Change Demand* and *FICD*, each one in state *defined*. The latter artifact, produced by the *Instrumentation Team*, is required only if the *instrumentation* option is activated. The optional task *Outline Instrumentation Model* belongs to the option *instrumentation*. The process fragment of the role *Instrumentation Team* includes two activities *Design Instrumentation* and *Generate Bench Specification*. The former is comprised of a mandatory task and the latter is comprised of two mandatory tasks and an optional task which belongs to option *instrumentation*.

Notice that some properties of the SPML concepts described in the abstract syntax are not represented in the process fragments models. These properties are assumed to have a default value. For instance, an artifact by default is considered as a single artifact in which the value of property *kind* is set to *single*. Moreover, in concert syntax, we define the list of options in the scope of an activity. Each of defined options is realized by an optional task of the activity.

2.3.2 SPML Well-formedness Rules

The proposed meta-model properly captures all the concepts that are necessary to create a model in SPML, as well as all the valid relationships that can exist between these concepts in a model. However, we require some additional constraints that restrict the way concepts can be assembled to form correct and consistent SPML process models. These constraints are defined in form of a set of OCL [2] well-formedness rules.

- $taskOutputArtifact$ A task can consume many artifacts as input but can produce only zero or one artifact as output. The reason behind such a constraint is threefold: (1) better reflecting the actors vision on their process as they know their manipulating artifacts, (2) providing finer change impact analysis which this fact

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement options instrumentation
            MandatoryTask Define Component Requirements
                Artifact Wiring Change Demand(in, defined)
                Artifact Component Requirements(out, defined, collection)
            MandatoryTask Define Electrical Specification
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(in, defined, toFinish) ifOption instrumentation
                Artifact Electrical Specification(out, defined)
            OptionalTask Outline Instrumentation Model ifOption instrumentation
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(out, outlined)
```

Figure 2.8: Process Fragment of *Analyst*

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Instrumentation Team
        Activity Design Instrumentation
            MandatoryTask Define Instrumentation Model
                Artifact FICD(in, outlined)
                Artifact FICD(out, defined)
        Activity Generate Bench Specification options interface
            MandatoryTask Define Bench Specification
                Artifact FICD(in, defined)
                Artifact Bench Specification(out, defined)
            OptionalTask Define Specific ICD ifOption interface
                Artifact FICD(in, defined)
                Artifact ICD(out, defined)
```

Figure 2.9: Process Fragment of *Instrumentation Team*

will be discussed in details in Chapter 4 and (3) constructing the lifecycle of each artifact in order to validate the model before enactment. The respective constraint is expressed in OCL as follows:

```
context Task inv taskOutputArtifact:
self.taskParameters->select(tp|tp.direction
=ParameterDirectionKind::out)->size()<=1
```

The Figure 2.10 represents an invalid model corresponding to this constraint.

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement
            MandatoryTask Define Component and Electrical Specification
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(in, defined, toFinish) ifOption instrumentation
                Artifact Component Requirement(out, defined)
                Artifact Electrical Specification(out, defined)
```

Figure 2.10: Invalid Model of constraint *taskOutputArtifact*

- *taskOutputArtifactUsage* As an output artifact only represents the post-condition of a task, thus its property *UsageKind* can be only set to *toFinish*. The respective constraint is expressed in OCL as follows:

```
context Task inv taskOutputArtifactUsage:
self.taskParameters->select(tp|tp.direction
=ParameterDirectionKind::out and tp.usage=UsageKind::toStart)
->isEmpty()
```

The Figure 2.11 represents an invalid model corresponding to this constraint where output artifact *Electrical Design Model* is set for the task's pre-condition.

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Electrical Designer
        Activity Design Electrical Model
            MandatoryTask Define Electrical Model
                Artifact Electrical Specification(in, defined, toStart)
                Artifact Electrical Design Model(out, defined, toStart)
```

Figure 2.11: Invalid Model of constraint *taskOutputArtifactUsage*

- *artifactStateNotSame* A task requires artifacts in certain states and produces an artifact in a certain state as well. In case of a task manipulating a same artifact as input and output, the state of the artifact must be different in its roles input and output. The respective constraint is expressed in OCL as follows:

```
context Task inv artifactStateNotSame:
let tps: Set(TaskParameter) = self.taskParameters->select(tp|tp
.direction=ParameterDirectionKind::in) in
if(tps->notEmpty()) then self.taskParameters -> select(tp|tp
.direction=ParameterDirection Kind::out)->forAll(tp|(tp.artifact
.name<>tps->first.artifact.name).or(tp.demandedState<>tps->first
.demandedState))
else true
endif
```

The Figure 2.12 illustrates the part of the process fragment model of *Instrumentation Team* representing an invalid model corresponding to this rule.

Process Modify Testbench Wiring **options** instrumentation, interface
  ProcessFragment Instrumentation Team
    Activity Design Instrumentation
      MandatoryTask Define Instrumentation Model
        Artifact FICD(**in**, outlined)
        Artifact FICD(**out**, outlined)

Figure 2.12: Invalid Model of constraint *artifactStateNotSame*

- *artifactStateMustExist* The demanded state associated to the relation between a task and an artifact must be an element of artifact's states as defined in the company assets. The respective constraint is expressed in OCL as follows:

```
context Task inv artifactStateMustExist:
self.taskParameters->forAll(tp| tp.artifact.artifactStates
->exists(tp.demandedState=tp.artifact.artifactStates.name))
```

  The Figure 2.13 illustrates the part of the process fragment model of *Instrumentation Team* where the artifact *FICD* has only two states as *outlined* and *defined*.

- *activityOptions* The list of options used by an activity must be a subset of options defined in the scope of the containing process. The respective constraint is expressed in OCL as follows:

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Instrumentation Team
        Activity Design Instrumentation
            MandatoryTask Define Instrumentation Model
                Artifact FICD(in, validated)
```

Figure 2.13: Invalid Model of constraint *artifactStateMustExist*

```
context Activity inv activityOptions:
self.processFragment.process.options->includesAll(self.options)
```

The Figure 2.14 illustrates the part of the process fragment model of *Analyst* where activity *Detail Change Requirement* includes an option which is not defined in the scope of process.

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement options mechanic
```

Figure 2.14: Invalid Model of constraint *activityOptions*

- *activityOptionalTask* For each option of the activity, there must be at least one optional task which corresponds to the respective option. The respective constraint is expressed in OCL as follows:

  ```
  context Activity inv activityOptionalTask:
  self.options->includesAll(self.tasks.activatingCondition)
  ```

  The Figure 2.15 illustrates an invalid model corresponding to this rule where the optional task *Outline Instrumentation Model* corresponds to an option which is not defined in the scope of activity *Detail Change Requirement.*

- *taskOption* If an activity of the process has an optional task, the respective activity must include an option corresponding to the respective optional task. The respective constraint is expressed in OCL as follows:

  ```
  context OptionalTask inv taskOption:
  self.activity.options->includesAll(self.activatingCondition)
  ```

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement options instrumentation
            OptionalTask Outline Instrumentation Model ifOption interface
```

Figure 2.15: Invalid Model of constraint *activityOptionalTask*

The Figure 2.16 illustrates the part of the process fragment model of *Analyst* representing an invalid model corresponding to this rule.

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement
            OptionalTask Outline Instrumentation Model ifOption instrumentation
```

Figure 2.16: Invalid Model of constraint *taskOption*

- *artifactOption* The option associated to a relation *Taskparameter* between an input artifact and a task has to be in the list of process's options. The respective constraint is expressed in OCL as follows:

```
context TaskParameter inv artifactOption:
self.task.activity.processFragment.process.options->includesAll
(self.option)
```

The Figure 2.17 illustrates an invalid model where artifact *FICD* is associated with an option which is not defined in the scope of process.

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement
            MandatoryTask Define Electrical Specification
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(in, defined, toFinish) ifOption mechanic
                Artifact Electrical Specification(out, defined)
```

Figure 2.17: Invalid Model of constraint *artifactOption*

In order to validate the presented OCL rules, we first defined the grammar of SPML concrete syntax via Xtext [5, 59] and obtained a textual editor for defining SPML models. Then we defined our OCL rules in the ecore file generated from the grammar and then validated the respective rules in the textual editor.

2.3.3 SPML Expressiveness

This section discusses the capacity of SPML in supporting the essential control flows when enacting process's tasks, i.e. the sequencing, branching, parallelism, iteration and also multi-instances execution.

Imperative process modeling languages describe explicitly the control-flows among process's tasks, thus, as described in [49, 1], they have to seek a way to correctly express various complex branching, merging patterns of tasks at modeling time and a way to correctly implement these patterns at enactment-time. In contrast to these languages, SPML does not model the control-flows among process's tasks, thus we do not need to support the complex control-flow patterns described in [49, 1]. However, we want to show that SPML can express the essential flows. This feature is enough to coordinate process's tasks at enactment-time thanks to an universal execution mechanism based on the consumption and production of artifacts to synchronize tasks. We aim at illustrating two points in this section:

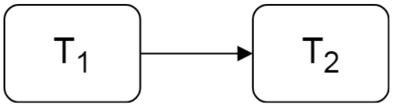
– Which main control-flow patterns can be implicitly supported in SPML.

– How simple is for process actors to define respective patterns compared to imperative languages, e.g., BPMN [37], SPEM [36], Petri Net [57], Event-Driven Process Chains (EPCs) [50], Subject-oriented Business Process Management (S-BPM) [15], etc.

In the following, we mention the SPML support for the main patterns.

- ***Sequence*** This pattern specifies a situation where a task in a process is enabled after the completion of a preceding task in the same process.
  In imperative languages, this pattern is supported by defining a work-sequence relation between two tasks as illustrated in the Figure 2.18a.
  SPML supports this pattern by specifying an artifact exchange between two tasks as illustrated in the Figure 2.18b through exchanging artifact art_1 in state s_1.

- ***Parallelism*** This pattern presents a situation where a set of tasks can be executed in parallel indicating that there is no dependencies (e.g., temporal, artifact exchange) among them.
  In imperative languages, this pattern is supported by using a parallel gateway as illustrated in the Figure 2.19a.
  In SPML, tasks can be concurrently executed as fas as there is no artifact exchange among them, i.e., illustrated in Figure 2.19b.

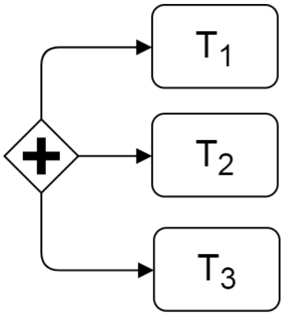


(a) Imperative language

MandatoryTask T_1
    Artifact $Art_1(\textbf{out}, s_1)$
MandatoryTask T_2
    Artifact $Art_1(\textbf{in}, s_1)$

(b) SPML

Figure 2.18: Sequence Pattern in SPML and imperative language



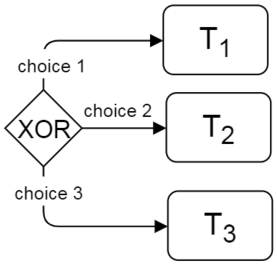
MandatoryTask T_1
MandatoryTask T_2
MandatoryTask T_3

(b) SPML

(a) Imperative language

Figure 2.19: Parallelism Pattern in SPML and imperative language

- ***Choice*** This pattern presents a situation where one or more tasks are selected for subsequent execution from a set of available tasks. Generally, a process model is used in different project contexts which result in different variants of the process. In particular, respective variants have certain activities, while at the same time differences due to their usage in different project contexts exist, e.g., certain activities are optional and relevant for only some of the process variants.
  Imperative languages express these variants with using choice construct (decision points) as illustrated in Figure 2.20a .
  In SPML, the concept of *Option* enables process actors to implicitly define different alternatives of their work process without overwhelming them with complex modeling constructs. Each option is associated with an optional task as illustrated in Figure 2.20b.
  The advantage of SPML over imperative languages is noticeable when considering the positioning of decision points. In imperative languages, decision points regarding choosing which path to be executed are fixed in the model. This fact makes the model super complex when it comes to complicated scenarios. For instance, in the example of Figure 2.20a, assume a scenario where only task T_1 is activated. However, the actor performing T_1 realizes that the tasks T_2 and T_3 must be selected as well. To support such situations, the choice constructs must



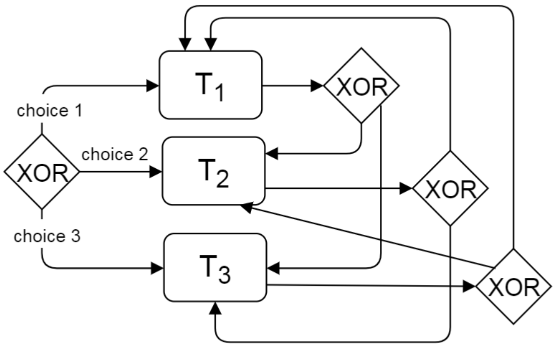
(a) Imperative language

Activity A_1 **options** choice 1, choice 2, choice 3
  OptionalTask T_1 **ifOption** choice 1
  OptionalTask T_2 **ifOption** choice 2
  OptionalTask T_3 **ifOption** choice 3

(b) SPML

Figure 2.20: Choice Pattern in SPML and imperative language

be defined after each task to verify whether other tasks required to be selected or not. By assuming defining the decision points in the whole process, this results in a super complex process as illustrated in Figure 2.21a.
In SPML the option is not fixed and can be activated at any moment during activity execution. This makes many situations to be covered by a simple and single model, i.e., Figure 2.21b.



(a) Imperative language

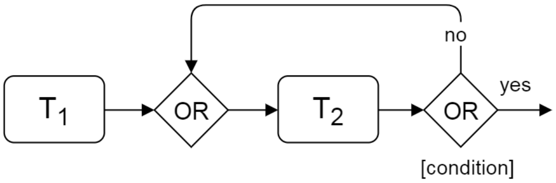
Activity A_1 **options** choice 1, choice 2, choice 3
  OptionalTask T_1 **ifOption** choice 1
  OptionalTask T_2 **ifOption** choice 2
  OptionalTask T_3 **ifOption** choice 3

(b) SPML

Figure 2.21: Complex scenario of choice pattern in SPML and imperative language

- ***Iteration*** This pattern describes a situation where one or more tasks can be

executed repeatedly. Note that iteration can be seen as a particular type of choice, where the join precedes the split.
In imperative language, this pattern is modeled using an exclusive gateway which either is controlled by a counter or is validated by a condition as illustrated in the Figure 2.22a.
In SPML, we do not provide any support to model such pattern as shown in the Figure 2.22b. We do not have condition for the loop and we bring this pattern under control of process actors. Suppression of work-sequence relations among activities and tasks facilitates the execution of such pattern as process actors can instantiate their tasks repeatedly if required.



(a) Imperative language

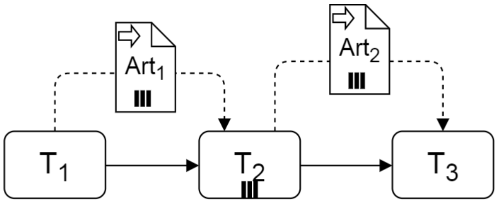
MandatoryTask T_1
MandatoryTask T_2

(b) SPML

Figure 2.22: Iteration pattern in SPML and imperative language

- ***Multiple Instances*** This pattern presents a situation where multiple concurrent instances of a task can be executed. The number of task instances depends on run-time factors, often the input data of a task.
  In imperative languages, this pattern can be supported as illustrated in Figure 2.23a where the output collection artifact Art_1 of task T_1 is the input of multi instance task T_2. In such case, when T_1 is completed the cardinality n of its output collection artifact Art_2 determines the number of task instances of the subsequent task T_2.
  In SPML, we do not provide any special construct for this pattern as illustrated in the Figure 2.23b. In our approach, we allow process actors to instantiate their tasks multiple times if needed, e.g., the task T_2 can be instantiated n times in which each instance takes one element of Art_1 in state S_1 as input.

To resume, our modeling language SPML is enough expressive to describe typical situations of process execution. Most importantly, to model such situations, SPML does not impose process actors with complex modeling constructs. SPML gives the control to process actors in order to handle situations described by patterns such as Multi Instance, Iteration, Cancellation and Termination. Suppression of work-sequence relation in the model along with including the concept of *Option* give more autonomy to process actors to decide different execution scenarios at runtime.



(a) Imperative language

MandatoryTask T_1
  Artifact $Art_1(\textbf{out}, s_1, \textbf{collection})$
MandatoryTask T_2
  Artifact $Art_1(\textbf{in}, s_1)$
  Artifact $Art_2(\textbf{out}, s_1)$
MandatoryTask T_3
  Artifact $Art_2(\textbf{in}, s_1, \textbf{collection})$

(b) SPML

Figure 2.23: Multiple instance Pattern in SPML and imperative language

2.4 Modeling Process

This section introduces our proposed modeling process in order to exhibit the characteristics of our artifact-centric process modeling approach, i.e., fragmented modeling, avoidance of work-sequence relations, direct enactability, etc. The proposed modeling process is illustrated in the Figure 2.24 by a map [13, 47]. A map is presented as a diagram where nodes are *intentions* and edges are *strategies*. A map includes two predefined intentions as *Start* and *Stop*, which mean accordingly the beginning and the end of the process. An important notion in process maps are the sections represented as a triplet <source intention, target intention, strategy>, in other terms, the knowledge corresponding to a particular process step to achieve an intention (the target intention) from a specific situation (the source intention) following a particular technique (the strategy). The different sections of our modeling process can be described as follows:

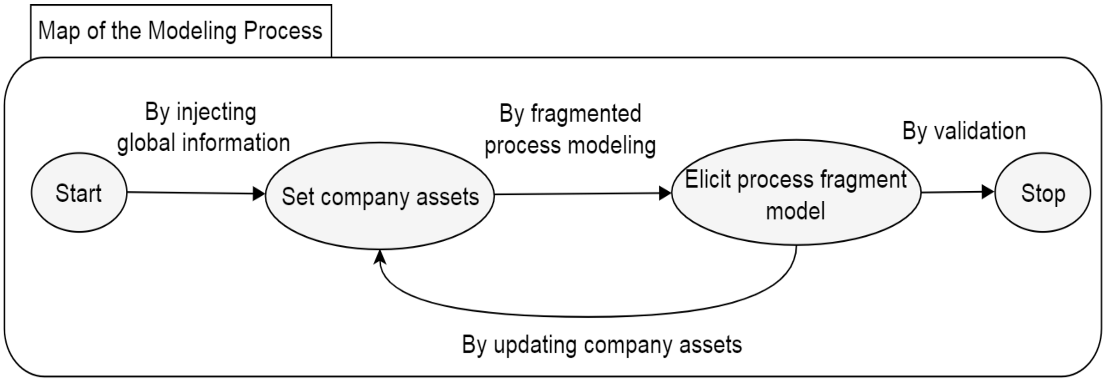

Map of the Modeling Process
Start
By injecting global information
Set company assets
By fragmented process modeling
Elicit process fragment model
By validation
Stop
By updating company assets

Figure 2.24: The proposed modeling process described as a map

- *<Start, Set company assets, By injecting global information>* This section represents the activity required by process owners (e.g., project manager, team responsible, etc.) who have the better vision on the global process to define the global and common information shared among process actors in the company as company assets. Note that all fragments are described based on common company assets definitions.

- *<Set company assets definitions, Elicit process fragments model, By fragmented process modeling>* This section presents the core step of our modeling process which enables process actors to model their process fragments via using the proposed modeling language SPML in a decentralized way. Each role specifies simply the structure of his process fragment as a list of activities under his responsibility. For each activity he defines the lists of mandatory and optional tasks. For each mandatory/optional task he specifies the required and produced artifacts which, generally, are pre-defined in the company assets as mentioned earlier. The artifacts may be associated with an option indicating on which context the respective artifacts are required or produced.

- *<Elicit process fragments model, Set company assets, By updating company assets>* In our approach, we consider that the information regarding company assets can be progressively defined by authorized actors (e.g., project manager, team responsible, etc.) whenever needed by process actors during modeling their process fragments.

- *<Elicit process fragments model, Stop, By validation>* After defining the process fragment model, the process actor can terminate the modeling process indicating that process fragment model is valid and no modification is required in the corresponding model. At this stage, the process fragment model is ready to be enacted, i.e., instantiating fragment's activities and tasks and starting their executions by associating the concrete artifacts and resources required in the project. This issue will be discussed in details in the next chapter.

Our artifact-centric process modeling approach gets the full benefits of fragmented process modeling in which it enables each role to model his process fragment independently of the other fragments. Obviously, each role knows his performing activities/tasks and the artifacts required or consumed by the respective tasks without knowing exactly where and by whom these artifacts are produced or will be used. Based on the common company assets definitions, two fragments of the process can be defined separately but using the same vocabulary for their exchanged artifacts. Thus, we assume that no negotiation and collaboration are required at modeling-time. This will avoid bringing a cognitive overload to the related roles and makes it easy for them to concentrate on their own contribution to the process. For instance, the two fragments of Figures 2.8 and 2.9 are defined separately, however, both roles implicitly define their exchange artifact *FICD* in certain states.

As illustrated in Figures 2.8 and 2.9, the work-sequence relations among activities/tasks are eliminated and process fragments are structurally defined. Moreover, the proposed SPML includes only the process elements known by process actors in their daily work (e.g., task, artifact, resource) and avoid introducing concepts requiring process-special knowledge to understand (e.g., different gateways and events in BPMN).

With the proposed process modeling approach, process actors describe exactly what they do in their routine work. Hence, the gap between a process model and its applications in concrete projects can be considerably reduced. Thanks to this simple and data-flow based modeling language, the enactment of a process becomes a simple instantiation of the process fragments models, i.e., creating concrete instances of the process and its defined activities together with used tasks, then providing these tasks the required effective parameters selected from the real artifact and resource instances. Therefore, process actors can easily enact their process fragment activities without need of a process designer for deploying and validating their process. In fact, our modeling approach, which its main objective is control, is solely conducted by process actors and does not require the participation of process designers, e.g., as a facilitator in modeling process or to validate the process model.

One advantage of our artifact-centric modeling approach is that it is not necessary to define the whole process from the beginning of the project. Thanks to the shared company assets, the process fragments can be defined separately and even progressively during the project execution. We remind that our main modeling objective is control than improvement. However, process improvement will be possible after reconstructing the global process model.

In the next section, we introduce and compare the existing modeling approaches which like us put main emphasize on process actors.

2.5 Related Work

In order to increase the operationality and user acceptance of current process environments, many conducted researches concentrate on improving the process model in order to better match with process actors needs. They advocate the idea of process actors participation in process modeling as they are the only ones who have the through knowledge and have to use the system at the end, and thus they should really know what is expected. This fact can result in a model which is more exact and close to the reality. The variety of user-centric modeling approaches such as participative modeling approaches [18, 52] and role-based modeling approaches [54, 53] with different objectives has been proposed so far. In the next section, we propose our criteria for evaluating current user-centric modeling solutions.

2.5.1 Evaluation Criteria

Agnes et al. [18] propose an evaluation criteria to assess different participative approaches which is composed of three criteria as *modeling process*, *modeling language* and *modeling* *objective*. Each criterion consists of different factors to carry out the evaluation. We extend this evaluation criteria by adding two factors as *fragmentation* and *progressiveness* for the modeling process criterion. Based on these considerations, the Figure 2.25 illustrates the outcome evaluation criteria.

Note that the corresponding evaluation criteria entails the fundamental characteristics of our artifact-centric modeling such as fragmentation and direct enactability. In the next, we present the description of each criterion along with its factors and their connection with the requirements.

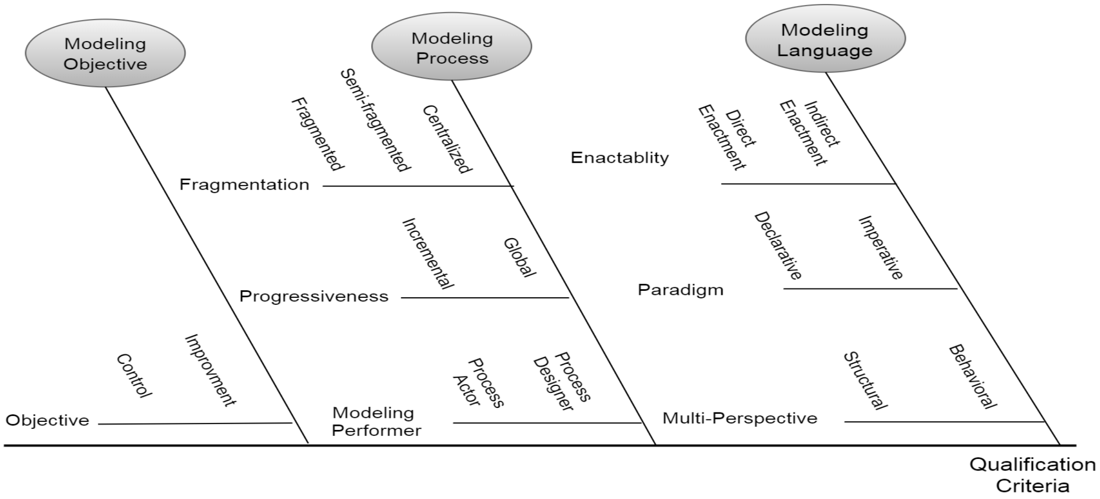


Figure 2.25: Modeling Evaluation Criteria

2.5.1.1 Modeling Objective

This criteria represents the aim of modeling approaches which can be to provide a model of a process in an organization in order to *improve* and communicate among process actors, i.e., by identification of the problem and bottlenecks [12] and/or to provide *control* over the process by providing enactable process models which later can be deployed and enacted by the process engine.

2.5.1.2 Modeling Process

This criterion is related to the modeling process to be followed during the process modeling and consists in three factors to evaluate:

- *Modeling performer* defines the profile of people who are required to participate to process modeling. It can be the *process designer* or the *process actors (end-users)* or participation of both.

- *Fragmentation* describes the required negotiations among process participants (e.g., process actor, process designer, etc.) during the process modeling which can be as follows:

    - *Centralized* where all process actors participate in a meeting to model their processes with the help of process designer as a facilitator.
    - *Semi-fragmented* where each process actor can model his own fragment of process and negotiates with other actors in case of artifacts exchanges.
    - *Fragmented* where each process actor is able to model his fragment of process separately of other fragments.

- *Progressiveness* defines the procedure of modeling process to be *global* by defining the whole process from the beginning or *incremental* by allowing continuously definition of process fragments during the project execution.

2.5.1.3 Modeling Language

This criterion is concerned with the language used during the modeling process. The variety of languages has been proposed for modeling processes in SSE [19] or business domains [33, 42] during last decade.

- *Multi-perspective* corresponds to capacity of modeling muti-aspect of process, each focusing on one aspect of the process [29, 10], which are required to represent complex processes. The main following perspectives can be considered:

    - *Structural aspect* (functional, organizational and informational) presents three modeling elements: who elements (organizational) representing actors, roles, or organizational entities, what elements (functional) representing activities/-tasks and exchange elements (informational) describing the artifacts that are produced, consumed, or otherwise manipulated by the activities/tasks.
    - *Behavioral aspect* provides control-flow information about the process, that is, when activities are performed and how they are performed (e.g., by describing the work-sequence relations between activities/tasks).

- *Enactability* defines the executability of process models, i.e. the capacity of enacting a process model in the context of a concrete project.

    - *Indirect enactment* indicates that the operational semantics of modeling language is not defined. Consequently, there is no supporting process engine for the language and deploying a process model requires efforts of process designer and process actors to map the modeling constructs to equivalent concepts of an operational process engine.

  - *Direct enactment* indicates that the operational semantics of the language is defined, i.e., specifies the behavior of the language's own process engine based on its modeling constructs. Thus, constructing a supporting process engine for the language is direct to deploy and operate process models.

- *Modeling paradigm* represents different approaches in constructing process modeling languages. There are two most important paradigms known as *imperative* or *declarative*.

  - *Imperative* modeling describes how the process has to enacted. It requires every possible execution sequence to be modeled explicitly [42]. Process actors must define the structural and behavioral aspects of their processes at modeling-time.
  - *Declarative* modeling describes the structural aspect of a process without specify how the process has to work exactly at enactment-time. The behavioral aspect is not modeled explicitly via the work-sequence relations. Declarative modeling allows for all possible behavior as long as it is not forbidden by specific constraints, thus can provide a better flexibility for enacting processes [58, 45, 24, 34].

2.5.2 Review of User-centric Modeling Approaches

In the next, we briefly presents some user-centric modeling approaches which, like us, put the main emphasize on process actors in process modeling. However, they put the primary emp have different

We compare them related to our criteria in order to investigate their benefits and shortcomings.

2.5.2.1 Participative Approaches

Participative approaches are mainly based on methods related to quality tools such as seven tools of Total Quality Control [25]. These methods tend to involve the stakeholders of a process in the proposition of ideas for process improvement, use techniques to stimulate and motivate people and help them to solve problems and to propose creative solutions, e.g., SIX Sigma [43, 28], PAWS [8], etc. Participative approaches try to facilitate the process discovery step of modeling phase by organizing joint modeling sessions with a centerlized collaboration and participation of process actors and process designers [52, 4]. In the following, we present some participative approaches.

COMA

Rittgen [46] proposes the COllaborative Modeling Architecture (COMA) to support active participation of the process actors unlike traditional process modeling in order to make models more understandable and more agreeable. He focuses on providing support for modeling and consolidating models during collaborative modeling with a language-agnostic negotiation approach. COMA has four main activities for negotiation which are propose, support, challenge and accept. Basically, each group member creates a global model and proposes that model to others. The acceptance of the proposals is decided based on either rule of seniority or rule of majority. The COMA adopts UML (Unified Modeling Language) to enables actors to model the processes.

ISEA

Front et al. [18] propose a participative modeling approach named ISEA which concerns participative and playful activities realized by the process actors to obtain, evaluate and improve intermediary models of the business process by using a domain specific language (DSL) that covers structural and behavioral perspectives of a process. ISEA proposes the mappings rules between intermediary models and standard models (BPMN) in order to automate the business process. The transformation is obtained by mapping each concept of the meta-model to a BPMN collaboration diagram concept. The transformed process is not enactable and requires a process designer to deploy the process in the target process environment (e.g., Bonita [7]).

2.5.2.2 Role-Based Modeling Approaches

Role-based approaches differ from participative approaches on the way of process actors participation in process modeling. Role-based approaches put more stress on process actors to model their processes in a decentralized (fragmented) fashion. Each role in the organization is responsible for modeling its own contribution to the process. Due to the collaborative nature of role-based and fragmented process modeling, negotiations among the actors who play different roles have a crucial impact on modeling activities of overall process. Based on the communication and negotiation time among the actors, these types of process modeling approaches vary. In the following, we resume some important works.

Plural

Turetken et al. [54, 55] propose the Plural method based on a multi-perspective modeling paradigm, which focuses on representation of individual work contributions in models and subsequently merges them into a common model by agreeing upon the interfaces (artifact exchanges) among the individual models. In Plural, the notation used for describing the structural and behavioral aspects are based on extension of Event-driven Process Chain (eEPC) [50] technique, which is semi-formal, imperative and has limitations particularly in relation to the process execution. It is assumed that actors are familiar with this language. Plural uses tool support built as add-on upon a commercial modeling environment ARIS Toolset [51], which identifies inconsistencies between individual models. Currently, however, it does not incorporate mechanisms to support the enactment of the defined processes based on enactable definitions generated from these models. Vandenhurk et al. [56] explore the feasibility of applying BPMN [37] in Plural as it provides great execution support. Plural defines a third party participant called coordination team in their process modeling.

RoaDMap

Ertugrul et al. [14] propose RoaDMap (ROle-based And Decentralized process Modeling) to allow process actors to model their fragments of process. The overall process model is formed progressively as long as the roles model their own internal behaviors and identify interactions from their own perspectives. If there exist inconsistencies, roles identify and resolve them in an ongoing way. Their proposed modeling language is an extension of S-BPM [15]. As S-BPM notation has simple and few number of elements, it is easy to learn for the inexperienced people. S-BPM comprises two types of diagrams, subject interaction diagram (SID) to illustrate the subjects and artifact exchanges among them and subject behavior diagram (SBD) to describe the structural and behavioral aspect of each subject. However, S-BPM receives great enactment support but the authors did not examine the enactment possibility of their extended S-BPM in the existing S-BPM process engines.

CTM

Stoitsev et al. [53] develop a collaborative task management tool CTM (as add-in to Microsoft Outlook) to enable end-users creating hierarchical to-do lists (tasks and sub tasks). Tasks can be delegated over email exchange. Tracking of email exchange for task delegation integrates the end-user's personal to-do list to overall Task Delegation Graphs (TDG). TDGs represent weakly-structured process models that are captured as results of actual process enactment. Their approach supports process elicitation through transformations of user-defined TDGs to formal workflows based on the task change and evolution history. Task changes which alter task status, percent complete or task artifacts, are considered as task processing changes, denoting that the user is acting on a given task. The transformation is acheived by usage of jBOSS Business Process Management (jBPM) solution [26] which uses an imperative modeling language (jPDL) to model workflows. As the work-sequence relations among tasks are defined as suggestion based on the time of tasks processing changes (e.g., alter task status, percent complete), the resulting process is no exact and must be validated by end-users. Furthermore, as TDGs are not enactable, their solution required the process designer to transform the TDGs into formal process models and deploy into target process environment.

Oppl

Oppl [40] proposes a modeling approach which is driven by process actors and allows them to model and align their views on a work process, and still leads to a syntactically correct and semantically sound process model for further processing in IS (enactable models). They apply case-based approaches that avoids the need for control-flow constructs which reduces the number of modeling elements to make modeling easier for non-expert modelers. The modeling process starts with defining a common understanding of the relevant concepts and the scope of the process. Then the process modeling continues with a set of concurrent individual modeling sessions, in which all involved process actors structurally define their process fragments. Then, the individual models are brought together and aligned to form a coherent and agreed-upon model of the global process model. Finally, by applying a set of transformation rules [39], the case-based model is mapped to target business process model in languages such as S-BPM [15] or BPMN [37]. As the obtained model initially reflects only one variant of the process, the aim of this step is creating a semantically correct representation of the work process in all its variations. Their approach requires process actors to know the S-BPM language , which follows an imperative paradigm, in order to improve the transformed model to include all variants of their working process.

2.5.3 Synthesis

Table 2.1 synthesizes the different modeling solutions studied in the previous section in relation to our criteria.

In general, participative approaches are considered as solutions mainly focus on keeping the complexity of process modeling low so that process actors can actively participate during the modeling phase in a centralized manner. To do so, they attempt to propose a simple modeling language to be more convenient for process actors to express their knowledge about their processes. Particularly, the objective is to obtain a process model by means of communication and improvement. They emphasize on high degree of collaboration in process modeling by holding a centralized modeling session which requires participation of all involved process actors and process designer as a facilitator to organize and guide the modeling sessions and resolve inconstancies.

Role-based approaches attempt to reduce the degree of collaboration among process actors and give more freedom to them in modeling their own contribution to the process by providing fragmented (decentralized) process modeling. To create a sound process model, all involved roles need to communicate and negotiate in case of artifact exchanging among them, either before, during or after modeling their process fragments. This fact brings limitation in getting full advantage of fragmented process modeling as it makes fragments to be dependent on each other.

Our proposed modeling approach gets the benefits of participative approaches in simplicity of modeling language and role-based approaches in fragmentation of process modeling, and strengths their shortcomings in order to achieve a solution which better suits process actors needs. In the following, we resume the advantages of our proposed modeling approach.

- *Structural process model* Mosltly all discussed works follow the imperative paradigm in their process modeling which emphasizes on behavioral aspect of the process which forces process actors to pre-define all situations in their working process that may happen at enactment-time. The work in [40] pursues declarative paradigm by adopting a modeling language which allows process actors to model their working process structurally. In order to provide enactment support, they transform the structurally defined models to imperative process models. Then process actors must improve and validate obtained models by considering all possible variants (situations) of their working process. This requires first process actors to know the imperative modeling language (e.g., BPMN [37], S-BPM [15]) and most importantly, it leads to the problem of imperative modeling as discussed in Section 2.1.
  Our modeling language SPML follows the declarative and artifact-centric paradigm by enabling process actors to define only structural aspect of their processes. Note that specification of input and output artifacts of each task implicitly defines the model constraints. The behavioral aspect of process emerges at enactment-time by real exchange of artifacts among process actors. Moreover, the proposed concept of *Option* in SPML enables process actors to specify different variants of their working process without overwhelming them by using work-sequence modeling constructs. Suppression of work-sequence relations makes SPML a less rigid language and gives more autonomy to process actors to control their processes at enactment-time, i.e., some conditions such as multi-instance or loop are not explicitly define in the model and process actors should control the number of their tasks at enactment-time.

- *Enactable process model* Current end-user modeling approaches have lack of enactment support. Mainly they aim at process improvement than control and so the obtained models are far from being further processed in the concrete project by means of enactment. The task of transforming these models into representations having a (semi) formal and executable modeling language that can be processed in process environments, however, is left to process designers. Only few works [18, 40] provided some support for process control by adopting transformation rules between their modeling language and the target (semi) formal language. But the transformed models are complex which required further effort of process actors and process designer to be evaluated, redefine, validated and deployed into the target process environment, i.e., by establishing a data mapping concerning information flow from the process context to a particular task and from the task to the process.
  Our approach enables process control by providing direct enactment and synchronization support of the structurally defined process fragment models. Our proposed

modeling language SPML is simple and data-flow based which can facilitate the precise definition of its execution semantics. Moreover, it leads to ease of process deployment where the deployment of a process fragment is achieved by associating the artifacts and resources with the concrete ones in the project.

- *Fragmented process modeling* Role-based approaches emphasize on fragmented process modeling in which their main objectives have been the process improvement. Thus, to obtain a valid global model, process fragments are dependent on each other by means of negotiation and cannot be modeled separately than each other, e.g., process actors who exchange an artifact, cannot continue modeling their processes until agreeing on the exchanged artifacts.
  As the objective of our modeling approach is control, it minimizes the degree of collaboration in process modeling and allows each role to concentrate on his own work regardless of other process fragments. The company assets definition allows process actors to reach an agreement on the content level. Then, we assume that each actor knows his tasks, respective artifacts to consume or produced. Thus, they do not require to know the source and target of their required or produced artifacts. In contrast to role-based modeling approaches, the negotiation among process fragments are automatically conducted when process actors enact their tasks and exchange concrete artifacts.

- *Incremental (Progressive) process modeling* To the best of our knowledge, none of the existing end-user modeling approaches provide support for incremental process modeling in which some fragments can be modeled and enacted where other process fragments are missing. In fact, in these approaches, the overall process model is constructed at modeling-time and so it is required before enactment. In our approach, the creation of process fragment models occurs progressively. In fact, we do not require the overall model to be defined before enactment, but it is progressively constructed based on the real enactment of separate fragments. As the dependency among process fragments are eliminated in the modeling-time, this dependency will be deduced at enactment time to construct the global process model. However, lack of global model hinders its validation before enactment.

To conclude, by inversion of the direction of process modeling and adopting an artifact-centric paradigm, our modeling approach can reduce the gap and bring both simplicity and enactability to process modeling.

2.6 Summary

This chapter introduced our artifact-centric process modeling approach which puts main stress on process actors to model their enactable working process. To this end, a process modeling language named SPML dedicated to process actors is proposed. SPML is considered as an end-user friendly process modeling language which allows process actors to structurally define their process fragments containing activities performed by their roles. It includes only the process elements known by process actors in their daily work and avoids introducing concepts requiring process-special knowledge to understand. The key feature of the SPML is to remove the work-sequence relations among activities and describing in detail the artifacts entering and leaving the activities to provide enough information to deduce the behavioral aspect of the process at enactment-time. Our modeling approach gets the full benefits of fragmented process modeling in which fragments are defined separately than each other. The procedure of fragments creation is incremental where some fragments can be modeled and enacted while other process fragments are missing. Our proposed modeling language SMPL is expressive enough to define executable process fragments models. However, fragmented and structural process modeling create new challenges for enacting and synchronizing the separate fragments of the process. In the next chapter, we present the solution to enable coordinating a process which is (partially) modeled in separate fragments.

Table 2.1: Synthesis of End-user Modeling Approaches

Related work		Comparison						
		Modeling Langugae			Modeling Process			Modeling Objective
Category	Name/Auther	Multi-Perspective	Enactability	Paradigm	Modeling Performer	Progressiveness	Fragmentation	Objective
Participative Modeling Approaches	COMA	Structural Behavioral	Indirect	Imperative	Process designer Process actor	Global	Centeralized	Improvement Semi-control
	ISEA	Structural Behavioral	Indirect	Imperative	Process actor	Global	Centeralized	Improvement Semi-control
Role-based Modeling Approaches	Plural	Structural Behavioral	Indirect	Imperative	Process actor	Global	Semi-decentralized	Improvement Semi-control
	RoaDMap	Structural Behavioral	Indirect	Imperative	Process actor	Global	Decentralized	Improvement Semi-control
	CTM	Structural Behavioral	Indirect	Imperative	Process designer Process actor	Global	Decentralized	Improvement Semi-control
	Oppl	Structural Behavioral	Indirect	Imperative	Process actor	Global	Decentralized	Improvement Semi-control
Artifact-centric Process Modeling		Structural	Direct	Declarative	Process actor	Incremental	Decentralized	Control

Appendix

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Analyst
        Activity Detail Change Requirement options instrumentation
            MandatoryTask Define Component Requirements
                Artifact Wiring Change Demand(in, defined)
                Artifact Component Requirements(out, defined, collection)
            MandatoryTask Define Electrical Specification
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(in, defined, toFinish) ifOption instrumentation
                Artifact Electrical Specification(out, defined)
            OptionalTask Outline Instrumentation Model ifOption instrumentation
                Artifact Wiring Change Demand(in, defined)
                Artifact FICD(out, outlined)
```

