Flexible Deviation Handling during Software Process Enactment

Marcos Aurélio Almeida da Silva, Reda Bendraou, Jacques Robin, Xavier Blanc

Flexible Deviation Handling during Software Process Enactment

Marcos Aurélio Almeida da Silva, Reda Bendraou and Jacques Robin
LIP6
Paris Universitas, France
Email: first-name.family-name@lip6.fr

Xavier Blanc
LaBRI
Université de Bordeaux 1, France
Email: xavier.blanc@labri.fr

***Abstract*—Software process models formalize the way a group of agents (e.g. developers, testers, managers etc) interact in order to produce a desired outcome (e.g. a product, an artifact etc). In this context, a "deviation" is a mismatch between the process executed by the agents and the process model. Existing approaches for deviation detection and handling force the agents to either pursue a deviation-free process execution, which is unrealistic; or to selectively ignore them, which may be risky to the desired outcome of the project. In this paper, we propose an approach that allows agents to deviate from the process specification, but also allows them to correct these deviations later in the process enactment. Additionally, they are informed about the risks implied by each non-handled deviation. During the correction phase, the process agents are assisted by the means of a set of correction plans that are automatically generated by the approach. These plans aim at reducing the risk of non resolved deviations. This paper presents a preliminary evaluation of this approach as a prototype implementation.**

***Keywords*-Process Enactment, Deviations, Consistency Management**

I. Introduction

A Software Process Model (SPM) is an abstraction of the partially ordered set of activities that should be performed by agents to produce a desired outcome [13]. Such a model is interpreted by a Process-centered Software Engineering Environment (PSEE), that guides the agents throughout the process enactment [4], [6]. PSEEs both make sure that agents execute the process activities in the right order, and that artefacts are produced and consumed as expected.

A deviation is an inconsistency between the SPM and the execution as observed by the PSEE. They may be harmful regarding the objectives of the process. This mainly depends on the context in which the deviation occurs and on the level of expertise of the agent that is deviating. Undeniably, performing actions that are not compliant with what has been planned introduces a risk for the success of the project.

Unfortunately, deviations are not exceptional events in process enactment. This has been confirmed by the empirical study conducted by Lanubile and Vissagio [11]. This study has shown that agents do not follow 100% of an assigned process. In fact, experts usually deviate and act opportunistically, using the process as a guide without necessarily reducing the quality of their work [19]. This implies that the risk introduced by deviations can not be fully avoided and has to be managed.

The main claim of this work is that managing the risk of deviations should be part of the PSEE responsibilities. A PSEE should both enforce the execution of the SPM and manage the agent's deviation. This claim contrasts with what is currently supported [6], [10], [5], [1], [17]. Existing PSEEs poorly support agent deviations either by ignoring them and continuing the process enactment, or by automatically undoing the actions that have led to them. In the former case, ignored deviations do represent a risk regarding the process objectives [20]. In the latter case, agents cannot act opportunistically, as they usually do. Both cases are very counter-productive.

The main contribution of this paper lies in a process enactment approach that supports agent deviations. In particular, the following specific goals are addressed:

- **Late Deviation Handling:** Existing PSEEs that allow process agents to selectively ignore deviations [10], [5] suggest that some deviations may be left to be handled later. Extra evidence for this hypothesis has been provided by our previous experiments in comparing consistency management techniques with process enactment ones [7]. This experiment showed us that if one allows agents to act opportunistically, most of the inconsistencies will be naturally fixed in a later time by the agents.
- **Risk Assessment**: Even though evidences suggest that most deviations will be fixed in a later time by agents, some will not. Since not all deviations have the same impact on the continuity of the process [10], [20], [5], the process agents need to have a clear picture of the risk represented by these deviations. This way, they can decide which ones will be handled next and which ones will not.
- **Correction Guidance:** As suggested by existing work in the field of model consistency management [9], [8], [18], providing guidance in fixing inconsistencies becomes even more necessary as the number of inconsistencies to be handled increases. This happens because a correction for one inconsistency may generate other inconsistencies.

