Contents

1 Overview

1.1 What is PRODAN?

PRODAN is a software process enactment tool that allows to enact the process models in an environment where the modeler is guided throughout the execution to avoid process deviations. Different approaches follow different techniques for handling process deviations. Some tools restrict users (e.g. project managers) to execute the activities only in the order that they appear in the original process model, while others ignore process deviations completely.

PRODAN analyses the user's decision for activity execution and detects if it would result in a process deviation. In case, a deviation may result from the execution of chosen activity, the user is notified of the deviation. This deviation notification shows the list of constraints that would be violated, if the user wants to continue anyway. However, the user is not restricted to follow the original process model and may choose to deviate from it. In all cases, user is supported by the tool to take informed decisions.

1.2 Who can use PRODAN?

PRODAN, as a process enactment tool can be used by project managers to enact process models. This enactment of process models simulates the control flow perspective of the real life processes, carried out in the enterprises. They help in developing the understanding of the flows of a process, before its actual implementation. It can also be used by the process modelers to analyze the effect of different deviations that can occur during the enactment of the process. The control flow behavior of a process deviation can be analyzed by the suggested continuation options offered by tool, after the deviation. This helps to guide project managers to make informed decisions for the enactment of processes.

1.3 How does it work?

PRODAN uses the UML designer from Obeo, to edit and visualize UML 2.4 activity diagrams for the specification of process models. This process model and the current execution sequence is used to generate a process enactment ruleset in Alloy. Alloy Analyzer, as a constraint solver takes this process enactment ruleset as a constraint satisfaction problem (CSP) and provides a viable solution that does not violate any of the given rules. Current execution sequence is taken into account, each time a solution is provided by the constraint solver. It allows the detection of deviations during the enactment of the process model. The violated constraints are presented to the user to inform him/her of the potential risks that can arise, in case of deviation.

1.4 License

PRODAN is a plug-in developed under the research project, MERGE. It is open source and is distributed under GNU General Public License GPL v3.0.

2 Plug-in Installation

A prerequisite for installing PRODAN V1.0 is the installation of Merge platform version 2.x onwards (or Eclipse Indigo onwards). Two mechanisms are offered to install the plug-in in the merge platform. It can be installed either by using the provided update site or by using the plug-in archive provided alongside this user guide.

! → The contents of archive can manually be copied into the Merge platform plugins folder, but we strongly recommend to use one of the suggested approaches for the installation of PRODAN plug-in.

2.1 Installation - update site

1. Open Merge platform (Eclipse), and go to `Help` > `Install New Software`.
2. This shall open an install window as shown in figure **??**. Click the `Add` button. This shall open an Add Repository dialog box. Add the following information and click `OK`:

Name : *PRODAN*

Location : *http://pagesperso-systeme.lip6.fr/Yoann.Laurent/PRODAN* [1]

Figure 1: Installing the Plug-in - update site

3. Click `OK` and uncheck the option `Group items by category`.
4. Select the plug-in `fr.lip6.move.meta.TracedEnactment` and click `Next`. Click `Next` once again for the installation details.
5. Accept the license agreement and click `Finish`.

2.2 Installation - Local archive

1. This user guide is distributed alongside the plug-in and an example process model. The plug-in is archived in a file named `PRODAN v1.0.2`. Locate the plug-in archive file and uncompress it in a desired location.
2. Open merge platform (eclipse), and go to `Help` > `Install New Software`.

Figure 2: Installing the Plug-in - local archive

3. This shall open up a window for plug-in installation. Click the `Add` button. This shall open an Add Repository dialog box, as shown in figure **??**. Add the name of the plug-in as PRODAN.
4. Click `Local` and browse to the uncompressed plug-in folder.
5. Click `OK` and uncheck the option `Group items by category`.
6. Select the plug-in `fr.lip6.move.meta.TracedEnactment` and click `Next`. Click `Next` once again for the installation details.
7. Accept the license agreement and click `Finish`.

[1] Update site will soon be moved to http://merge.lip6.fr/PRODAN

3 Tool Layout

PRODAN is presented as an eclipse plugin, which alloys to model and enact process models. The tool is composed of different views. Each of these views is discussed in detail below.