Figure 2.26: Process Fragment of Analyst

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Supplier
        Activity Purchase Component
            MandatoryTask Supply Component
                Artifact Component Requirements(in, defined)
                Artifact Component(out, defined)
```

Figure 2.27: Process Fragment of Supplier

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Instrumentation Team
        Activity Design Instrumentation
            MandatoryTask Define Instrumentation Model
                Artifact FICD(in, outlined)
                Artifact FICD(out, defined)
        Activity Generate Bench Specification options interface
            MandatoryTask Define Bench Specification
                Artifact FICD(in, defined)
                Artifact Bench Specification(out, defined)
            OptionalTask Define Specific ICD ifOption interface
                Artifact FICD(in, defined)
                Artifact ICD(out, defined)
```

Figure 2.28: Process Fragment of Instrumentation Team

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Wiring Team
        Activity Fix and Install Components
            MandatoryTask Fix Component
                Artifact Component(in, defined)
                Artifact Installed Component(out, defined)
        Activity Wiring All Components
            MandatoryTask Wiring components
                Artifact Installed Components(in, defined, collection)
                Artifact Testbench Wired(out, defined)
```

Figure 2.29: Process Fragment of Wiring Team

```
Process Modify Testbench Wiring options instrumentation, interface
    ProcessFragment Electrical Designer
        Activity Define Electrical Model
            MandatoryTask Specify Electrical Model
                Artifact Electrical Specification(in, defined)
                Artifact Electrical Design Model(out, defined)
```

Figure 2.30: Process Fragment of Electrical Designer