The challenge in accomplishing these three goals is in their interconnection. Allowing agents to delay deviation handling means that the guidance offered to them should take into account a larger number of deviations at the same time. Additionally, associating different risk levels to different deviations means that they must be handled differently in terms of guidance. For example, guidance provided to agents should prioritize correcting higher risk deviations.

In this paper we deal with these challenges by proposing an approach based on the so-called *deviation rules*. These rules are logical formulas that detect deviations, their causes and risks to the process objectives. This way, an agent can have a global view of the deviations that have not yet been addressed and of their impact to the process objectives (Risk Assessment). The next step consists in using a planning algorithm [8] to produce correction plans for the detected deviations that are then suggested to the process agents (Correction Guidance). On top of that, this approach does not force process agents to repair deviations as soon as they have been detected (Late Handling).

This paper is organized as follows. Section II describes the problem of deviation handling during process enactment along. Section III presents our approach and Section IV describes the steps to be followed in order to apply it in a process enactment context. Section V describes our preliminary evaluation of this approach by the means of a prototype PSEE and its application to a software development process. Finally, Section VI presents our related work and Section VII concludes.

II. Deviation Detection & Handling during Process Enactment

This section discusses the problem of deviation handling during process enactment. The following development process is used as an illustration throughout this paper.

Example 1: The software development process represented as a UML2 Activity Diagram in Figure 1 consists of three activities that should be executed in order. During the **Use case modeling (ucm)** activity the agent should gather the system requirements and build a use case model of the system. During the **Class diagram modeling (cdm)** activity, the agent should build a UML Class Diagram with the help of the use cases described in the previous activity. Finally, during the **Coding (code)** activity, the agent should implement the class diagram built in the previous activity in the Java programming language. A postcondition of this activity is that every class that appears in the class diagram should be implemented as a Java class.

A process model encodes desired process outcomes. In this example, the process outcome are the use case model, the class diagram and the Java source code. We call such desired outcomes the *process goal*.

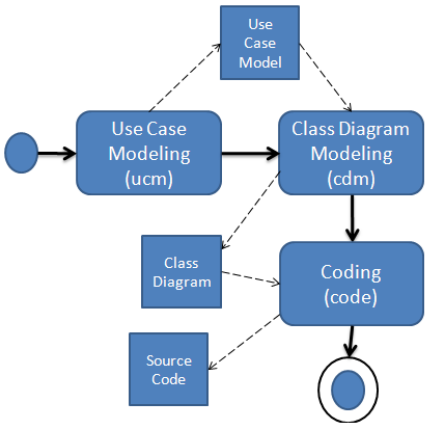

Figure 1. Example Software Development Process

Deviation Detection. During process enactment, the PSEE performs three kinds of verification for each activity. First, when the agent **launches** the activity, the PSEE verifies if the required artefacts are present and consistent with the process model. Second, **during** the execution of an activity, the PSEE verifies if the actions performed by the process agents are consistent with the process model. Third and last, when the agent **finishes** the activity, the PSEE verifies if the produced artefacts are consistent with the process model. When one of those verifications fails, we say that the process agents are **deviating** from the process model. At this point we propose the following definition of deviation:

Definition 1 (Deviation): A deviation is an inconsistency between the process specification and the process as observed by the PSEE.

Deviation Handling. After detecting deviations, a PSEE should be able to guide the process agents into handling them. We divide the general objective of handling deviations into the following three ones:

Objective 1 (Late Handling): Supporting delayed deviation correction.

Objective 2 (Risk Assessment): Assessing the impact of a process deviation to the process objectives.

Objective 3 (Correction Guidance): Guiding the process agents in finding a deviation correction plan.

An ideal deviation handling approach should allow agents to delay the resolution of detected deviations. For example, if the class diagram produced at the end of the second activity (cdm) is not consistent w.r.t the UML meta-model, this deviation could be handled by correcting the class diagram during the third activity ($code$). This kind of support is important in terms of flexibility because it allows the agent to act opportunistically using the SPM as a guide. Common deviations may be used as basis of further process refactoring and improvement.