3.1 Process Editor

Process editor allows the development of process models using UML 2.4 activity diagrams. The tool relies on the control flow of a process model to detect process deviations. This editor allows to add activities, actions, initial node, final node, and other control flow nodes like decision, merge, fork and join nodes. Apart from these nodes, the `activity tools` toolbox at the right side also allows to add control flows between these nodes. These control flows link the nodes to get a process graph. Figure **??** shows a testing sub-process that is a part of a complete software development process. It contains four activities: *Prepare release candidate*, *Regression test*, *Ad hoc testing* and *Release for user acceptance test*. This process occurs in the final stage of testing after integration & system testing and before the user acceptance testing.

Figure 3: Process editor view

Activities are added to the process model through the notion of Activity states. Opaque actions are atomic activities of a process model that can not be decomposed any further through activities. The description of an opaque action describes its inner implementation. This implementation can be added to the action using its "body" property under the semantics tab of properties view. Different roles associated with the process model can be modeled using partitions in activity diagram, which is a similar concept as swimlanes in BPMN.

3.2 Process Enactment view

Process enactment view of the PRODAN tool allows to enact the process model. Controls for executing and terminating a process are provided by this tool. Once the execution of the process model is started, this view lists all the activities that can be executed in the process model. It offers the possibility to start and finish an activity from the process model. The state of the activity (i.e. currently running or not) is also presented in this view. Figure **??** presents the process enactment view when *Prepare release candidate* activity has already completed its execution. Process enactment view suggests the user to execute any of the two activities: *Regression testing* or *Ad hoc testing*.

Figure 4: Process enactment view

Apart from presenting the list of executable activities and their current state, this view is also important for guiding the user for the next activities to be executed. Next activities are not calculated on the basis of the overall process automaton, instead they are calculated through the solution provided by the constraint solver. The constraint solver provides the solution of the constraint solving problem (i.e. process enactment ruleset in our case) using the process execution trace. Thus it can effectively suggest activities to be performed which were skipped due to

certain deviations in the past. Deviation in process enactment results in the violation of certain constraints. Some of these constraints still remain satisfiable, if the user chooses to perform these activities at a later stage.

3.3 Execution Trace

The process execution trace view provides a trace of process enactment. It lists all the activities that were executed from the process model. This list of executed activities is presented in temporal order, which can be used to make some key decisions for further enactment of a process model. Figure **??** shows the trace view of our process example, where *Regression testing* activity was executed after the *Prepare release candidate* activity. *(none)* shows the state of the process enactment, where no activity is executing.

Figure 5: Execution trace view

3.4 Deviations View

Figure 6: Deviations view

The deviation view of the PRODAN tool lists all the constraints associated with the process model. These constraints are generated from the process model. This view also presents the state of each constraint, which can be `Satisfied`, `Satisfiable` or `Deviated`, as shown in Figure **??**.

Further information about a specific constraint of a process model can be obtained by double clicking it in deviations view. It opens up an information dialog box that gives a description for the constraint, as shown in figure **??**. It also gives the reason behind the implementation of a particular constraint.

Figure 7: Constraint Information

4 Getting started by example

Let us try to execute a simple example of a process already shipped with this tool. This would help in understanding the process enactment and deviation aspects of the tool.

4.1 Setup the environment & example

Following steps can be used to setup the environment for process enactment. This will allow us to execute *Finalize Testing* example later on.

1. Open Merge platform (or Eclipse) and go to `File` > `Import`. Then select `General` > `Existing Projects into Workspace` and click `Next`. (Consult figure **??**)

Figure 8: Import UML Project

2. Select `archive file` option and browse to the example archive file, *Finalize Testing Example.zip*, provided alongside this plugin. Click `Finish`.

3. This will import a UML project in your current workspace. In Model explorer view, open this project. Double click the *example* diagram to open it in the editor. (Consult figure **??**)

Figure 9: Expand Model explorer

4. This shall open up the graphical model (UML activity diagram) for this example in the editor, as shown in figure **??**

Figure 10: Example Process model in editor

5. Use `Window` > `Show view` > `Process Guidance` to open up Process Enactment, Trace, Guidance and Deviations views.

4.2 Execute example process

You can follow the following steps to execute the *Finalize Testing* example:

1. Select the *Finalize Testing* process in the editor and click `Start` to initiate the execution of the process. It takes a little time to load the process model and when its ready, you will see a list of all the activities in the process enactment view. You will be able to notice that no activity is running at the moment. Even though the process in initiated at this state, no particular activity started its execution yet. You can see that *Prepare release candidate* is the first activity proposed by the tool.