```
Process Modify Testbench Wiring options instrumentation, interface
  ProcessFragment Bench Coordinator
    Activity Build Bench Specification
      MandatoryTask Define Bench Specification
        Artifact Bench Specification(in, defined)
        Artifact ICD(in, defined) ifOption interface
        Artifact Testbench Configuration(out, defined)
    Activity Validate
      MandatoryTask Validate Bench
        Artifact Testbench Configuration(in, defined)
        Artifact Testbench Wired(out, validated)
```

Figure 2.31: Process Fragment of Bench Coordinator

Bibliography

[1] Workflow pattern, http://www.workflowpatterns.com/.

[2] OMG Object Constraint Language (OCL), Version 2.4, http://www.omg.org/spec/OCL/2.4, February 2014.

[3] Y. Alotaibi. Business process modelling challenges and solutions: a literature review. *Journal of Intelligent Manufacturing*, 27(4):701–723, Aug 2016.

[4] J. Barjis. *Collaborative, Participative and Interactive Enterprise Modeling*, pages 651–662. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[5] L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend.* Packt Publishing, 2013.

[6] M. Blay-Fornarino, A. Pinna-Dery, K. Schmidt, and P. Zaraté, editors. *Cooperative Systems Design, A Challenge of the Mobility Age, Proceedings of COOP 2002, Saint-Raphaël, France, 4-7 June 2002.* IOS, 2002.

[7] BonitaSoftware. http://www.bonitasoft.com/.

[8] M. R. Borges and J. A. Pino. Paws: Towards a participatory approach to business process reengineering. In *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, SPIRE '99, pages 262–, Washington, DC, USA, 1999. IEEE Computer Society.

[9] C. Britton and S. Jones. The untrained eye: How languages for software specification support understanding in untrained users. *Hum.-Comput. Interact.*, 14(1):191–244, Mar. 1999.

[10] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35(9):75–90, Sept. 1992.

[11] J. C. de A. R. Gonçalves, F. M. Santoro, and F. A. Baião. Business process mining from group stories. In M. R. S. Borges, W. Shen, J. A. Pino, J.-P. A. Barthès, J. Luo, S. F. Ochoa, and J. Yong, editors, *CSCWD*, pages 161–166. IEEE, 2009.

[12] R. M. de Freitas, M. R. S. Borges, F. M. Santoro, and J. A. Pino. *Groupware Support for Cooperative Process Elicitation*, pages 232–246. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[13] R. Deneckere, E. Kornyshova, and C. Rolland. Enhancing the guidance of the intentional model "MAP": Graph theory application. In *Research Challenges in Information Science (RCIS)*, pages 13–22, Fès, Morocco, Apr. 2009.

[14] A. M. Ertugrul and O. Demirors. A method for modeling business processes in a role-based and decentralized way. In *Proceedings of the 8th International Conference on Subject-oriented Business Process Management*, S-BPM '16, pages 4:1–4:4, New York, NY, USA, 2016. ACM.

[15] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Brger. *Subject-Oriented Business Process Management*. Springer Publishing Company, Incorporated, 2014.

[16] M. Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[17] J. Friedrich and K. Bergner. Formally founded, plan-based enactment of software development processes. In *Proceedings of the 2011 International Conference on Software and Systems Process*, ICSSP '11, pages 199–203, New York, NY, USA, 2011. ACM.

[18] A. Front, D. Rieu, M. Santorum, and F. Movahedian. A participative end-user method for multi-perspective business process elicitation and improvement. *Software & Systems Modeling*, pages 1–24, 2015.

[19] L. García-Borgoñón, M. A. Barcelona, J. A. García-García, M. Alba, and M. J. Escalona. Software process modeling languages: A systematic literature review. *Inf. Softw. Technol.*, 56(2):103–116, Feb. 2014.

[20] G. Grambow, R. Oberhauser, and M. Reichert. *User-Centric Abstraction of Workflow Logic Applied to Software Engineering Processes*, pages 307–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] M. Hajmoosaei, H. N. Tran, and C. Percebois. A user-centric process management for system and software engineering projects. In *presented at the 7th IESM Conference, October 11–13, 2017, Saarbrucken, Germany*, 2017.

[22] D. Harel and B. Rumpe. Meaningful modeling: What's the semantics of "semantics"? *Computer*, 37(10):64–72, Oct. 2004.

[23] T. Herrmann, K. Loser, and I. Jahnke. Sociotechnical walkthrough: A means for knowledge integration. *The Learning Organization*, 14(5):450–464, 07 2007.

[24] T. T. Hildebrandt and R. R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010.*, pages 59–73, 2010.

[25] K. Ishikawa. *What is total quality control? The Japanese way.* Prentice Hall business classics. Prentice-Hall, 1985.

[26] JBoss. jbpm, http://www.jbpm.org/.

[27] S. Kabicher and S. Rinderle-Ma. *Human-Centered Process Engineering Based on Content Analysis and Process View Aggregation*, pages 467–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[28] P. Keller and T. Pyzdek. *The Six Sigma Handbook, Fourth Edition.* McGraw-Hill Education, 2014.

[29] J. Krogstie. *Perspectives to Process Modeling – A Historical Overview*, pages 315–330. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[30] J. Krogstie and H. D. Jørgensen. Interactive models for supporting networked organisations. In *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, pages 550–563, 2004.

[31] M. Kuhrmann, G. Kalus, and M. Then. The process enactment tool framework-transformation of software process models to prepare enactment. *Sci. Comput. Program.*, 79:172–188, Jan. 2014.

[32] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven process modeling guidelines (7pmg). *Inf. Softw. Technol.*, 52(2):127–136, Feb. 2010.

[33] H. Mili, G. Tremblay, G. B. Jaoude, E. Lefebvre, L. Elabed, and G. E. Boussaidi. Business process modeling languages: Sorting through the alphabet soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, Dec. 2010.

[34] M. Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing.* Springer, 2010.

[35] M. Z. Muehlen and J. Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Proceedings of the 20th International Conference on Advanced Information Systems Engineering*, CAiSE '08, pages 465–479, Berlin, Heidelberg, 2008. Springer-Verlag.

[36] OMG. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0, Apr. 2008.

[37] OMG. Business Process Model and Notation (BPMN) Version 2.0.2, Dec. 2013.

[38] S. Onoda, Y. Ikkai, T. Kobayashi, and N. Komoda. Definition of deadlock patterns for business processes work ow models. In *Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 5 - Volume 5*, HICSS '99, pages 5065–, Washington, DC, USA, 1999. IEEE Computer Society.

[39] S. Oppl. Articulation of subject-oriented business process models. In *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*, S-BPM ONE '15, pages 2:1–2:11, New York, NY, USA, 2015. ACM.

[40] S. Oppl. Articulation of work process models for organizational alignment and informed information system design. *Information & Management*, 53(5):591 – 608, 2016.

[41] S. Oppl. Towards scaffolding collaborative articulation and alignment of mental models. *Procedia Comput. Sci.*, 99(C):125–145, Oct. 2016.

[42] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers. *Imperative versus Declarative Process Modeling Languages: An Empirical Investigation*, pages 383–394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[43] S. Pimsakul, N. Somsuk, W. Junboon, and T. Laosirihongthong. *Production Process Improvement Using the Six Sigma DMAIC Methodology: A Case Study of a Laser Computer Mouse Production Process*, pages 133–146. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[44] J. Recker, N. Safrudin, and M. Rosemann. How novices design business processes. *Information Systems*, 37(6):557 – 573, 2012. BPM 2010.

[45] H. A. Reijers, T. Slaats, and C. Stahl. *Declarative Modeling–An Academic Dream or the Future for BPM?*, pages 307–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[46] P. Rittgen. Collaborative modeling - a design science approach. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10, Jan 2009.

[47] C. Rolland, N. Prakash, and A. Benjamen. A multi-model view of process modelling. *Requirements Engineering*, 4(4):169–187, 1999.

[48] J. Roschelle. Computers, communication and mental models. chapter Designing for Cognitive Communication: Epistemic Fidelity or Mediating Collaborative Inquiry?, pages 15–27. Taylor & Francis, Inc., Bristol, PA, USA, 1996.

[49] N. Russell, A. H. Ter Hofstede, W. M. Van Der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. *BPM Center Report BPM-06-22, BPMcenter. org*, pages 06–22, 2006.

[50] A.-W. Scheer, O. Thomas, and O. Adam. *Process Modeling Using Event-driven Process Chains*, pages 119–145. Wiley, Hoboken, New Jersey, 2005.

[51] SoftwareAG. Software ag. http://www.softwareag.com/, 2014. retrieved january 2015.

[52] J. Stirna, A. Persson, and K. Sandkuhl. Participative enterprise modeling: Experiences and recommendations. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, CAiSE'07, pages 546–560, Berlin, Heidelberg, 2007. Springer-Verlag.

[53] T. Stoitsev, S. Scheidl, F. Flentge, and M. Mühlhäuser. From personal task management to end-user driven business process modeling. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 84–99, Berlin, Heidelberg, 2008. Springer-Verlag.

[54] O. Turetken and O. Demirors. Plural: A decentralized business process modeling method. *Information and Management*, 48(6):235 – 247, 2011.

[55] O. Turetken and O. Demirors. *Business Process Modeling Pluralized*, pages 34–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[56] H. van den Hurk, O. Turetken, and J. van Moll. Subject-oriented plural method meets bpmn: A case study. In *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*, S-BPM ONE '15, pages 5:1–5:9, New York, NY, USA, 2015. ACM.

[57] W. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach*. The MIT Press, 2011.

[58] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

[59] Xtext. https://eclipse.org/xtext/documentation/index.html.

Chapter 3

Artifact-driven Process Enactment

Contents

This chapter introduces our process enactment approach which assists process actors to perform their processes encoded in SPML. We aim at proposing an automated mechanism to enact and synchronize different process fragments without confining process actors to predefined execution scenarios. Section 3.1 discusses the main challenges to obtain this objective and Section 3.2 characterizes our approach to enact a fragmented process model while offering some degree of flexibilities to process actors. Section 3.3 presents the architecture of our process environment Bottom-up Artifact-centric Process Environment (BAPE) while Section 3.4 and Section 3.5.3 describe in detail two main components of BAPE: a Process Dependency Graph (PDG) giving information about process instances at enactment-time and a Process Engine executing process instances. We emphasize the flexibilities of our enactment approach in Section 3.6. Section 3.7 compare our approach to related work and Section 3.8 summarizes this chapter.

3.1 Challenges of Enacting SPML Process Models

Once the process is modeled, it can be put at working environment where real execution happens by process actors performing their process model activities. Process enactment is supported by a process engine whose responsibility is to control and monitor the execution of operational processes by synchronizing their activities according to the process models.

In conventional activity-driven process environments, process engines synchronize process activities based on the work-sequence relations defined between activities in the model [12, 6, 4]. They require then the whole process to be completely defined before enactment in order to deduce the dependencies between process activities at enactment-time, i.e., run-time. In this way, the conventional process engines use principally the information on model-level to synchronize process activities in an established manner.

Obviously, the conventional activity-driven environments are unsuitable for enacting SPML process models because of the characteristics of our modeling approach as discussed in the next.

3.1.1 Structurally and Partially Defined Processes

We face two major challenges regarding enacting SPML process models which particularly originate from the characteristics of our user-centric process modeling approach:

- SPML processes are structurally modeled in separate fragments where there is no work-sequence relations defined among process activities or tasks. This fact poses a question regarding how to know the inter-dependencies among activities and among tasks coming from different fragments in order to synchronize them correctly at enactment-time.

- SPML processes can be partially and progressively modeled because the process fragments can be defined or not, and can be entered in the process environment at different times. The second challenge here is how to enact a partially modeled process with some process fragments are missing.

In our enactment approach, the main challenge is to propose a mechanism in order to provide control over structurally and partially modeled processes encoded in SPML. The enactment approach cannot solely rely on model-level information and should adopt another mechanism to deduce dependencies among process tasks at enactment-time in order to provide synchronization among them.

3.1.2 User-centric Enactment

Generally, process management tries to provide a balance between control and flexibility over processes [33]. On the one hand, there is a desire to control processes and to avoid incorrect or undesirable executions of these processes. On the other hand, process actors want flexible process environments that do not put constraint on them in carrying out their daily works. This apparent paradox has limited the application of process environments, since, as indicated by many authors [17, 25, 5, 10], process environments are too restrictive and have problems when dealing with unforeseen situations that occur at enactment-time according to process actors needs. Our main objective is to ease the use of process environments, thus our enactment approach should give process actors more flexibility and autonomy in performing their processes.

In our user-centric approach, defining all execution situations at modeling-time is undesirable for process actors from the standpoint of model complexity due to a large number of possible execution paths. Thus the approach *flexibility by design* [28] which emphasizes on defining all foreseen situations in the models is not suitable for our intention. The challenge for our enactment approach is then to support more flexibility to handle both foreseen and unforeseen situations without requiring process actors' extra efforts when using the process environment.

To propose an efficient solution, we identify the fundamental characteristics of such an enactment approach which are discussed in details in the next section.

3.2 Data-Driven and User-Centric Process Enactment

Our ultimate goal is not only allowing process actors to create enactable models but also providing an operational process engine to execute their models [9]. Considering the characteristics of our modeling approach and the need of providing more flexibility to process actors, the process engine implementing the operational semantics of our process modeling language SPML should have two following features:

- *Data-driven synchronization.* SPML is defined for modeling especially software and system processes where the execution of each activity will change the state of an artifact representing a part of the final product. Therefore, we consider artifacts as the main driver for process progression and define the operational semantics of an activity based on the evolution of the artifacts that it uses or produces.
  SPML process models are fragmented and structurally defined without work-sequence relations among activities. However, their inter-dependencies between the activities inside or among process fragments can be deduced at enactment-time from the specification of the artifacts that they exchange. Our process engine adopts then an artifact-driven synchronisation which uses the states of artifacts from the real enactment circumstances to make activities progress. One advantage of this feature is that it enables the enactment of partially modeled processes where some fragments are missing or progressively defined. Because the synchronization of the fragments defined in the model with those performed by external systems/agents is based on the artifacts managed by a shared database, the process engine does not need a complete process model to start executing its activities.

- *User-centric control.* By suppressing work-sequence relations in process models, SPML does not prescribe the way process's activities are enacted. Besides the constraints on the availability of the used artifacts, process actors have more freedom in enacting their activities. The enactment approach must offer then some degree of flexibility to allow different execution possibilities of activities in response to different execution contexts (e.g., optional activities) or unforeseen situations (e.g., rework) that may happen at enactment-time.

In the next, we present how our enactment approach exhibits the characteristics mentioned above. To do so, first we introduce the overall architecture of our developed Bottom-up Artifact-centric Process Environment (BAPE) which allows process actors to model their processes fragments via SPML and enact them. Then, we thoroughly present the main components of the BAPE enactment environment which are (1) a Process Dependency Graph (PDG) as a run-time storage keeping the information of process element instances coming from different process fragments and therefore represent the global view of the system; (2) a process engine which provides a synchronization mechanism to manage process activities and tasks. Finally, we introduce the flexibility that our enactment approach supports.

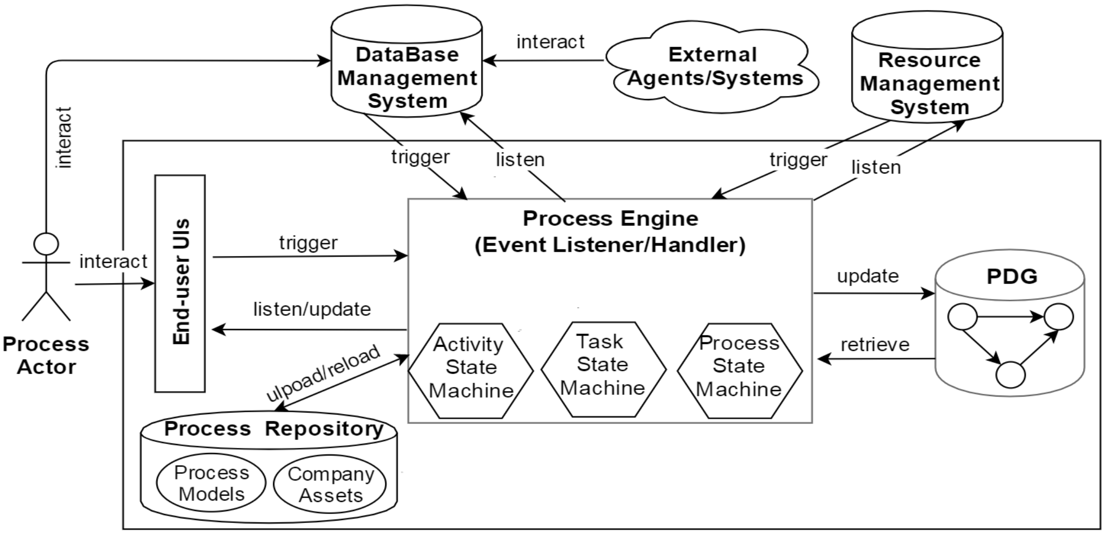


Figure 3.1: Architecture of BAPE

3.3 Process Environment BAPE

We introduce in this section the overall architecture of our developed Bottom-up Artifact-centric Process Environment (BAPE) which allows process actors to model their processes fragments via SPML and enact them. Figure 3.1 illustrates the overall architecture of BAPE. In the following, a brief description of its components is provided whereas the more technical and detail explanations of them will be discussed in the Chapter 5.

1. *End-user UIs* This component provides interfaces that allow process actors to interact with process environment during the modeling, enacting and monitoring phases. The modeling interface enables process actors to define company assets and model the fragments of their processes and then store them into the *Process Repository*. The enactment interface enables process actors to manage their activities and tasks (e.g., create activities, start and complete tasks, etc.). Each user's action on an activity or a task triggers a specific event that is handled by the process engine. The monitoring interface enables process actors to assess the performance of their running processes.

2. *Database Management System (DBMS)* We suppose that resources (e.g., process actors, tools, etc.) and working artifacts are externally managed by each company proper tools. However, those external tools use a central DBMS which is connected to BAPE. This connection makes the process environment being aware of any events changing the state of artifacts and resources in external systems.

3. *Process Dependency Graph (PDG)* This component stores the information of

running processes managed by BAPE. PDG is progressively updated by the process engine to reflect always the current state and the global view of the system at run-time. Hence, PDG is a source of information for the process engine to coordinate and synchronize processes.

4. *Process Engine* This component is in charge of enacting and synchronizing the processes managed by BAPE. Process engine implements the operational semantics of SPML which define the behavior of process's enactable elements at enactment-time, i.e. processes, activities and tasks. The behavior of the process engine is thus defined based on the state machines specifying the life cycles of process, activity and task instances.

We used the conventional architect for our process environment but proposed innovations in developing the two core components PDG and process engine in order to enable a more flexible enactment.

3.4 Process Dependency Graph PDG

In the process environment, process information exists at two levels: model and instance levels. The former relates to process information existing at modeling-time and extracted from process models which are static and stateless. The modeling-level information was discussed in the precedent chapter in details. On the other side, the instance-level process information which is dynamic and statefull, is obtained when a process model is enacted. When enacting a process model, the instances of process elements in the model are created and the mappings from stateless model elements to their corresponding statefull instances are established. It is necessary for a process environment to adopt a mechanism to store the instance-level information.

Process Dependency Graph (PDG) is our solution to monitor the instances of process elements enacted in the process environment as well as the relations among them. PDG represents the instances of process elements along with the dependencies among them at enactment-time and is progressively updated by the process engine to reflect always the current state of the system. As in our approach process models are fragmented and are progressively enacted during the project execution, PDG establishes the global view of the system from separate process fragments at enactment-time via shared resources among respective fragments. This global view helps the process engine in synchronizing the process activities/tasks inside or among process fragments. In our process environment PDG has two important functionalities as following:

- PDG is the source of information about process element instances. This instance level information is required by the process engine in order to coordinate and synchronize the process activities/tasks instances. This functionality of PDG is discussed in the Section 3.5.3 in details.

- PDG, by having the global view of the system, provides the base for interesting evaluations such as analysis of process change or analysis of process itself. The former concerns the impact analysis of the change that may happen at enactment-time, e.g., changing an already produced artifact which has being used by several people. The latter concerns providing the process owner (e.g., project manager) with information about the current status of the running projects. These aspects will be discussed in the Chapter 5 in details.

PDG has a graph structure and is stored in a graph database. Graph structures are particularly beneficial in areas where information about data inter-connectivity is more important, or as important as the data itself. This fact is completely reflected in SPML process models comprised of several connected elements. Moreover, SMPL process fragments are defined separately but the dependencies among them is deduced at enactment-time based on sharing resources. When assuming a real context with thousands of running processes, the relations established among the process fragments result in a highly connected data structure. When it comes to the process synchronization where process engine requires to know these dependencies among running process elements, having a mechanism implementing a graph structure is a promising solution which perfectly represents the connected information and provide highly effective query and traversal mechanisms.

3.4.1 Structure of PDG

PDG is defined as a directed graph composed of a set N of nodes and a set E of edges. Nodes and edges of a PDG are typed in which each node represents an instance of an SPML element and each edge represents the relationship between SPML elements at instance-level. PDG is comprised of six types of nodes and five types of edges which are specified in the following definitions:

$PDG = (N\ ,\ E)$

$N = ProcessNode \ \cup \ ActivityNode \ \cup \ TaskNode \ \cup \ ArtifactNode \ \cup \ ActorNode \ \cup \ ResourceNode$

$E = BelongEdge \ \cup \ ChildEdge \ \cup \ DataEdge \ \cup \ PerformEdge \ \cup \ UseEdge$

Figure 3.2 illustrates the structure of PDG along with presenting the mappings between PDG elements with their corresponding SPML elements. We have mostly direct mappings between elements of SPML and PDG. For instance, an element *ProcessNode* has a property *reference* which directly assigns the PDG *ProcessNode* to the element *Process* from SPML. Some of the mappings given in the Figure 3.3 have to be defined based on indirect connections between SPML elements and PDG elements to reflect the right semantics and to fulfil a particular purpose as explained in the Section 3.4.1.2. In the next, we present each node and edge of the PDG in details.

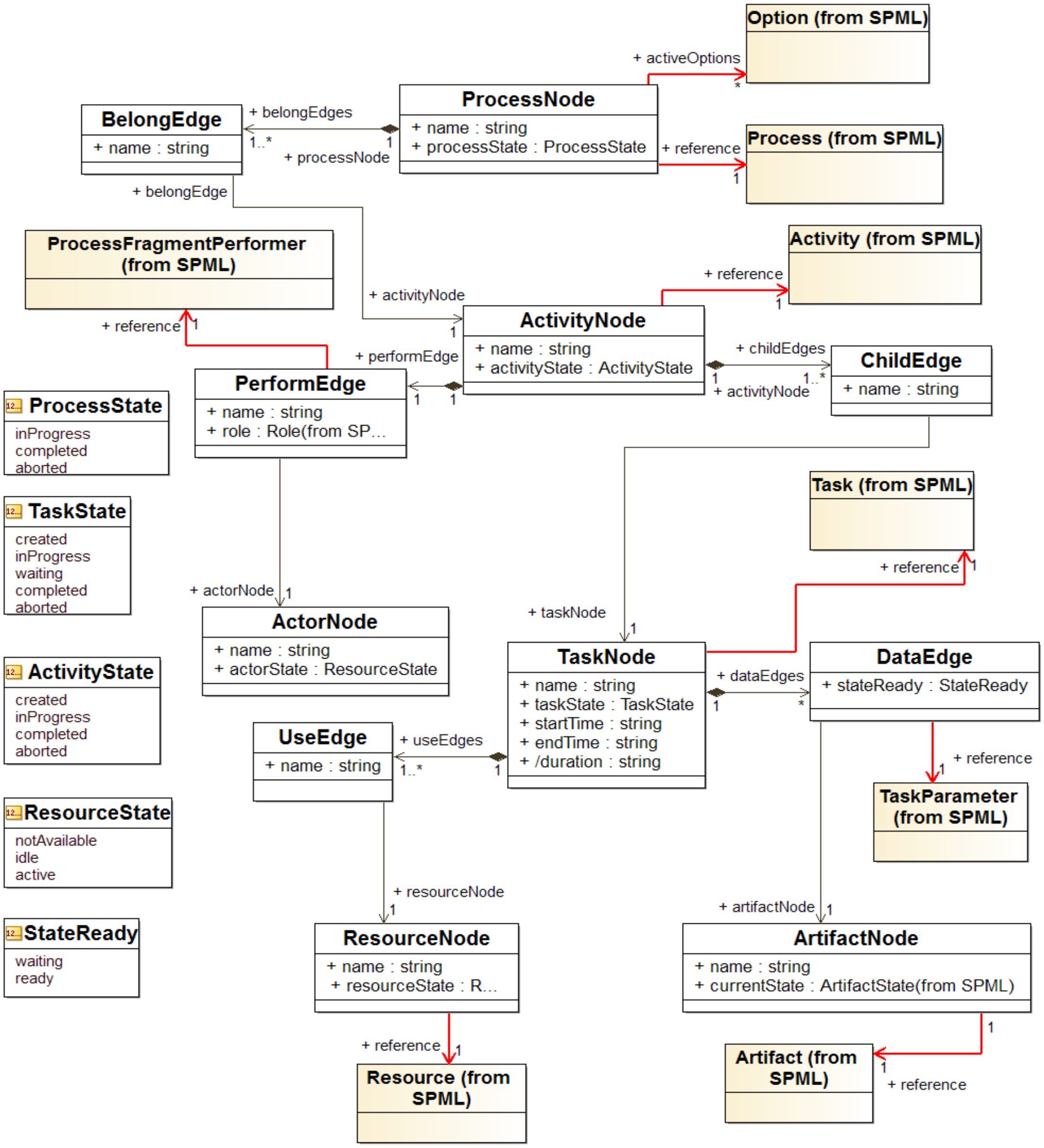


Figure 3.2: Structure of PDG along with references to SPML

3.4.1.1 Nodes of PDG

PDG nodes represent process element instances. PDG presents the dynamic information and behavior of the system, thus each PDG node has the property describing its current state in the system. Each type of nodes along with their properties are defined as follows:

- *ProcessNode* represents an instance of a process whose attributes are defined in the Table 3.1.

Table 3.1: Properties of a $ProcessNode$

Property	**Description**
reference: Process	Links to the corresponding element *Process* in SPML.
activeOptions: Option	Indicates the list of options which are defined in the process model, and are activated in the process instance.
processState: ProcessState	indicates the current state of the process instance. The values of this property are define by the enumeration *ProcessState* as follows: *processState = (inProgress, completed, aborted).*

- *ActivityNode* represents an instance of an activity whose attributes are defined in the Table 3.2.

Table 3.2: Properties of an $ActivityNode$

Property	**Description**
reference: Activity	Links to the corresponding element *Activity* in SPML.
activityState: ActivityState	Indicates the current state of the activity instance. The values of this property are define by the enumeration *ActivityState* as follows: *activityState = (inProgress, completed, aborted).*

- *TaskNode* represents an instance of a task. Note that all mandatory and optional tasks share common attributes and behavior. Thus, they are both instantiated as *TaskNode*s. Table 3.3 represents the attributes of a $TaskNode$.

- *ArtifactNode* represents an instance of an artifact. Thanks to the information provided by the DBMS which manages company's data, the PDG's *ArtifactNodes* can be created and updated. In fact, PDG is connected to the DBMS in which always reflects the current states of artifacts as they defined in the model. Table 3.4 represents the attributes of an $ArtifactNode$.

- *ActorNode* represents a real process actor in the system. An actor may involve in several process instances. Moreover, an actor may play several roles which are generally defined in the organizational model of a company. Table 3.5 represents the attributes of an $ActorNode$.

Table 3.3: Properties of a $TaskNode$

Property	Description
reference: Task	Links to the corresponding element $Task$ in SPML.
taskState: TaskState	Indicates the current state of the task instance. The values of this property are define by the enumeration $TaskState$ as follows: *taskState = (created, inProgress, waiting, completed, aborted).*
startTime	Indicates the moment of starting executing a task.
endTime	Indicates the moment of finishing executing a task.
duration	Indicates the duration of executing a task which is derived from properties $endTime$ and $startTime$.

Table 3.4: Properties of an $ArtifactNode$

Property	Description
reference: Artifact	Links to the corresponding element $Artifact$ in SPML.
currentState: ArtifactState	Indicates the current state of the artifact instance. Its property *currentState* receives one possible value defined by the *demandedState* of its referenced *TaskParameter* in the SPML process model.

Table 3.5: Properties of an $ActorNode$

Property	Description
actorState: ResourceState	Indicates the current state of the process actor. The values of this property are define by the enumeration *ResourceState* as follows: *resourceState = (notAvailable, idle, active).*

Table 3.6: Properties of an $ResourceNode$

Property	Description
reference: Resource	Links to the corresponding element *Resource* in SPML.
resourceState: ResourceState	Indicates the current state of the resource instance. The values of this property are define by the enumeration *ResourceState* as follows: *resourceState = (notAvailable, idle, active).*

- *ResourceNode* represents an instance of a resource that an actor uses to perform a task which its attributes are defined in Table 3.6. Note that the procedure of PDG to manage resources (human and non-human) is completely similar to the artifacts. Thanks to the information provided by the RMS which manages company's resources (human and non-human), the PDG's *ActorNode*s and *ResourceNode*s can be created and updated. In fact, PDG is connected to the RMS which always reflects the current states of artifacts and actors as they exist in the RMS.

3.4.1.2 Edges of PDG

Edges in PDG are used to represent different types of relationships among process instance elements. Each type of edges along with their properties are defined as follows:

- *BelongEdge* represents the association between a process instance and its activities instances. In SPML, the dependency between a *Process* and an *Activity* is created by *ProcessFragment*. However, we remind that PDG represents the global view of the system where all instances of process fragments models become unified. Thus, as illustrated in Figure 3.3, we do not require a specific node to represent the element *ProcessFragment* of SPML.

- *ChildEdge* represents the association between an activity instance and its task instances. As illustrated in Figure 3.3, this edge is the mapping of composition relation defined in SPML between an *Activity* and a *Task* elements.

- *PerformEdge* represents the association between an activity instance and an actor who performs the activity and all its tasks. An actor may play several roles but he has only one role in each activity as defined in the SPML meta-model. We remind that the activities of a process fragment can be associated to only one role in SPML. However, at enactment-time, the respective role instance can be played by several process actors (i.e., *ActorNode*s). But each activity instance (i.e., *ActivityNode*) and all its task instances (i.e., *TaskNode*s) can be performed by only one process actor. Thus, as illustrated in Figure 3.3, we add the role as the property of relation (i.e., *PerformEdge*) between process actor and respective activity instance. Table 3.7 presents the properties of a *PerformEdge*.

Table 3.7: Properties of a *PerformEdge*

Property	**Description**
reference: ProcessFragmentPerformer	Links to the corresponding element *ProcessFragmentPerformer* in SPML.
role: Role	Indicates the role of process actor as defined in the model, i.e., computed by traversing from *ProcessFragmentPerformer* in the SPML.

- *DataEdge* represents the association between a task instance and its artifact instances whose attributes are defined in the Table 3.8. This edge is the direct mapping of *TaskParameter* concept in SPML into PDG.

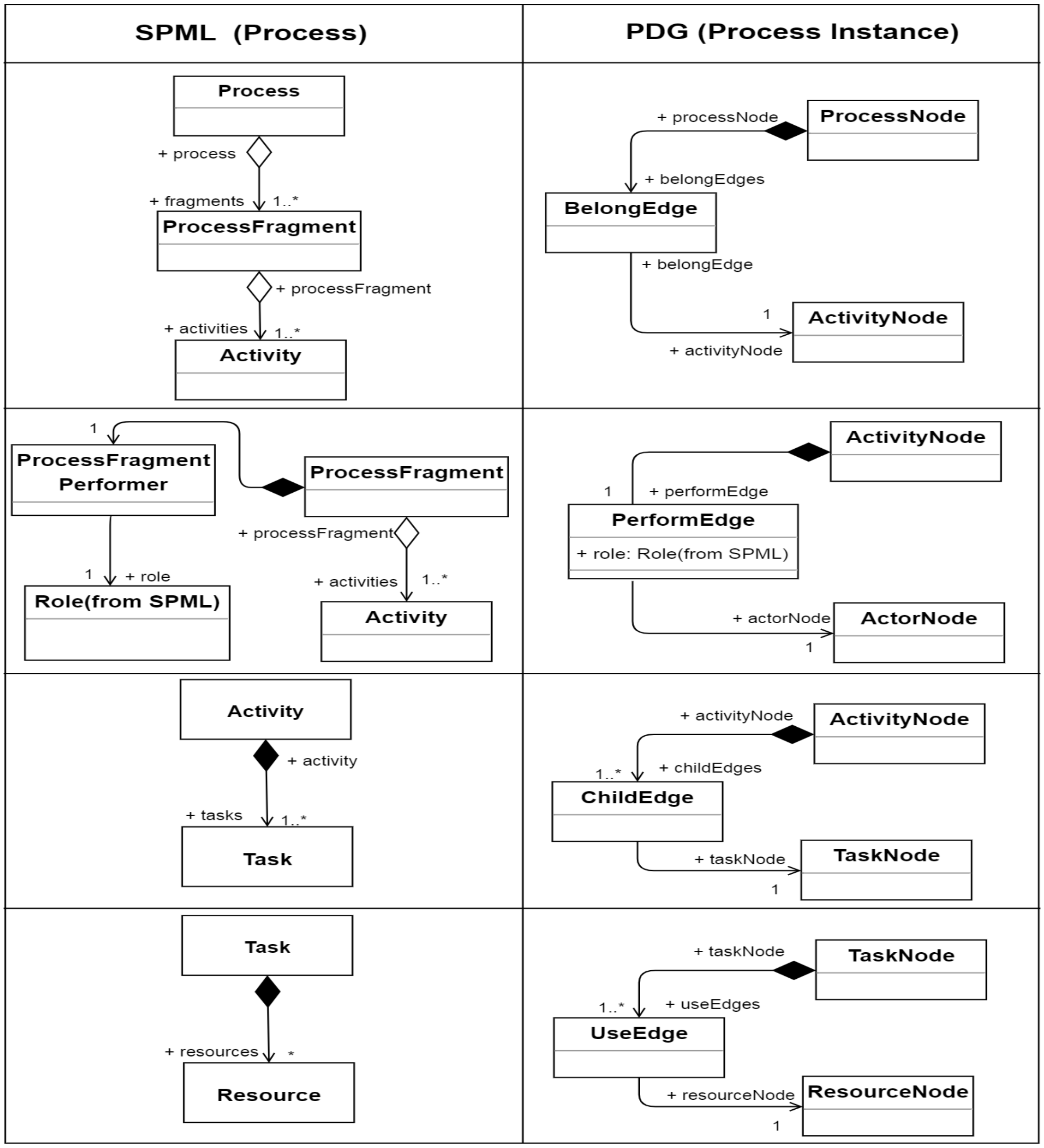


Figure 3.3: Indirect mappings between SPML and PDG

- *UseEdge* represents the association between a non-human resource and a task which uses the respective resource. As illustrated in Figure 3.3, this edge is the mapping of composition relation defined in SPML between a *Task* and a *Resource* elements.

Table 3.8: Properties of a *DataEdge*

Property	**Description**
reference: TaskParameter	Refers to the corresponding *TaskParameter* element in SPML. Thanks to this reference, a *DataEdge* can access to the *reference.demandedState* of the linked *ArtifactNode* to know in which state it is required or produced by the *TaskNode*, i.e., to know the pre- and post-conditions of a *TaskNode* defined in the model.
stateReady: StateReady	Indicates whether the artifact is currently *ready* or *waiting* for the $reference.demandedState$ so that the $TaskNode$ can use it.

3.4.2 PDG Example

Figure 3.4 provides an extraction of the PDG representing the system for a process instance P_1 instantiated from our examined process *Modify Testbench Wiring* of the previous chapter. We assume that several process actors are involved in the project by performing their activities/tasks. In this example, the process is initiated for the artifact *Wiring Change Demand* WCD_1. In this process instance, we assume that all activities are completed and two options *instrumentation* and *interface* are activated. The complete information of P_1 is presented in the Table 3.9.

The PDG shows the activities performed by each actor, and the tasks executed inside each activity. In fact, PDG presents the unification of all process fragments models instances. The dependency among tasks, both inside or among process fragments, are established based on their exchanged concrete artifacts and shared concrete resources. PDG represents these dependencies based on the concrete information arising at enactment-time. For example, if we consider process fragments instances of *Analyst* and *Instrumentation Team*, the task OIM_1, performed by the actor A_1, has produced the artifact $FICD_1$ in the state *outlined*; the task DIM_1, performed by the actor IT_1, has taken an *outlined* $FICD_1$ as input, then modifies it to produce the output $FICD_1$ in the state $defined$; the task DES_1 is in state *waiting* as it needs a $defined$ $FICD_1$ to finish. By combining two process fragments into one common structure, the PDG can establish a global view with following dependencies among two process fragments: OIM_1 has to finish so that DIM_1 can start and DIM_1 has to finish so that DES_1 can finish. This mechanism applies to all process fragments instances which result in a global process.

As mentioned earlier, the PDG presents the dynamic information; this information emerges progressively. In the next section, we present how PDG is progressively constructed by the process engine and how it provides the engine the necessary information to synchronize process activities/tasks inside and among process fragments instances.

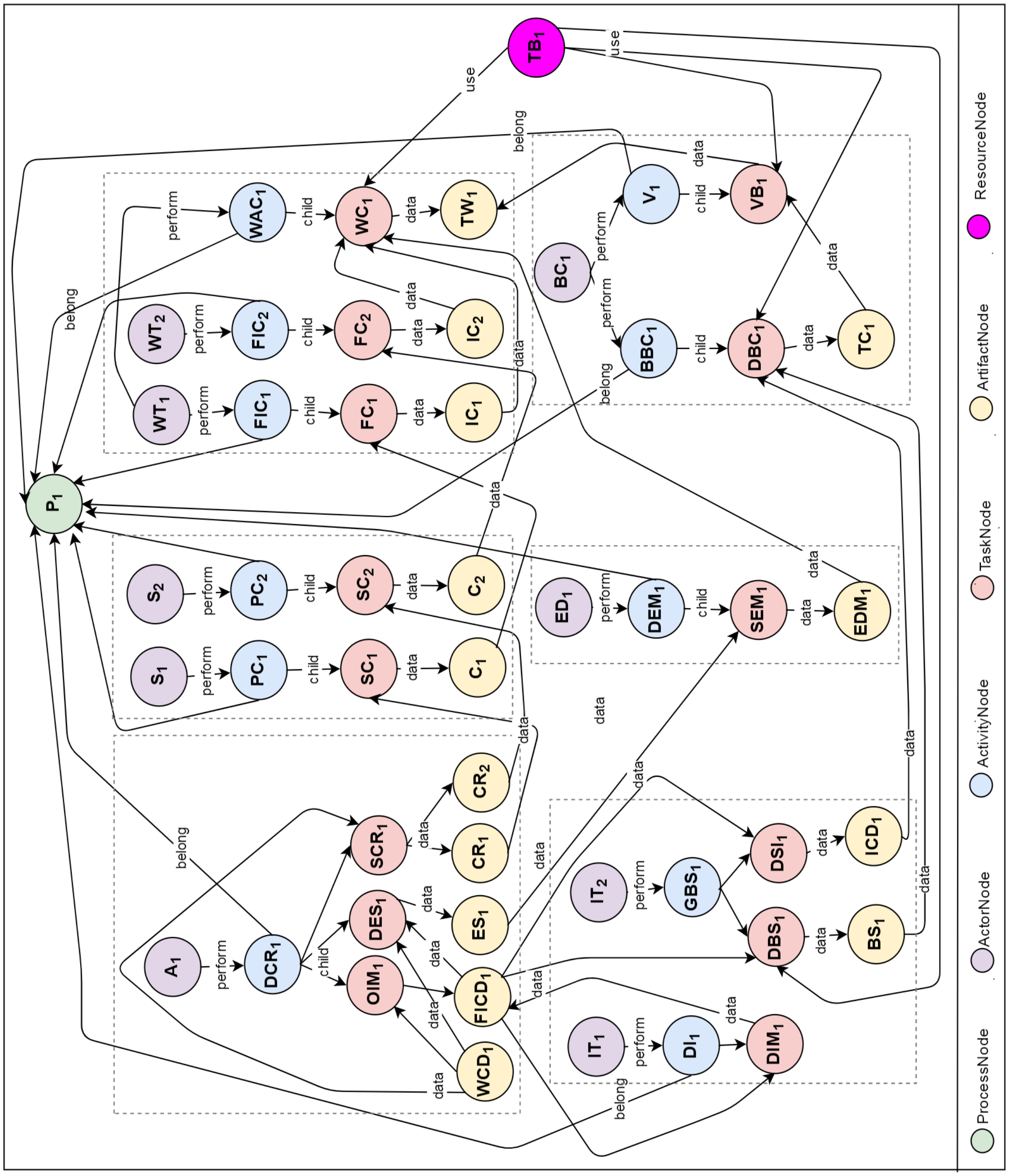


Figure 3.4: PDG representing the process fragments instances

Table 3.9: Information of P_1

Process Actors	**Non-human Resources**
Analyst = A_1 Instrumentation Team = $\{IT_1, IT_2\}$ Supplier = $\{S_1, S_2\}$ Wiring Team = $\{WT_1, WT_2\}$ Electrical Designer = ED_1 Bench Coordinator = BC_1	TestBench = TB_1
Artifacts	**Activate Options**
Wiring Change Demand=WCD_1 Component Requirement= $\{CR_1, CR_2\}$ Component= $\{C_1, C_2\}$ Installed Component= $\{IC_1, IC_2\}$ Testbench Wired=TW_1 FICD=$FICD_1$ Bench Specification=BS_1 ICD=ICD_1 Electrical Design Model=EDM_1 Testbench Configuration=TC_1	instrumentation interface
Activities	**Tasks**
Detail Change Requirement=DCR_1 Purchase Components=$\{PC_1, PC_2\}$ Fix & Install Components=$\{FIC_1, FIC_2\}$ Wire All Components=WAC_1 Define Instrumentation=DI_1 Generate Bench Specification=GBS_1 Define Electrical Model=DEM_1 Build Bench Configuration=BBC_1 Validate=V_1	Outline Instrumentation Model=OIM_1 FICD=$FICD_1$ Specify Component Requirement=SCR_1 Supply Component=$\{SC_1, SC_2\}$ Fix Component=$\{FC_1, FC_2\}$ Wire Components=WC_1 Define Instrumentation Model=DIM_1 Define Specific ICD=DSI_1 Specify Electrical Model=SEM_1 Define Bench Specfcation=DBS_1 Validate Bench=VB_1

3.5 Process Engine

Enactment of a process requires to define the precise operational semantics of each process element. Operational semantics accurately prescribes how a proper SPML process engine will realize the operation of process model elements at enactment-time. In SPML process models, we have five elements which need their operational semantics to be defined so that process engine can manage the instances of these elements. These elements are *Process, Activity, Task, Artifact* and *Resource*. Notice that the *Process, Activity* and *Task* elements are directly created and managed by the process engine.

Artifact and *Resource* elements are normally managed by the external systems, i.e., respectively Database Management System (DBMS) and Resource Management System (RMS). We assume that our process engine is connected with these systems and is aware of those events changing the state of these elements in the external systems. Our process engine does not have a right to manipulate the state of artifacts and resources.

In BAPE, we define a life cycle for each instance of operational elements directly managed by the process engine, i.e., *Process, Activity* and *Task* instances. These life cycles generally characterize the operational semantics of process elements instances. The conforming process engine launches process elements instances and controls their internal states precisely as they are described by their life cycles. These life cycles are defined by state machines comprising a set of possible states and transitions. Each transition may have a trigger, a guard and an effect, as below:

- *Trigger* is the cause of the transition, which could be an event. In BAPE, events are triggered by process actors from their Modeling/Enactment UIs or external systems (e.g., DBMS and RMS).

- *Guard* is a condition which must be true in order for the trigger to cause the transition. In BAPE, all conditions are verified in the PDG by the process engine to enable or refuse the transition.

- *Effect* is an action which will be invoked directly on the process element that owns the state machine as a result of the transition. In BAPE, all actions are taken by the process engine which result in creating and updating the PDG nodes and edges.

In the next we describe the life cycle of each process element and illustrate how these life cycles define the overall behavior of process engine in enacting and synchronizing the structurally defined process fragment models which results in progressively updating the PDG. The operational semantics of SPML is given by the state machines defined for each executable SPML element. The actions of these state machines which update the PDG are defined by algorithms implemented in Java.

3.5.1 Process Instance Life cycle

In SPML, a process is composed of several process fragments which each fragment comprises several activities. This fact makes the life cycles of a process and an activity to be bound with each other, i.e., a process can only be completed if all its activities are completed or an activity can only be enacted if its process is running as well. We propose a generic state machine to define the life cycle of a process which is illustrated in the Figure 3.5. The *completed* and *aborted* states are end states in which processes cannot be performed.

In many cases it will be useful to store information about the execution of process instances, so that this information can be used afterwards. For example, sometimes we want to verify which actions have been executed for a particular process instance, or in general, we want to be able to monitor and analyze the efficiency of a particular process. For this reason, all completed and aborted processes are kept in database until the issue of *DiscardProcess* event, i.e., manual deletion. We apply the same procedure for activity and task instances. The detail description of main transitions of the process state machine are defined as follows:

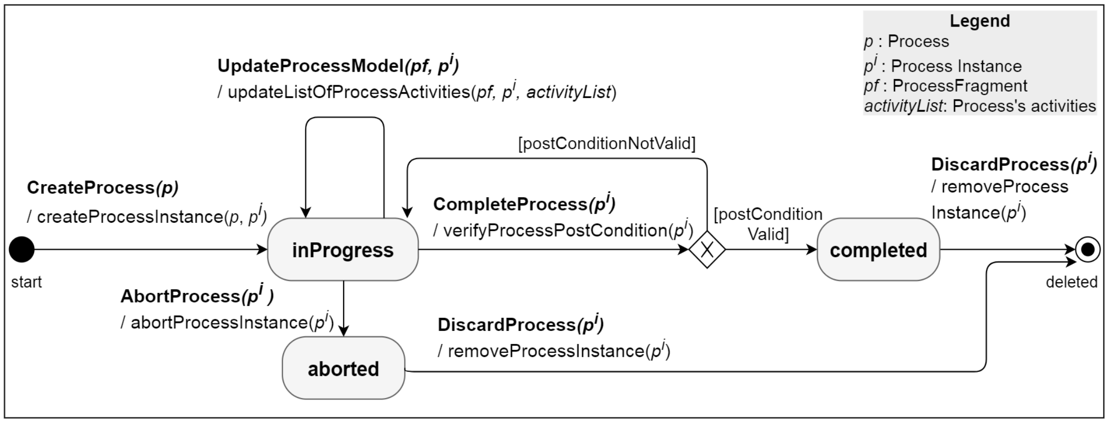

Legend
p : Process
p^i : Process Instance
pf : ProcessFragment
activityList: Process's activities
UpdateProcessModel(pf, p^i)
/ updateListOfProcessActivities(pf, p^i, activityList)
[postConditionNotValid]
CreateProcess(p)
/ createProcessInstance(p, p^i)
CompleteProcess(p^i)
/ verifyProcessPostCondition(p^i)
[postCondition Valid]
DiscardProcess(p^i)
/ removeProcess Instance(p^i)
inProgress
completed
start
deleted
AbortProcess(p^i)
/ abortProcessInstance(p^i)
DiscardProcess(p^i)
/ removeProcessInstance(p^i)
aborted

Figure 3.5: Process State Machine

- **CreateProcess** A process p is instantiated/enacted via triggering the event *CreateProcess*, e.g., event can be triggered by process owners. An event *CreateProcess* has a parameter p with type *Process*. As the effect of this transition, process engine takes an action *createProcessInstance* which itself constitutes two sub-actions:

    1. *createProcessNode* Process engine creates p^i as the instance of process p. This is done by creating the corresponding PDG *ProcessNode* with a state *inProgress*. The Figure 3.6 shows the PDG as a result of the corresponding action regarding the enactment of our examined process *Modify TestBench Wiring (MTW)* which results in a process instance MTW_1 in state $inProgress$.

    2. *getListOfProcessActivities* Process engine looks up the process repository to extract the list of activities of p from the process model which later can be enacted by process actors.

- **CompleteProcess** A process instance p^i can be completed via triggering the event *CompleteProcess*. To enable the transition, a post-condition is defined which specifies that a process instance p^i can be completed if and only if all activities of the process p are enacted and completed. As the guard of this transition, process

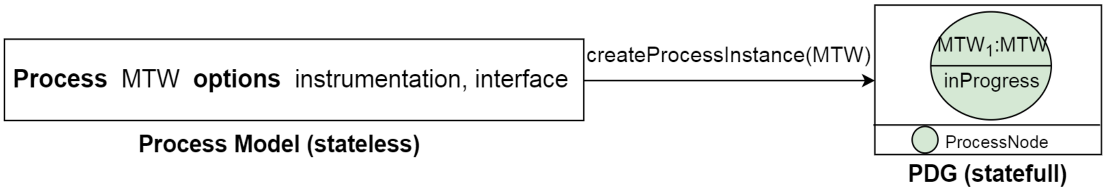

Process MTW options instrumentation, interface
Process Model (stateless)
createProcessInstance(MTW)
MTW1:MTW
inProgress
ProcessNode
PDG (statefull)

Figure 3.6: PDG as a result of creating process instance

engine verifies the process post-condition by first getting list of all activities of the process *p* stored in the process repository. Then, process engine verifies inside the PDG whether the respective activities are instantiated or not. Normally, two situations may exist:

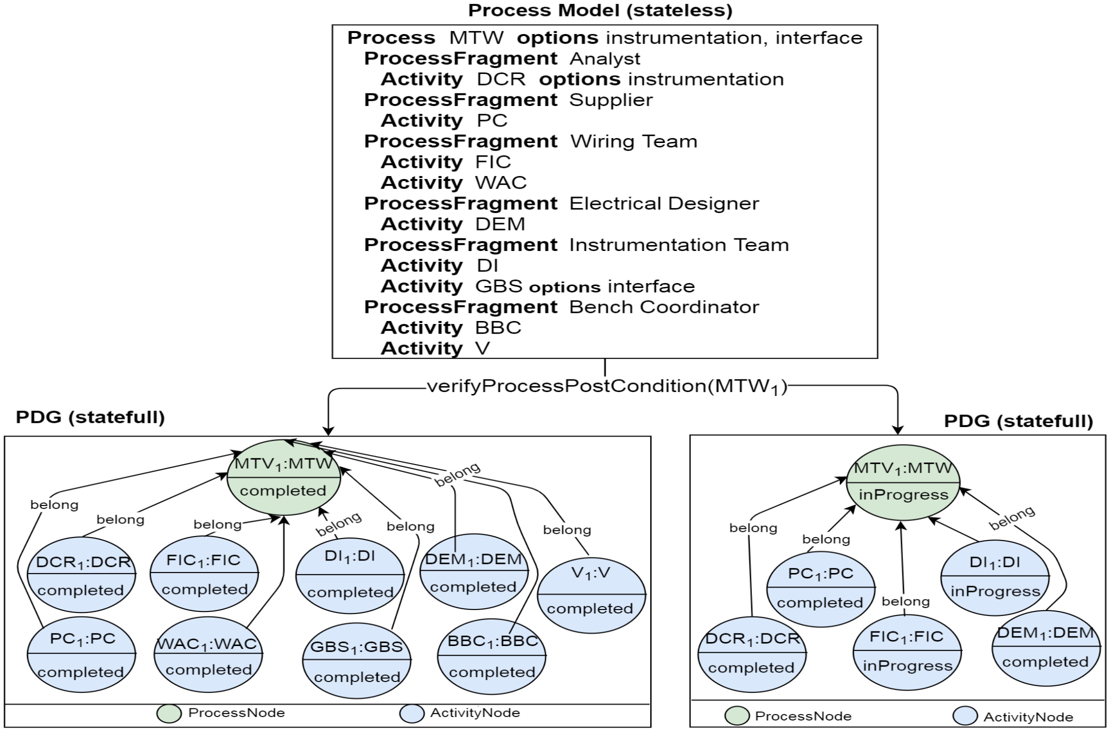
1. All process activities defined in the model are instantiated and completed. In this case in which the post-condition is met, process engine updates the state of the corresponding PDG *ProcessNode* p^i to *completed*.
2. All process activities defined in the model are not instantiated or are instantiated but they are not completed yet. In this case in which the post-condition is not met, process engine keeps the state of the corresponding PDG *ProcessNode* p^i in *inProgress*.

The Figure 3.7 shows two mentioned situations regarding activities of process instance MTW_1.

- **UpdateProcessModel** In our approach, processes are composed of separate fragments which can be progressively modeled during process execution, i.e., P^i in state $inProgress$. Thus, whenever a process actor adds a new fragment into the process repository, the process engine invokes the event *ProcessUpdate* of P^i which has a new process fragment pf as parameter. As the effect of this transition, process engine updates the list of process's activities including the activities of a new added fragment.

3.5.2 Activity Instance Life cycle

While the process instance p^i is running, activities of p^i which are defined in different process fragments can be enacted. In SPML, an activity is composed of tasks, i.e., mandatory and optional tasks. This encapsulation binds the life cycles of an activity and its tasks together, i.e., an activity can be completed if all its mandatory tasks and required optional tasks are completed or a task can be enacted if its activity is running as well. Each process actor can enact activities of process corresponding to his role. The life cycle of an activity is defined by a state machine as illustrated in Figure 3.8.



(a) Valid process post-condition (b) Invalid process post-condition

Figure 3.7: PDGs as results of completing process instance

The detail description of main transitions of the activity state machine are described as follows:

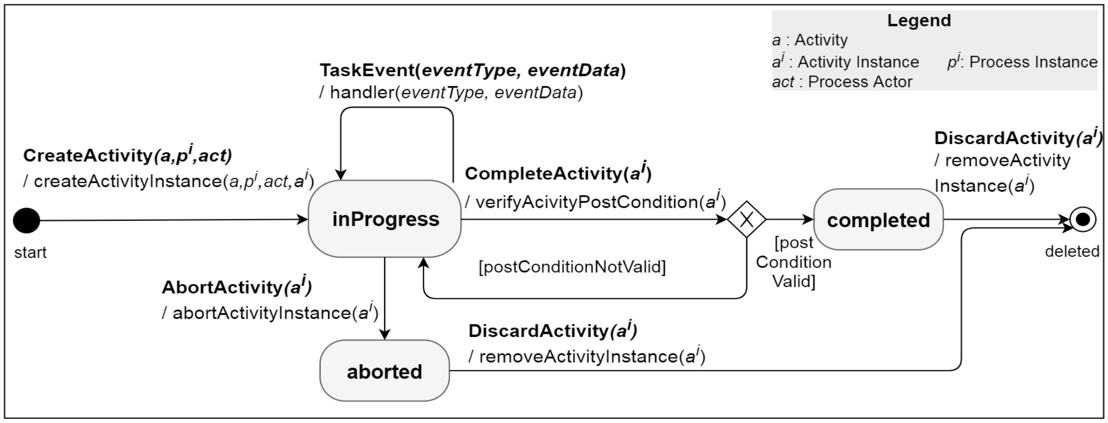

Legend
a : Activity
a^i : Activity Instance
p^i: Process Instance
act : Process Actor
TaskEvent(eventType, eventData)
/ handler(eventType, eventData)
CreateActivity(a,p^i,act)
/ createActivityInstance(a,p^i,act,a^i)
start
inProgress
CompleteActivity(a^i)
/ verifyAcivityPostCondition(a^i)
completed
DiscardActivity(a^i)
/ removeActivity Instance(a^i)
deleted
[postConditionNotValid]
[post Condition Valid]
AbortActivity(a^i)
/ abortActivityInstance(a^i)
aborted
DiscardActivity(a^i)
/ removeActivityInstance(a^i)

Figure 3.8: Activity State Machine

- **CreateActivity** An activity a of a running process p^i is instantiated via triggering the event *CreateActivity*. After the creation of an activity, the process actor can enact the activity's mandatory and required optional tasks. An event *CreateActivity* has three input parameters as:

  - p^i is the process instance,
  - a is an activity of process p extracted from the process model,
  - act is the process actor who enacts the activity a.

  As the effect of this transition, process engine takes an action *createActivityInstance* comprising two sub-actions as:

  1. *createActivityNode* Process engine creates an instance a^i of an activity a. This is done by creating the corresponding PDG *ActivityNode* a^i with a state *inProgress*. Then, process engine establishes a PDG *BelongEdge* between the corresponding PDG *ActivityNode* a^i and the PDG *ProcessNode* p^i. Finally, process engine establishes a PDG *PerformEdge* between the corresponding *ActivityNode* a^i and the *ActorNode* *act* along with assigning the role of process actor as the property of the PDG *BelongEdge*. Remember that a process actor performs the activity and all its tasks. The Figure 3.9 shows the PDG as a result of this transition regarding the enactment of activity *Detail Change Requirement (DCR)* by *Analyst* A_1.

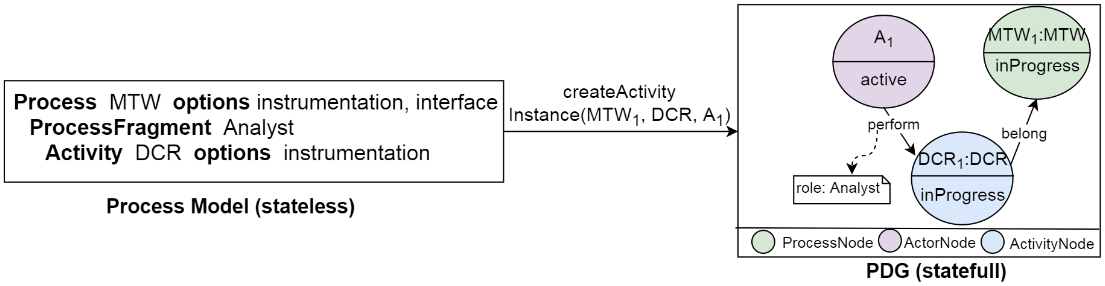

Process MTW options instrumentation, interface
ProcessFragment Analyst
Activity DCR options instrumentation
Process Model (stateless)
createActivity Instance(MTW1, DCR, A1)
A1
active
MTW1:MTW
inProgress
perform
belong
role: Analyst
DCR1:DCR
inProgress
ProcessNode
ActorNode
ActivityNode
PDG (statefull)

Figure 3.9: PDG as a result of creating an activity instance

  2. *getListOfActivityTasks* Process engine extracts the list of activity's tasks from the process model which later can be enacted by process actors.

- **CompleteActivity** A process actor can complete an activity instance a^i by triggering the event *CompleteActivity*. To enable the transition, a post-condition is verified by the process engine which is regarding the tasks of the activities extracted from the model. An activity instance a^i is completed if and only if the following post-condition holds:

1. all mandatory tasks of $a^i \rightarrow reference \rightarrow tasks$ are enacted and completed,
2. if an option o is activated in the activity, the optional tasks corresponding to o in $a^i \rightarrow reference \rightarrow tasks$ must be completed as well.

As the guard of this transition, process engine verifies the activity post-condition by traversing the PDG *ActivityNode* a^i through all *ChildEdges* to verify the state of its mandatory and activated optional tasks instances. If all respective task instances are completed, process engine updates the state of the corresponding PDG *ActivityNode* a^i to *completed*. If not, process engine keeps the state of PDG *ActivityNode* a^i as *inProgress*. Figure 3.10 shows PDGs in two cases: (a) all mandatory and required optional tasks of activity instance *Detail Change Requirement* DCR_1 are completed; (b) an optional task *Outline Instrumentation Model* OIM_1 corresponds to option *instrumentation* is still running.

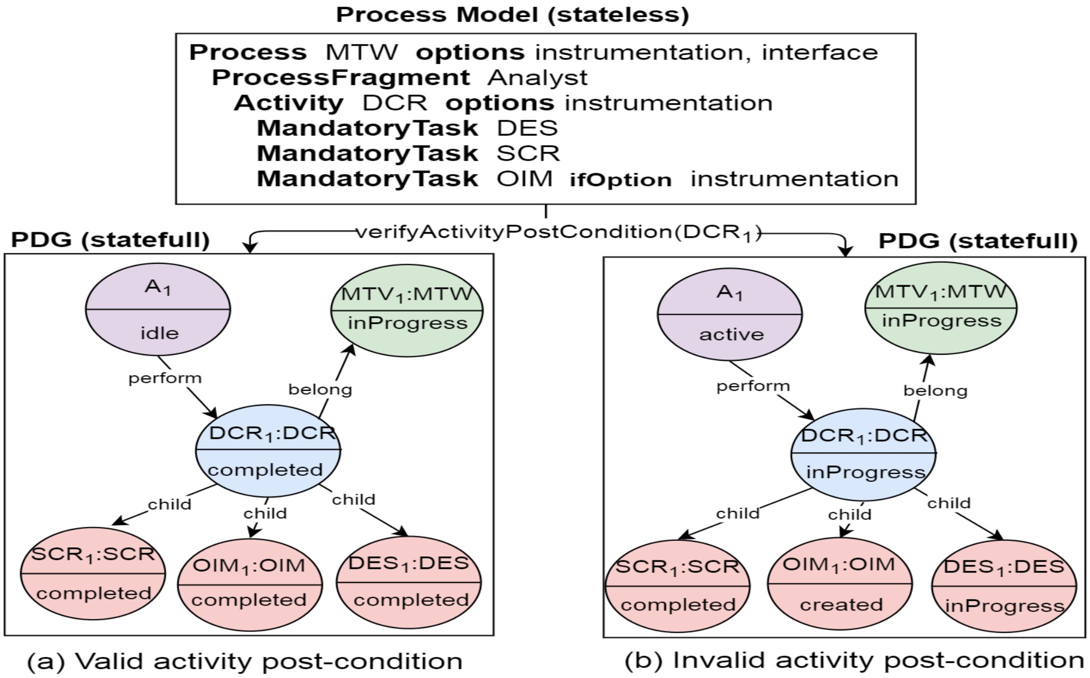


Figure 3.10: PDGs as results of completing an activity instance

- **TaskEvent** While an activity is running (i.e., in state *inProgress*), it listens to events related to execution of its tasks. As the effect of this transition, for each type of event, process engine triggers the appropriate handler. This solution facilitates different execution configurations of an activity which is discussed in details in the Section 3.6.1.

3.5.3 Task Instance Life cycle

An activity is comprised of mandatory and optional tasks. When a process actor enacts an activity, he performs the activity's tasks including all mandatory and required optional tasks. Each task is associated with artifacts. A task consumes artifacts and is meant to produce an artifact. Tasks are the real actions that change artifacts' state and progress the process's execution. Similarly to conventional process engines, we adopt a state machine inspired from the standard WS-HumanTask Specification [22] to define the life cycle of any task managed by our process engine. Figure 3.11 gives our proposed task state machine.

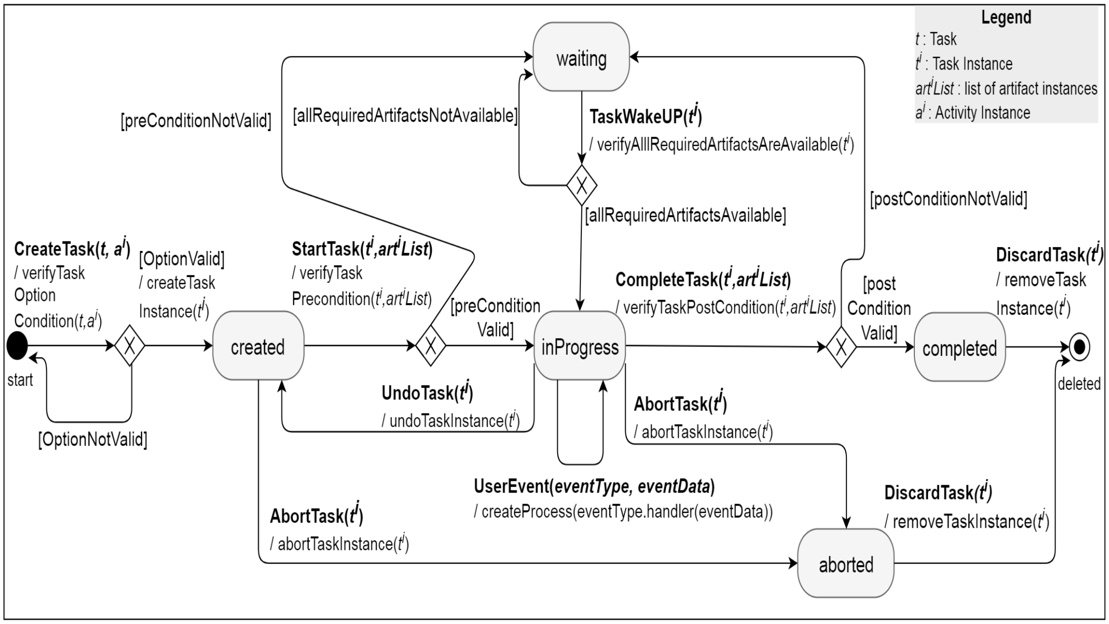

Legend
t : Task
t^i : Task Instance
art^iList : list of artifact instances
a^i : Activity Instance
waiting
TaskWakeUP(t^i)
/ verifyAlllRequiredArtifactsAreAvailable(t^i)
[preConditionNotValid]
[allRequiredArtifactsNotAvailable]
X
[allRequiredArtifactsAvailable]
[postConditionNotValid]
CreateTask(t, a^i)
/ verifyTask Option Condition(t,a^i)
[OptionValid]
/ createTask Instance(t^i)
StartTask(t^i,art^iList)
/ verifyTask Precondition(t^i,art^iList)
[preCondition Valid]
CompleteTask(t^i,art^iList)
/ verifyTaskPostCondition(t^i,art^iList)
[post Condition Valid]
DiscardTask(t^i)
/ removeTask Instance(t^i)
start
X
created
X
inProgress
X
completed
deleted
[OptionNotValid]
UndoTask(t^i)
/ undoTaskInstance(t^i)
AbortTask(t^i)
/ abortTaskInstance(t^i)
UserEvent(eventType, eventData)
/ createProcess(eventType.handler(eventData))
DiscardTask(t^i)
/ removeTaskInstance(t^i)
AbortTask(t^i)
/ abortTaskInstance(t^i)
aborted

Figure 3.11: Task State Machine

As tasks are bound to artifacts, the life cycle of a task is highly dependent on artifacts and their states. Artifacts are shared among tasks in the way that an artifact in a specific state can be consumed by several tasks. However, tasks can not collaborate to produce an artifact in a specific state, i.e., no sharing on artifact production. As mentioned earlier, we assume that artifacts are managed by DBMS. Process actors directly interact with DBMS in order to create or update their artifacts. To enable the coordination between DBMS and our process environment, our process engine continuously listen to any events related to creating or updating the artifacts in the DBMS and takes the appropriate actions in order to make PDG always reflects the current state of artifacts. The corresponding events are defined as follows:

- *CreateArtifact* Whenever an artifact instance art^i with state s_1 is created into the DBMS, e.g., by process actors or external systems/agents, DBMS informs the process engine of the new artifact instance art^i. As a result, process engine creates the corresponding *ArtifactNode* art^i with state s_1 in the PDG.
- *UpdateArtifact* Whenever an artifact instance art^i is updated into the DBMS, i.e., by changing its state s_1 to s_2, DBMS informs the process engine of the art^i's state change. Consequently, process engine takes two actions as:
  1. *updateArtifactInstance* Process engine updates the state of corresponding PDG *ArtifactNode* art^i to s_2.
  2. *findWaitingTaskInstances* Process engine looks for those task instances which are waiting for the respective artifact in the state s_2. This procedure of engine is explained in detail where we present the transitions of task sate machine.

The detail descriptions of main transitions of the task state machine are provided as follows:

- **CreateTask** A task t (mandatory or optional) of an activity instance a^i is enacted via triggering the event *CreateTask*. An event *CreateTask* has two parameters as:
  - a^i is the active activity instance that wants to execute the task,
  - t is a task in the list of possible tasks of a^i to be executed.

  To enable the transition, a condition is verified by the process engine and is hold if either:

  - t is a mandatory task,
  - t is an optional task and the option o corresponding to t is activated in the process instance.

  Based on the result of the verification, two situations may happen:

  - If the condition is satisfied, process engine creates t^i as the instance of task t. This is done by creating the corresponding PDG *TaskNode* t^i with a state *created*. Additionally, process engine establishes a *ChildEdge* between the corresponding *TaskNode* t^i and the *ActivityNode* a^i.
  - If the condition is not satisfied, process engine rejects the enactment of the corresponding task, i.e., optional task.

  The Figure 3.12 shows the PDG regarding the enactment of a mandatory task *Define Electrical Specification (DES)* by *Analyst* A_1.

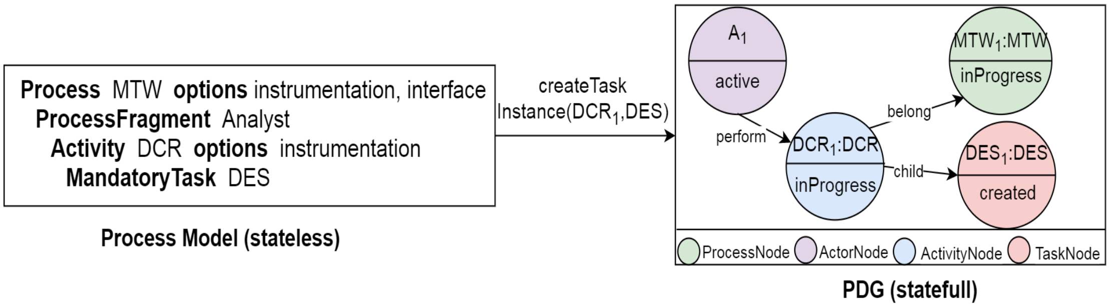


Figure 3.12: PDG as result of creating task instance

Note that in conventional process engines, the condition of a transition between task's states is principally based on work-sequence relations defined among tasks. For example, if the tasks A and B are related by a relation $finishToStart$ [1], the task B can only start its execution if the task A is completed. This solution cannot be applied to our process engine as we do not describe the work-sequence relations among tasks in process models. Instead of work-sequence relations, our process engine deduces the dependencies among tasks from real enactment circumstances and uses artifacts as the main source of synchronization to enable the task state transitions. In our task state machine, the conditions of transitions as starting a task, completing a task and waking up a task are defined based on the state of artifacts. In the next, we define each of these transitions in details.

- **StartTask** A process actor can start the execution of his task instance t^i by triggering the event *StartTask*. Concretely, an event *StartTask* entails two parameters as:

  1. t^i is the task instance.
  2. $art^i List$ is the list of artifacts instances given by process actor. These artifacts should fulfil following criteria as defined in the model:
     • All input artifacts having the property *usage* set to *toStart*.
     • The input artifact, which the property *usage* set to *toStart*, associated with an option and the corresponding option has been activated in the process instance.
     With respect to the mentioned criteria, process engine selects required artifacts from the model and demands the process actor to give the name of concrete artifact (i.e., artifact instance) for each selected artifact.

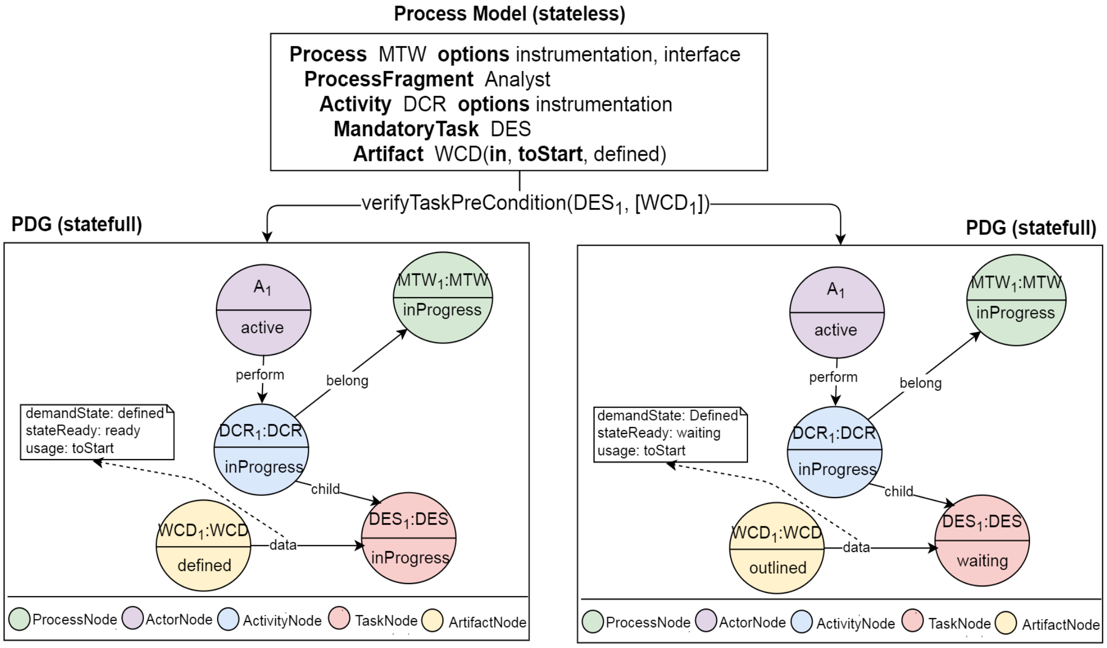
  After triggering the event *StartTask* including its effective parameters, a precondition is defined which specifies the artifacts instances art^i given by the process

[1] means that there is a work-sequence relation from A to B

actor, in a specific state s, required to start the task instance t^i. As the guard of this transition, process engine verifies the pre-condition of a task instance by first finding the corresponding artifacts instances in the PDG and checking if the states of the concerned PDG's *ArtifactNodes* are the required ones. Normally two situations may arise:

1. If the pre-condition is satisfied, the process engine updates the task's state from *created* to *inProgress* and establishes a *DataEdge* with the property *ready* between the required PDG *ArtifactNode* art^i and the corresponding PDG *TaskNode* t^i.
2. If not, the task will be put in *waiting* state to wait until the required input artifact is available. As a result, in the PDG, a *DataEdge* with the property *waiting* is established between the required *ArtifactNode* art^i and the waiting *TaskNode* t^i so that the process engine can keep trace of the waiting tasks and wake up them later when the needed artifact's state is set.

Figure 3.13 illustrates the PDGs regarding two situations where process actor A_1 starts his task DES_1 by selecting artifact WCD_1 in state $defined$.

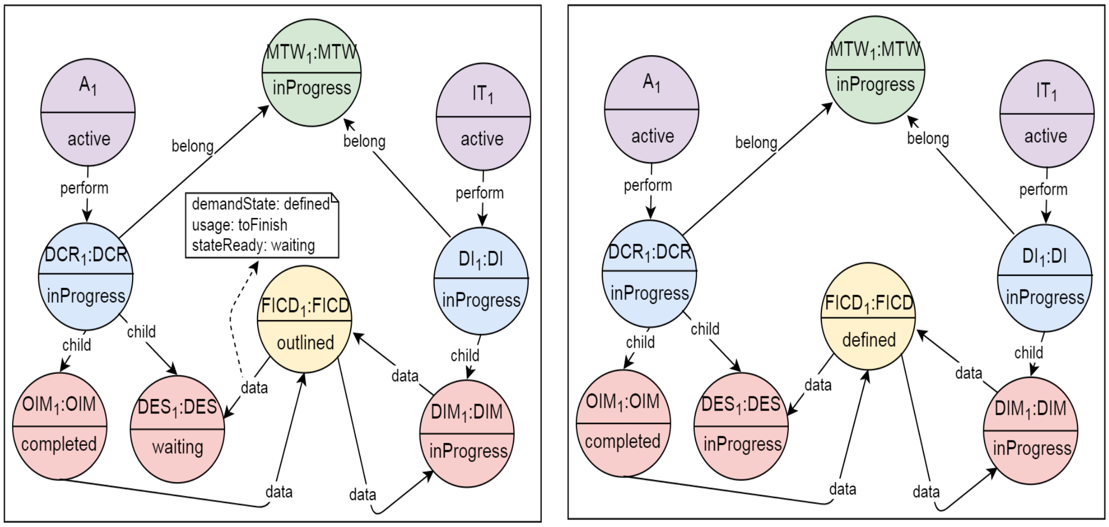


(a) Valid task pre-condition

(b) Invalid task pre-condition

Figure 3.13: PDGs as results of starting task instance

- **TaskWakeUP** As mentioned earlier, when a pre- or a post-condition of a task is not satisfied, process engine puts the task in the *waiting* state. The condition to leave the *waiting* state is related to availability of artifacts in the demanded state. Thus, when an artifact instance is updated in the PDG, i.e., by changing its state, process engine looks for the tasks which are waiting for the corresponding artifact instance with the new state. Thanks to the *dataEdge*s together with their properties (e.g., *demandedState*) established between waiting *TaskNode*s and *ArtifactNode*s, process engine can find *TaskNodes* which are waiting for the respective artifact in the required state. Afterwards, process engine triggers the *TaskWakeUp* event of the corresponding task instances as specified in their state machines. To enable the transition, the process engine verifies whether all task's required artifacts are in demanded states or not.
The Figure 3.14 illustrates part of PDG regarding two fragments of *Analyst* and *Instrumentation Team* regarding two situations: In Figure 3.14(a), task OIM_1, performed by the actor A_1, has produced the artifact $FICD_1$ in the state *outlined*; the task DIM_1, performed by the actor IT_1, has taken an *outlined* $FICD_1$ as input then modifies it to produce the output $FICD_1$ in the state $defined$; the task DES_1 is in state *waiting* as it needs a $defined$ $FICD_1$ to finish. Figure 3.14 (b) illustrates the moment when task OIM_1 has produced the $defined$ $FICD_1$ in which process engine wakes up the task DES_1.



(a) Required Artifacts are not available

(b) Required Artifacts are available

Figure 3.14: PDGs showing results of task wake up

- **UserEvent** This transition does not change the state of the task instance. However, it gives a possibility to the process actor who performs the task instance to handle some specific user defined events emerging during the task execution. For each type of user events, process engine executes a pre-defined process comprising the event handler. This feature offers a simple mechanism to extend the functionalities of the process engine to better response to process actors' needs. In Section 4, we report an application of such feature to enable process actor to signal their change requests.

3.5.4 Discussion

By using the conditions based on the artifact states to decide the progress of tasks executions, our process engine enables the enactment of structurally-defined process fragments models defined by our modeling language SPML. The behavioral aspect of process is deduced at enactment-time based on artifact exchanges among processes' tasks. This leads to progressive unification of different process fragments inside PDG as process actors enact their process activities/tasks. Thanks to the suppression of work-sequence relations, our process engine allows activities and tasks to be enacted several times if needed. Overall, our approach gives more autonomy to process actors in controlling their activities.

By considering artifacts as a source of synchronization, our process engine better integrates data into tasks life cycles and thus provides a finer control on the execution of tasks. While conventional process engines rely exclusively on the process actors to make progress a task execution, our process environment ensures that the execution of a task makes progress the concerned artifacts as required in the process models.

Notice that resources (e.g., process actors, machines, etc.) can be considered as another source of synchonization as well. The synchronization mechanism of process engine is completey similar to the artifact one, i.e., conditions based on availability of resources. However, the focus of our work is on artifacts which are significant in process actors daily activities [14, 20, 15, 11].

Thus far, we have presented how our artifact-driven enactment approach provides control over processes defined by our language SPML. We described the behavior of the process engine in enacting and synchronizing the tasks which result in progressively updating the PDG. As an advantage of our enactment approach we can mention offered flexibilities for process actors which are presented in the next section.

3.6 Enactment Flexibilities

Due to the emerging situations occurring at enactment-time, a process cannot always be executed as modeled. Therefore, a process environment must provide some degree of flexibility to deal with these unforeseen situations while preserving the control over processes [25, 28].

The conventional enactment approaches following activity-driven execution mechanism, tend to use the work-sequence relations in process models to explicitly specify the execution order. Thus, the process execution, based on the work-sequence relations prescribed in process model, is too strict in many cases thus cannot let process actors dealing with emergent situations. On the contrary, by keeping constraints on exchanged artifacts, our approach can have more balance between control and flexibility in enacting processes. We want to emphasize two types of flexibilities offered by our enactment solution: *Loose Inter-Tasks Execution Order* and *Execution of Partially Modeled Processes (Partial Modeling)*. In the next, we discuss each of these flexibilities in details.

3.6.1 Loose Inter-Tasks Execution Order

In SPML, tasks are encapsulated into activities. This abstraction allows an activity to separately manage the execution of its tasks. Moreover, without prescribed work-sequence relations specifying the order for executing the tasks inside an activity, tasks can be progressively and freely instantiated in order to support different execution situations, especially rework situations. The key idea is giving more liberty to process actors so that they can decide how to handle different situations emerging during the enactment-time. For this purpose, we defined our activity state machine with a loop transition on its *inProgress* state to allow an activity catching different events triggered during its execution by the activity's performer or by external systems.

In the first time, we handle specifically events causing activation of an option. This transition is regarding the activation of an option via triggering event *ActivateOption*, i.e., *TaskEvent* having *eventType* as *ActivateOption* and *eventData* as Option. Figure 3.15 illustrates respective part of the activity state machine.

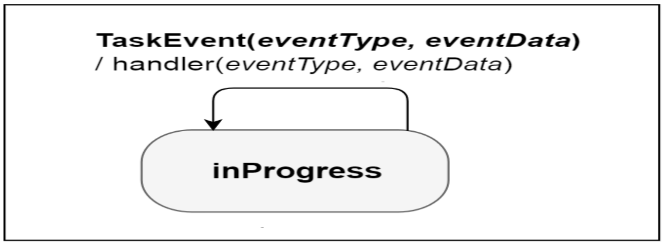

TaskEvent(eventType, eventData)
/ handler(eventType, eventData)
inProgress

Figure 3.15: Task event transition of activity state machine

As described in the SPML, an optional task is associated with an option and must be enacted if the corresponding option is activated. The activation of an option enables the execution of an optional task (i.e. process actors create a new task instance). The activation of a new option may cause a reconfiguration of other tasks, i.e., activation of an option makes another task to need an extra artifact. The information required by the process engine to reconfigure the task conditions are defined in the model by the property *activatingCondition* of each *TaskParameter*.

The actions taken by the process engine for the reconfiguration depends on (1) the state of reconfigured task (2) pre- or post-conditions of the reconfigured task. We distinguish three cases among which the activation of an option requires:

- Reconfiguration of a task's pre-condition which is in state *created*.
- Reconfiguration of a task's post-condition which is in state *created*.
- Reconfiguration of a task's post-condition which is in state *inProgress*.

The above-mentioned cases are supported by the process engine through the *Task Start* and *Task Completion* transitions specified in the task state machine. Remember that regarding starting or completing a task, process engine considers the associated option when extracting the required artifacts from the model.

There exist three other cases in which the activation of an option requires reconfiguring the pre-condition of an *inProgress* task, reconfiguring the pre-condition of a *completed* task or reconfiguring the post-condition of a *completed* task should be also considered by the process engine.

In the first case, a task is in state *inProgress* which has been started with pre-conditions that are no longer valid due to the activation of a new option. To handle this situation, process engine undoes the configured task. Undoing a task leads to moving control to the moment before the execution of a task i.e., changing the state of task from *inProgress* to *created* as specified via event *UndoTask* in the task state machine. The process actor has to re-start the task in which process engine can verify the task conditions with the new pre-condition. Undoing a task is only possible if the corresponding task has not been produced its output artifact and can cause the lost of modifications done on the output artifact.

The process engine handles differently if the reconfigured task is in the state *completed* or is in the state *inProgress* which the output artifact is produced. This mean that, the task has produced an artifact may no longer be valid due to the activation of an option, i.e., the task may require some extra input artifacts to correctly produce it output artifact corresponding to the activated option. In such situation, activating a new option implies to rework a completed task in order to produce the new version of the artifact. The rework of a task is regarding creating a new instance of the task. Thanks to the suppression of work-sequence relations, our process engine allows activities and tasks to be enacted several times if needed.

Regarding the multi-instance tasks support, process actors can easily enact their activities and tasks and start their tasks with requiring and producing different concrete artifacts, i.e., elements of collection artifacts. However, we give control to process actors in order to determine the number of their instances.

The rework of a task should be thoroughly considered as it implies producing a new version of the artifact. As the corresponding artifact may have already being used by other process actors, producing the new version of the artifact may cause major impacts on affected people. This issue first requires that the impact of corresponding change has to be evaluated. If the change is accepted the re-configured task can be reworked. This relates to the process of change management which provides better control over rework and will be discussed in the Chapter 4 in details.

To conclude, by enabling loose inter-task execution order, process engine provides more flexibility in enactment of tasks. Therefore, process actors instead of explicitly following the ordering of their model tasks as in conventional process environments, are able to enact their tasks in different execution orders in response to different contexts or unforeseen situations.

3.6.2 Partial Modeling

Partial modeling is the flexibility issue provided by our solution which is about enabling enactment of partially-modeled processes where some fragments are missing. While conventional process environments require the whole process to be defined and instantiated to enable its enactment, our environment can enact a process with only the defined process fragments that are managed by the system.

In our environment as mentioned earlier, artifacts are created/updated by process actors or external agents/systems directly into the central DBMS. Our process engine is aware of these events and keeps the PDG to be updated with any artifact changes in the DBMS. In fact, PDG keeps the global image of artifacts existing in the DBMS. This is done by the process engine which listens to the events related to creating or updating the artifacts in the DBMS and takes the appropriate actions to create or update the state of respective artifact nodes in PDG.

Thanks to this organization, we can enact and synchronize the managed fragments by the process environment with those that are performed by external agents. The reason is that synchronization information is based on the exchanged artifacts, which are surely managed by the DBMS. This fact enables external agents/systems to participate into the process without explicitly using our process environment but being under control of the process environment.

3.6.3 Examples of Enactment Flexibilities

To better illustrate the discussed flexibilities of our enactment approach, we consider our examined process *Modify Tesbench Wiring*. Figure 3.16 illustrates the whole procedure of our process environment for enacting and synchronizing the activities/tasks of *Analyst* and *Instrumentation Team* process fragments models in the process instance *Modify Tesbench Wiring* MTW_1. We suppose that the fragment of *Analyst* is defined and performed by A_1. Moreover, due to the organizational changes, the activities of

Instrumentation Team are performed by an external subsidiary who does not interact with the process environment, but directly interacts with the central DBMS.

To illustrate the loose inter-task execution order in our enactment approach, the tasks of the activity *Detail Change Requirement* DCR can be executed in any order. For instance the *Analyst* A_1 creates first instances of mandatory tasks *Specify Component Requirements* SCR and *Define Electrical Specification* DES. Then by realizing the need of *instrumentation*, he activates the option *instrumentation* which necessitates him to create an optional task *Outline Instrumentation Model* OIM, i.e., to apply modification on *instrumentation* component. The creation of task OIM is done regardless of other task instances of its activity *Detail Change Requirement* DCR_1. Moreover, once the option *instrumentation* is activated, the process engine re-configures the activity DCR_1 particularly by adjusting the post-condition of the task instance DES_1 as it now requires an extra artifact *FICD* in state *defined*. Note that the external agent is responsible to produce the *defined FICD*. Such situations, if not specified in the model, are not supported in conventional process environments. In fact, we allow process actors to enact their tasks in any order in response to their real execution context without explicitly defining their execution orders in the model.

To illustrate the enactment of partially-modeled processes in our enactment approach, we suppose that the fragment of *Instrumentation Team* is performed by an external subsidiary who is not managed by the process environment. As a big strength of our enactment solution, following situation is remedied as shown in the Figure 3.16 where A_1 completes his task *Define Electrical Specification* DES_1, process engine verifies the post-condition and thus suspends the task as the artifact $FICD_1$ is not in state *defined*. Process engine allows the *Analyst* to start execution of his task DES_1 as soon as the artifact $FICD_1$ with state $defined$ is produced by the external subsidiary inside the DBMS. Then, process engine updates the PDG regarding the state change of $FICD_1$ and wakes up the task DES_1. Conventional process environments are unable to deal with such situations as the global process is partially modeled or partially instantiated. For instance, the *Analyst* will be blocked on the task *Define Electrical Specification*, waiting for the termination of the unmanaged activity *Design Instrumentation Model*.

To conclude, the artifact-driven enactment supports fully the execution of partially and structurally modeled process models. It offers significant flexibilities to support unforeseen situations while preserving control over the processes. It makes the process execution less restrictive and closer to the reality which results in gaining the process actors' satisfaction and reducing the gap between model and its real enactment.

3.7 Related Work

Process environments mostly adopt two paradigms as being activity-driven or artifact-driven in order to enact and synchronize processes. In activity-driven enactment, the

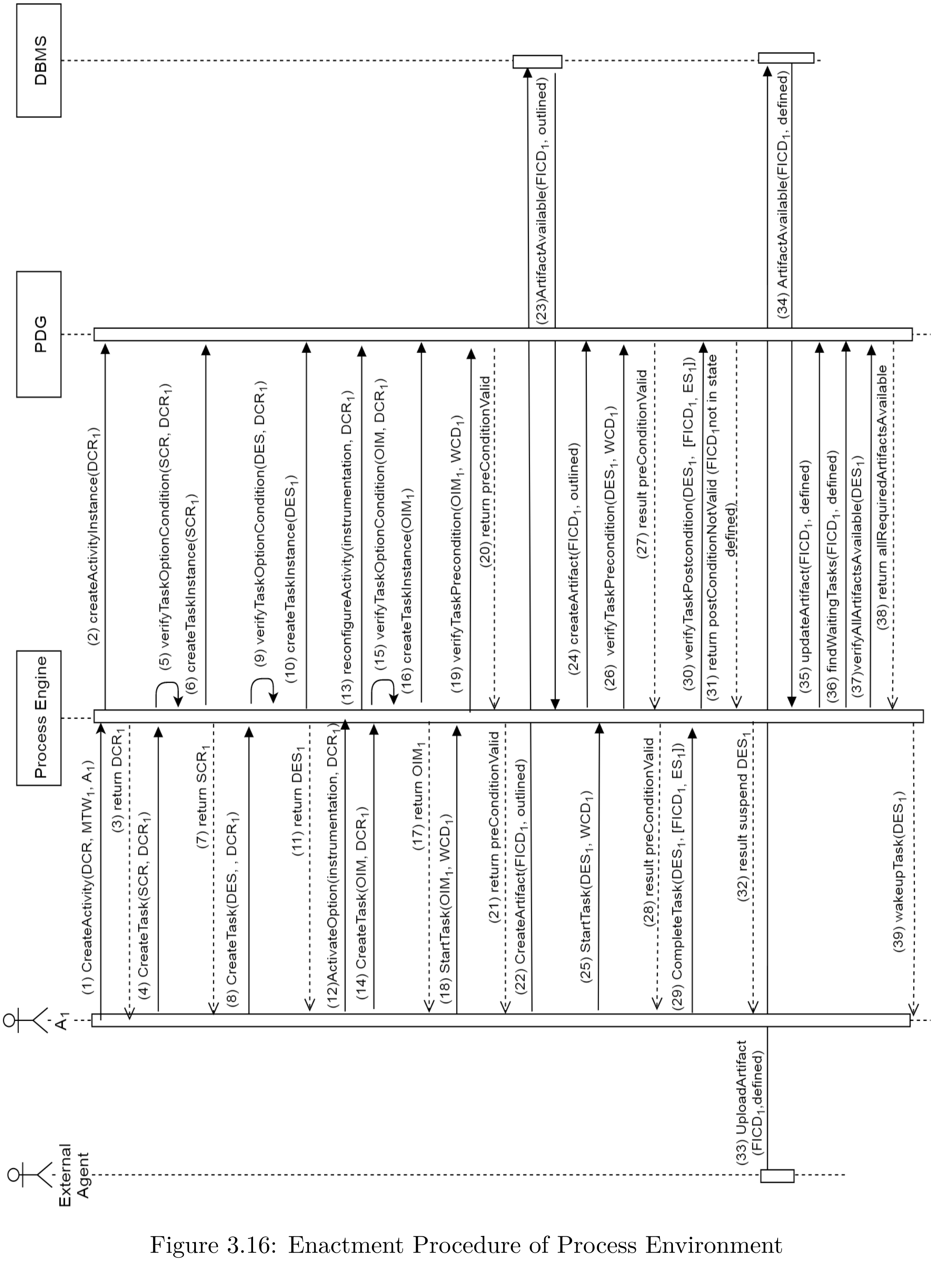

DBMS
PDG
Process Engine
A_1
External Agent
(1) CreateActivity(DCR, MTW$_1$, A$_1$)
(2) createActivityInstance(DCR$_1$)
(3) return DCR$_1$
(4) CreateTask(SCR, DCR$_1$)
(5) verifyTaskOptionCondition(SCR, DCR$_1$)
(6) createTaskInstance(SCR$_1$)
(7) return SCR$_1$
(8) CreateTask(DES, , DCR$_1$)
(9) verifyTaskOptionCondition(DES, DCR$_1$)
(10) createTaskInstance(DES$_1$)
(11) return DES$_1$
(12)ActivateOption(instrumentation, DCR$_1$)
(13) reconfigureActivity(instrumentation, DCR$_1$)
(14) CreateTask(OIM, DCR$_1$)
(15) verifyTaskOptionCondition(OIM, DCR$_1$)
(16) createTaskInstance(OIM$_1$)
(17) return OIM$_1$
(18) StartTask(OIM$_1$, WCD$_1$)
(19) verifyTaskPrecondition(OIM$_1$, WCD$_1$)
(20) return preConditionValid
(21) return preConditionValid
(22) CreateArtifact(FICD$_1$, outlined)
(23)ArtifactAvailable(FICD$_1$, outlined)
(24) createArtifact(FICD$_1$, outlined)
(25) StartTask(DES$_1$, WCD$_1$)
(26) verifyTaskPrecondition(DES$_1$, WCD$_1$)
(27) result preConditionValid
(28) result preConditionValid
(29) CompleteTask(DES$_1$, [FICD$_1$, ES$_1$])
(30) verifyTaskPostcondition(DES$_1$, [FICD$_1$, ES$_1$])
(31) return postConditionNotValid (FICD$_1$not in state defined)
(32) result suspend DES$_1$
(33) UploadArtifact (FICD$_1$,defined)
(34) ArtifactAvailable(FICD$_1$, defined)
(35) updateArtifact(FICD$_1$, defined)
(36) findWaitingTasks(FICD$_1$, defined)
(37)verifyAllArtifactsAvailable(DES$_1$)
(38) return allRequiredArtifactsAvailable
(39) wakeupTask(DES$_1$)

Figure 3.16: Enactment Procedure of Process Environment

main driver for process progression are the events related to activity completions. In fact, activity completions enable subsequent activities according to the work-sequence relations defined in the model. In turn, business artifacts are rather unknown to the process engine of an activity-driven process environment. On the other side, artifact-driven enactment emphasizes on tight integration of processes, artifacts (data), and users where the main driver for process progress is not the events related to activity completion, but availability of artifacts in certain states.

One of the main challenges of process environments has been offering flexibility due to the dynamicity of current environments. A key consideration of effective process environments is their ability to deal with both foreseen and unforeseen situations in the context in which they operate [28]. This quality of a process environment reflects its ability to deal with such situations by varying or adapting those parts of the process that are affected by them, while maintaining the essential format of those parts that are not impacted by the variations.

In the next, we present our evaluation criteria in order to evaluate the existing enactment approaches.

3.7.1 Evaluation Criteria

Our evaluation criteria is inpired form the one proposed by Van der Aalst et al. [32, 28] who presents a comprehensive description of the various approaches that can be taken to facilitate flexibility within a process. All these strategies improve the ability of processes to respond to (un)foreseen situations in their operating environment without necessitating the complete redesign of the underlying process definition. However they differ in the timing and manner in which they are effected. The respective evaluation criteria is presented in the Figure 3.17 comprising two criteria as *Control* and *Flexibility*.

3.7.1.1 Control

This criterion concerns the whole procedure of the enactment approach in order to enact and synchronize the process activities. It contains two factors as described below:

- *Synchronization* defines the main driver for the process progression which can be the *activity* or the *artifact*.

- *Enactment Requirement* defines the required degree of process model completeness to enable the enactment and synchronization of process activities which can be as follows:

  - *Complete Model* is the ability to enact a complete process model in which all possible execution paths in a process model are completely defined at modeling-time.

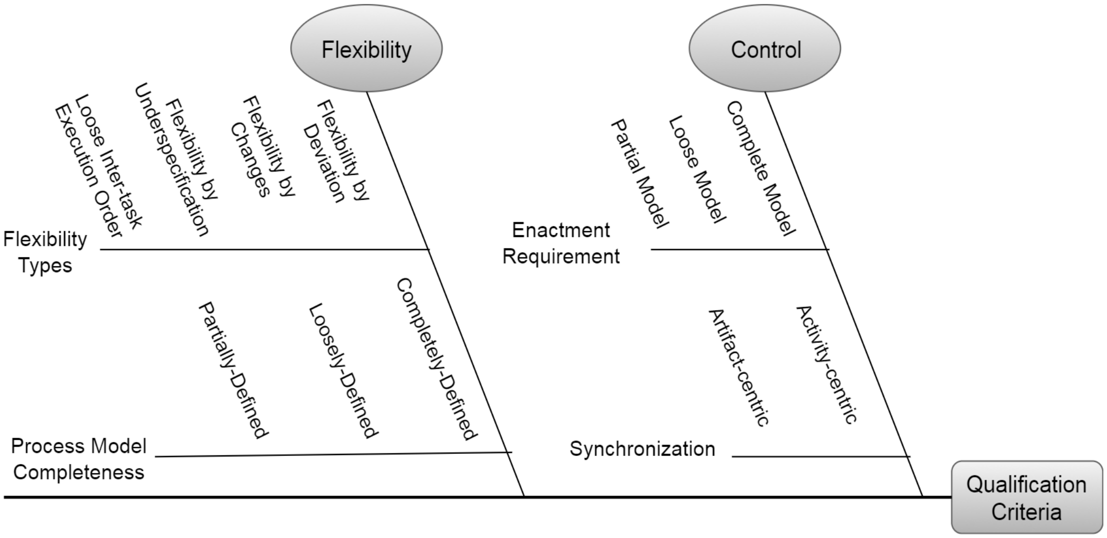


Figure 3.17: Enactment Evaluation Criteria

- *Loose Model* is the ability to enact an incomplete process model at enactment-time, i.e. one which does not contain sufficient information to allow it to be executed. The model does not need to be changed at run-time, instead it needs to be completed by providing a concrete realisation for the undefined fragments in order to enable the completion of process execution.
- *Partial Model* is the ability to enact a process in which some process fragments are missing. This does not require the model to be completed to finish the process execution but the engine can enact and synchronize the fragments defined in the process model with those that have not been defined yet, i.e., performed by external agents/systems.

3.7.1.2 Flexibility

This criterion presents the ability of enactment solutions to respond to anticipated or unforeseen situations in their operating environment without necessitating the complete redesign of the underlying process model. Four different types of flexibility are identified as following in which they differ in the timing and manner:

- *Flexibility by Deviation* is the ability for a process instance to deviate at enactment-time from the execution path prescribed by the original process without modifying its process model. Two important operations that characterize the support of deviation are *undo* and *rework/redo* a task. The former concerns switching the control to the moment before the execution of a task and the latter concerns re-enacting the previously executed task without shifting control.

- *Flexibility by Change* is the ability to modify a process model at enactment-time. It allows processes to adapt to changes that are identified in the operating environment in an *anticipated* or an *ad-hoc* manner. Changes may be introduced both at process instance and process model levels, i.e., one or all of the currently executing process instances are migrated to a new process definition.

- *Flexibility by Underspecification* is the ability to execute an incomplete process model at enactment-time. It comprises three types as follows:

  – *Late Binding* is the approach where the realisation of an unspecified part is selected from a set of available process fragments. Note that one process fragment has to be selected from an existing set of process fragments which are predefined at modeling-time. This approach is limited to selection, and does not allow a new process fragment to be constructed.

  – *Late Modeling* is the approach where a new process fragment is constructed in order to realise a given unspecified part at enactment-time.

  – *Partial Modeling* is the approach where the process can be enacted and synchronized without requiring all fragments to be defined. In contrast to *late binding* and *late modeling*, process models do not need to be defined completely during process execution.

- *Loose Inter-task Execution Order* is the ability of the enactment solution to allow enactment of processes tasks in different execution orders according to different contexts or (un)foreseen situations without imposing any change in the process model, i.e., enactment of optional activities.

3.7.2 Review of Artifact-driven Enactment Approaches

Several approaches have already perceived the inability of current process environments to adequately capture the relation between process and data such as Case Handling [10, 19, 34], Object-aware Processes [15, 21, 24] and Product-based Workflow Support [26, 35]. The next section evaluates to what extent these approaches support the enactment of structurally and partially modeled processes via SPML.

3.7.2.1 Case Handling

van der Aalst et al. [34] propose *Case Handling (CH)* approach aiming at more flexible process enactment by relaxing the work-sequence relation of processes and emphasizing on artifact-driven synchronization. Cases, data objects and activities are the main concepts of CH. CH supports multiple instantiation patterns through dynamic sub-plans (i.e., sub-process instances). CH coordinates the execution of activities based on the data assigned to a case. CH differentiates between free, restricted, and mandatory data elements. An activity is considered as completed if its mandatory data elements all have an assigned value. CH associates three types of role to activities as execute role to carry out an activity, rework role to redo a completed activity and skip role to omit the execution of activities. Their approach is formalized by defining generic state machines for activity and data objects, as well as Event Condition Action (ECA) rules that describe the execution semantics. CH requires the whole process model to be defined before enactment. Moreover, the rework of an activity is only possible if it is explicitly defined in the model.

3.7.2.2 Object and data aware Process Enactment

Kunzle et al. [15] emphasize on the integration of process and data to offer more flexible and efficient process enactment through the PHILharmonicFlows framework. They address modeling and enactment of object-aware processes. Object types and object relations are defined in a data model, while object behavior are expressed in terms of a process whose execution is driven by object attribute changes. Enactment flexibility is provided by allowing the rework and the explicit consideration of user decisions.