The impact of the detected deviations to the objectives of the process should also be measured. This is accomplished

by Objective 2. It states that a deviation handling approach should be able to assess the risk of each deviation according to the process goals. For example, starting the second activity (cdm) before the first one (ucm) would represent a lower risk to the process goals than finishing the third activity ($code$) without having implemented each UML class as a Java class. This happens because making sure that all classes have been implemented is part of the objective of the process.

Furthermore, being able to guide the process agents in reducing this impact is also important. This is accomplished by Objective 3. This guidance is provided to the process agent by the means of *correction plans* that are sequences of actions to execute in order to correct or reduce the risk represented by a set of deviations.

III. Approach for Flexible Deviation Handling

This section presents our approach for flexible deviation handling. It consists in, provided that a set of deviations has been detected, informing the process agent about the risks of not fixing them immediately and in assisting her in actually fixing them. The PSEE does that by proposing sequences of actions that will correct deviations. The agents just need to choose the sequence of actions that fit their objectives and to perform them. When no solution that fixes every deviation exists, the PSEE proposes partial solutions to the process agent. These solutions are sorted from the lower to the higher overall risk level.

This section is organized as follows: Section III-A formally defines the problem of deviation detection and handling. This formalization is reused throughout the discussion of the approach. Sections III-B, III-C and III-D present our approach for accomplishing respectivelly the objectives of Risk Assessment, Guidance and Late Handling.

A. Background

In this section we formally define the problem of deviation detection and handling. This formalization is based on the one used to detect inconsistencies over sequences of actions initially defined in [3] and is reused throughout the presentation of our approach.

As a first step, we need to represent an observed process state. Let us define $s = a_0, \ldots, a_n$ a sequence of actions observed by the PSEE during process enactment. The vocabulary of actions must be able to represent every possible action during process enactment. For now, let us focus on the actions we call *process enactment actions*. For the sake of simplicity, let us consider that this group is composed of two kinds of actions. The action $launch(a, u)$ represents the beginning of the execution of the activity a by the process agent u. The action $finish(a, u)$ represents the end of the execution of the activity a by the process agent u.

An inconsistency rule over a sequence of actions is a rule R, written as the first order logic formula $R(s) \equiv F(s)$ that states that the inconsistency R is found in the observed process s if and only if the logical formula F holds over s. These rules can then be used to verify if an observed process is consistent or not with the process model. A deviation R is said to be present in an observed process s if and only if the inconsistency rule R holds for s.

Let us analyze the following example of inconsistency rule:

$$M'(s) \equiv \exists u \exists a \in s | a = launch(cdm, u) \wedge \\ \neg \exists a' \in s | a' = launch(ucm, u)$$

This rule states that a deviation exists in an observed process if the cdm activity has been launched by u if she has not launched ucm yet. This is a deviation because the process model states that cdm must be executed after ucm, and this implies that one cannot start cdm without having started ucm. Consider the observed process state $s = launch(cdm, u)$. In this observed process state, the deviation M' is detected because $M'(s)$ holds.

B. Risk Assessment

In our approach, the risk assessment is obtained by the means of the *deviation rules*. We define a deviation rule as a logical formula $M(s, r, c) \equiv F(s, r, c)$ that means that the deviation M is present in the sequence s if and only if the logical formula $F(s, r, c)$ holds. It extends the concept of inconsistency rule with r, that is the risk represented by this deviation and c, that represents its cause. The values for r come from the enumeration low, $medium$ or $high$ representing increasing risk levels.

Example 2: Let us consider the sample process in Example 1. The model states that ucm should be executed before cdm. This can be represented by the following deviation rules:

$$\begin{aligned} M(s,&low, c) \equiv \exists u \exists a, a' \in s \\ &a = launch(cdm, u) \wedge a' = finish(ucm, u) \\ &\wedge a <_s a' \wedge c = a \\ M'(s,&medium, c) \equiv \\ &\exists u \exists a \in s | a = launch(cdm, u) \wedge \\ &\neg \exists a' \in s | a' = launch(ucm, u) \wedge \\ &\wedge c = a \end{aligned}$$

The first rule states that a deviation happens if, at some point in the sequence s, the process agent u started the activity cdm *before* the end of the activity ucd. It constitutes a low risk deviation and its cause is the action a that launches the cdm activity. The second rule, M', represents another deviation related to the same statement in the process model: it states that starting the cdm activity without having done the ucm activity constitutes a medium risk deviation and the cause of this deviation is the action a that launched the cdm activity.