2. Click on *Prepare release candidate* from the list in enactment view or the graphical representation of this activity in model editor and click `start`. This executes the activity and its status changes in the enactment view.

3. After it has started, you can `Finish` it. Once this activity is finished, you are proposed to go for *regression testing* and *ad hoc testing*, as shown in figure **??**. This does not mean that you can execute both of them, it simply means to go for one of these options.

Figure 11: Enactment view - Termination of *Prepare release candidate*

4. Start and finish *Regression testing*. Then start *Ad hoc testing*. When you try to execute *Ad hoc testing*, a message box pops up that shows the violated constraints if you continue with this execution, as depicted in figure **??**. Continue the execution anyway.

Figure 12: Warning message - violated constraint

5. You can notice the `deviations view` of the tool that lists all the constraints of the process, as illustrated in figure **??**. The state of each constraint shows if it is `satisfied`. In case the constraint is not satisfied, whether it is `satisfiable`, during the current execution. It also shows the `violated` constraints of the process model.

Figure 13: Deviation view - Violated constraint

6. The `Trace view` can be consulted to see the order of execution of the activities that were executed till now. Resulting from our execution scenario, we have *Prepare release candidate*, *Regression testing* and *Ad hoc testing* activities respectively, as shown in figure **??**. A *(none)* state at sequence 0 specifies that there was no activity running. Then *prepare release candidate* was executed and then again at sequence 2, we see another *(none)*. Sequence 2 shows that *prepare release candidate* was finished and thus no activity was executing at that time. The final sequence is 5, that shows that *Ad hoc test* was executed last. However, it also shows that this activity was not finished, when the trace was taken.

Figure 14: Deviation view - Violated constraint

7. You can terminate the process model execution at this time or continue till the *Release for user acceptance test* activity is finished.

5 Process Development

PRODAN uses Obeo UML Designer for the development of process models. These process models are then made available to the tool for enactment. Application of specific profiles developed for the PRODAN process models allows to add specific attributes to them.

5.1 Creating a new process model

1. In Merge platform, go to `File` > `New` > `UML Project`. Then give this project a name as "Sales Example". This will create a new project in your workspace. Click `Finish`.
2. Use `Window` > `Show view` > `Process Guidance` to open up Process Enactment, Trace, Guidance and Deviations views.
3. This will create a UML project in your current workspace. In `Model explorer view`, open the project hierarchy for this project. Open the hierarchy for *model.uml* and then right click `NewModel` > `New Representation` > `Activity Diagram`. Give it a name like "Sales Activity Diagram". (Consult figure **??**)

Figure 15: Creating an activity diagram

4. Select the NewModel in the process editor and enlarge it so that multiple activities can be placed inside it. Go to `properties view` and change its name to *Sales Process*.
5. Choose `Partition` from the activity tools at the right side and create two partitions in this model. Using `Properties view`, change their names to *Sales department* and *Warehouse*. Place an Initial Node and an Activity Final Node outside the partitions, as illustrated in figure **??**.

Figure 16: Creating partitions for different roles

6. Now create opaque actions inside the partitions, as shown in the figure **??**. You can change the name of the activities from `properties view` > `General` and remove the body text from `Properties` > `Semantic` > `Body`. You can also resize the activities according to your personal liking.
7. From the activity tools, select `decision node`, `merge node`, `fork node` and `join node` and place them in the process model as shown in figure **??**. Using the `control flow` link the *initial node* to *Receive order* activity. Keep linking the nodes till *activity final node*.

Figure 17: Complete Sales Process

8. We have developed a process in PRODAN tool. Now save it before its execution.

5.2 Adding guidance to activities

We will take the process developed in section **??** and add guidance to each activity in the process model. This will complete the development

of the process model. The guidance added for each activity is available to the user, at the time of enactment.