Muller at al. [20, 21] propose a data-driven coordination framework COREPRO applied in the automotive domain. COREPRO enables the coordination of multiple processes based on objects and object relations. Single objects are defined in terms of states and (internal) transitions between them. The latter can be associated with complex actions (i.e., processes) that must be completed before the subsequent state may be entered. According to the relations between objects, external transitions connect the states of different objects to coordinate their processing. In addition to modeling and enactment, COREPRO allows dynamic adaptation of process structures based on certain product model. COREPERO applies two modifications as add and remove system or subsystem life cycles. Their work adopts the top-down approach that requires a global view of process model before enactment.

3.7.2.3 Product-based Workflow Support

The core of product-based workflows [26, 35] is a *Product Data Model* described in terms of a tree-like structure [26]. Processes are designed in respect to a certain product structure comprising atomic data elements and operations. An operation is executable if specific values are available for all input data elements. Finally, the product is fully processed as soon as a value for the top element of the product data model becomes available. Each process model solely contains data elements mandatorily required for process enactment. Optional data is not taken into account and thus optional activities are not considered. The process model must be completely defined before enactment and also there is no possibility of rework or any other flexibility types.

3.7.3 Review of Activity-driven Enactment Approaches

Many solutions have been proposed to support the flexible enactment of activity-driven processes. Some of them try to avoid change, e.g. by generating implicit alternative paths [30], or by deferring the selection or modeling of the desired behavior, e.g. by late binding or late modeling [3, 33]. Some approaches allow for changing the process model for a single instance by allowing process actors to deviate from the model. The deviation can be carried out in a planned [5, 7] or ad-hoc manner [4, 16]. Finally, some works focus on process evolution by enabling the changing of a process model while migrating all running process instances to the new version of process model. This solution is very costly and may generate other problems especially if there are several running instances of that model. However, to set a focus, we exclude issues related to process evolution [25] as this section focuses on process enactment.

3.7.3.1 Traditional Process management

Several process environments adopting Business Process Management (BPM) decipline are commercially developed and offer similar functionalities to users, e.g., jBPM [12], Bonita BPM [6], Signavio Business Transformation Suite [1], etc. Among them, we consider jBPM [12] which is a flexible Business Process Management (BPM) Suite. The core of jBPM [12] is a light-weight, extensible process engine that allows you to execute business processes using the latest BPMN 2.0 specification. On top of the core engine, a lot of features and tools are offered to support business processes throughout their entire life cycle, e.g., human task service, runtime persistence, process simulation, Business Activity Monitoring (BAM), etc. jBPM is developed towards process experts and offers a low degree of flexibility. Process engine requires a whole process to be modeled before enactment. It support late selection by allowing the definition of the ad-hoc sub-process in the model.

3.7.3.2 Flexible Enactment by Change

Changes/deviations are known as unexpected situations that could arise during process enactment [29]. Bendraou et al. [5, 7] focus on detection of change on activity and artifact by comparing execution trace to set of pre-defined rules. They provide no support for correcting the deviation. Zazworka et al. [36] propose a solution to detect and correct the deviation of software development project. Correction is done manually by assisting project managers with only pieces of advice to help them finding out some resolutions that are supposed to fix the occurred deviations once applied. Proposed solutions are mainly tool specific and can be applied in small specific domains where deviations are limited and known beforehand.

Lanz et al. [4, 16] propose AristaFlow BPM Suit as an adaptive process environment based on ADEPT technology [8] which enables ad-hoc deviations from pre-specified processes without affecting correctness and soundness of the process it implements. AristaFlow uses BPMN [23] as its modeling language and state machine based process engine which process progression is driven by activities' completion. Moreover, the process engine requires the global model to be defined before enactment. It offers handling of exceptions by allowing process actors to adopt the process by adding/removing/moving activities.

3.7.3.3 Loosely-specified Process Enactment

Some approaches focus on enactment of processes which are not fully pre-defined, where parts of the process model are known at modeling-time, other parts are unknown and can be specified during enactment-time [3, 27, 33], i.e., flexibility by underspecification. Adams et al. [3] propose the Worklet approach that allows late selection by enabling process designer to create a model which, except for its placeholder activities, is entirely predefined. The enactment of a placeholder is selected from a set of defined process fragments (i.e., activities and sub-processes).

Sadiq et al. [27] propose the Pockets of Flexibility approach that allows late modeling of placeholder activities at enactment-time. They define declarative constraints, as relations between tasks, to limit the construction of new models in which process engine verifies these constraints to avoid undesirable enactment of process. In general, as more constraints are defined for a process, less execution paths are possible, i.e., constraints limit the flexibility. The process engine allows the enactment of incomplete processes. However, the process engine requires the unspecified parts of the process to be completely defined in order to completion of process execution. Moreover, the main driver for the synchronization is activity completion.

3.7.3.4 Subject-oriented Process Enactment

The Subject-oriented Business Process Management (S-BPM) conceives a process as a collaboration of multiple subjects (roles) organized via structured communication. It aims to distributed modeling and execution of processes. Several process environments following this paradigm exist either in commercial/academic level [13, 18, 31, 2]. S-BPM engines (e.g., metasonic [18]) are activity-driven and are mostly rigid where there is no flexibility offered such as rework, optional activities, etc. To enact a process, all process fragments along with their artifact exchanges must be completely-defined.

3.7.4 Synthesis

Table 3.10 synthesizes the different enactment approaches discussed in the previous section in relation to our evaluation criteria.

Artifact-driven enactment approaches provide a high degree of control over the processes which considers artifacts as the main driver for process progression. On the other side, they offer a low degree of flexibility to process actors by requiring the completely defined process for enactment and synchronization. Moreover they are mostly unable to deal with unforeseen situations. Many prototypes have been proposed by offering means to model and enact processes [34, 15], but the lack of flexibility within these environments has led to being less adopted within industry [11].

Activity-driven enactment approaches in which the main driver for process progression is activity completion, provide a better balance between control and flexibility. Although, they ignore the importance of process and data integration in which data is rather unknown to their process engines. In turn, the process engine follows a control-flow based process model to enact processes. These models are (loosely) completely defined. To provide less rigid enactment, several prototypes have been proposed by offering different types of flexibility to cope with unforeseen situations in academic (e.g., AristaFlow [4], Declare [33], etc.) or commercial level (e.g., jBPM [12], Bonita [6], etc.). Among them flexibility by underspecification [33, 27] appears to better keeping the balance between control and flexibility. Process changes [4] offer high degree of flexibility to process actors by allowing them to freely modify the process model which can cause control problems on already running processes.

However, artifact and activity-driven enactment approaches differ in their way of providing control and flexibility but both have difficulties in terms of enacting and synchronizing the partially and structurally modeled processes. In contrast to activity-driven approaches, artifact-driven approaches are able to enact the structurally defined process fragments models. Our enactment approach gains this benefit of artifact-driven approaches but most importantly, it provides some degree of enactment flexibilities to deal with both foreseen and unforeseen situations while preserving the control over processes.

Our enactment approach provides enactment flexibilities by proposing an activity state machine which allows different possibilities of task executions, i.e., through optional activities and rework. As there is no work-sequence relations defined among tasks, tasks can be enacted less restrictive, e.g., activity and tasks can be enacted several times if needed. Moreover, thanks to the connection between process engine and DBMS, the enactment of partially modeled processes are supported (i.e., partial modeling). Notice that, partial modeling encompasses late modeling as well by considering the process fragments as unspecified parts of the model which can be defined during project execution.

In our enactment approach, process deployment is simple as mapping between SPML and PDG is direct. Having a mechanism as PDG which stores the global vision of the running processes, leads to a better emergence of global models and dependencies among process instance elements. This global model can provide a base for some interesting analysis such as impact analysis of the change or process analysis. In the next chapter, we discuss this feature in details.

3.8 Summary

This chapter introduced our artifact-driven enactment approach which puts the main stress on artifacts as the main driver for process progression. The objective of our enactment approach is to provide the balance between control (i.e., enacting and synchronizing the activities/tasks of the structurally defined process fragments) and flexibility (i.e., coping with (un)foreseen situations). To do so, we develop a Bottom-up Artifact-centric Process Environment (BAPE). The enactment environment of BAPE is comprised of (1) a Process Dependency Graph (PDG) as a run-time storage keeping the information of process element instances coming from different process fragments and therefore represent the global view of the system; (2) an artifact-driven process synchronization mechanism (process engine) to manage tasks' life cycle. Our enactment approach offers some degree of autonomy to process actors in order to deal with (un)foreseen situation such as loose inter task execution orders, reworks and enactment of partially defined processes. In our approach, process models are fragmented but they progressively emerge from enacting the activities/tasks of separate process fragments models.

Table 3.10: Synthesis of Enactment Approaches

Related Work		Comparison		
Category	Approach	Flexibility	Control	
			Synchronization	Enactment Requirement
Artifact-driven Process Enactment	Case Handling	Flexibility by Deviation: semi-support for rework	Artifact	Complete Model
	Object and Data-Aware Process Enactment	Flexibility by Deviation: Rework	Artifact	Complete Model
	Product-based Workflow Support	No Support of Flexibility	Artifact	Complete Model
Activity-driven Process Enactment	Traditional Process Management	Flexibility by underspecification: late binding	Activity	Complete Model
	Process Adaptation	Flexibility by Change: Ad-hoc Changes	Activity	Complete Model
	Loosely-specified Process Enactment	Flexibility by underspecification: late binding, late modeling	Activity	Complete Model Loose Model
	Subject Oriented Process Enactment	No Support of Flexibility	Activity	Complete Model
Data-driven Process Enactment		Flexibility by Inter Task Execution Order Flexibility by Underspecification: Partial Modeling, Late Modeling	Artifact	Partial Model Complete Model Loose Model

Bibliography

[1] Signavio business transformation suite, https://www.signavio.com/products/business-transformation-suite/.

[2] ActnConnect. http://www.actnconnect.de/.

[3] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. *Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows*, pages 291–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[4] AristaFlow. http://www.aristaflow.com/.

[5] R. Bendraou, M. A. A. da Silva, M. Gervais, and X. Blanc. Support for deviation detections in the context of multi-viewpoint-based development processes. In *Proceedings of the CAiSE'12 Forum at the 24^{th} International Conference on Advanced Information Systems Engineering (CAiSE), Gdansk, Poland, June 28, 2012*, pages 23–31, 2012.

[6] BonitaSoftware. http://www.bonitasoft.com/.

[7] M. da Silva, X. Blanc, and R. Bendraou. Deviation management during process execution. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 528–531, Washington, DC, USA, 2011. IEEE Computer Society.

[8] P. Dadam and M. Reichert. The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - Research and Development*, 23(2):81–97, 2009.

[9] M. Hajmoosaei, H. N. Tran, and C. Percebois. A user-centric process management for system and software engineering projects. In *presented at the 7th IESM Conference, October 11–13, 2017, Saarbrucken, Germany*, 2017.

[10] M. Hewelt and M. Weske. *A Hybrid Approach for Flexible Case Modeling and Execution*, pages 38–54. Springer International Publishing, Cham, 2016.

[11] R. Hull. *Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges*, pages 1152–1163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[12] JBoss. jbpm, http://www.jbpm.org/.

[13] F. Krenn, C. Stary, and D. Wachholder. Stakeholder-centered process implementation: Assessing s-bpm tool support. In *Proceedings of the 9th Conference on*

Subject-oriented Business Process Management, S-BPM ONE '17, pages 2:1–2:11, New York, NY, USA, 2017. ACM.

[14] M. Kuhrmann and S. Beecham. Artifact-based software process improvement and management: A method proposal. In *Proceedings of the 2014 International Conference on Software and System Process*, ICSSP 2014, pages 119–123, New York, NY, USA, 2014. ACM.

[15] V. Künzle and M. Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance and Evolution: Research and Practice*, 23(4):205–244, June 2011.

[16] A. Lanz, M. Reichert, and P. Dadam. Making business process implementations flexible and robust: Error handling in the aristaflow bpm suite. In *CAiSE'10 Demos*, June 2010.

[17] R. Matinnejad and R. Ramsin. An analytical review of process-centered software engineering environments. In *IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2012, Novi Sad, Serbia, April 11-13,2012*, pages 64–73, 2012.

[18] MetasonicSuite. http://www.metasonic.de/.

[19] A. Meyer, N. Herzberg, F. Puhlmann, and M. Weske. Implementation framework for production case management: Modeling and execution. In *Proceedings of the 2014 IEEE 18th International Enterprise Distributed Object Computing Conference*, EDOC '14, pages 190–199, Washington, DC, USA, 2014. IEEE Computer Society.

[20] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08)*, number 5074 in LNCS, pages 48–63. Springer, June 2008.

[21] D. Müller, M. Reichert, J. Herbst, D. Köntges, and A. Neubert. Coreprosim: A tool for modeling, simulating and adapting data-driven process structures. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 394–397, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] OASIS. Web Services Human Task (WS-HumanTask) Specification Version 1.1, August 2010.

[23] OMG. Business Process Model and Notation (BPMN) Version 2.0.2, Dec. 2013.

[24] G. Redding, M. Dumas, A. H. M. ter Hofstede, and A. Iordachescu. A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, 4(3):191–201, 2010.

[25] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies.* Springer, Berlin-Heidelberg, 2012.

[26] H. A. Reijers, S. Limam, and W. M. van der Aalst. Product-based workflow design. *Journal of Management Information Systems*, 20(1):229–262, 2003.

[27] S. W. Sadiq, M. E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, July 2005.

[28] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst. Towards a taxonomy of process flexibility. In *CAiSE Forum*, volume 344 of *CEUR Workshop Proceedings*, pages 81–84. CEUR-WS.org, 2008.

[29] M. Smatti, M. C. Oussalah, and M. Ahmed-Nacer. Supporting deviations on software processes: A literature overview. In *Software Technologies - 10th International Joint Conference, ICSOFT 2015, Colmar, France, July 20-22, 2015*, pages 191–209, 2015.

[30] B. Staudt lerner, S. Christov, L. Osterweil, R. Bendraou, U. Kannengiesser, and A. Wise. Exception Handling Patterns for Process Modeling. *IEEE Transactions on Software Engineering*, 36(2):162–183, Mar. 2010.

[31] F. Strecker, R. Gniza, T. Hollosy, and F. Schmatzer. Business-actors as base components of complex and distributed software applications. In *Proceedings of the 8th International Conference on Subject-oriented Business Process Management*, S-BPM '16, pages 9:1–9:8, New York, NY, USA, 2016. ACM.

[32] W. M. P. van der Aalst. Process-aware information systems: Design, enactment, and analysis. In *Wiley Encyclopedia of Computer Science and Engineering.* John Wiley & Sons, Inc., 2008.

[33] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

[34] W. M. P. van der Aalst and M. Weske. Case handling: A new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, May 2005.

[35] I. T. P. Vanderfeesten, H. A. Reijers, and W. M. P. van der Aalst. Product-based workflow support. *Inf. Syst.*, 36(2):517–535, 2011.

[36] N. Zazworka, V. R. Basili, and F. Shull. Tool supported detection and judgment of nonconformance in process execution. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 312 – 323, 2009/10// 2009.

Chapter 4

Change Aware Process Enviroment

Contents

This chapter introduces the change management support provided by our process environment which enables process actors to report emergent changes, analyze the possible impacts and inform the affected people by the changes. The challenges of managing process changes along with the importance of PDG are presented in Section 4.1. We identify the essential objectives of our change management approach which are discussed in Section 4.2. To support change management, architecture of our process environment BAPE is extended as presented in Section 4.3. Section 4.4 introduces our proposed change management process in order to handle change requests. The core step of our process is regarding analyzing the impact of change which achieved by developing a change impact analyzer component. Section 4.5 presents the feasibility of using some metrics in order to enrich the result of change impact analysis. Section 4.6 studies some essential works and compared them with our solution. Finally, Section 4.7 summarizes this chapter.

4.1 Problems of Unofficial Changes

Nowadays, changes in SSE processes are almost inevitable due to evolving requirements, resources and technologies. Changes occurring in a running task can affect, in a chaining-fashion, other tasks either inside one organization or among different organizations. Badly managed changes can lead to unwanted reworks and cause projects to fall behind schedule and go over budget. Applying a holistic, structured approach to manage changes is then crucial to avoid adding extra cost and risk to both the project and organizational levels.

In [15], the authors classified changes existing inside companies into two types: official and unofficial. An official change happens often at a company or product level and requires to be handled by a formal protocol. Normally the process dealing with official changes is rather well-defined and conducted in each company. An unofficial change happens generally inside technical processes, during the pre-certification phases, as backwards patching/debugging redesign processes [5] where process actors attempted to fix a problem quickly during their work. This type of changes is often informal or semi-formal and poorly managed due to the lack of coordination among process actors. In this work we focus on managing such unofficial changes.

4.1.1 Example of an Unofficial Change

To illustrate the issues of unofficial changes, we consider the process of our industrial partners as shown in the Figure 4.1.

The respective process can be applied to various projects initiated for different wiring change demands corresponding to different specific testbenches. We discuss the impacts of unofficial changes based on one of the most common situation that frequently occurs during the modification of testbench wiring. We suppose that there is no any systematic support for handling changes. The illustrating scenario is described as follows:

Execution scenario Δ
We assume that the company has launched three projects by enacting three process

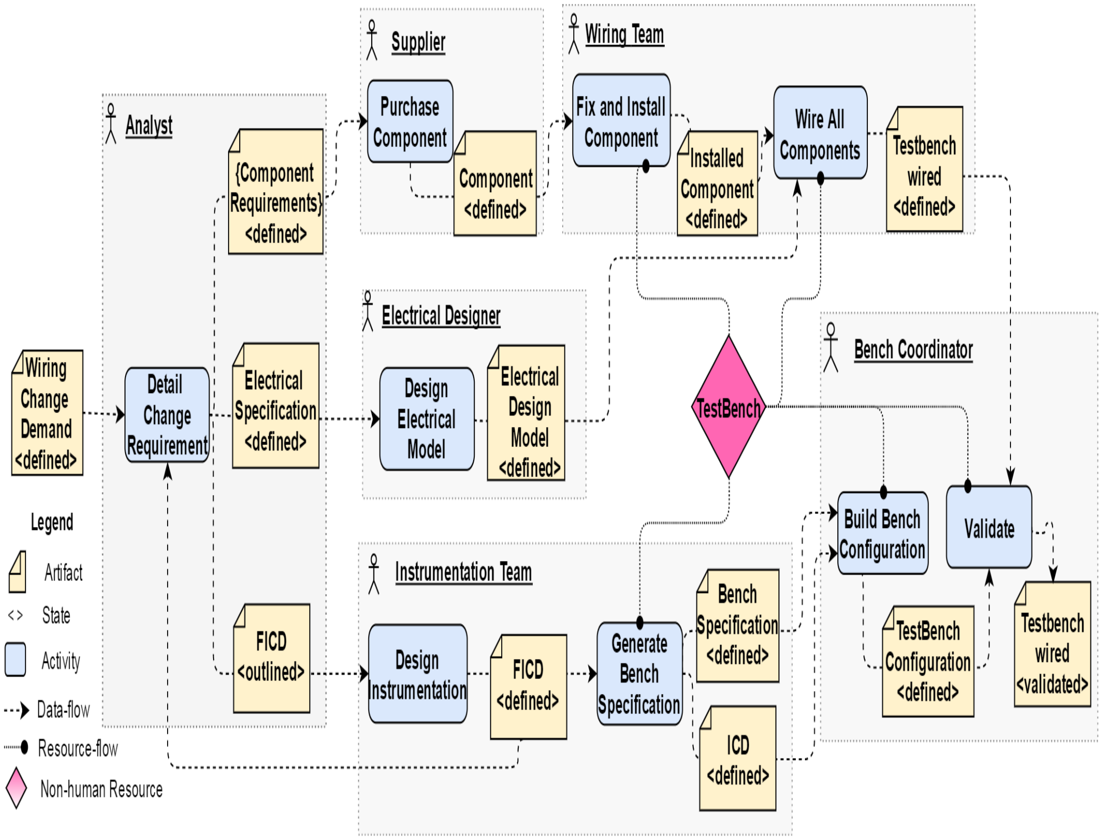

Analyst
Supplier
Wiring Team
Electrical Designer
Instrumentation Team
Bench Coordinator
Wiring Change Demand <defined>
Detail Change Requirement
{Component Requirements} <defined>
Electrical Specification <defined>
FICD <outlined>
Purchase Component
Component <defined>
Fix and Install Component
Installed Component <defined>
Wire All Components
Testbench wired <defined>
Design Electrical Model
Electrical Design Model <defined>
TestBench
Design Instrumentation
FICD <defined>
Generate Bench Specification
Bench Specification <defined>
ICD <defined>
Build Bench Configuration
Validate
TestBench Configuration <defined>
Testbench wired <validated>
Legend
Artifact
<> State
Activity
Data-flow
Resource-flow
Non-human Resource

Figure 4.1: Process of *Modify Testbench Wiring*

instances P_1, P_2 and P_3 respectively for three wiring change demands WCD_1, WCD_2 and WCD_3. Particularly, we examine the process instance P_1 initiated from change demand WCD_1 regarding the testbench TB_1. We suppose that P_1 shares some artifacts and resources with other process instances. The main information along with the execution scenario of P_1 are respectively presented in Table 4.1 and Figure 4.2. We suppose that in P_1, two instances of activity *Purchase Component* PC_1 and PC_2 are performed, the components C_1 and C_2 are produced and installed on the testbench TB_1. At that moment, the *Wiring Team* WT_2 is performing his task *Wiring Components* WC_1 by consuming artifacts *Instal Components* IC_1 and IC_2. *Wiring Team* WT_3 involved in P_2 is performing task *Fix Component* FC_3 of his activity *Fix and Install component* FIC_3 in order to set up the component C_1 on the testbench TB_2. Moreover, the *Bench Coordinator* BC_3 and *Wiring Team* $WT4$ involved in P_3 are waiting to use the *Testbench* TB_1 in order to perform their tasks.

Table 4.1: Information of P_1

Process Actors	Analyst = A_1 Instrumentation Team = IT_1, IT_2 Supplier = $\{S_1, S_2\}$ Wiring Team = $\{WT_1, WT_2\}$ Electrical Designer = ED_1 Bench Coordinator = BC_1	
Non-human resources	TestBench = TB_1	
Components	$\{C_1, C_2\}$	
Shared Resources & Artifacts	$P_1 \wedge P_2$	C_1
	$P_1 \wedge P_3$	TB_1

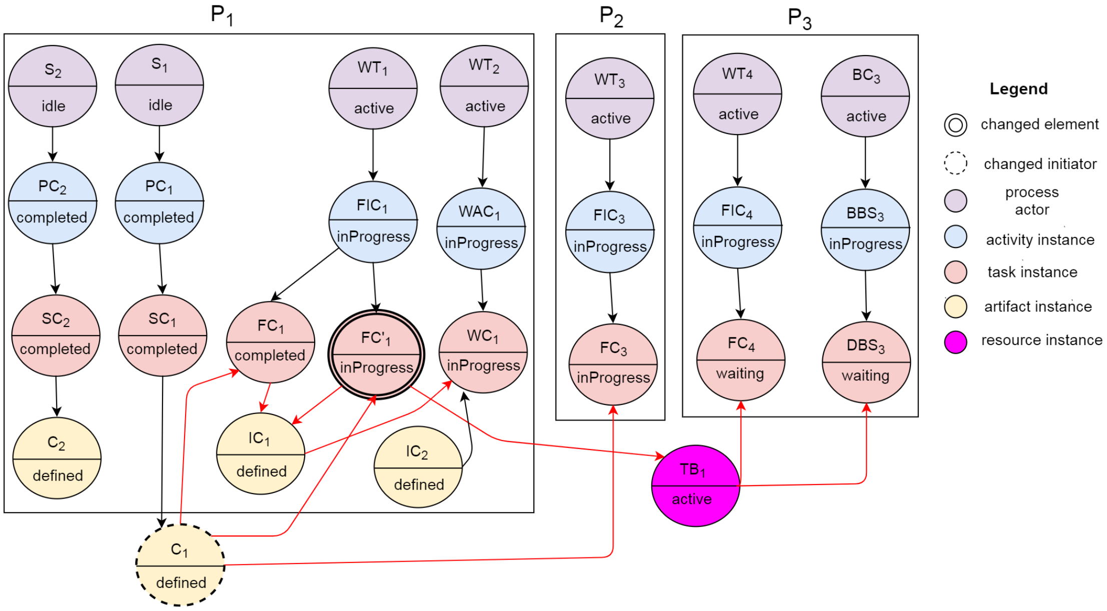


Figure 4.2: Execution Scenario

Description of change scenario

In P_1, the *Wiring Team* WT_1 realizes a problem in the already produced *Component* C_1. Thus WT_1 requires to change C_1 by redoing his task FC_1 which will result in a new version of IC_1.

To continue and finish his installation, WT_1 wants to modify C_1 which is out of his responsibility. If the change was controlled, WT_1 had to inform the *Supplier* S_1 of the problem and then wait for a new version of C_1. However, in reality, in a loosely controlled development environment, WT_1 may change directly C_1 without reporting the change because (1) he cannot wait and (2) he does not know the concerned people to notify them of the change.

In this example, the possible impacts of the change on C_1 initiated by WT_1 are:

- *Impacts inside P_1* Any change in C_1 will immediately affect the process fragment instance of *Wiring Team WT_2* who is wiring up the installed components IC_1 and IC_2 for testbench TB_1. Changing C_1 without notifying WT_2 may lead to an incompatible testbench TB_1 which is obsolete and not corresponding to the real component C_1 supplied by S_1. This change may require reworks of the supplier S_1 to insure the consistency between the artifacts *Component Requirement CR_1* and the *Component C_1*.

- *Impacts among other process instances* As process instances share artifacts and resources, the change inside one process instance may propagate through other instances as well. For instance, the change on C_1 in P_1 leads to rework of *Wiring Team WT_3* in order to install the new version of *Component C_1* on testbench TB_2. Additionally, the change causes reworking of the *Wiring Team WT_1* to install the new version of C_1 for the test bench TB_1. Consequently, WT_1 may have to keep TB_1 for a longer period. Because TB_1 is also required in the *Wiring Team* and *Bench Coordinator* process fragment instances of P_3, the change in P_1 can have significant impact on P_3. The impact concerning shared resources requires often stressful and costly planning adjustments for the whole company.

4.1.2 PDG as a Global View

The impacts of the mentioned change scenario cannot be deduced from the process (fragment) models as dependencies among process instances arise only at enactment-time, e.g., list of concrete artifacts along with their states, the state of tasks and the shared resources. By performing the analysis at the model-level we can deduce just some coarse-grained impacts of change. For example, we can say that a change on the artifact type *Component* will impact the tasks *Fix Component* and *Wiring Components*. But if these activities have many concurrent task instances performed by different members of wiring team, we cannot say exactly who and which task instances are affected. Moreover, at the model-level, we cannot know the inter-process impacts via shared resources.

We emphasize the use of instance-level process information in order to establish a global view on the state of all elements inside the development environment. As observed in the examined process, while some of the dependencies existing among elements inside one process instance can be easily extracted at build-time from the process model, there are other dependencies which only emerge at enactment-time. It is especially the case of dependencies among tasks which are instantiated multiple times and collection of artifacts. It is also the case of dependencies via shared resources among different running processes. In contrast to most of existing works which have concentrated on the dependencies at process model-level [6, 10], our PDG describes dependencies at enactment-time in the process instance-level, thus allows a more thorough change impact analysis.

As mentioned in the previous chapter, PDG monitors the instances of process elements enacted in the process environment as well as the relations among them. In fact, PDG represents the process elements and the dependencies among them at enactment-time (instance-level) and is progressively updated by the process engine to reflect always the current state of the system. As in our approach process models are fragmented and are progressively enacted during the project execution, PDG establishes the global view of the system from separate process fragments at enactment-time via shared resources among respective fragments. This global view provides the only base for analyzing the impact of process change and determining the affected elements of the change, e.g., process actors, artifacts, resources, etc. For an efficient storing and querying of enactment-time process data, the PDG is implemented in a graph database. Thanks to the graph structure of PDG, when a change request occurs, PDG provides a sound and effective basis to traverse intra-process instances and inter-process instances based on shared artifacts and resources in order to derive the affected elements. This is achieved by developing traversal algorithms which look up PDG in order to extract affected process elements.

Figure 4.3 illustrates the whole instance-level information of the process instance P_1 and only parts of other process instances illustrating the dependencies among process instances. For instance, we can see the dependencies among process fragments inside P_1 through exchanging artifacts (e.g., Components C_1 and C_2 between process fragments of *Supplier* and *Instrumentation Team*, FICD $FICD_1$ between process fragments of *Analyst* and *Electrical Designer*, etc.) and sharing resources (e.g., Testbench TB_1 between process fragments of *Supplier*, *Instrumentation Team* and *Bench Coordinator*).

We can also observe the dependencies among process instances via exchanging artifacts (e.g., Component C_1 between process fragments of *Supplier* in P_1 and *Wiring Team* in P_2, Component C_2 between process fragments of *Supplier* in P_1 and *Wiring Team* in P_4) and sharing resources (e.g., Testbench TB_1 between process fragments of *Supplier* and *Wiring Team* in P_1 and *Wiring Team* in P_2).

4.1.3 Discussion

As observed, based on the information existing at instance-level, in order to thoroughly assess change impacts, managing and analyzing changes should be done at instance-level through whole system as the change impact propagates inside and among process instances.

We are interested in this unofficial change and motivated by development environ-

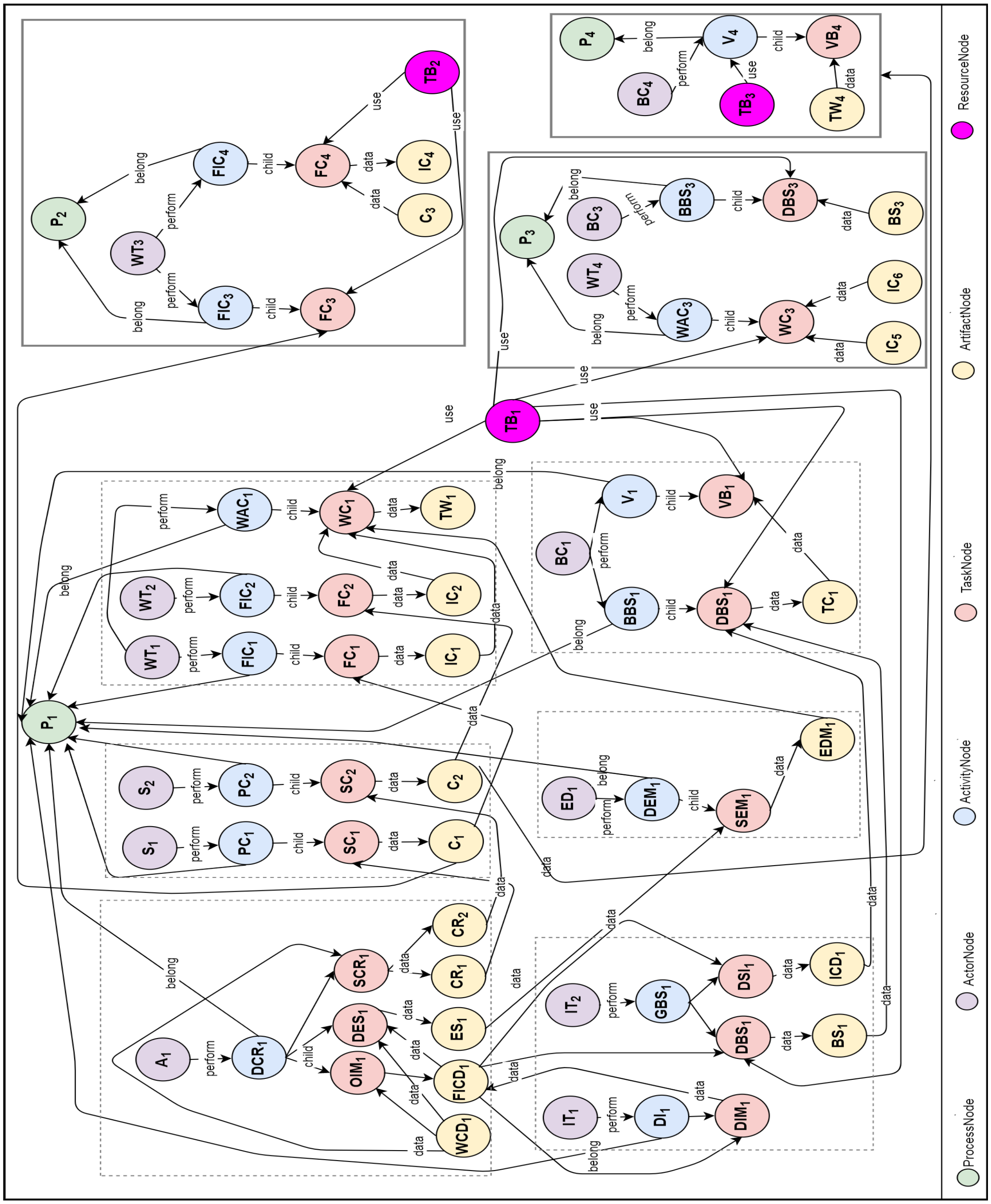


Figure 4.3: PDG representing the process instances

ments where changes are handled manually and communication between concerned actors is free and ad-hoc. In such environments, the poor coordination can lead to unawareness of changes, and therefore to possible reworking. Our challenge is regarding proposing a mechanism that helps coordinating better the process actors and therefore managing unofficial changes more efficiently and systematically.

Notice that we don't have the ambition to propose a complete solution for change management. We are primarily interested in the change notification issue. Change implementation is out of the scope of the presented work. We consider only changes that affects the content of artifacts, i.e., producing a new version of artifact. Our objective is to provide an effective assistance for coordinating process actors in order to keep their artifacts consistent.

In the next section, we illustrate how the architecture of BAPE can be extended in order to support the systematic change signals and usage of PDG information in order to traverse the PDG to extract the affected elements of the change.

4.2 Change Impact Analysis Objectives

In our work, we seek to remedy the problem of unnoticed changes in order to permit concerned people to anticipate and respond to changes so that they can avoid obsolete works. The ultimate goal of our work is providing a *Change-Aware Process Environment* being reactive to change requests but proactive to change implementations [25, 12]. We identify three main objectives of such a Change-Aware Process Environment which are described as follows:

- *Capturing change requests* The proposed approach should provide support for capturing in a centralized and continuous way all change requests sent asynchronously by various process actors. This requires consolidating the process management mechanism and a change management mechanism. On the one hand a *Process Management System* is needed to monitor process actors' activities. On the other hand a *Change Management mechanism* is needed to allow process actors signaling change requests during their tasks executions.

- *Analyzing change impacts* The approach should provide a mechanism to enable analyzing the potential impacts of a change on the whole system. The corresponding analysis should not be limited to one process instance as the change impact can be propagated through whole running processes via shared artifacts and resources. In order to enable analyzing the impacts of change among process elements in a timely and systematically manner, we need to integrate the process management mechanism and the query mechanism on the run-time process data. This requires a mechanism to represent dependencies existing among all process elements of the system at enactment-time, and therefore provides process actors with information of the whole development environment - what they cannot know from their local viewpoint.

- *Informing affected people of change* As the change may impact other process actors on their completed or running tasks, the proposed approach should provide support for notifying process actors affected by the change in a timely manner which avoids obsolete work result and save time and budget.

With respect to the aforementioned objectives, we proposed a change impact analysis approach whose role is turning our process environment BAPE to a change-aware process environment. The change-aware BAPE facilitates run-time change management by providing three mechanisms for: capturing changes occurring in the development environment, (2) analyzing change impacts and (3) informing affected process actors, all in a timely and systematically manner. In the next, we explain how we implement our solution in details.

4.3 Change-aware Process Environment BAPE

Our aim is to consolidate two mechanisms of process management and change management to obtain a change-aware process environment that enables process modeling and process enacting with the capacity of handling changes coming from running processes. In the previous chapters, we thoroughly explained our solution for process management by presenting our modeling and enactment approaches which were implemented in our developed process environment BAPE.

The focus of this chapter is to provide a mechanism to turn BAPE into a change-aware process environment. Figure 5.6 shows the architecture of the change-aware BAPE including the extended (sub)components as illustrated in a gray color. The description of the extended components are as follows:

1. *End User UIs* In addition to a set of user interfaces to allow process actors to interact with the core process management system to model and enact their activities/tasks, a new interface offering process actors the possibility to send a change request during their task execution and also receiving change signals from other process actors is developed. Thanks to this new functionality, process actors can integrate emergent changes into the process execution, those are not described in their process model. Also a new role *Change Manager* together with the specific interface are defined to centralize the change requests coming from different process actors and illustrating the result of the change impact analysis. *Change Manager* can decide about the change (i.e., to accept or reject) and then inform affected process actors.

2. *Change Analyzer Component* We use this component to implement our change management policy, in particular the change impact analysis that traverses the

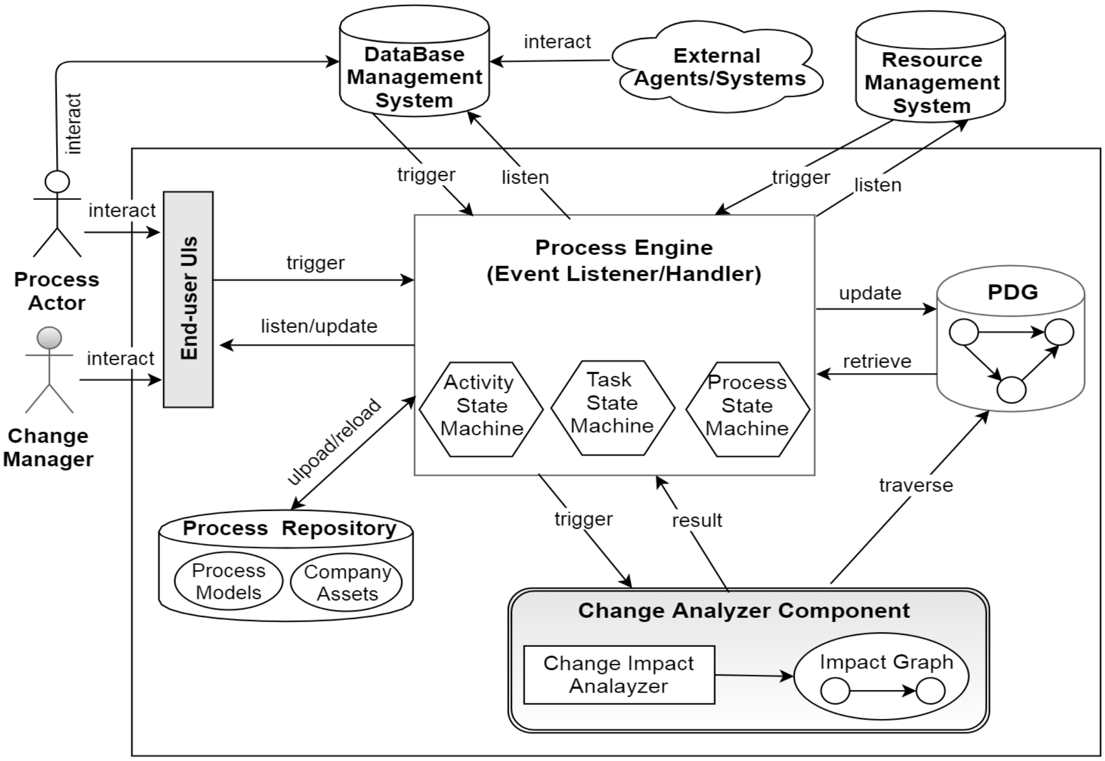


Figure 4.4: Architecture of Change-aware BAPE

PDG and extracts the elements potentially affected by the change request. The result of the change impact analysis is an *Impact Graph (IG)* that gives some information to help the process owners on the change implementation decision and in informing the concerned elements in a timely and systematically manner.

In the next section, we present how BAPE facilitates the implementation of our change management process to allow process actors signal their change requests and be informed of the impact of their changes.

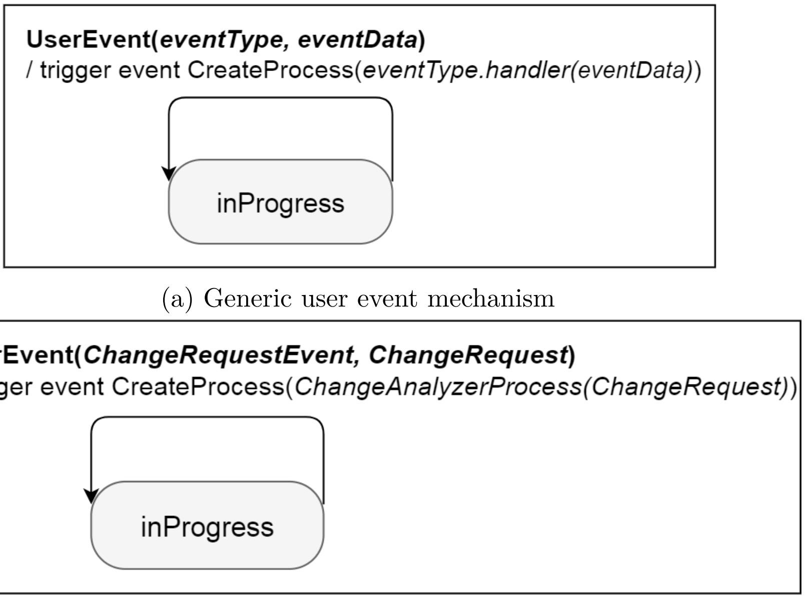
4.4 Change Management Mechanism

We remind that the change management provided in this study is regarding providing a systematic mechanism to pursue three main objectives:

- Signalling an intentional change.
- Estimating the impact of the requested change.

- Notifying potential affected process actors of the change.

To achieve the aforementioned objectives, we defined a process *Change Analyzer* representing one possible policy to handle change requests asynchronously sent by process actors. To integrate this new functionality into BAPE, we used the feature of *UserEvent* defined in the task state machine illustrated in Figure 4.5a. As mentioned in Chapter 3, this event permits process actors to manage some specific user defined events while executing their tasks. Thus, we consider the change request event as a user defined event as illustrated in Figure 4.5.


UserEvent(eventType, eventData)
/ trigger event CreateProcess(eventType.handler(eventData))
inProgress
UserEvent(ChangeRequestEvent, ChangeRequest)
/ trigger event CreateProcess(ChangeAnalyzerProcess(ChangeRequest))
inProgress

(a) Generic user event mechanism

(b) Application of user event mechanism

Figure 4.5: User defined event in task state machine

By using the feature of *UserEvent*, concretely we define a *UserEvent* having *eventType* as *ChangeRequestEvent* and *eventData* as a *Change Request CR*.

In a change request *CR (changeInitiator, changedElement, changeResponsible, changedType, changeEstimation, changeComment, changeAffectedElements, changeDecision)*:

- *changeInitiator* is the process actor who initiates the change.

- *changedElement* is the element that he wants to modify. The *changedElement* must be one of the elements related to the *changeInitiator*'s current task, e.g., input artifact.

- *changeResponsible* is the process actor who is responsible of the changed element, i.e., the one who produced the artifact which required to be changed.

- *changeType* is the type of change that the *changeInitiator* has initiated. When a change initiator signal a change, he can evaluate the strength of the change and can make *minor change* for correcting small problems of an artifact without causing reworking in other tasks. If he wants to evolve an artifact whose changed content can result in reworking on other tasks, he has to make his change as *major change*.

- *changeEstimation* is the initial temporal estimation for the delay caused by the change which is given by the *changeInitiator* or *changeResponsible*.

- *changeComment* is the description of the change provided by *changeInitiator* and *changeResponsible*.

- *changeAffectedElements* are the elements affected by the *changedElement*. This property is the result of analysing the impact of change, i.e., impact graph (IG).

- *changeDecision* is the decision made by change manager after analyzing the impact of change. The decision can be either accepting or rejecting the change request.

By considering the change management policy as a separate executable process, BAPE allows different change management policies to be defined. Whenever a process actor signals a change request, process engine enacts the corresponding change management process and executes its activities. The advantage of BAPE is that it allows the change management process to be defined separately from the core engine. This separation allows different policies to be adopted without requiring modifying the core of process engine.

In the next, we present our process as an example of a policy adopted to handle the change requests.

4.4.1 Change Analyzer Process

This process comprises automatic activities which implement the algorithms to analyze the impacts of a change and inform affected people. We defined the *Change Analyzer* process as the handler of change request event as illustrated in Figure 4.5. Figure 4.6 illustrates our change management process comprising major implemented activities. When a process actor initiates the *ChangeRequestEvent*, process engine triggers the event *CreateProcess (Change Analyzer Process)* to execute the defined change management policy.

The properties of the change request are filled step by step during the change management process. When a process actor triggers signals a change request, he evaluates the change by determining the *changeType* as minor change or major change. As the effect of corresponding transition, process engine initiates the corresponding user handler which is the change management process.

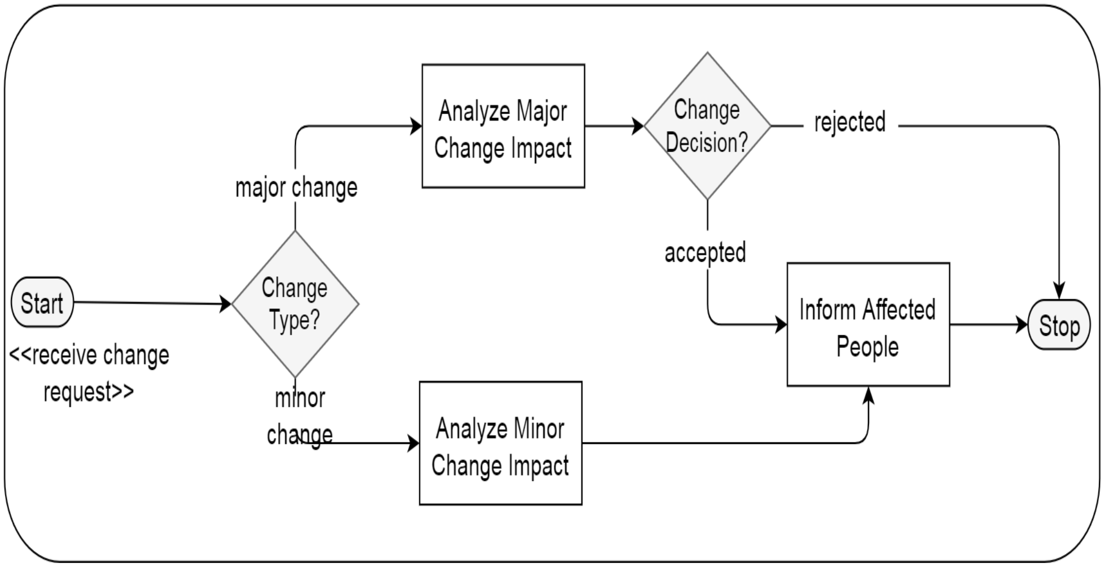


Figure 4.6: Change analyzer process