The risk associated to a deviation indirectly encodes the objective of the process. In the present example, executing ucm and cdm in any order may still lead to correct model, whereas creating the class diagram without the associated use case diagram is much less satisfactory. That is why the first deviation has a lower risk value than the second. Notice that it is up to the process modeler to decide the risk level and the cause that should be associated to each kind of deviation.

This deviation rule can now be evaluated in an observed process. Let us consider that the action $create(o, t)$ represents the creation of an artifact o of type t. Consider the following sequence:

$$s = launch(cdm, u), create(c, class), finish(cdm, u), \\ lauch(ucm, u), create(u, usecase), finish(ucm, u)$$

This sequence represents an observed process in which the process agent u executed the cdm activity before the ucm activity. Since $M(s, low, launch(cdm, u))$ holds, a deviation has been detected.

C. Correction Guidance

In our approach, a planning algorithm is used to compute the set of possible *correction plans* that are then presented to the process agent to guide her. A correction plan is a sequence of actions that when executed fix the deviations found in the observed process. Computing these plans is an inherently incomplete process, because there is an infinite number of potential sequences to be considered as correction plans.

The present approach reuses the algorithm defined in [8] to explore these possibilities. The overall risk represented by the deviations present in a sequence is used to heuristically classify all possible correction plans. We consider that one solution is better than another if it has less high risk deviations. If both have the same number of high risk deviations, the one with less medium risk deviations is better. If both have the same level of high and medium risk deviations, the one with less low risk deviations is better. They are not comparable otherwise[1].

This algorithm proceeds in three steps. In the first step, a set of *deviation rules* is used to detect deviations, their causes and risks. In the second step, a set of potential correction plans for fixing each cause of deviation is computed by the means of the so called *generator functions*. Finally, in the third step, the potential correction plans computed in step two are composed into final correction plans that are then presented to the process agent.

[1]Notice that this is just an heuristic we have adopted in the current work. In an specific context other strategies for comparing proposed correction plant may be more suitable.

Generator functions define a mapping from observed processes and deviation causes to correction plans. Given a sequence s and cause of deviation c, the expression $G(s, c)$ represents the set of correction plans for c in s. This function is defined in the form of logical rules expressed each one by a formula $E \implies s_c \in G(s, c)$, where E is a logical expression and s_c is a correction plan. The meaning of such rule is, if E holds, s_c is a correction plan for c in s.

An example of generator function follows:

$$\exists b, activity(b) \implies \\ (finish(a, u), launch(b, u)) \in G(s, launch(a, u))$$

This function states that every time a deviation has been caused by launching an activity a (represented by the action $launch(a, u)$), finishing this activity and launching another activity b may fix the deviation. Every possible activity b in the process model is going to be tried by the planning algorithm but only the ones that reduce the overall risk level will be presented to the agent.

Example 3: Let us take the Example 2. Using the generator function over the detected deviation we obtain:

$$G(s, launch(cdm, u)) \\ = \{(finish(cdm, u), launch(ucm, u)), \\ (finish(cdm, u), launch(cdm, u)), \\ (finish(cdm, u), launch(code, u)), \\ (cancel(launch(cdm, u)))\}$$

This indicates that, from that source of deviation, finishing the current activity and starting another and simply canceling the action that launches the offending activity are possible solutions to this deviation.

D. Late Handling

The late handling objective is achieved by our approach by simply allowing the process agents to continue process enactment even after a deviation has been detected. This can be done because, once the Risk Assessment objective has been fulfilled, the process agent knows which deviations have been detected, their risk to the process objectives and their causes. Once the Correction Guidance objective has been achieved, the PSEE can guide the agent into performing the actions that are needed to reduce the overall risk to process objectives.