1. In `Model editor view`, click the profiles button to add profiles to the model. This button is accessible only when no activity or process is currently selected in the editor. This shall open up the `Profiles toolbox` at the right. First marking in figure **??** shows the profile button to add `Profile toolbox`, which is highlighted as second marking.

Figure 18: Applying profile

2. From the `Profiles` toolbox, select `Apply profile` and click on the *Sales Process*. This opens up a window to select the profile. Select *SoftwareProcessProfile* and click `Add` (or double click it). Click `Finish`. This adds our profile to the *Sales Process*.
3. Now we need to apply stereotype to each activity in the process model. For doing this, select `Apply stereotype` from the `Profiles` toolbox and apply it on the chosen activity. This opens up a window to select the stereotypes to apply. Choose *SoftwareAction* stereotype by either double clicking or by pressing the `Add` button. Finally click `Finish`.
4. We can repeat this process for every activity in the process model. We can also choose to attach guidance to only few of the activities, in which case, we add stereotype to the chosen activities only.
5. Once the stereotypes are applied to the activities, save the project. Expand the *Sales example* project hierarchy in `Model Explorer view`. At the end of the hierarchy, we can see a list of `Software Actions`, as shown in figure **??**.

Figure 19: softwareActions in Model Explorer

6. Select the first `SoftwareAction` and open the `Properties view`. You can see the name of the concerned activity in the properties view. (These software actions will be named according to the activity names in next version.) You can add guidance for the enactment of each activity in the `Description` property.
7. Keep adding the description to each activity, so that they are available as guidance during the enactment. Figure **??** shows the *Sales Process* after a guidance is added to all its activities.

Figure 20: Applying profile

8. Finally, save the process before attempting to execute it.

6 Process Enactment

The execution of a process model is called its enactment, in the process domain. We shall execute the process model that we developed in the earlier section, the *Sales Example* process.

6.1 Starting & terminating a process

1. From the *Sales example* UML project, open the *Sales Activity Diagram* in the editor view, if it is not open.
2. Open the `process enactment view` of the tool. In the top row of process enactment view, it shall state that process is not selected. We have to select the *Sales Process* from the `editor view`, so that it can be loaded in this view.
3. Once, the process is loaded, we can see two buttons across the process name: `Start` and `Terminate`. `Start` button shall execute the process and `Terminate` button shall terminate it.

! → For demonstrating the control flow of the processes and the deviations from the original process model, we focused on the transfer of active execution between the activities. For this reason, we did not go for a detailed lifecycle for the process and activities.

4. Once the process is executed, we can see its activities in the table shown in `process enactment view`.
5. The `deviation view` lists all the constraints generated from the *Sales Process*. It also shows the state of each constraint.
6. Once `process enactment view` and `deviations view` is populated with activities and constraints respectively, we can continue the execution of individual activities.

After the execution of all the activities, one should terminate the execution of the process.

6.2 Starting & completing an activity

1. Once the process is started, the user can execute its activities. Before executing this activity, we can see that *Receive order* activity is the suggested by the tool, as shown in the `process enactment view`. We can also see that *initial_Activities* constraint in the `Deviations view` is satisfiable. (Consult figure **??**)

Figure 21: Enactment view - Start of Sales Process

2. We start with executing the first activity i.e. *Receive order*. This activity can either be selected from the graphical model or from the process enactment view. Once the activity is selected, the start button on right of the table can be used to start its execution. During the execution of the activity, we can see that it is running from the `process enactment view`. The running state of the activity is enabled in the table. (Consult figure **??**)

Figure 22: Enactment view - Running *Receive order* activity

As soon as the execution of the *Receive order* activity started, the state of the *Initial_ Activities* constraint in the `deviation view` changed from *satisfiable* to *satisfied*. A user can forcibly start each activity, but this would violate the constraint that the process execution should start with *Receive order* as first activity after the *initial node*.

3. Let us imagine that this activity is performed. And now we want to finish the execution of this activity. The `Finish` button in the process enactment view can be used to stop the execution of this activity. After the completion of *Receive order* activity, we can see that no activity is running in the `process enactment view`. However there are two suggested activities to be executed next: *Reserve order* and *Order stock*. (Consult figure**??**)

! → Suggestion of multiple activities in process enactment view does not mean that all of those activities should be executed. It suggests to execute of these activities.