The process starts with process engine which receives the change request sent by process actor and sends the respective change requests to the change manager. The procedure of change management is slightly different depending on the *changeType*, i.e., minor change or major change. If the change is a minor change which do not impose rework on other affected tasks and do not require to be further evaluated by *changeResponsible*, change manager can request process engine to analyze the change impact. To do so, process engine uses the *Change Analyzer Component* in order to extract the affected process elements of the change and inform affected people, specially the change responsible, i.e., one who produced the changed artifact. We explained detailed traversal procedure of our *Change Analyzer Component* in the next section.

By choosing the *major change*, first, the *changeResponsible* must be identified in order to precisely evaluate the change, e.g., reporting initial temporal estimation for the rework. Thus, change manager requests from process engine to find the *changeResponsible*. To do so, process engine uses Change Analyzer Component in order to find the *changeResponsible*, sends the change request to him and receives the evaluated change request and sends back to the change manager. By receiving the evaluated change request, change manager requests process engine in order to analyze the impact of change. Similarly, process engine invokes the Change Impact Analyzer whose role is to traverse the PDG, extract the affected elements and send the result to the engine which later is sent to the change manager. At this step, change manager requires to make decision about the change based on the information provided from change impact analysis. In fact, two situations may arise:

- In case change manager accepts the change, he can decide which affected tasks requires to be reconfigured, i.e., to be reworked, aborted or undone and then inform affected people of the change. Notice that, as the state of required artifacts may already changed by other tasks, rework or undo a task may require first to modify the state of required artifacts in the DBMS in order to enable process actors to re-start their tasks. However, this action is out of scope of change management process and is assumed to be done separately by change manager or process owner.

- In case change manager rejects the change, only the *changeInitiator* and *changeResponsible* are informed of the change decision.

We summarize the whole procedure of our change-aware BAPE in the Figure 4.7.

In our change management process, the core activity is regarding the analyzing the impact of change by identifying the affected elements of the change. The impacted elements of the change are derived from traversing the PDG by Change Analyzer Component. In the next section, we explain the traversal procedure of Change Analyzer Component in details.

4.4.2 Change Analyzer Component

This component is the core of our change management proposal which is responsible to catch the sent change request, analyzes the impact of the change described in the and sends the result to the process engine in order to inform affected process actors of change.

The sub-component Change Impact Analyzer carries out the impact analysis by traversing the PDG from the changed element *changedElement* thorough possible types of edges. The result of the traversal is a digraph so-called Impact Graph (IG) which is an extraction of the global PDG but contains only the process elements impacted by the change, e.g., process actors, artifacts instances, resources instances, etc. These elements are detected by the emerging dependencies among enactment-time process elements in the PDG based on shared data and resources.

The impact can be direct if an element works with the changed element, or indirect if an element works with an element impacted by the change. Moreover, thanks to the PDG that keeps trace of all elements in the development, we can have a thorough analysis on different axes: by examining both nodes inside and outside of the *changeInitiator*'s process instances, we can identify the impacts on the elements in the scope of a process instance or in other process instances; by examining all existing task nodes - completed and current - we can know which elements are really impacted.

Our change-aware BAPE can handle change situations conform to two change patterns correction and evolution. These patterns directly depend on the type of the change which can be minor or major. In the next we describe each pattern in details.

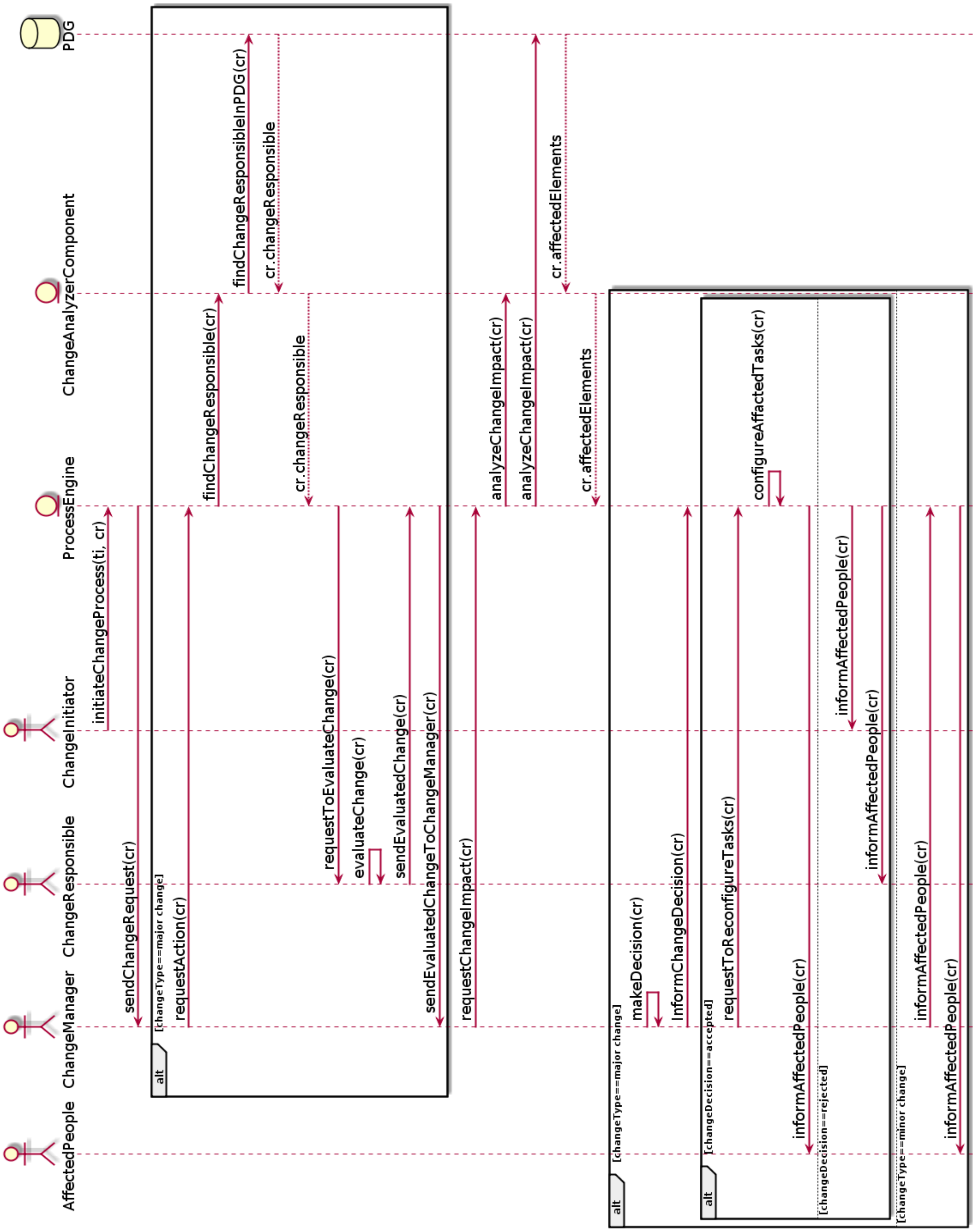


Figure 4.7: Procedure of change-aware BAPE

4.4.2.1 Correction Pattern

As illustrated in the Figure 4.8, this pattern concerns a situation when a process actor faces a *minor change* during his task t_2 in order to produce its output artifact. To continue his task, he needs to change (correct) his input artifact art_1 which is produced by the completed task t_1. At the moment of change, tasks t_3 and t_4 are in parallel with t_2.

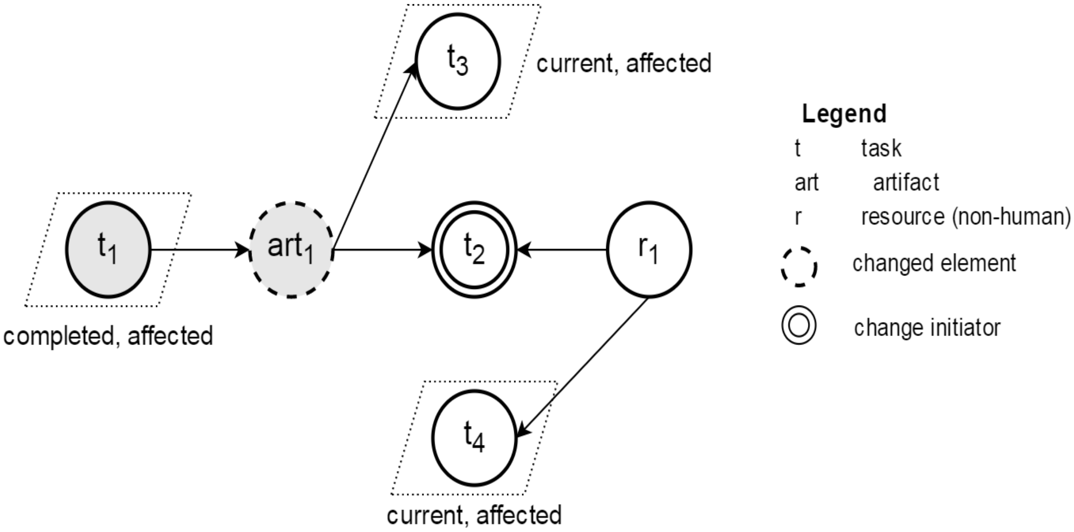


Figure 4.8: Correction Pattern

First, Change Impact Analyzer component carries out a backward analysis to detect the completed elements concerned by the change that need to be informed but do not need to rework on the changed element. Backward analysis is implemented by the traversing the PDG through incoming data edge of changed element art_1 that results in t_1.

Afterwards, it runs a forward analysis to deduce the current elements that can be concerned by the change. Forward analysis is implemented by recursively traversing the PDG through all outgoing edges of changed element. According to the pattern, based on the shared artifact art_1, forward analysis gives us t_3 as an affected element. The correction on the art_1 can add delay time on completing task t_2. Consequently, all the tasks needing resource r_1 (e.g., t_4) will be impacted by the respective delay.

In order to illustrate an example of correction pattern, we consider the following scenario that happens in our examined process.

Execution scenario θ
We imagine a same set-up as we had in scenario Δ.
We suppose that in P_1, *Electrical Designer* ED_1 is using artifact ES_1 to produce the artifact *Electrical Design model.* Moreover the artifacts *Component Requirement* CR_1 and CR_2 are produced by *Analyst* A_1, and *Wiring Team* WT_1 and WT_2 are waiting for the test bench TB_1 for commencing theirs tasks *Supply Component* SC_1 and SC_2.
We suppose that in P_3, *Wiring Team* WT_4 and *Bench Coordinator* BC_3 are waiting for the test bench TB_1 for commencing theirs tasks respectively *Wire Component* WC_3 and *Define Bench Specification* DBS_3.

Correction change scenario
The change happens in P_1 when the *Instrumentation Team* IT_2 faces a problem during his task *Define Bench Specification* DBS_1 to prepare the artifact *Bench Specification* for TB_1. This problem requires a minor change on his input artifact $FICD_1$ which had been previously produced by IT_1 in the task *Define Instrumentation Model* DIM_1.

Table 4.2 shows the result of change impact analysis based on the correction pattern regarding the respective scenario. As a result of impact analysis, the impacted actors inside P_1 (i.e., ED_1, BC_1, WT_1 and WT_2) as well as outside P_1 (i.e., WT_4 and BC_3) must be informed of the change. The figure 4.9 illustrates the impact graph extracted from the traversal corresponding to this scenario.

4.4.2.2 Evolution Pattern

The pattern in Figure 4.10 presents a situation when a major change occurs, e.g., normally caused by the environment of the project. This major change is requested on the artifact art_1 which is previously produced by t_1. The change necessitates the rework of t_1 by concretely creating a new task instance t'_1 to modify art_1 in order to produce the new version of art_1. As the consequence of such rework, other current or completed tasks may feel considerable impact, i.e., completed tasks need to rework in order to produce the artifacts coherent with the change and current/running tasks may need more time regarding their task execution in order to integrate the change in their current tasks.

As explained in the previous chapter, our process environment facilitates the rework for process actors, i.e., process actors can create new activity/task instances. Therefore, the change responsible can create t'_1 as a new instance of t_1 and initiates the change request entailing the changed element art_1. Change Impact Analyzer performs a forward

Table 4.2: Correction scenario

Pattern Element	**Process Element**	**Status**	**Responsible Process Actor**
t_1	DIM_1	completed	IT_1
	OIM_1		A_1
t_2	DBS_1	current	IT_2
t_3	SEM_1	current	ED_1
t_4	FC_1	current	WT_1
	FC_2		WT_2
	WC_3		WT_4
	DBC_3		BC_3
art_1	$FICD_1$	completed	IT_1
r_1	TB_1	current	WT_1
			WT_2
			WT_4
			BC_3

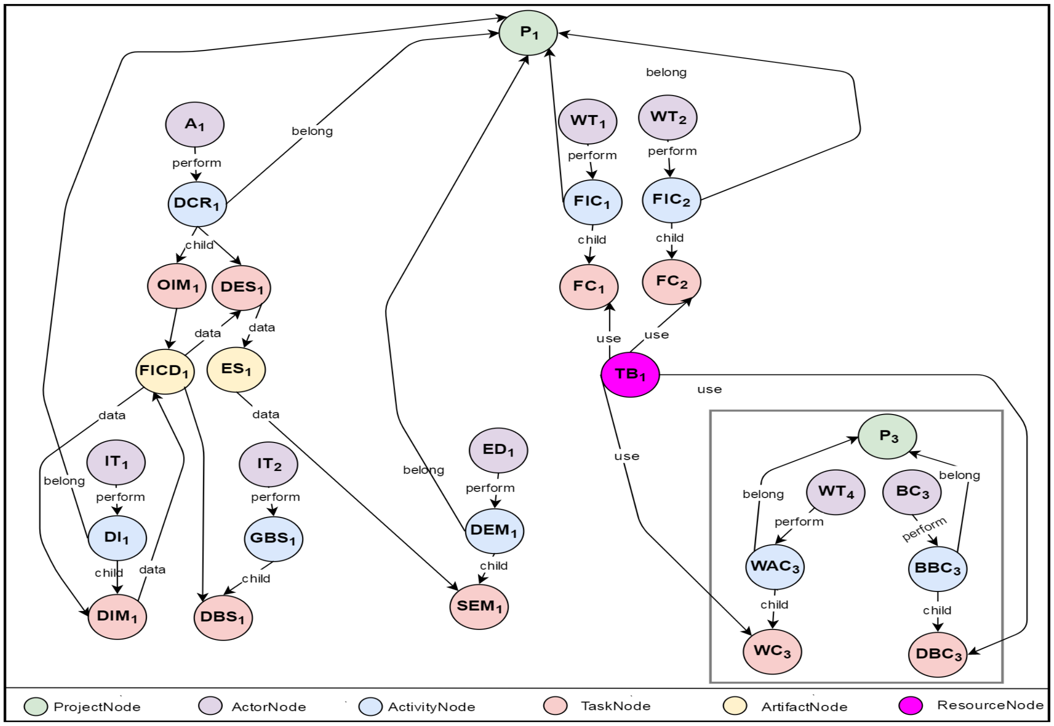


Figure 4.9: Impact Graph of scenario θ

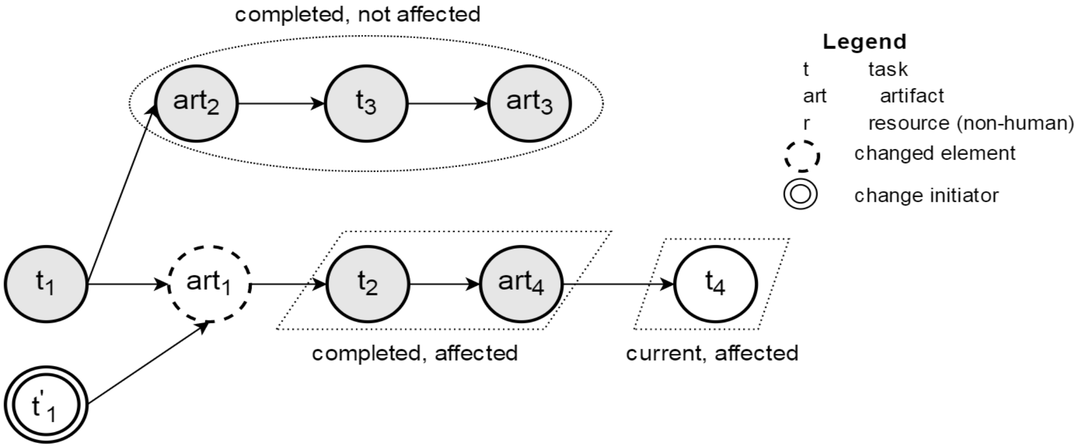


Figure 4.10: Evolution Pattern

analysis by traversing the PDG through outgoing data edges of changed element art_1 to detect all the elements that are dependent on art_1. As shown in the Figure 4.10, this analysis gives the list of impacted elements comprising the completed elements t_2 and art_4 and the current task t_4. However, the rework on art_1 does not have any impacts on elements t_3, art_2 and art_3.

Normally, major changes require an in-depth analysis to better estimate the impact. These changes may consider situations regarding system evolution or new business regulations where the impact must be perceived on all current and completed process elements. Thus, it is important to identify the relevant and irrelevant elements to a change. This issue is especially demanding in the situation where the changed element is a collection of artifact and is related to a task which has been enacted multiple times. Concretely each instance of the task is consuming one element in the collection of artifact. Concretely, if the type of art_1 is collection and t_2 has been enacted several times, when reworking art_1, it may happen that only some of its elements are modified. In this case, it would be useful to identify which instances of t_2 will be affected by the changed elements in art_1.

To better illustrate the evolution pattern we reconsider the execution scenario Δ where a major change happens as following.

Evolution change scenario

In P_1, the change happens when A_1 needs to enact once again the task *Specify Component Requirements SCR* to solely modify the previously completed artifact CR_1. Note that the task instance SCR_1 has previously produced two artifacts CR_1 and CR_2 but in this scenario we suppose that only the element CR_1 which is an input of task instance SC_1 is considered as the *changeElement*.

Table 4.3 shows respective scenario of the pattern *Evolution* which emphasizes on the multi-instance elements. Figure 4.11 illustrates the IG corresponding to this scenario that Change Impact Analyzer provides to the change manager by traversing the PDG. The impacted actors are distinguished by the cause of impact: A_1 as the change responsible and S_1 as the one has directly impacted, are affected by the change on their artifacts concerning the components C_1, BC_3. WT_4 in process instance P_3 are affected by sharing the testbench TB_1 with WT_1. Moreover, WT_3 in P_2 is also affected by using the shared artifact C_1. We can see that the actors S_2 and WT_2 in P_1 and the actor WT_5 in P_4 who worked on the component C_2 are not affected by the change.

Table 4.3: Evolution scenario

Pattern Element	**Process Element**	**Status**	**Responsible Process Actor**
t_1	SCR_1	completed	A_1
t'_1	SCR'_1	current	A_1
t_2	SC_1	completed	S_1
	FC_1		WT_1
	FC_3		WT_3
t_3	SC_2	completed	S_2
	IC_2		WT_3
	FC_5		WT_5
t_4	WC_1	current	WT_1
	DBS_3		IT_4
	WC_3		WT_4
art_1	CR_1	completed	A_1
art_2	CR_2	completed	A_1
art_3	C_2	completed	S_3
	IC_2		WT_3
	IC_5		WT_5
art_4	C_1	completed	S_1
	IC_1		WT_1
	IC_3		WT_3

The obtained IG is considered as the base to conduct impact analysis at different levels according to a specific need of the change manager. For instance, IG can show only the dependencies at process, task, data or resource (process actor and non-human resources) levels. Another analysis can be regarding the in-degree and out-degree of the nodes of the IG. Obviously, a node with maximum in-degree is the most affected

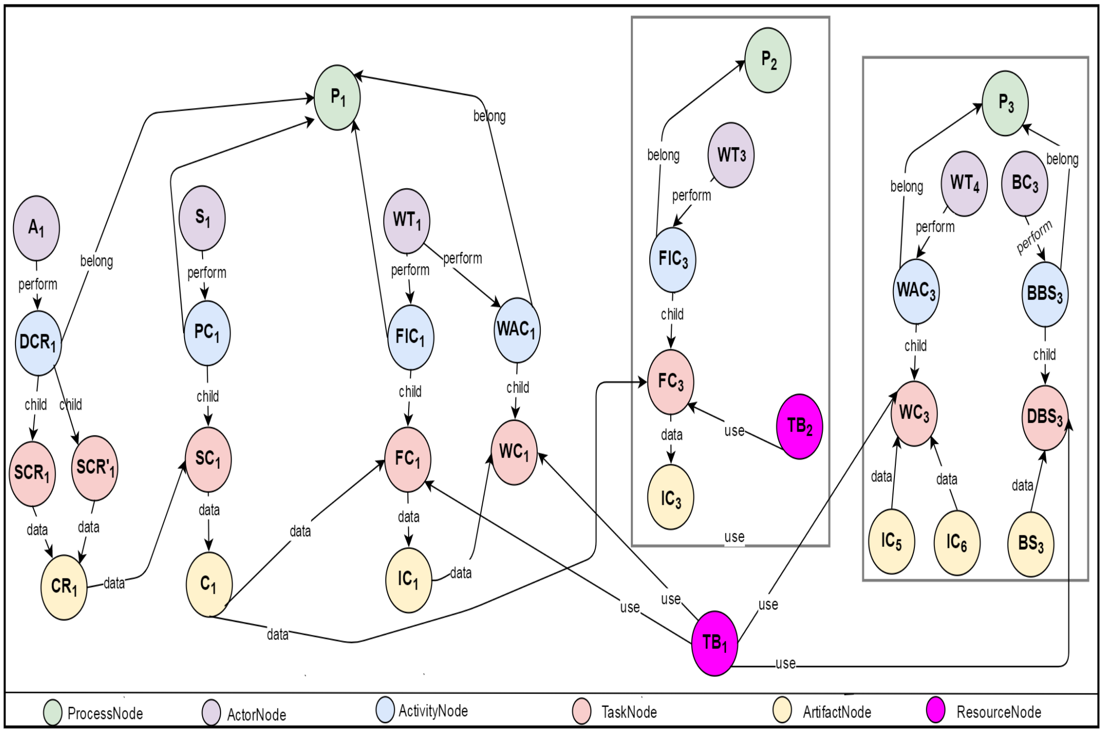
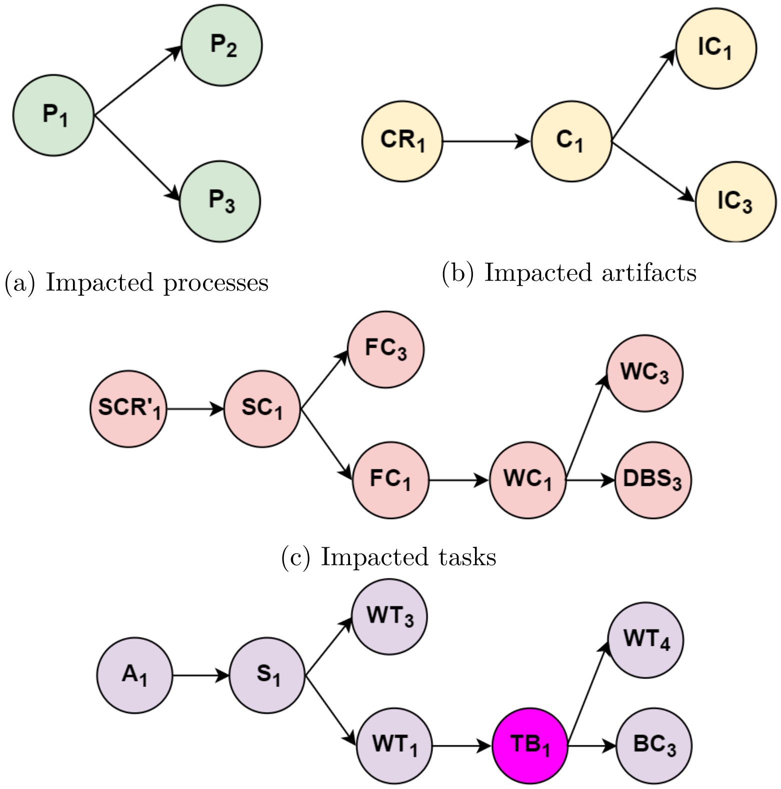


Figure 4.11: Impact Graph of scenario Δ

element of the change and a node with maximum out-degree is considered as a crucial element which propagates the change throughout other affected elements. These analysis can be easily achieved by defining some queries that look up the IG from different points of interest, i.e., by traversing the impact graph and extracting specific nodes such as *ProcessNodes, ArtifactNodes, etc.* For instance, Figures 4.12a, 4.12b, 4.12c and 4.12d respectively illustrate dependencies at process, artifact, task and resource levels regarding the impact graph of scenario Δ.

4.5 Assessing the Impact of Change

The IG stores the affected entities of the process change. In principal, we can go further in such analysis by annotating the nodes of the IG with some interesting metrics defined by user according to specific needs of change impact analysis [11]. The applied metrics are necessary to derive the strength of side-effects of change on certain aspects. This enrichment of IG can facilitate understanding of the impact of change for the change



(a) Impacted processes

(b) Impacted artifacts

(c) Impacted tasks

(d) Impacted resources (human and non-human)

Figure 4.12: Impact Graphs of scenario Δ in different levels

manager and help him on procedure of decision making about the change.

In order to illustrate the benefits regarding usage of metrics in evaluating the change, we consider two following cases that may take place in the scenario Δ:

- Case 1 as a result of the change, Analyst A_1 estimates two days for the rework task SCR'_1 in order to produce the new version of CR_1.
- Case 2 thanks to the availability of Analyst A_1, the rework task SCR'_1 requires only 3 hours to be completed. Also we assume that there is an idle testbench TB_4 in the system.

The Impact Graph derived from both cases would be the same, however the impact of the change in the case 1 is far more than in case 2. The reason is the delay time that results from the rework in both cases. Moreover, importance and availability of the resources affected by the change can be significant on estimating the strength of the change impact. For instance, the change is propagated from P_1 to P_2 through only available testbech TB_1. Therefore, availability of testbench TB_4 in case 2 can reduce the impact of change between P_1 and P_2. Therefore, enriching the IG by useful information can aid the process of decision-making about the change.

In our proposal we adopt the Quality of Service (QoS) metrics to assess the impact of change. According to [2], QoS metrics can be classified into quantitative and qualitative metrics. By applying the QoS metrics into process change context, we assume that applicability of QoS metrics is entirely associated with the degree of change. This means that quantitative metrics are more related to ad-hoc changes (changes at enactment-time) and qualitative metrics are consolidated with changes that occur in the infrastructural level. Since our work addresses the problem of unofficial changes at enactment-time, we apply four quantitative metrics known as *Artifact Completion Percentage*, *Delay Time*, *Realization Cost* and *Resource availability.* These metrics are defined as follows:

- *Artifact Completion Percentage ACP* For any affected artifact, the percentage of its completion at the time of change can be estimated based on the duration of execution of the task that produces it. For instance, if a task producing the affected artifact is completed, the respective artifact will have the maximum impact.

- *Delay time DT* It corresponds to the delay time that a change causes as a result of reworking. The delay time can be in occupying a resource, producing an artifact and/or participation of an actor in other activities. As mentioned earlier, when a process actor evaluates a change, he estimates the temporal delay regarding the change, i.e., resulted from reworking. As IG entails the temporal information such as task duration and start time of task instances, the delay time can be propagated through IG.

- *Realization cost RC* It represents the cost of resources (process actors and non-human resources) used to perform the tasks of the processes. The information regarding the resources can be obtained from the Resource Management System (RMS). Information about realization cost is highly domain-specific and each company has their own strategy to estimate it. In our work, realization cost of a resource *res* is defined as RC_{res} which is sum of two elements [2]: labor cost LC_{res} is the cost associated with each role of the domain and resource expertise RE_{res} is a factor to differentiate actors playing the same role based on their expertise or non-human resources based on their quality.

- *Resource availability* RA_{res} This metric represents the list of idle resources having the same type as *res* at the time of change [28]. This information can be beneficial in a situation that a change affects an actor or a non-human resource who is participated in several process instances, e.g., testbench TB_1 in case 2 of scenario Δ. At this situation, information of resource availability can help the change manager to allocate the idle resource to one of the affected process instances in order to decrease the impact of change.

By considering the above properties, we have defined sample realization cost RC for the resources of the scenario Δ for cases 1 and 2 (for the sake of simplicity, actor expertise $RE = 0$ for all resources). The result is illustrated in Table 4.4.

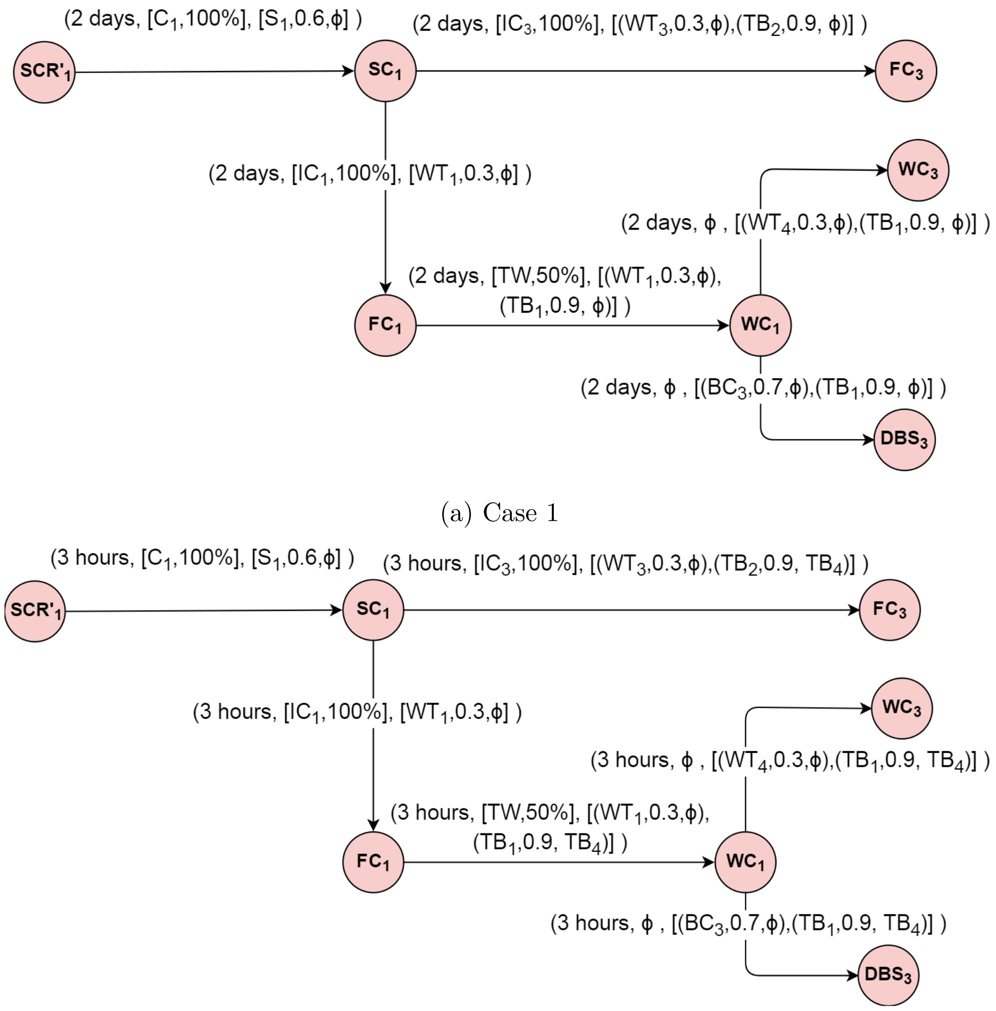
In order to enrich the IG with mentioned metrics, we consider the task-level impact graph. The enriched impact graph is defined as follows:

$IG = (V,\ E)$
$\forall e_{ij} \in E \mid e_{ij} = (t_i, t_j, DT, artList_j, resList_j)$ where
t_i is the source TaskNode,
t_j is the target TaskNode,
DT is the delay time,
$artList_j$ is the list of affected $ArtifactNode$s where each one has two properties as the name and artifact completion percentage (ACP),
$resList_j$ is the list of affected $ActorNode$s and $ResourceNode$s where each one has three properties as the name, realization cost RC_{res} and resource availability RA_{res}.

Table 4.4: Resource information of scenario Δ for cases 1 and 2

Resource		RC	RA($case_1$)	RA($case_2$)
Human	Analyst	0.6	ϕ	ϕ
	Supplier	0.4	ϕ	ϕ
	Electrical Designer	0.7	ϕ	ϕ
	Wiring Team	0.3	ϕ	ϕ
	Instrumentation Team	0.5	ϕ	ϕ
	Bench Coordinator	0.7	ϕ	ϕ
Non-human	Testbench	0.9	ϕ	TB_4

The enriched task-level IGs of the scenario Δ for cases 1 and 2 are illustrated respectively in Figures 4.13a and 4.13b. Thanks to the enrichment, the change manager can better assess the impact of change. This assessment is mainly about delay time, affected artifacts with their completion percentage and resource cost and availability. For instances, in case 1, delay time propagated through tasks is 2 days compared to case 2 where the delay time is only 3 hours. In both cases artifact completion percentage is kept same. Another difference between two cases is regarding resource availability. any change imposed on the testbench TB_1 in case 1 that may be concerned as a crucial resource ($RA_{testbench} = \varnothing$) can increase the strength of the change impact. Existence of the testbench TB_4 in case 2 may lead to allocation of the tasks WC_3 and DBS_3 of P_2 to TB_4 in order to cut the impact of change propagation in process instance P_2. Based on these assessment metrics, acceptance of the change in case 2 can be more reasonable than in case 1 for the change manager.


(2 days, [C1,100%], [S1,0.6,ϕ])
(2 days, [IC3,100%], [(WT3,0.3,ϕ),(TB2,0.9, ϕ)])
SCR'1
SC1
FC3
(2 days, [IC1,100%], [WT1,0.3,ϕ])
WC3
(2 days, ϕ , [(WT4,0.3,ϕ),(TB1,0.9, ϕ)])
(2 days, [TW,50%], [(WT1,0.3,ϕ),
(TB1,0.9, ϕ)])
FC1
WC1
(2 days, ϕ , [(BC3,0.7,ϕ),(TB1,0.9, ϕ)])
DBS3
(3 hours, [C1,100%], [S1,0.6,ϕ])
(3 hours, [IC3,100%], [(WT3,0.3,ϕ),(TB2,0.9, TB4)])
SCR'1
SC1
FC3
(3 hours, [IC1,100%], [WT1,0.3,ϕ])
WC3
(3 hours, ϕ , [(WT4,0.3,ϕ),(TB1,0.9, TB4)])
(3 hours, [TW,50%], [(WT1,0.3,ϕ),
(TB1,0.9, TB4)])
FC1
WC1
(3 hours, ϕ , [(BC3,0.7,ϕ),(TB1,0.9, TB4)])
DBS3

(a) Case 1

(b) Case 2

Figure 4.13: Enriched task impact graphs of case 1 and case 2

4.6 Related Work

Change management can be tackled from different perspectives, such as process perspective, tool perspective and product perspective [15, 5]. According to [15], tools and methods to support the change process can be divided into two groups: (1) those that help managing the process or documentation of the process and (2) those which support process actors in making decisions at a particular point in the engineering change process (e.g. the risk/impact analysis phase).

We use this structure to discuss some similar works on the tool perspective in Business Process, System and Software Engineering communities.

4.6.1 Process/Documentation Support

Computer-based tools have been recognized as an essential to support engineering changes [13]. In terms of academic works, Chen et al. [3] proposed a tool to support distributed engineering change management linking with concurrent engineering. Lee et al. [17] introduced a prototype for collaborative environment for engineering change management which combines ontology-based representations of engineering cases, case-based reasoning for retrieval and a collaboration model. In business process community, many of existing works on change management focus on proposing mechanisms to enable process adaptation and changes propagation [24, 1, 19, 21]. However, these studies focus on facilitating the change implementation but the analyzing impact of change is outside the scope of their works.

4.6.2 Decision Making Support

A wide variety of techniques are used in the context of impact analysis and change propagation [13, 26] in the engineering domain. There is currently no commercial package that helps predict the effect of a change, however some work is being carried out in academic institutions [15]. Eckert et al. [4, 9] based on a study conducted in Westland Helicopters, identified two types of changes: the emergent changes and the initiated changes. This particular study was based on the interviews conducted with the company's employees. They proved that by capturing the design knowledge and experience (e.g. source of change, interdependencies between parts and systems, etc.), in the form of experienced designers in the company, an automatic tool to identify the engineering change propagation can be developed.

The respective work has further led to the development of a computer support tool by Jarrat et al. [14] to identify the risk of a change. They apply the *Change Prediction Method (CPM)* to realize how changes spread through out a product by using *Design Structure Matrix (DSM)* as the basis of the product model. The tool uses a simple model of risk, where the likelihood of a change propagating is derived from the past experience in terms of their occurrence and the impact such changes would have. A product model consisting of two numerical DSMs is created, which show the likelihood and impact of change propagation occurring between directly connected components. A route counting algorithm is then used to calculate the combined impact of change propagation, which is the sum of the direct risk and indirect impacts. This technique has been used in many other works [22, 16, 9, 18].

Reddi and Moon's [23] approach is another dependency model technique, harvesting dependencies in the early phases of design for use in later stages of the life cycle. It captures the type of change at both initiator and target as well as the likeliness of the specific change propagating between the two in terms of discrete levels (low, medium, or high). Search algorithms iterate through the model to identify all possible propagation paths.

Impact analysis of change is also an important topic in the research area of business process domain. Fdhila et al. [6, 7] propose a change propagation approach to deal with change in process choreographies. They apply *Refine Process Structure Tree (RPST)* to define public and private models of involved partners as well as achoreography model. When applying a change operation to a partner's private model, they extract all message exchange activities concerned by the change. The list of affected interactions is analyzed to identify all affected partners involved. Then, for each of these partners, a relative change computation is accomplished to determine the changes to be propagated. Then, a negotiation phase is launched with each affected partner. Their work is solely restricted to modeling-time by analyzing the models and extracting the affected partners and their change impact analysis is limited to inside one process.

Muler et al. [20, 21] deal with logical failures management in inter-workflow collaboration scenarios by investigating temporal and qualitative implications of workflow adaptation. They propose a predictive strategy estimating whether constraints for delivery times or result qualities will be violated due to the dynamic adaptation. Temporal implications of an adaptation are determined by estimating the duration required to execute the dynamically adapted workflow and by comparing it with originally fixed time constraints. If temporal constraints are expected to be violated, affected collaboration partners are informed immediately. For qualitative implication, they introduce so-called quality-measuring objects. These numerical objects are used in workflow's data flow in order to measure the quality of result provided by the workflow. To this aim, they extend the workflow model by adding communication nodes COMM-IN and COMM-OUT which respectively specify when information has to be sent to some collaboration partners and when information is expected to be received from some collaboration partners. The metrics used in their approach are for deriving the essence of adaptation not for measuring the impact of the adaptation. Impact of change among process instances is not investigated as they consider only the impact of adaptation inside one process instance.

Wang et al. [27] study the impact of change between services and business processes which expose services to business partners. They develop a service-oriented business process model, based on which the taxonomy of change types (i.e., service changes, process changes) is provided. Then a generic solution for identifying, analyzing, and reacting to various types of changes in business processes and services is also developed. The prototype SCA (Service Change Analyzer) implements their change management approach for analyzing the change impact. SCA focuses on a type of dependency that multiple services are supported by a single business process. For each input service change, the SCA calculates the impact scopes and provides suggestions for the possible change impact patterns. These analysis result for a specific service change help developers to understand the direct impact and also the cascading impact in services and their supporting business processes.

4.6.3 Synthesis

Process support approaches focus on implementing the change by allowing process actors to change the model and propagate the change to the running instances. They neglect the fact that a change can impact other process elements particularly at enactment-time.

Decision making support approaches like us focus on investigating the impact of a change. In the following, we define their shortcomings compared to our proposed change management approach.

- *Integrating the change management into process management* Most presented approaches lack the integration of change management into process managements. Some only integrate change management into the modeling phase by enabling the analysis the change impact in the model [6, 7, 23, 8].
  In our approach, the change management is integrated with process management, i.e., particularly enactment phase. This brings great advantages as:

  - *Signaling change in a systematic manner* In our process environment, process actors can easily signal a change request through their enactment UI. In fact, we enable process actor to integrate emergent changes, which are not described in their process model, into the process execution.
  - *Informing affected people in a timely manner* After extracting the result of impact analysis, our process environment aids change manager to inform all affected people of the change. This result in avoiding producing the obsolete work and reduce the cost of rework.

- *Usage of enactment-time information* Generally, the impact of a change is analyzed based on the dependencies among process elements, i.e., dependencies based shared artifacts and resources (human and non-human). Mostly all discussed works analyze the impact of change based on dependencies at model-level [6, 7, 23, 8]. In fact, by assuming a change happening in the model (e.g., a change on an artifact definition), their approaches can extract affected process elements of the change from the process model. The result will be regarding which task definitions and which roles or resource definitions can be impacted by the change. This type of impact analysis limits the precision of the analysis result. There are particular information that uniquely emerge at enactment-time which make dependencies among process elements, i.e., information regarding collection of artifacts, multi-instance tasks, concrete resources (process actors, machines, etc). Moreover,

modeling-time analysis cannot provide any information about current state of process element instances, i.e., task, artifact and resource states.
In our approach, impact analysis is conducted at instance-level. This leads to have a more precise impact analysis where we can precisely extract concrete dependencies among process elements, i.e., concrete artifacts and resources instances. This is achieved via PDG which stores instance-level information regarding process elements, particularly representing the instance-level dependencies among them. Thanks to the graph structure of PDG and developed traversal algorithms, affected process elements of the change can be efficiently extracted and informed.

- *Analysis the impact of change inter and intra processes* All discussed works limit the impact analysis to one process(including its sub-processes). They neglect the fact that a change can be propagated inside a process and among other processes as well. In fact, several projects may share artifacts and resources where a change on one can impact the others as well, e.g., a process actor or a resource who is planned to be used in two process instances. In our approach, as the impact analysis is conducted in a PDG having the global view of system, impact of changes can be easily extracted among process instances (inter-process) based on their shared artifacts or resources.

4.7 Summary

This chapter introduced our change management approach which provides support for capturing in a centralized and continuous way all change requests sent asynchronously by process actors. Then, we provide a mechanism to enable analyzing the potential impacts of a change on the whole system and finally notifying process actors affected by the change in a timely and systematically manner. To do so, we extended BAPE by defining our change manage policy as a separate process and implemented its activities which turn BAPE to a change-aware process environment. We extend the task state machine in order to enable process actors to send change requests while executing their tasks and developed interfaces to support the change. A Change Analyzer Component implementing traversal algorithms (correction and evolution) were developed in order to traverse the PDG and extract the affected elements of change which are illustrated in a graph so-called Impact Graph (IG). The obtained IG is considered as the base to conduct impact analysis at different levels according to a specific need of the change manager, e.g., process, task, etc. We discussed the feasibility of going further in such analysis by annotating the nodes of the IG with some interesting metrics such as resource cost or delay time. By conducting the change impact analysis at instance-level, we can achieve to a more precise and accurate analysis regarding affected process elements.

Bibliography

[1] AristaFlow. http://www.aristaflow.com/.

[2] J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1:281–308, 2004.

[3] Y.-M. Chen, W.-S. Shir, and C.-Y. Shen. Distributed engineering change management for allied concurrent engineering. *Int J Comput Integr Manuf*, 15(2):127–151, 2002.

[4] C. Eckert, P. Clarkson, and W. Zanker. Change and customisation in complex engineering domains. *Research in Engineering Design*, 15(1):1–21, March 2004.

[5] C. Eckert, P. J. Clarkson, and W. Zanker. Change and customisation in complex engineering domains. *Research in Engineering Design*, 15(1):1–21, 2004.

[6] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert. Dealing with change in process choreographies: Design and implementation of propagation algorithms. *Information Systems*, 49:1–24, April 2015.

[7] W. Fdhila, S. Rinderle-Ma, and M. Reichert. Change propagation in collaborative processes scenarios. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 452–461, Oct 2012.

[8] T. I. Grantham-Lough K, Stone MC. Prescribing and implementing the risk in early design (red) method. In *Proceedings of ASME 2006 International Design Engineering Technical Conferences*, volume 4a, pages 431–439, Philadelphia, Pennsylvania, USA, 2006. American Society of Mechanical Engineers (ASME).

[9] M. Griffin, O. de Weck, G. Bounova, R. Keller, C. Eckert, and P. J. Clarkson. Change propagation analysis in complex technical systems. *Journal of Mechanical Design*, 131(8):081001, 2009.

[10] C. Gupta, Y. Singh, and D. S. Chauhan. Dependency based process model for impact analysis: A requirement engineering perspective. volume 6, pages 28–33. International Journal of Computer Applications, 2010.

[11] M. Hajmoosaei, H. N. Tran, C. Percebois, A. Front, and C. Roncancio. Impact analysis of process change at run-time. In *24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2015, Larnaca, Cyprus, June 15-17, 2015*, pages 156–161, 2015.

[12] M. Hajmoosaei, H.-N. Tran, C. Percebois, A. Front, and C. Roncancio. Towards a change-aware process environment for system and software process. In *Proceedings of the 2015 International Conference on Software and System Process*, ICSSP 2015, pages 32–41, New York, NY, USA, 2015. ACM.

[13] G. Huang and K. Mak. Computer aids for engineering change control. *Journal of Materials Processing Technology*, 76:187 – 191, 1998.

[14] T. A. W. Jarratt, C. M. Eckert, P. J. Clarkson, and L. Schwankl. Product architecture and the propagation of engineering change. In *7th International Design Conference (Design 2002)*, pages 75–80, 2002.

[15] T. A. W. Jarrett, C. M. Eckert, N. H. M. Caldwell, and P. J. Clarkson. Engineering change: an overview and perspective on the literature. *Research in Engineering Design*, 22(2):103–124, April 2011.

[16] S. F. KöNigs, G. Beier, A. Figge, and R. Stark. Traceability in systems engineering - review of industrial practices, state-of-the-art technologies and new research solutions. *Adv. Eng. Inform.*, 26(4):924–940, Oct. 2012.

[17] H. J. Lee, H. J. Ahn, J. W. Kim, and S. J. Park. Capturing and reusing knowledge in engineering change management: A case of automobile development. *Information Systems Frontiers*, 8(5):375–394, 2006.

[18] S. Li and L. Chen. Pattern-based reasoning for rapid redesign: a proactive approach. *Research in Engineering Design*, 21(1):25, 2009.

[19] M. Minor, R. Bergmann, S. Görg, and K. Walter. Towards case-based adaptation of workflows. In *Proceedings of the 18th International Conference on Case-Based Reasoning Research and Development*, ICCBR'10, pages 421–435, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] R. Müller and E. Rahm. *Dealing with Logical Failures for Collaborating Workflows*, pages 210–223. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[21] R. Müller, U. Greiner, and E. Rahm. Agentwork: a workflow system supporting rule-based workflow adaptation. *Data and Knowledge Engineering*, 51(2):223 – 256, 2004.

[22] S. Oh, B. Park, S. Park, and Y. S. Hong. Design of change-absorbing system architecture for the design of robust products and services. In J. A. Jacko, editor, *HCI (4)*, volume 4553 of *Lecture Notes in Computer Science*, pages 1110–1119. Springer, 2007.

[23] K. R. Reddi and Y. B. Moon. A framework for managing engineering change propagation. *International Journal of Innovation and Learning*, 6(5):461–476, 2009.

[24] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies.* Springer, Berlin-Heidelberg, 2012.

[25] H. N. Tran, M. Hajmoosaei, C. Percebois, A. Front, and C. Roncancio. Integrating run-time changes into system and software process enactment. *Journal of Software: Evolution and Process*, 28(9):762–782, 2016.

[26] I. Ullah, D. Tang, and L. Yin. Engineering product and process design changes: A literature overview. *Procedia CIRP*, 56:25 – 33, 2016. The 9th International Conference on Digital Enterprise Technology – Intelligent Manufacturing in the Knowledge Economy Era.

[27] Y. Wang, J. Yang, W. Zhao, and J. Su. Change impact analysis in service-based business processes. *Service Oriented Computing and Applications*, 6(2):131–149, 2012.

[28] J. Xu, C. Liu, X. Zhao, S. Yongchareon, and Z. Ding. Resource management for business process scheduling in the presence of availability constraints. *ACM Trans. Manage. Inf. Syst.*, 7(3):9:1–9:26, Oct. 2016.

Chapter 5

Evaluation

Contents

This chapter introduces the prototype of our developed process environment BAPE along with the experiments conducted with BAPE regarding process modeling, process enactment and change management. The main components of BAPE are presented in Section 5.1. Section 5.2 illustrates two experiments, regarding the real process of our industrial partners, conducted with BAPE in the context of ACOVAS project. These experiments evaluate the modeling, enactment and change management support provided by BAPE. Finally, Section 5.3 summarizes this chapter.

5.1 Proof-of-Concept Prototype

To demonstrate the technical feasibility of the propose framework, this section presents the proof-of-concept prototype of BAPE. It aims at demonstrating the functionalities of BAPE and evaluating how end-users (e.g., process owner, process actor and change manager) will perceive this framework.

5.1.1 Overall Architecture

BAPE is a Java-based tool developed based on PAC architectural pattern (Presentation-Abstraction-Controller) [2, 3] as illustrated in the Table 5.1.

Table 5.1: BAPE's components based on PAC pattern

PAC Pattern	BAPE
Controller	Process Engine
Abstraction	SPML Process models
	Company Assets
	PDG
Presentation	Modeling UI
	Enactment UI
	Change Management UI
	Monitoring UI

In PAC pattern, controller is considered as the core. Usage of this pattern allows us to separate out our application into two major components as Presentation and Abstraction which can be replaced or worked on in relative isolation. This makes BAPE an extensible and pluggable framework, which can be easily replaced and customized. The Presentation and Abstraction components never speak to each other. The Controller takes input, not the Presentation component. The Controller has all the business logic and routing information. The Presentation component is essentially just a filter that takes and display raw data that the Controller pushes through it. Because there is a separation between the Presentation and the Abstraction, with a well-defined way of communicating (through controller), this allows multiple presentations (i.e., user interfaces or views) of the same underlying data.

5.1.2 Process Engine as Controller

The process engine is the heart of BAPE. It is a light-weight engine that executes processes and can be deployed on almost any device that supports a simple Java Runtime Environment (JRE).

According to PAC pattern, process engine as the controller plays the role of event listener and handler. In BAPE, process engine mainly listens to two types of events as:

- Events initiated from different presentations such as modeling, enactment, change management and monitoring.

- Events initiated from external systems such as DBMS and RMS.

Process engine acts as an interface between Abstraction and Presentation components to handle all incoming events, manipulates data using the Abstraction component and interacts with the Presentation component to render the final output. As an example, the enactment events are defined in the process, activity and task state machines which are implemented in the process engine and triggered from the enactment interface. The code snippet 5.1 illustrates some implemented methods regarding the Controller, Abstraction and Presentation components of BAPE.