IV. Adoption Process

In the present approach the following workflow is imposed:

1) The process modeler defines the process model.
   a) The process modeler defines the deviation rules to be used during enactment.
   b) The process modeler associates a risk level and cause to each rule.

c) The process modeler defines the generator functions that should be used to provide guidance to process agents in deviation scenarios.
2) The process agent enacts the process model by the means of the PSEE.
3) The PSEE continuously displays the set of currently detected deviations and their associated risks.
4) The process agent may ask the PSEE to compute a correction plan to guide her into fixing detected deviations.

The first step consists in defining a process model. In fact, our approach does not mandate any specific process model language, but in order to detect and handle deviations, the notion of what is allowed or not by the process model should be translated into a set of deviation rules (Step 1-a). This example will take into consideration process models represented as UML2 activity diagrams. Suppose that the $precedes(a, b)$ predicate means: the activity a precedes the activity b in the software process. The following deviation rule P encodes the fact that if a_1 precedes a_2 one should not start a_2 without having executed a_1:

$$\begin{aligned} P(s,medium,c) \equiv\ & precedes(a_1, a_2) \wedge \\ & \exists u \exists a \in s | a = launch(a_2, u) \wedge \\ & \neg \exists a' \in s | a' = launch(a_1, u) \wedge \\ & c = a \end{aligned}$$

This is an example of rule that does not depend on the actual process model being enacted. That means that it could be automatically generated without any extra overhead to the process modeler.

The next step (Step 1-b) would be assigning a risk level and a cause to each rule. In the case of the P rule, this has already been done, by stating that the risk represented by this deviation is $medium$ and that its cause is the action a that launched the activity a_2 before launching the activity a_1. Different risk level and cause may be more appropriate in different contexts.

Step 1-c consists in defining the set of generator functions to be used to provide guidance during process enactment. The following example of generator function states that if c is a cause of deviation during activity a a correction plan is canceling c, finishing the current activity and starting one in which c is allowed:

$$\begin{aligned} & activity(a) \wedge activity(b) \wedge precedes(a, b) \implies \\ & \quad (cancel(c), finish(a, u), launch(b, u), c) \in G(s, c) \end{aligned}$$

Notice that Steps 1-a b and c do not represent a burden to the process modeler but bring more flexibility to the whole process. A library of generic deviation rules and generator functions could be defined in advance for each process model language. The work of the process modeler would be reduced to choosing the rules that she would like to apply for a given process, customizing their parameters and writing process specific rules ones, if necessary.

The Steps 2-4 are automatically executed by the PSEE. In Step 2, it allows the process agent to enact the process model; in Step 3, it detects violations to deviation rules and displays the list of detected deviations and associated risks; and in Step 4, it computes correction plans for the current set of deviations.

The essential difference between this workflow and the classic one is that in the second one the process agent may delay the handling of deviations as long as she wants to (Late Handling). The detected deviations are also prioritized according to the risks they represent to the process objectives as defined by the process modeler in Step 1 (Risk Assessment) and guidance is optionally provided in order to assist the process agent into fixing the detected deviations (Correction Guidance).

V. Preliminary Evaluation

In this section we present our preliminary evaluation of the approach described in the previous section. It has been prototyped as a PSEE that allows a process agent to enact a process, get the list of the currently detected deviations, their risks and get correction plan proposals. Furthermore, a process enactment scenario based on the process presented in Figure 1 was executed. The prototype, the complete source code for the presented scenarios and other ones are available for download in the website of the prototype[2].

A. Process Implementation

Even though the considered process (Figure 1) has only three activities, it allowed us to represent different kinds of deviation rules and generator functions with a conveniently small number of rules. Figure 2 overviews the nine deviation rules we implemented as part of this experiment.

Five of them are completely independent of the process model. They verify inconsistencies on the use of the `launch` and `finish` actions (the four rules whose name start with `wfr`, which stands for *Well-formedness Rule*) and the order in which the actions were enacted in the observed process when compared to the process model (the rule called `order`). The other four rules implement additional constraints that are not directly part of the activity diagram but come from the natural language description of each activity. Three of these rules define, for each activity, which actions may be performed by the agents during each activity (the rules called `ucm`, `cdm` and `code`). The last rule verifies the postcondition of the `code` activity: it makes sure that every UML class is implemented by at least one Java class (the rule called `codep`).