Figure 23: Enactment view - Two suggested activities

4. The user can continue the execution of activities by following the suggested activities by the tool. This would ensure that process does not deviate from the original planning. Once the final activity is finished, the process can be terminated.

! → A user has access to the guidance of each activity through the `Guidance` view. Figure **??** shows the guidance for the *Receive order* activity in *Sales Process*.

Figure 24: Activity guidance for *Receive order* activity

6.3 Understanding process trace

Let us follow the steps 1 - 3 of section **??** (starting and completing an activity), this gives us the state where the process is executing and only *Receive order* activity was executed and finished by the user. After *Receive order* activity, we can choose any of the two activities *Reserve stock* and *Order stock*. Let us execute *Order stock* activity and then finish it. We can continue further to the execution of *Forward to warehouse* activity.

At this point if we open the `Trace view` of the PRODAN tool, we can see the execution trace for the process enactment of *Sales Process*, as shown in figure **??**. This trace gives us the order in which the activities were executed till the current state. Thus we can see that the execution started from *Receive order* activity, followed by *Order stock* activity and finally *Forward to warehouse* activity was executed. The *(none)* value represents the instance when no activity is running in the process.

Figure 25: Process enactment trace

7 Process Deviations

7.1 Deviating from a process

1. Let us follow the steps 1 - 3 of section **??** (Starting and completing an activity), this gives us the state where the process is in execution and only *Receive order* activity was executed and finished by the user. After *Receive order* activity, we can choose only one of the two activities *Reserve stock* and *Order stock*.
2. Execute *Order stock* activity and then finish it. At this point, the tool suggests us to execute *Forward to warehouse* activity. It would be against the original process plan to execute *Reserve stock* activity, as we can not execute both *Order stock* and *Reserve stock* activities.
3. Try to execute the *Reserve stock* activity at this point, against the original flow of the process model. This triggers a warning message box that informs the user of the constraints that would be violated by the execution of this activity. This error box is illustrated in figure **??**. It also gives a little description of the constraint. The user can choose to cancel the demanded action or continue anyway.

Figure 26: Warning message box for deviation

4. Click `OK` in the message box, where we would intentionally like to deviate from the process model. This means that tool informs us about the constraints that are being violated by the execution of an activity that shall deviate from the process model, but it does not restrict us from executing that activity.
5. We can continue to execute the process till its termination or continue with the next section (Understanding the violated constraints) from the current state.

7.2 Understanding the violated constraints

1. Let us follow the steps 1 - 4 of section **??** (Deviating from a process), this gives us the state where the process is in execution and we executed *Reserve stock* activity after the execution of *Order stock* activity, which caused a deviation.
2. In the deviations view of the tool, we can see all the constraints related to the *Sales Process*. There are three types of constraints that are generated from the process model, other than the *initial node* constraint:

Response(a,b) : This constraint specifies that activity b must be executed anytime during the execution of the process, once activity a is executed. This does not constrain the execution of activity b to be executed directly after the execution of a.

Precedence(a,b) : This constraint specifies that activity b can be executed only if activity a has already been executed in the process. Again, this does not constrain activity a to be a direct precedent of activity b.

Existence(a) : This constraint specifies that activity a must be executed exactly once during the execution.

3. We can also notice the states for each constraint shown in the deviation view. There are three kinds of states possible for a constraint. These are

Satisfied : A constraint on a process model is satisfied if there is no action taken that violates the satisfiability of the given condition.

Violated : A constraint is violated if the specified condition was not followed during the execution of the process model.

Satisfiable : A constraint that is not currently satisfied (as a special case of violated constraint) but some future action of user might make it possible to satisfy this condition.

8 Support for Control-flow patterns

Workflow control flow patterns are offered by the *Workflow Pattern Initiative*[2]. These patterns can be used to analyze the strengths and weaknesses of different process modeling approaches. We have implemented the first five patterns to show the applicability of this tool. We have taken some steps ahead to classify different kinds of deviations that may arrive in each pattern. This section presents the analysis for enactment and deviations of these patterns.

8.1 Pattern 1 - Sequence

A task in a process in enabled after the completion of a preceding task in the same process. Two tasks can be connected directly, so as to show a sequence of their executions. They are connected through the control-flow arrow in UML activity diagram.