```
Controller:

// this method defined in the activity state machine, updates the PDG
// and enactment presentation of the respective process actor
public void updateAbstractionView(ActivityCreateEvent event)

Abstraction:

// this method deletes an activity from the process fragment model
public void deleteActivity(String process, String role,
String activity)

// this  method  starts execution of a task by verifying the state of
// required artifacts in the PDG
public boolean taskStart(long taskID, List<ArtifactType> artifacts)

Presentation:

// this method  updates the activity instance table by adding
// information of a new enacted activity instance
public void reloadActivityInsTable(Activity actIns)

// this method provides the result of change impact in the impact
// analysis table
public void reloadChangeImpactAnalysisTable(ChangeRequest cr)
```

Code 5.1: Some specifications of PAC methods

5.1.3 End-User Interfaces

This section gives an overview of the user interfaces provided by the developed proof-of-concept prototype. BAPE is comprised of four user-friendly interfaces as *Modeling UI*, *Enactment UI*, *Change Management UI* and *Monitoring UI*. To develop the interfaces, we investigated the usage of graphical (i.e., window-based) and textual (i.e., xtext editor) forms. We chose graphical interfaces according to the feedback of our industrial partners.

However, as mentioned earlier, usage of PAC architecture makes it very easy to plug other forms of presentations into BAPE.

Java provides a rich set of libraries to create graphical user interfaces in a platform independent way. We adopt Java Swing API which is a part of Java Foundation Classes (JFC) that is used to create window-based applications. Java Swing provides rich controls, lightweight and highly customizable components which follow PAC architecture.

Each user interfaces exposes different functionalities of BAPE to end-users (i.e., process actor, process owner and change manager). For instance, users can model the company assets and their fragments of process through *Modeling UI*; they can enact their processes and processes' activities and tasks through *Enactment UI*; they can manage the change requests through *Change Management UI* and finally they can monitor their process executions through *Monitoring UI*.

Notice that users have different types of accessibilities to these interfaces. In order to provide better organization through the rights and accessibilities of users to BAPE's functionalities, we defined three types of users as:

- *Process Owner* has full access to make any changes, e.g., defining company assets, enacting processes, monitoring processes, etc.
- *Process Actor* carries out the activities/tasks defined in a process model. He can be either:
  - *Team Responsible* has the right to define company assets and model and enact the fragment of process belonging to his role.
  - *Executer* has the right solely to enact process activities/tasks of his fragment model.
- *Change Manager* has full access to the *Change Management UI* to handle and analysis the change requests triggered by process actors.

In BAPE, all the information regarding end-users (i.e., name, role, email, password and responsibility) is defined in the organizational model as illustrated in the Figure 5.1. In the next, we explain the usability of each developed interface in details.

5.1.3.1 Modeling UI

Authorized users (i.e., process owner and team responsible) can easily define or update the company assets and the process fragment models via SPML. Figure 5.2 illustrates the developed user-friendly *Modeling UI* dedicated to the process owner (e.g., project manager PM_1) who has the maximum rights and access to define company assets and process fragments models. For instance, Figure 5.2 illustrates modeling the *Analyst* process fragment of process *Modify Testbench Wiring* as presented in Chapter 2. Generally, *Modeling UI* comprises two panels which are described as follows:

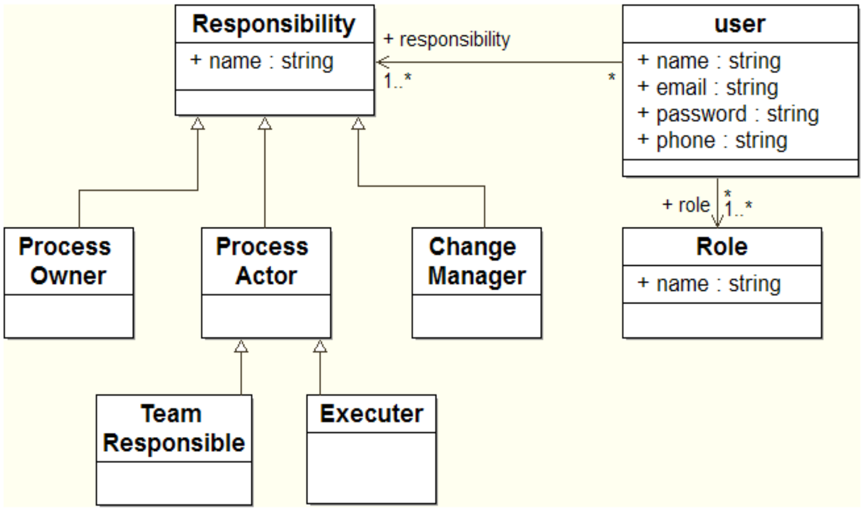
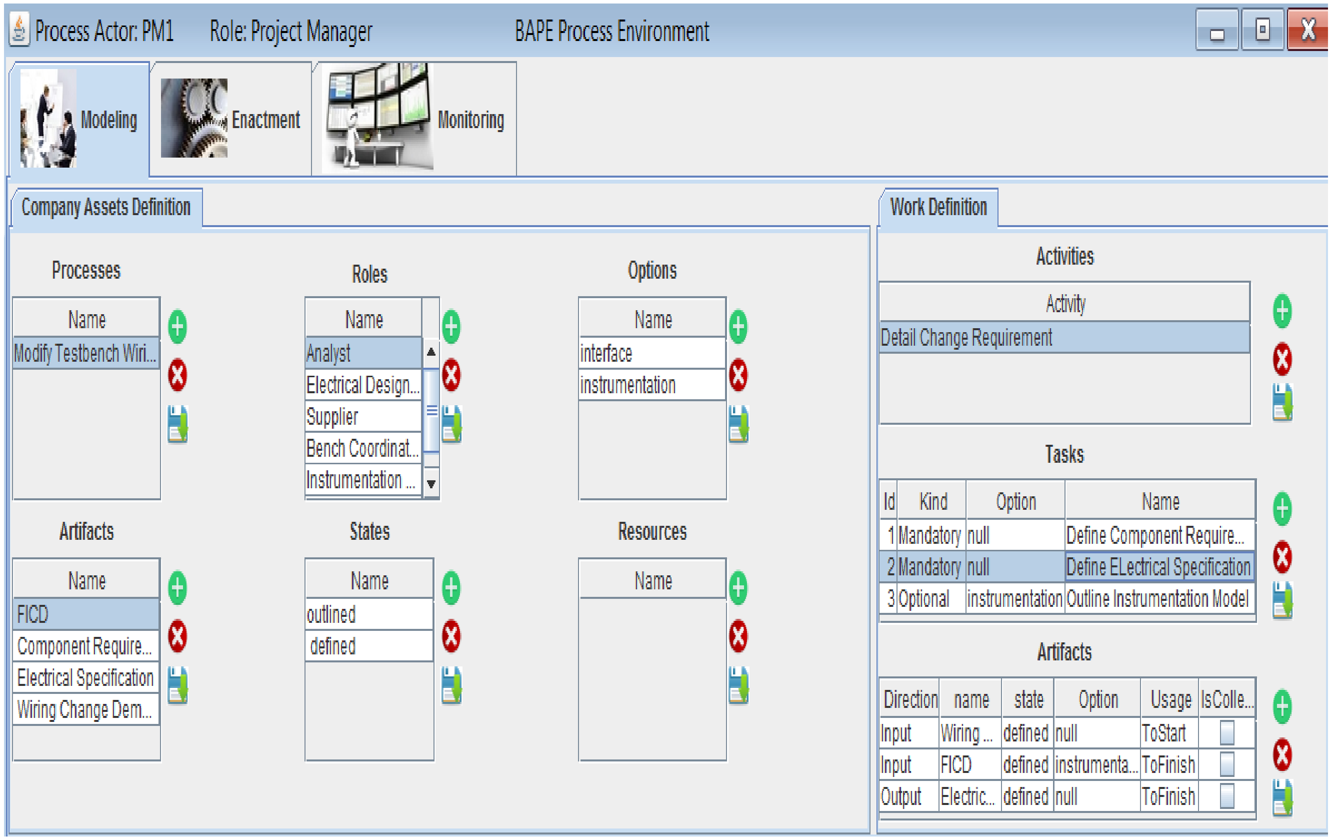


Figure 5.1: BAPE's organizational model

- *Company Asset Definition* This panel is comprised of several tables allowing an authorized user to define first the list of processes. Second, for each process, he selects the list of roles required to be involved in the respective process. Notice that information regarding the roles are extracted from the organizational model. Third, he defines the list of options, artifacts and resources needed in the process. Finally, for each artifact he defines the list of possible states that the artifact may pass through its life cycle. The company assets are stored in the process repository.

- *Work Definition* This panel is comprised of three tables which allows user to first define the list of activities belonging to his role. Then for each activity, he defines the list of options, tasks along with setting the tasks' properties as defined in SPML meta-model, e.g., *kind (mandatory* or *optional), name, etc.* For each task, he selects the list of artifacts as defined in the company assets. For each artifact, he selects the demanded state along with setting other artifact's properties as defined in the SPML meta-model. Each activity along with its tasks is added to respective process fragment stored in the process repository.

Notice that the well-formedness rules defined for SPML are built-in in the modeling interface and are verified in order to avoid inconsistencies in the process fragment models. For instance, Figure 5.3 illustrates a situation where *Instrumentation Team* IT_1 creates a task with an input and output artifacts in which both artifacts have the same state.



Process Actor: PM1 Role: Project Manager BAPE Process Environment

Processes

Name
Modify Testbench Wiri...

Roles

Name
Analyst
Electrical Design...
Supplier
Bench Coordinat...
Instrumentation ...

Options

Name
interface
instrumentation

Artifacts

Name
FICD
Component Require...
Electrical Specification
Wiring Change Dem...

States

Name
outlined
defined

Resources

Name

Activities

Activity
Detail Change Requirement

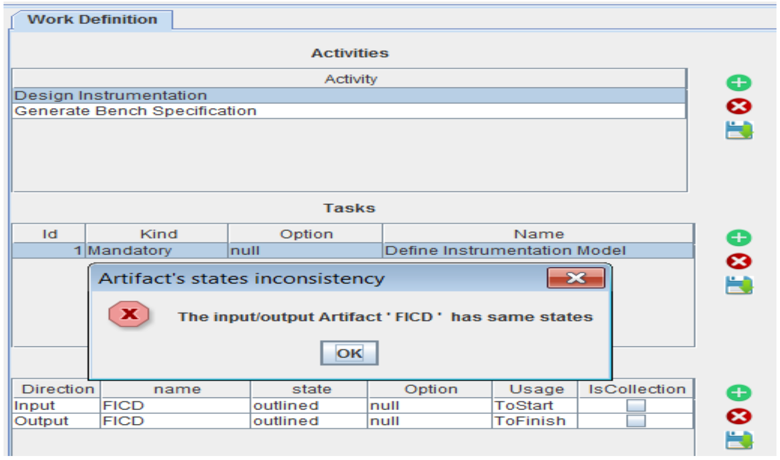
Tasks

Id	Kind	Option	Name
1	Mandatory	null	Define Component Require...
2	Mandatory	null	Define ELectrical Specification
3	Optional	instrumentation	Outline Instrumentation Model

Artifacts

Direction	name	state	Option	Usage	IsColle...
Input	Wiring ...	defined	null	ToStart	□
Input	FICD	defined	instrumenta...	ToFinish	□
Output	Electric...	defined	null	ToFinish	□

Figure 5.2: Modeling UI



Activity
Design Instrumentation
Generate Bench Specification

Id	Kind	Option	Name
1	Mandatory	null	Define Instrumentation Model

Artifact's states inconsistency

The input/output Artifact ' FICD ' has same states

Direction	name	state	Option	Usage	IsCollection
Input	FICD	outlined	null	ToStart	□
Output	FICD	outlined	null	ToFinish	□

Figure 5.3: Verifying well-formedness rules regarding artifact's states

5.1.3.2 Enactment UI

The enactment-time functionalities of BAPE is exposed to process actors through *Enactment UI* as illustrated in the Figure 5.4. The *Enactment UI* is comprised of five panels as following:

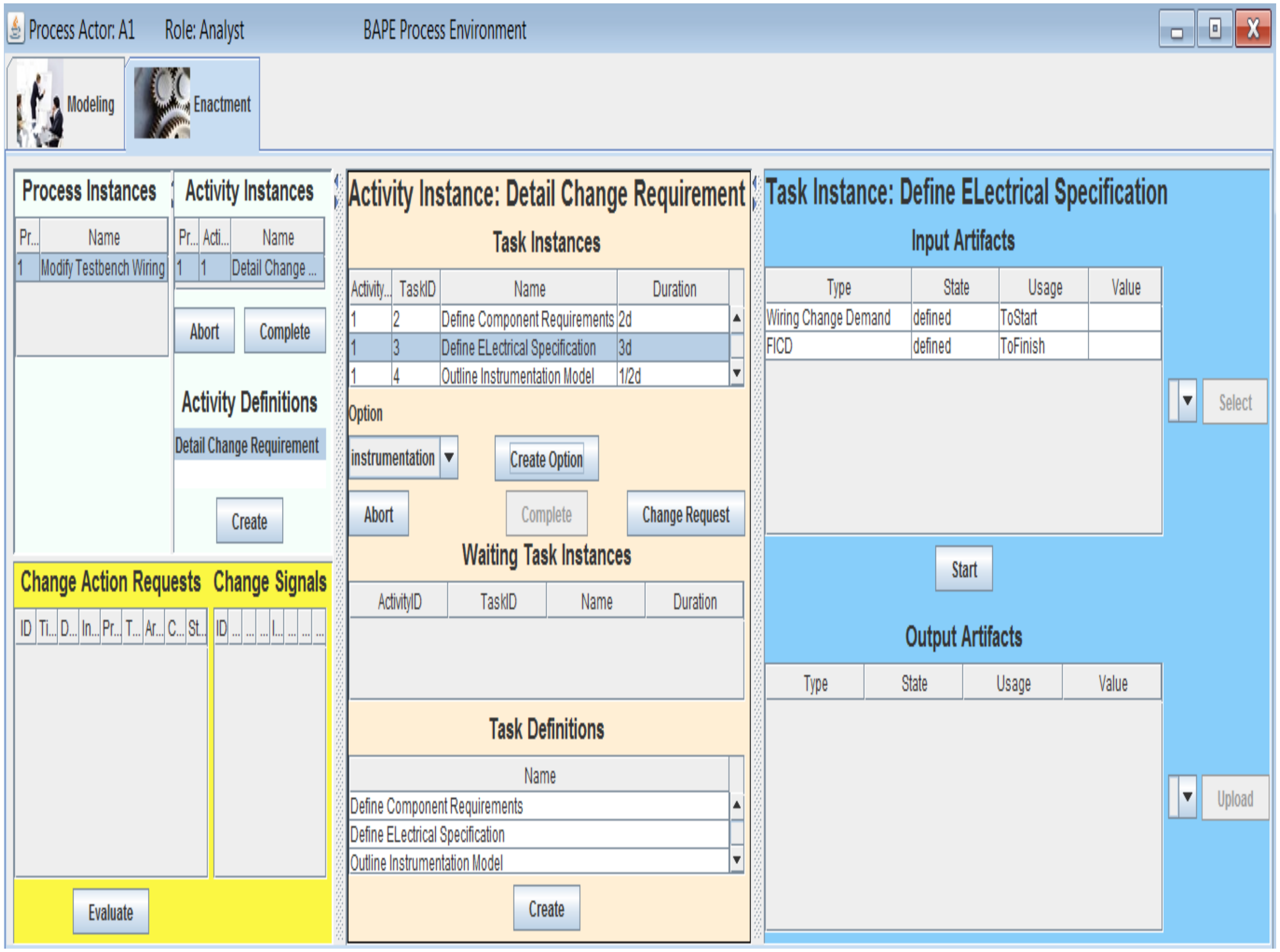
- *Process Panel* This panel enables process owner to trigger events regarding processes as defined in the process state machine (i.e., events to create, complete and abort process instances).

- *Activity Panel* This panel illustrates the list of activity definitions extracted from the process fragment model along with the list of activity instances for each process instance. A process actor can trigger events to create, abort or complete an activity instance.

- *Task Panel* This panel illustrates the list of task definitions extracted from the model along with list of running and waiting task instances for each activity instance. Through this panel process actors are enable to create, start, complete or abort their task instances.

- *Artifact Panel* This panel shows the list of input and output artifacts extracted from the model which their values required to be set respectively before starting and before completing the task instance.

- *Change Panel* This panel illustrates the information regarding change requests as following:



Process Instances

Pr...	Name
1	Modify Testbench Wiring

Activity Instances

Pr...	Acti...	Name
1	1	Detail Change ...

Activity Definitions

Detail Change Requirement

Change Action Requests

ID	Ti...	D...	In...	Pr...	T...	Ar...	C...	St...

Change Signals

ID	...	...	...	I...	...	...	...

Activity Instance: Detail Change Requirement

Task Instances

Activity...	TaskID	Name	Duration
1	2	Define Component Requirements	2d
1	3	Define ELectrical Specification	3d
1	4	Outline Instrumentation Model	1/2d

Waiting Task Instances

ActivityID	TaskID	Name	Duration

Task Definitions

Name
Define Component Requirements
Define ELectrical Specification
Outline Instrumentation Model

Task Instance: Define ELectrical Specification

Input Artifacts

Type	State	Usage	Value
Wiring Change Demand	defined	ToStart	
FICD	defined	ToFinish	

Output Artifacts

Type	State	Usage	Value

Figure 5.4: Enactment UI

  - List of change action requests which is regarding the change requests required to be evaluated.
  - List of change signals which is regarding informing the process actor who is affected by a change.

5.1.3.3 Monitoring UI

This interface provides real time information about the state of process instances along with their tasks and artifacts to authorized process actors (e.g., process manager). The information of this interface is feed from data coming from the graph database PDG via implemented queries. Every time the BAPE process engine updates the information stored into PDG, the data becomes automatically available to the *Monitoring UI*.

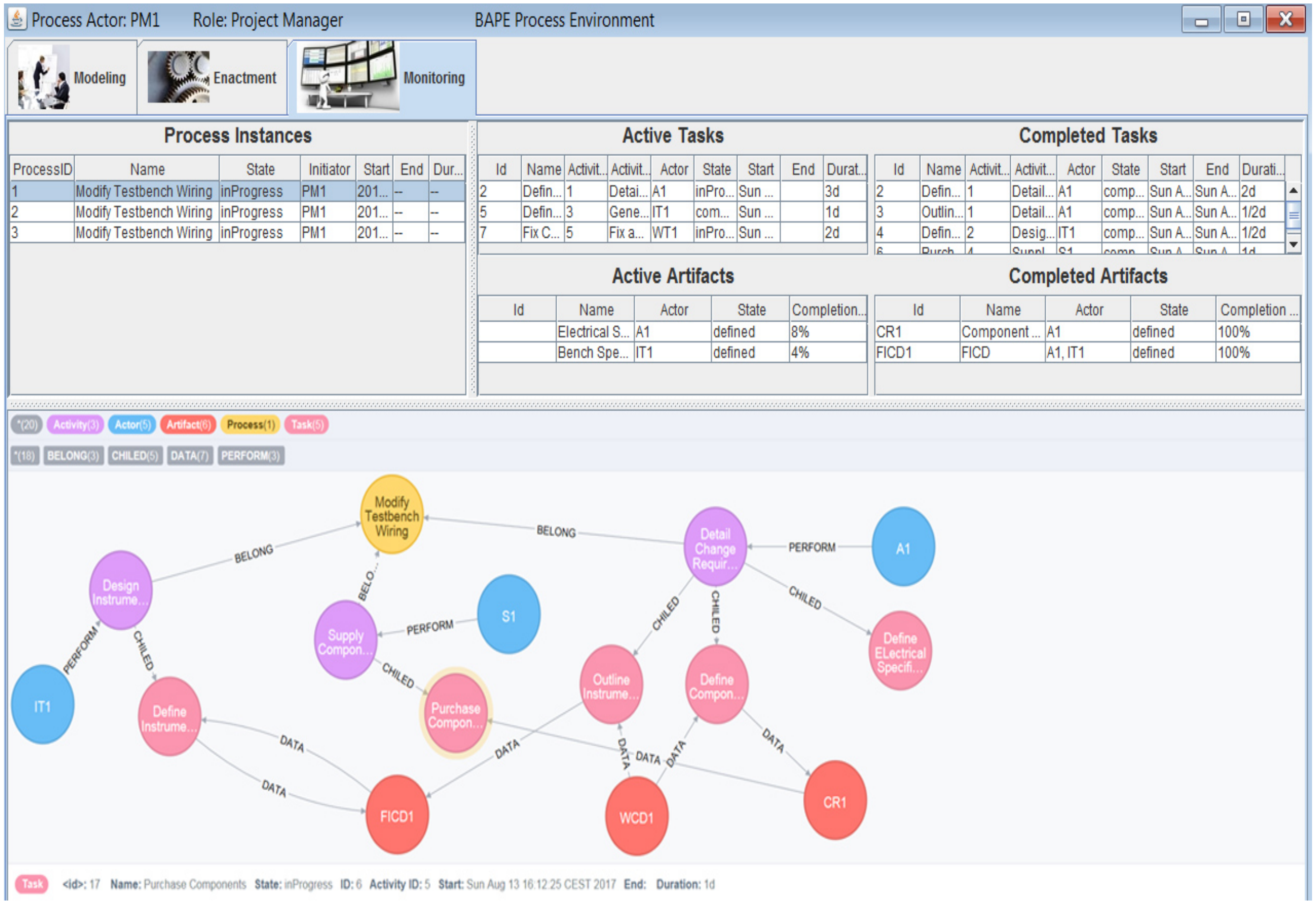
As in our approach processes are fragmented and progressively modeled and enacted, having a mechanism to show the emerging process models is very beneficial in order to give process owners good visibility of the activities of organization and identifying possible problems in the process. As mentioned earlier, PDG establishes the global view of the system from separate process fragments at enactment-time via shared resources among respective fragments. Therefore, to illustrate the emerging process model, the *Monitoring UI* comprises a snapshot of PDG. Currently, we illustrate the global model in the form of Neo4j graph which utilize a simple notation to represent nodes and their relationships. For instance, Figure 5.5 illustrates the *Monitoring UI* of a *Project Manager* by presenting run-time data about tasks, artifacts and snapshot of running process instances. However, in the future versions of BAPE, we consider to illustrate the global model in other forms to be more understandable and visualized for process actors.

5.1.3.4 Change Management UI

The change management functionalities of BAPE are exposed to the change manager through *Change Management UI* as illustrated in the Figure 5.6. Concretely, *Change Management UI* is comprised of two panels as:

- *Change Request Panel* This panel entails two tables as *Change Action Requests* and *Evaluated Change Requests*. The former enables change manager to request action from the change responsible in case the initiated change request is a major change. The latter shows the list of minor changes or the major changes evaluated by the change responsible which requires their impacts to be analyzed.

- *Impact Analysis Panel* This panel illustrates the result of impact analysis comprising the affected elements, e.g., artifacts, tasks, process actors, impact percentage, etc. It enables process manager to make decision about the change and reconfiguring affected tasks and informing affected people of change. Note that affected people



Process Actor: PM1 Role: Project Manager BAPE Process Environment

Process Instances

ProcessID	Name	State	Initiator	Start	End	Dur...
1	Modify Testbench Wiring	inProgress	PM1	201...	--	--
2	Modify Testbench Wiring	inProgress	PM1	201...	--	--
3	Modify Testbench Wiring	inProgress	PM1	201...	--	--

Active Tasks

Id	Name	Activit...	Activit...	Actor	State	Start	End	Durat...
2	Defin...	1	Detai...	A1	inPro...	Sun ...		3d
5	Defin...	3	Gene...	IT1	com...	Sun ...		1d
7	Fix C...	5	Fix a...	WT1	inPro...	Sun ...		2d

Completed Tasks

Id	Name	Activit...	Activit...	Actor	State	Start	End	Durati...
2	Defin...	1	Detail...	A1	comp...	Sun A...	Sun A...	2d
3	Outlin...	1	Detail...	A1	comp...	Sun A...	Sun A...	1/2d
4	Defin...	2	Desig...	IT1	comp...	Sun A...	Sun A...	1/2d
6	Purch...	4	Suppl...	S1	comp...	Sun A...	Sun A...	1d

Active Artifacts

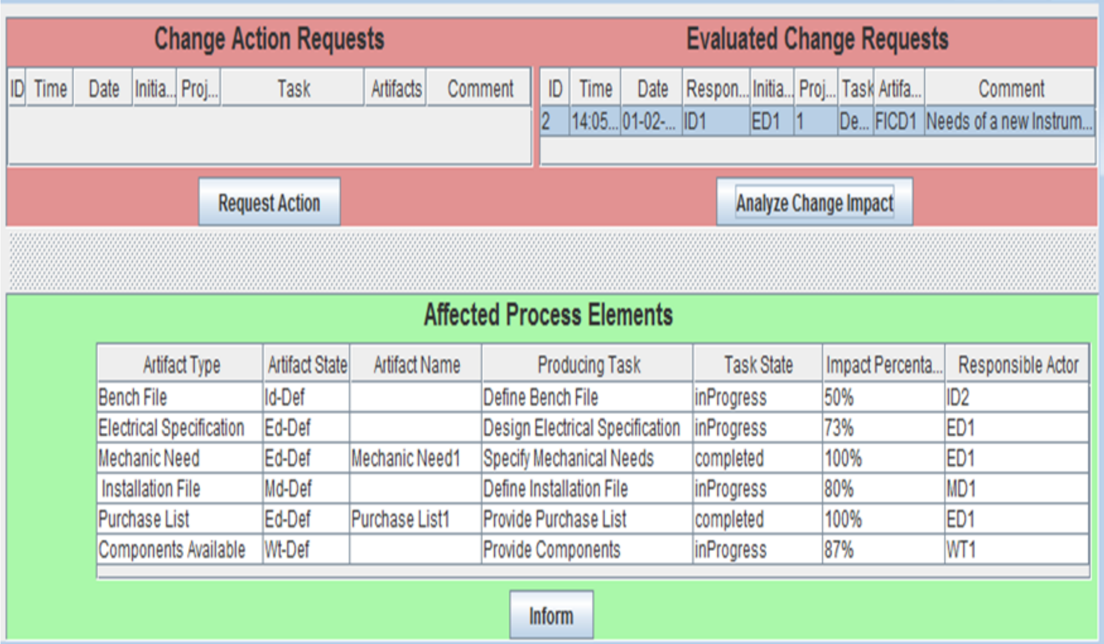
Id	Name	Actor	State	Completion...
	Electrical S...	A1	defined	8%
	Bench Spe...	IT1	defined	4%

Completed Artifacts

Id	Name	Actor	State	Completion ...
CR1	Component ...	A1	defined	100%
FICD1	FICD	A1, IT1	defined	100%

<id>: 17 Name: Purchase Components State: inProgress ID: 6 Activity ID: 5 Start: Sun Aug 13 16:12:25 CEST 2017 End: Duration: 1d

Figure 5.5: Monitoring UI



Change Action Requests

ID	Time	Date	Initia...	Proj...	Task	Artifacts	Comment

Evaluated Change Requests

ID	Time	Date	Respon...	Initia...	Proj...	Task	Artifa...	Comment
2	14:05...	01-02-...	ID1	ED1	1	De...	FICD1	Needs of a new Instrum...

Affected Process Elements

Artifact Type	Artifact State	Artifact Name	Producing Task	Task State	Impact Percenta...	Responsible Actor
Bench File	Id-Def		Define Bench File	inProgress	50%	ID2
Electrical Specification	Ed-Def		Design Electrical Specification	inProgress	73%	ED1
Mechanic Need	Ed-Def	Mechanic Need1	Specify Mechanical Needs	completed	100%	ED1
Installation File	Md-Def		Define Installation File	inProgress	80%	MD1
Purchase List	Ed-Def	Purchase List1	Provide Purchase List	completed	100%	ED1
Components Available	Wt-Def		Provide Components	inProgress	87%	WT1

Figure 5.6: Change Management UI

are notified by receiving the change signal in their change panel along with an email.

5.1.4 Implementation of PDG

Generally, a process environment requires a data management mechanism in order to store enactment-time information, i.e., information regarding processes, activities, tasks, artifacts and resources instances. In BAPE, enactment-time information is stored in PDG. To implement our PDG we explored the usage of NoSQL data management system. We focused on systems proposing native graph data management. This section introduces their main features and our implementation choice.

A graph database management system is an online database management system with *Create, Read, Update,* and *Delete* (CRUD) methods that expose a graph data model. Graph databases are generally built for use with transactional (OLTP) systems. Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability.

A graph database stores data in a graph, uses nodes, relationships between nodes and key-value properties instead of tables to represent information. By assembling the simple abstractions of nodes and relationships into connected structures, graph databases enable us to build arbitrarily sophisticated models that map closely to our problem domain.

This model is typically substantially faster for connected data sets and uses a schemaless, bottoms-up model that is ideal for capturing ad-hoc and rapidly changing data [7]. As illustrated in the Figure 5.7, structure of the graph database is defined as follows:

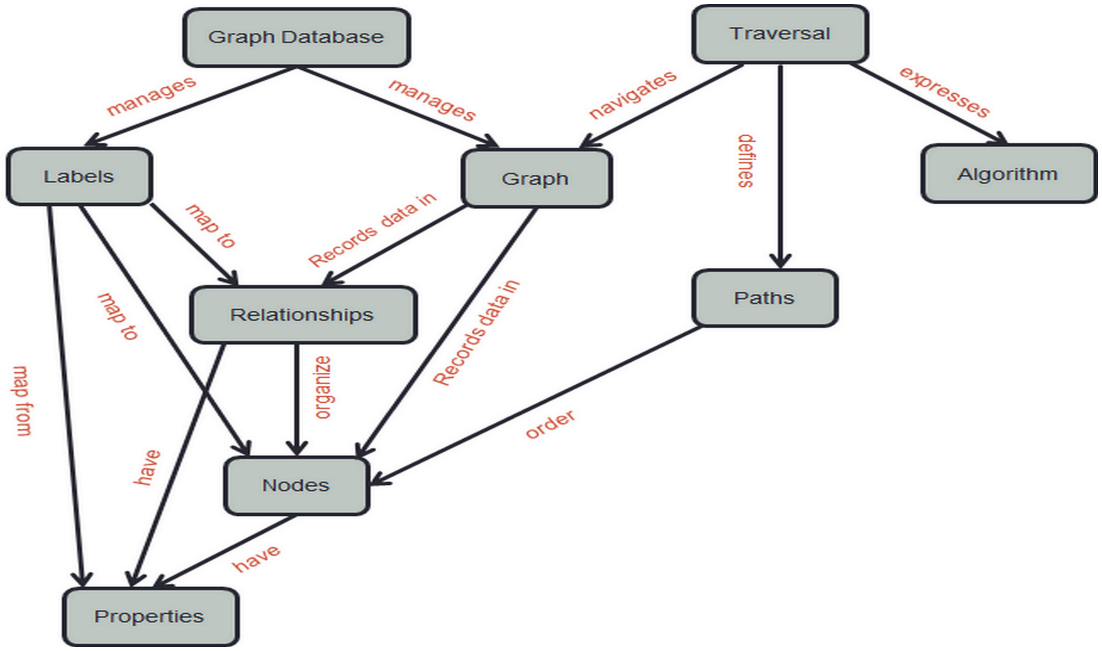


Figure 5.7: Structure of graph database

- A graph records data in Nodes which have Properties.
- Nodes are organized by Relationships which also have Properties. Relationships connect and structure nodes. A relationship always has a direction, a label, and a start node and an end node. There are no dangling relationships. Together, a relationship's direction and label add semantics to the structuring of nodes. Like nodes, relationships can also have properties. The ability to add properties to relationships is particularly useful for providing additional metadata for graph algorithms, adding additional semantics to relationships (including quality and weight), and for constraining queries at run-time.
- A Traversal navigates a Graph. It identifies Paths which order Nodes. A Traversal is how you query a Graph, navigating from starting Nodes to related Nodes according to an algorithm. In our case, we are interested in finding answers to questions such as "if an event happens, what processes and partners are affected?"

- An Index maps Properties to either Nodes or Relationships.
- A Graph Database manages a Graph and the related Indexes.

Graph databases are applied in areas where information about data inter-connectivity is more important, or as important as the data itself. In these applications, the data and relations among the data, are usually at the same level. One compelling reason, then, for choosing a graph database for our bottom-up approach is the importance of existing dependencies among process instance elements. These dependencies are playing the vital role in process engine's synchronizations as well as the change impact analysis. In the former, availability and connectivity of artifacts and tasks enable process engine to provide synchronization. In the latter, dependencies enable more thorough traversing over all process instance elements in order to extract affected elements of the change. Finally, more efficient storing and querying of enactment-time process data is achieved in graph database. For instance, queries are localized to a portion of the graph. As a result, the execution time for each query is proportional only to the size of the part of the graph traversed to satisfy that query, rather than the size of the overall graph.

To resume, introducing graphs as a modeling tool has several advantages as follows:

- as they depend less on a rigid schema, graph databases are more suitable to manage ad hoc and changing data with evolving schemas,
- graph databases are often faster for associative data sets,
- graph databases map more directly to the structure of object-oriented applications,
- graph databases scale more naturally to large data sets as they do not typically require expensive join operations.

In our proposal, PDG is implemented by using Neo4j [5]. Neo4j is an open-source native and JVM-based graph database. The architecture is designed for optimizing fast management, storage, and traversal of nodes and relationships. Neo4j provides true data safety through ACID transactions and supports fast development of graph powered systems.

Neo4j can be run in embedded as well as server mode. We used the Neo4j Core-Java-API to develop a Neo4j embedded application in our process environment BAPE. Neo4j provides access to the full range of APIs for creating and querying data: the Core API, traversal framework, and the Cypher query language [7]. Using the Core API, we can control the transactional life cycle, executing an arbitrarily complex sequence of commands against the database in the context of a single transaction. The Java APIs also expose the transaction life cycle, enabling us to plug in custom transaction event handlers that execute additional logic with each transaction.

Neo4j offers two ways in order to query the data base as using pure Java traversal

APIs or using Cypher as a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Cypher is a relatively simple but still very powerful language. Complicated database queries can easily be expressed through Cypher. This allows process owners to focus on their domain instead of getting lost in database access and query the PDG from their interest points. However it requires to know the Cypher language. In BAPE, we used Neo4j Cypher Java API in order to query the PDG. For instance, code snippet 5.2 shows a simple query which finds all task instances of a process instance MTW_1 and returns only running task instances.