We also needed to associate each rule to a risk level. The associations we present here only represent a possible

[2]http://lip6.fr/Marcos.Almeida/

Rule Name	Description	Risk Level
wfr1	Makes sure that every launch starts an activity that was not in execution	High
wfr2	Makes sure that every finish finishes an activity that was under execution.	High
wfr3	Makes sure that no activity can be launched more twice by the same agent at the same time.	High
wfr4	Makes sure that every artifact modification action happens during the enactment of some activity.	Medium
order	Makes sure that the activities are executed in the order defined by the process model	Low
cdm	Makes sure that only actions that create or modify a class diagram are allowed during the cdm activity.	Medium
ucm	Makes sure that only actions that create or modify a use case diagram are allowed during the ucm activity.	Medium
code	Makes sure that only actions that create or modify the source code are allowed during the code activity.	Medium
codep	Makes sure that by the end of the code activity every class in the class diagram has a Java implementation in the source code.	High

Figure 2. Description of Deviation Rules.

Rule#	Cause	Proposed Correction
1	Any	Finish current activity and start another one in which this event is allowed.
2	Any	Cancel cause.
3	launch(a,u)	Start another activity.
4	launch(a,u)	Finish current activity.
5	create(c, uml_class)	Create missing Java classes.

Figure 3. Description of Generator Functions.

scenario. Four rules detect high risk level deviations: wfr1, wfr2, wfr3 and codep. That was done because when one of the first three kinds of deviation is detected that means that the current observed process state is meaningless (e.g. it makes no sense to finish an activity that has not been started yet). The last one is marked as a high risk level one because it encodes part of the process objective: obtain the set of Java classes that implements the requirements and the class diagram.

Four rules detect medium risk level deviations: wfr4, cdm, ucm and code. That was done because these rules detect actions that were executed when they were not expected to (e.g. during the wrong activity). Even though this constitutes a deviation, we consider it as a lower risk level deviation when compared to not producing an implementation for every class in the class diagram. It happens because, executing actions when they were not expected is not necessarily an obstacle to the process objective.

Finally, one rule detects low level deviations: order. This rule makes sure that the activities in the process model were executed in the correct order. We consider that executing all activities is much more important in the present process model than executing them in the specified order.

Six Prolog rules implement the Generator Functions for this process. Two of them propose plans based on canceling the deviating action (rules 1 and 2). Two others propose plans based on changing the current activity, e.g. finishing the current activity and starting another one (rules 3 and 4). One of them proposes a mixed plan, based on canceling and changing current activity, and, finally, the last one proposes a fix to a structural deviation concerning the postcondition of the code activity: if a UML class is not reflected as a Java class, it proposes creating the missing class (rule 5).

B. Sample Execution Scenario

This section shows how our approach accomplishes our three goals for flexible deviation handling in a sample execution scenario. This scenario has been chosen because, even though it is very short, it is representative of a case in which deviations were ignored from the beginning of the process enactment, and still can be fixed later. It contains many different kinds of deviations and depicts a particular case in which the solution for one deviation also fixes another one. The output of our sample PSEE is presented at each step of the deviation handling process.

Late Handling. The tested scenario consists in 9 actions presented below:

```
Timestamp       Event
      1         launch(cdm,u)
      2         create(controller,uml_class)
      3         finish(cdm,u)
      4         launch(ucm,u)
      5         create(model,uml_class)
      6         finish(ucm,u)
      7         launch(code,u)
      8         create(helper,java_class)
      9         finish(code,u)
```

The first action consists in the user u starting the cdm activity. She then creates the UML class controller and finishes the activity. At timestamp 4 she launches the activity

ucm and creates the UML class model and finishes the activity. Finally, she launches the code activity and creates the helper Java class. Notice that the process agent is able to execute her actions without being forced to deal with the eventual deviations when they are first detected.

Risk Assessment. At any point the list of currently detected deviations can be consulted. This list at timestamp 9 is as presented below:

```
Timestamp       Action               Rule    Risk Level
  2    create(controller,uml_class) codep      high
  5      create(model,uml_class)    codep      high
  5      create(model,uml_class)     ucm       medium
  1            launch(cdm,u)        order      low
```

This table displays the list of detected deviations ordered first by decreasing risk level and then chronologically. In the present scenario, four deviations have been detected. The first two lines point out that the code activity has been finished but the controller and the model UML classes still have not been coded into Java classes. The third deviation points out that the model class was created during the ucm activity and finally, the fourth deviation states that the agent launched the cdm activity before the ucm activity in timestamp 1.

Correction Guidance. The agent is now able to obtain a list of possible plans to reduce the risk level by fixing some of these deviations, the result of this computation is provided below:

```
Searching for plans...
2 solutions found...

Correction Plan
====== ====

cancel(9)
create(controller,java_class)
create(model,java_class)
finish(code,u)
launch(cdm,u)
cancel(5)
create(model,uml_class)
finish(cdm,u)
launch(code,u)

Risk assessment at the end: 0,0,0

...
```

This printout shows that 2 solutions were found. Both of them reduce the risk level to what is indicated by 0,0,0 which means 0 high risk deviation, 0 medium risk deviation and 0 low risk deviation. The presented solution starts by canceling the last action, that finishes the code activity and then creating the two missing Java classes: controller and model. tmodel class in the correct activity. Finally, the cdm activity is finished and the code activity is initiated again.

C. Discussion

This section discusses the merits and the limits of our approach from the viewpoints of the two stakeholders related to this context: the process agent, and the process modeler. From the process modeler's viewpoint, our study showed that a process containing three activities was implemented as a set of 14 rules: 9 deviation rules and 5 generator functions. At first sight, this could appear to be a lot, but one needs to take into consideration that most of them (64%) are independent from the process model. These rules could therefore be reused as a library or be automatically generated to be applied to another process model. The remaining 5 rules are specific to this process model. That means that our prototype would still be fully functional without them. It would be able to verify that the process activities were executed in the correct order and with the correct artifacts.

From the process agent's viewpoint, our results show that our approach is implementable in a PSEE and that it is able to support enactment scenarios in which process agents are able to deviate from the process specification while accomplishing the three objectives of this work. As we suggested in the introduction, these features provided by our approach are more appropriate to the opportunistic way of working employed by process agents. However, since we only executed scenarios with only our approach; we could not assess if our approach would be perceived as more useful by process agents when compared with a scenario using a different approach. This has been left as future work.

VI. Related Work

To the best of our knowledge, no current approach in software process enactment fully accomplishes the three objectives of our approach: Late Handling, Risk Assessment and Correction Guidance. These approaches can be divided into three groups. The first group does not allow any deviation during process enactment [17] and therefore does not support any of the objectives. The second group allows deviations during process enactment [5], [10], assess their risk but are not able to provide any correction guidance or late handling. Finally, the third group [20] realizes an *à posteriori* analysis of a log of the actions executed by the process agents to detect each deviation and assess their risk. Correction guidance is however left to process agents.

These objectives however have been separatedly addressed in the Business Process Management domain before. Business Process are more general then software processes, that describe the services provided by a set of agents (most of them not human) [12]. These services are also structured into activities whose ordering constraints are defined by the process model. Works like [2], [14] define languages for expressing deviation rules and generator functions and therefore can automatically generate correction plans for deviations. However they focus on mostly automatized processes executed by WebServices, and they do not provide then

late handling and risk assessment during process enactment. Other works focus on business processes mostly executed by humans [16]. They focus on identifying and assessing the risks of deviations during process enactment. They usually provide late handling handling for these deviations, but do not offer any correction guidance.