```
try (Transaction ignored = db.beginTx();
   Result result = db.execute("MATCH (processNode {name: ’MTW1’})
   -[:belong]->()-[:child]->(top)
  WHERE top.state = ’inProgress’
  RETURN top.name, top.Id"))
```

Code 5.2: A simple query in Cypher

In the aforementioned code snippet, the $MATCH$ clause which is the most common way to get data from the graph is used. It traverses PDG through edges *belong* and *child* with specifying a constraint to obtain active tasks instances and returns their name and id.

5.2 Industrial Case Study

This section reports the use of BAPE for a case study along with the feedback of our industrial partners about BAPE's usability. The examined case study is a simplified version of a real case study conducted in the context of the project ACOVAS. Our contribution to the project has been providing tool support to model, control, monitor and managing changes of existing system development complex processes.

To do so, we carried out our evaluations in two parts as modeling and enactment, and change management, concretely regarding our examined process described in Figure 5.8.

Our evaluation has been carried out among three process environments as following:

- *jBPM* [4] A traditional process environment which is light-weight, fully open-source (distributed under Apache license) and written in Java. It supports modeling (i.e., described in BPMN 2.0), enacting, and monitoring (business) processes throughout their life cycle.

- *AristaFlow* [1] An adaptive process environment providing flexible PAIS (Process-Aware Information System) for a large number of processes from different domains, e.g., healthcare, disaster management, logistics, and software engineering. In particular, at the process instance level authorized users may deviate from the pre-modeled process if required (e.g., to deal with unforeseen situations). Examples

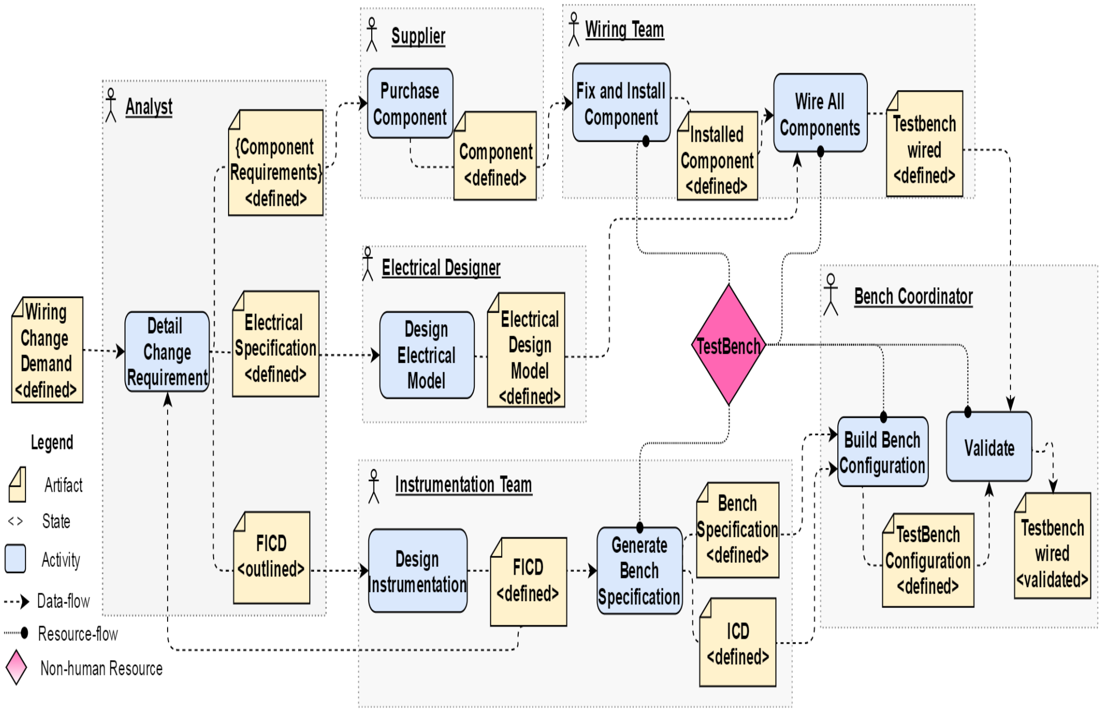

Analyst
Supplier
Wiring Team
Electrical Designer
Instrumentation Team
Bench Coordinator
Wiring Change Demand <defined>
Detail Change Requirement
{Component Requirements} <defined>
Electrical Specification <defined>
FICD <outlined>
Purchase Component
Component <defined>
Fix and Install Component
Installed Component <defined>
Wire All Components
Testbench wired <defined>
Design Electrical Model
Electrical Design Model <defined>
TestBench
Design Instrumentation
FICD <defined>
Generate Bench Specification
Bench Specification <defined>
ICD <defined>
Build Bench Configuration
TestBench Configuration <defined>
Validate
Testbench wired <validated>
Legend
Artifact
<> State
Activity
Data-flow
Resource-flow
Non-human Resource

Figure 5.8: Process of *Modify Testbench Wiring*

of such ad-hoc changes include the dynamic insertion, deletion or movement of one or multiple process activities.

- *BAPE* Our developed process environment enabling process actors to model, enact and monitor their processes along with providing some degree of change management support.

In the next, we explain our experiments in details.

5.2.1 Modeling and Enactment Evaluations

In order to carry out our modeling and enactment evaluations, we consider our examined process and applied a simple setup for a process instance P_1 including six process actors who were teams responsible playing different roles in the process. Table 5.2 summarizes the main information of process instance P_1.

Our evaluation on modeling phase has been based on the required roles to participate into two sub-phases as process elicitation and process formalization. The former concerns gathering domain knowledge to refine and elaborate the process and the latter is regarding modeling the process using a formal modeling language. Both jBPM and

Table 5.2: Information of P_1

Process Actors	Analyst = A_1 Instrumentation Team = IT_1 Supplier = S_1 Wiring Team = WT_1 Electrical Designer = ED_1 Bench Coordinator = BC_1
Non-human resources	TestBench = TB_1
Activated Options	instrumentation
Artifacts	$\{C_1, C_2\}$

ArtistaFlow adopt top-down methodology and respectively use BPMN [6] and Petri Nets [9] as their modeling language. Hence, when using these two environments, we had to play the role of process designer to model the process. Process actors from our industrial partners used the BAPE environment to model themselves their process fragments.

Our evaluation on enactment phase was regarding the support provided by the examined environments based on two criteria: the requirements for process synchronization and offered flexibility to deal with emerging situations which are not pre-defined in the model. To do so, we simulated the enactment of our examined process and experiment several execution scenarios. The enactment of the process in jBPM and ArtistaFlow were done by our academic research team, whereas in BAPE, the enactment of the process were conducted by industrial partners.

The result of our modeling and enactment evaluations is presented in the Table 5.3.

	jBPM	AristaFlow	BAPE
Elicitation	process designer process actor	process designer process actor	process actor
Formalization	process designer	process designer	process actor
Synchronization	complete-defined	complete-defined	partial-defined
Flexibility	late-binding	ad-hoc adaptation	loose-inter-task execution order

Table 5.3: Modeling and enactment phases evaluations

As learned from our experience, jBPM and ArtistaFlow require an important effort and time to model the process for both process designer (to understand and to formalize the process) and process actors (to describe the process in process terminology). It was difficult to ask process actors learning BPMN or Petri Nets to model themselves the process. Moreover, they had difficulty to understand the final model in BPMN. Therefore, we spent four months through conducting several interviews with process actors in order to obtain their operational process.

Six process actors playing different roles used BAPE in order to model the fragment of their process. They got all required information regarding the process modeling (introduce the SPML concepts such as option) and modeling tool aspects, through holding a modeling session. During this session, we asked process actors to define the company assets in order to resolve some wording issues regarding their shared process information. Then each process actor started to model his own fragment of process. In order to examine the simplicity of our modeling language, we played the role of facilitator to observe possible modeling difficulties.

By proposing SPML with a user-friendly modeling interface developed in BAPE, process actors conveniently defined their own executable processes fragments without requiring any guide during modeling process. Process actors found the modeling process straightforward and fast. They believed SPML was enough expressive in order to define their working process. They found fragmented process modeling beneficial which avoided bringing a cognitive overload to the related roles and made it easy for them to concentrate on their own contribution to the process. Besides the benefits of our modeling approach, process actors pointed a limitation regarding the visualization of their process fragment.

About process synchronization, while both jBPM and ArtistaFlow require that the whole process is modeled before enactment so that the process engine knows how to synchronize and coordinate the process's activities/tasks, BAPE can enact a partially modeled process. On process adaptation, the case study shows the rigidity of jBPM faced to unforeseen enactment situations. jBPM only supports the late binding by allowing to define ad-hoc sub-process in the model. ArtistaFlow supports better the inadequate anticipation in process modeling by offering ad-hoc deviations from predefined process models. However, the resulting adaptation may cause enactment inconsistencies.

BAPE does not propose a proper solution for process adaptation but offers a less restrictive manner to perform activities so that adapt better to real circumstances. In BAPE, at enactment-time, an activity can manage freely and separately its inner tasks because the process model does not prescript the inter-task execution order. Furthermore, thanks to the activity state-machine which allows an activity to listen to different task events, an activity is reactive to specific execution contexts. Together, these characteristics of BAPE gives more autonomy to process actors in order to control their processes, e.g., facilitating the reworks and enabling variable ways to perform an activity.

5.2.2 Change Management Evaluation

Our evaluation on change management was regarding the supports of examined tools to allow systematic and timely change signalling, impact analysis and informing of affected people. To do so, we simulated the execution of our examined process and play out several change scenarios. For instance, one of our examined scenarios is described as follows:

Execution scenario Δ
We assume that the company has lunched three projects by enacting three processes P_1, P_2 and P_3. Particularly, we examine P_1 initiated from change demand WCD_1 regarding the testbench TB_1. The main information about P_1 as well as the execution scenario are respectively presented in Table 5.4 and Figure 5.9. We suppose that in P_1, two instances of activity *Purchase Component* PC_1 and PC_2 regarding two *Component Requirements* CR_1 and CR_2 are performed, the installed components IC_1 and IC_2 are produced and *Wiring Team* WT_2 is wiring the respective components on the testbench TB_1. We suppose that at that moment, the *Wiring Team* WT_3 involved in P_2 is performing task *Fix Component* FC_3 of his activity *Fix and Install component* FIC_3 in order to install the component C_1 on the testbench TB_2. Moreover, the *Bench Coordinator* BC_3 and *Wiring Team* $WT4$ involved in P_3 are waiting to use the *Testbench* TB_1 in order to perform their tasks.

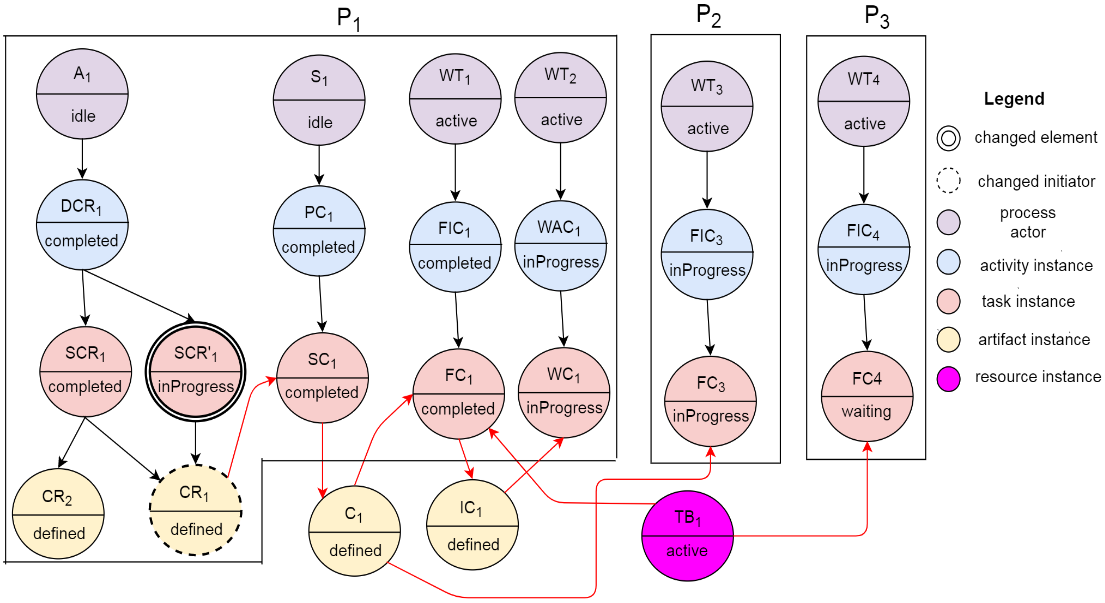


Figure 5.9: Execution Scenario Δ

Change scenario
In the P_1, A_1 realizes a modification on his already produced artifact CR_1. Thus, he enacts a new instance of task *Specify Component Requirements* SCR'_1 and signals

a major change on CR_1.

Table 5.4: Information of P_1

Process Actors	Analyst = A_1 Instrumentation Team = $\{IT_1, IT_2\}$ Supplier = $\{S_1, S_2\}$ Wiring Team = $\{WT_1, WT_2\}$ Electrical Designer = ED_1 Bench Coordinator = BC_1	
Non-human resources	TestBench = TB_1	
Components	$\{C_1, C_2\}$	
Shared Resources & Artifacts	$P_1 \wedge P_2$	C_1
	$P_1 \wedge P_3$	TB_1

We summarize in Table 5.5 different types of impacts that should be analyzed when a change occurs. The impact can be direct if an element works with the changed element, or indirect if an element works with an element impacted by the change. Then we point out which types of impacts that can be identified in three solutions: (A) without supporting tools (like actual solution of our partners in project ACOVAS), (B) with run-time information taken from a standard process environment (as jBPM or AristaFlow) and (C) with information taken from our change-aware process environment BAPE.

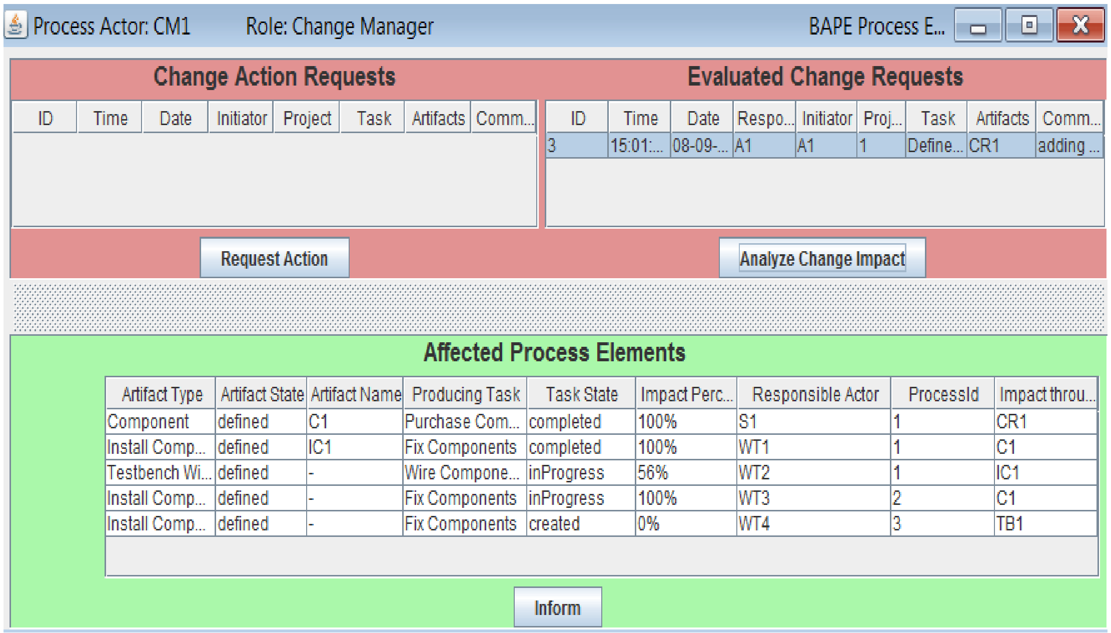
The Table 5.5 is not for comparing BAPE with the solutions (A) and (B), but to show the contributions of BAPE in analyzing inter-processes change impacts, impacts on tasks at different states (i.e., current, completed, potential) as well as in analyzing indirect change impacts. Thanks to the PDG that keeps trace of all elements in the development, we can have a thorough analysis on different axes: by examining both nodes inside and outside of the changed process instances, we can identify the impacts on the elements in the scope of a process instance or in other process instances; by examining all existing task nodes - completed and current - we can know which elements are really impacted. However, as in our approach the procedure of process modeling is incremental, to analyse potential impacts of a change, we have to simulate possible scenarios of process's evolution. Figure 5.10 shows the change manager UI as a result of analyzing the change impact of scenario Δ. All the information regarding affected elements is extracted and illustrated in the panel, e.g., affected artifacts and tasks, responsible, impact percentage, etc.

In contrast to BAPE where analysis and propagation of changes are systematic and immediate, change impact analysis takes more time in the solution (A) and (B). In (A), without supporting tools, changes propagation is done by human actors and often the change request has to be passed through many organizational levels so that its impact can be estimated. In (B), using only a standard process environment, impact analysis can be done by complex queries on various process logs, but lacking the information about the data and resources using by tasks, this analysis is rather limited.

Table 5.5: Simulation results

Impacts	**Impacted Tasks**	**Scope**	
		Intra-process	**Inter-processes**
Direct	**Current**	(A), (B), (C)	(C)
	Completed	(A), (C)	(C)
	Potential	(A), (C)	(C)
Indirect	**Current**	(C)	(C)
	Completed	(C)	(C)
	Potential	-	-

Legend	**A**	**B**	**C**
	Without Tool supports	jBPM/AristaFlow	BAPE



Change Action Requests

ID	Time	Date	Initiator	Project	Task	Artifacts	Comm...

Evaluated Change Requests

ID	Time	Date	Respo...	Initiator	Proj...	Task	Artifacts	Comm...
3	15:01:...	08-09-...	A1	A1	1	Define...	CR1	adding ...

Affected Process Elements

Artifact Type	Artifact State	Artifact Name	Producing Task	Task State	Impact Perc...	Responsible Actor	ProcessId	Impact throu...
Component	defined	C1	Purchase Com...	completed	100%	S1	1	CR1
Install Comp...	defined	IC1	Fix Components	completed	100%	WT1	1	C1
Testbench Wi...	defined	-	Wire Compone...	inProgress	56%	WT2	1	IC1
Install Comp...	defined	-	Fix Components	inProgress	100%	WT3	2	C1
Install Comp...	defined	-	Fix Components	created	0%	WT4	3	TB1

Figure 5.10: Change impact analysis results of scenario Δ

The feedback of our partners on the preliminary change management results of BAPE is positive. They affirmed the needs of managing unofficial changes during the development process, especially for dealing with problems in testing phase where time constrains are important, since physical and human resources are reserved and allocated. In Acovas project, a resource planning tool is dedicated to reserve resources required for system testing. Although additional works will be needed to develop a full solution that can be integrated to the real environment of our partners, they found that BAPE is helpful and motivated to continue working with us on the improvement of BAPE.

The development of the BAPE was carried out in six months. As illustrated in the Figure 5.11, developing BAPE comprises 10500 lines of code constituting three major modules in which abstraction and presentation modules comprise several sub-modules.

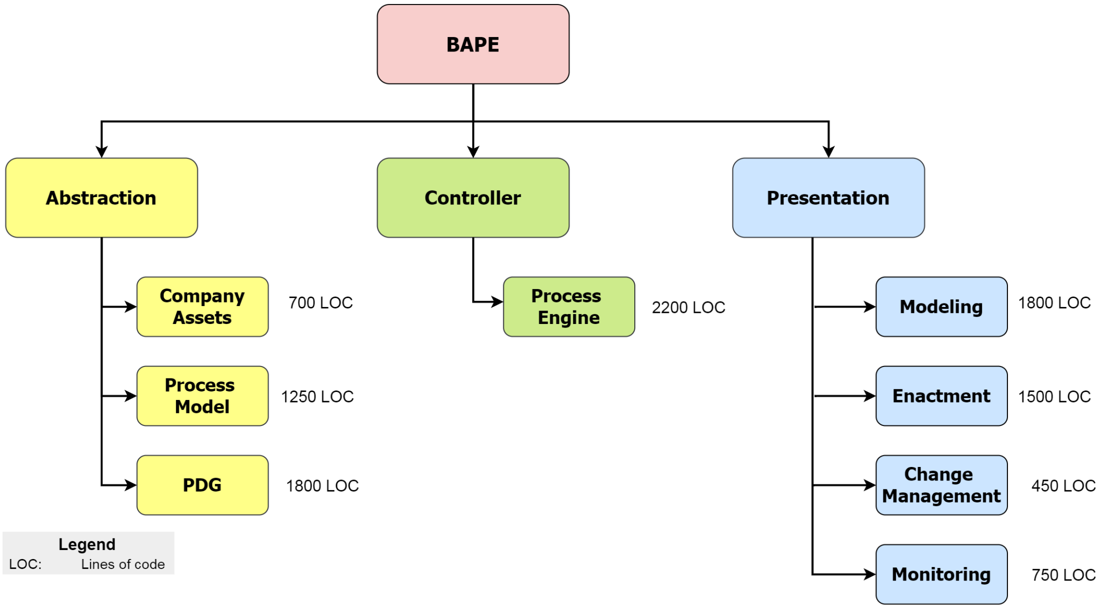


Figure 5.11: Number of code lines for BAPE

Our framework is based on the assumptions that a central graph database, a central resource repository and an application server (e.g. jBOSS) are provided to support process and change management in a distributed environment. However, due to the security policies of our industrial partners, currently we could not deploy our prototype directly in their real development environment. So we conducted the validation with the participation of our partners on a simulated development environment on one central machine.

Regarding the possible evolutions of the prototype BAPE, we consider adding following functionalities:

- Deploying and running BAPE in a web/application server along with developing web-based modeling tools to allow using BAPE in a distributed working environment.

- Integrating BAPE into process actors's development tools, e.g., Eclipse, Github, etc.

- Enabling definition of non-human tasks, e.g., script tasks and service tasks.

- Developing different types of SPML modeling editors, e.g., textual, diagram, etc.

- Improving process monitoring by adding different metrics and key performance indicators.

- Visualizing the global model in different forms in order to provide better communication among process actors to assess how processes are doing.

We developed BAPE for our bottom-up approach. However, it is a viable proposition to use its main components process engine and PDG in top-down activity-driven process environments [8]. The former, requires adaptation of state machines to implement the behavior of engine based on activity-driven enactment. The latter which has a graph structure to store process instance elements, can easily represent activity-centric process models. However, PDG structure must be extended in order to reflect the dependency established by work-sequence relations in the model, i.e., adding a new type of edge to associated task instances.

5.3 Summary

This chapter presented the technical aspect of our developed process environment BAPE along with our conducted experiments regarding modeling, enactment and change impact analysis. BAPE's architecture is based on PAC pattern which makes BAPE an extensible and pluggable process environment. BAPE comprises user-friendly interfaces to allow process actors to model, enact and monitor their processes as well as manage changes. The choice of graph database (Neo4j) to implement PDG brings an advantage to project dependencies among process instance elements. We conducted our experiments with participation of our industrial partners by usage of three process environments, i.e., jBPM, AristaFlow and BAPE. We received positive feedback from our partners. By providing a user friendly and an expressive modeling language, process modeling was straightforward and was not time and effort consuming. Our enactment experiment was performed by simulating several execution scenarios to illustrate the flexibility and possibility of enacting SPML fragment models. Finally, our experiments on change impact analysis show the strength of BAPE in automatically deducing the change impacts between and among process instances.

Bibliography

[1] AristaFlow. http://www.aristaflow.com/.

[2] J. Coutaz. Pac: An object oriented model for implementing user interfaces. *SIGCHI Bull.*, 19(2):37–41, Oct. 1987.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[4] JBoss. jbpm, http://www.jbpm.org/.

[5] Neo4j. Neo4j website : http://www.neo4j.com.

[6] OMG. Business Process Model and Notation (BPMN) Version 2.0.2, Dec. 2013.

[7] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[8] H. N. Tran, M. Hajmoosaei, C. Percebois, A. Front, and C. Roncancio. Integrating run-time changes into system and software process enactment. *Journal of Software: Evolution and Process*, 28(9):762–782, 2016.

[9] W. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach*. The MIT Press, 2011.

Chapter 6

Conclusion and Future Work

To better support process actors in managing their operational processes, we proposed a user-centric approach that allows process actors to model and enact their fragments of the process in a bottom-up manner. In our approach, the global view of the system can be constructed dynamically at enactment-time to allow synchronizing activities instantiated from different process fragments.

6.1 Contributions

Overall, the contributions of this thesis are:

- *Artifact-centric Process Modeling* The first contribution of this work is an executable and structural process modeling language (SPML) based on the artifact-centric paradigm. The proposed language eliminates the behavioral aspect in process modeling and focuses only on the structural aspect. SPML emphasizes on the data-flows perspective by describing in detail artifacts consumed and produced by activities. Moreover, it offers the ease of use to process actors by decomposing the process model into several role-based fragments and using concepts known by process actors in their daily work. The simplicity of SPML facilitates its operational semantics as well as the implementation of this semantic in the process environment. An advantage of the bottom-up and artifact-centric modeling is that it is not necessary to define the whole process from the beginning of the project. The process fragments can be progressively defined during the project execution. Moreover, if some process fragments are not provided, the rest of the process can be specified and enacted independently to the missing parts. However, this feature is under the condition of having a shared data management system between process actors and external agents/systems. By inverting the direction and changing the perspective, our bottom-up modeling approach makes process modeling realizable by process actors and the result process models give them more autonomy in enactment phase.

- *Data-driven Process Enactment* As second major contribution of this thesis, we proposed an approach to enact SPML process fragment models which are structurally and partially defined. Our enactment approach comprises two main components: a Process Dependency Graph (PDG) to store process instance elements and an artifact-centric process engine that enables enacting and synchronizing SPML processes. Due to similarity of PDG and SPML structures, we define a direct mapping between PDG and SPML concepts which makes process deployment very simple. Our process engine adopts a data-driven mechanism where the main driver to progress an activity is the availability of artifacts in the demanded states. The proposed process engine offers a user-centric control where process actors have enough autonomy to decide the way they enact their processes. This autonomy brings flexibility in order to handle foreseen and unforeseen situations at enactment-time, e.g., rework, optional activities, etc. Moreover, we provide an extension mechanism that enables adding new functionalities to our process environment without requiring any modification on core process engine.

- *Change-aware Process Environment* As last contribution of this thesis, we proposed a mechanism to integrate into our process management environment new functionalities to analyze the monitored processes and provide end-users the global view about the whole system. This contribution is especially necessary in a bottom-up approach where each process actors, as in the real life, has just a partial view on the system.

  In the first time, we proposed a change management mechanism to provide systematic support to handle change requests. We turned our process environment to a change-aware environment by using our extension mechanism. In this environment, run-time changes - which are not described in the process model - can be integrated into process enactment. By providing process actors with the possibilities of notifying and analyzing changes, our environment allows them to handle changes in a centralized and proactive way so that they can better anticipate and response to changes. The key strength of this work is using run-time process information to establish dependencies among processes elements. Thanks to PDG, the dependencies which are hidden on the model and only emerge at enactment-time via artifact exchanging or shared resources are revealed. By uncovering these dependencies, we could provide a more thorough and precise analysis on the impacts of changes inside or among process instance. This advantage should be particularly in areas where the development involves often various teams in different domains and on complex products.

- *Bottom-up Artifact-centric Process Environment (BAPE)* To validate our user-centric and bottom-up approach, we developed a proof-of concept prototype BAPE. BAPE offers user-friendly modeling interfaces and less restrictive manner to process actors, respectively to defined conveniently their own executable process fragments and to perform their activities so that adapt better to real circumstances. BAPE was validated by our industrial partners through conducting several experiments on their processes. The first feedback of our industrial partners regarding modeling, enactment and change management is rather positive. They found our approach end-user friendly and were satisfied about the given autonomy to model and enact their processes and react to different situations at execution. They stated that SPML is enough expressive to allow them defining their processes. They emphasized the need of the provided change management support in their daily work where a lot of rework and problems occur due to lack of systematic coordination and communication among actors.

6.2 Limitations and Perspectives

Our work aims at making process management realizable by process actors with a bottom-up approach which is more natural way for them to perform. To be able to concentrate on the essential points on modeling and enacting solutions, we make a strong assumption about the application context of this work: the operational process in practice has to be already defined, the company assets have to be defined, aligned and shared among process actors. In reality, such an ideal starting point is not always easy to obtain and some preprocessing works are needed before applying BAPE. This preparation requires process designers to explain how process actors should describe in details their activities, task, artifacts (with explicit states). However, our experiences show that this step is acceptable for process actors because it concerns principally their domain knowledge.

Our study has unveiled several aspects that should be addressed by further research.

- *Preventing possible modeling and enactment inconsistencies* In our approach, the global process model is progressively constructed by adding coming process fragments at enactment-time. That is why the verification of conflicts (e.g., deadlock situations) in the process model can be difficult before enactment. To remedy this problem, we are investigating the use of a separate state-machine for each type of artifact to establish finer constraints on activities sequences via the states of their exchanged artifacts. Such constraints can help us detect potential conflicts each time a new fragment is integrated into the global process.

- *Feasibility of integrating BAPE into actors' development tools* Currently, BAPE provides process actors to model, enact and report the progress of their work. However, it is beneficial to integrate BAPE into process actors development tools, e.g., Eclipse, Github, office suits, testing tools, etc. This fact can bring down the gap between process and existing tools for project operation.

- *Supporting of non-human tasks* Currently, our approach supports only human tasks. However, we are investigating the usage of non-human tasks such as service and script tasks. Our aim is to enable the usage of external services and most importantly the possibility of controlling process engine by exposing process engines' functionalities through its APIs. The latter can be achieved by task's script in a language that the engine can interpret and execute.
- *Usage of process mining for process improvement* A complete process environment should consider all three phases of modeling, enactment and improvement. However, the focus of this work was only on modeling and enactment. As future work, we consider using process mining techniques to construct the global process model extracted from enactments of different process instances. Verification and improvement then can be done on this extracted process model.

Overall, we will continue working on the BAPE environment to realize a more user-centric and flexible process management technology that allows process actors perform their daily work in a more natural and intuitive way.

List of Figures

List of Tables