VII. Conclusion

In the present work, the *deviation rules* are used to formally specify deviations in such a way that: (i) they may be fixed afterwards (Late Handling); (ii) they provide pieces of information necessary to estimate the risk of the current observed process (Risk Assessment) and (iii) they may be used as input to planning algorithms that derive correction plans that assist the process agents in reducing this risk (Correction Guidance). The present approach is process model independent and was prototyped and evaluated in a set of enactment scenarios. Our preliminary results show that it accomplishes the three desired goals, however a deeper evaluation is still necessary. In order to accomplish that, as perspectives of this work, we are studying the validation of our prototype in the context of a bigger case study in an industrial use case provided by our industrial partners. This use case should include the integration of our current prototype into an industrial environment.

References

[1] S. C. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the spade environment. *IEEE Trans. Softw. Eng.*, 19(12):1128–1144, 1993.

[2] L. Baresi, S. Guinea, and P. Plebani. Policies and aspects for the supervision of bpel processes. In J. Krogstie, A. L. Opdahl, and G. Sindre, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2007.

[3] X. Blanc, A. Mougenot, I. Mounier, and T. Mens. Detecting model inconsistency through operation-based model construction. In Robby, editor, *Proc. Int'l Conf. Software engineering (ICSE'08)*, volume 1, pages 511–520. ACM, 2008.

[4] A. G. Cass and L. J. Osterweil. Process support to help novices design software faster and better. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *ASE*, pages 295–299. ACM, 2005.

[5] G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Trans. Software Eng.*, 24(11):982–1001, 1998.

[6] M. A. A. da Silva, R. Bendraou, X. Blanc, and M.-P. Gervais. Early deviation detection in modeling activities of mde processes. In Petriu et al. [15], pages 303–317.

[7] M. A. A. da Silva, A. Mougenot, R. Bendraou, J. Robin, and X. Blanc. Artifact or process guidance, an empirical study. In Petriu et al. [15], pages 318–330.

[8] M. A. A. da Silva, A. Mougenot, X. Blanc, and R. Bendraou. Towards automated inconsistency handling in design models. In *Proceedings of the 22st International Conference on Advanced Information Systems, CAISE'10*, volume 6051 of *LNCS*, pages 348–362. Springer, June 2010.

[9] A. Egyed, E. Letier, and A. Finkelstein. Generating and evaluating choices for fixing inconsistencies in UML design models. In *Proc. ACM/IEEE Int'l Conf. Automated Software Engineering (ASE '08)*, pages 99–108, New York, NY, USA, 2008. ACM.

[10] M. Kabbaj, R. Lbath, and B. Coulette. A deviation management system for handling software process enactment evolution. In Q. Wang, D. Pfahl, and D. M. Raffo, editors, *ICSP*, volume 5007 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2008.

[11] F. Lanubile and G. Visaggio. Evaluating defect detection techniques for software requirements inspections, 2000.

[12] A. Lindsay, D. Downs, and K. Lunn. Business processes–attempts to find a definition. *Information & Software Technology*, 45(15):1015–1019, 2003.

[13] J. Lonchamp. A structured conceptual and terminological framework for software process engineering. In *In Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, 1993.

[14] K. Mahbub and G. Spanoudakis. A framework for requirents monitoring of service based systems. In *Proceedings of the 2nd international conference on Service oriented computing*, ICSOC '04, pages 84–93, New York, NY, USA, 2004. ACM.

[15] D. C. Petriu, N. Rouquette, and Ø. Haugen, editors. *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part II*, volume 6395 of *Lecture Notes in Computer Science*. Springer, 2010.

[16] S. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 149–164, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] T. J. Sliski, M. P. Billmers, L. A. Clarke, and L. J. Osterweil. An architecture for flexible, evolvable process-driven user-guidance environments. *ESEC/FSE*, ?:33–43, 2001.

[18] G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. In *IN HANDBOOK OF SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*, pages 329–380. World Scientific.

[19] W. Visser. More or less following a plan during design: Opportunistic deviations in specification. *International Journal of Man-Machine Studies*, 33(3):247–278, 1990.

[20] N. Zazworka, V. R. Basili, and F. Shull. Tool supported detection and judgment of nonconformance in process execution. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 312–323, Washington, DC, USA, 2009. IEEE Computer Society.