Figure 27: Pattern 1 - Sequence controlflow pattern

Enactment Analysis This pattern can be implemented in PRODAN for enactment, as shown in figure **??**. Four tasks (A, B, C, D) are modeled in a sequence. Execution of the process starts through the *initial node* and then A is proposed for the execution. After the execution of each activity, the next activity in sequence is proposed for execution.

Deviation Analysis The tool responds to the deviations, depending upon the different type of deviation in the process.

1. *Skip A instead execute B:* the tool issues a warning of deviation where two violated constraints are shown to the user. c1) A is the initial activity and c2) B should be performed after A. Proposed activity for further execution A and C. (why A, to be discussed ?)
2. *Skip B instead execute C:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) B should be performed before C. Proposed activity for further execution B and D.
3. *Skip D and complete the process:* the tool does not provide a way to do it, except terminating the process (which is different from the completion of a process).

8.2 Pattern 2: Parallel split

The divergence of a branch into two or more parallel branches each of which execute concurrently. It is captured in UML through a ForkNode. A fork node in UML is a control node that splits the flow into multiple concurrent flows.

Figure 28: Pattern 2 - Parallel Split controlflow pattern

Enactment Analysis This pattern can be implemented in PRODAN for enactment, as shown in figure **??**. Five tasks (A, B, C, D, E) are modeled such that B and C are forked after A. Tasks D and E are

[2] http://www.workflowpatterns.com/patterns/control/

modeled to study the post-conditions of B and C. Task F is not part of the pattern and is just modeled for the sound completion of model. Execution of the process starts through the *initial node* and then A is proposed for the execution. After the execution of A, B and C are proposed for execution. Both these tasks (B & C) can run concurrently on the tool.

Deviation Analysis The tool responds to the deviations, depending upon the different type of deviation in the process.

1. *Skip A instead execute B:* the tool issues a warning of deviation where two violated constraints are shown to the user. c1) A must be the initial activity and c2) B should be performed after A. Proposed activities for further execution A and D.
2. *Skip A instead execute B & C:* the tool issues a warning of deviation where two violated constraints are shown to the user for B (as in previous case) and for C only one constraint is shown c1) C should be performed after A. Proposed activities for further execution A, D and E.
3. *Skip B instead execute D:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) B should be performed before D. Proposed activity for further execution B and C.

8.3 Pattern 3: Synchronizaiton

The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled. It is captured in UML through a JoinNode. Multiple control tokens offered on the same incoming edge are combined into one before passing the token to the single subsequent flow (p.394). This is useful when multiple instances are being dealt.

Figure 29: Pattern 3 - Synchronization controlflow pattern

Enactment Analysis This pattern can be implemented in PRODAN for enactment, as shown in figure **??**. Five tasks (A, B, C, D, E) are modeled such that B and C are synchronized to D. Tasks E is modeled to study the post-conditions of D. Task A is not part of the (active) pattern and is just modeled for the sound completion of model. Execution of the process starts through the *initial node* and then A is proposed for the execution. After the completion of A, the next proposed tasks are B and C. After the execution of both B & C, the flow is converged to D and it is proposed for execution.

Deviation Analysis The tool responds to the deviations, depending upon the different type of deviation in the process.

1. *Skip B & C instead execute D:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be executed after B & C. Proposed activities for further execution B, C and E.
2. *Skip B instead execute D:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be

executed after B. Proposed activity for further execution B and E. [Same kind of deviation will apply if C is skipped and D is executed.]

3. *Skip B, C & D instead execute E:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be performed before E. [Constraint for the execution of B & C after A is not displayed]. B and C are still proposed for further execution.

4. *Skip B & D instead execute E:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be performed before E. [Constraint for the execution of B after A is not displayed]. B is proposed for further execution after the execution of E.

5. *Skip D instead execute E:* the tool issues a warning of deviation where a violated constraint is displayed to the user c1) D should be performed before E. After the execution of E, the task D is proposed for further execution.

8.4 Pattern 4: Exclusive Choice

The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a mechanism that can select one of the outgoing branches. In UML a DecisionNode is used to represent this pattern, where it is depicted by a diamond node. It is a control node that choses between the outgoing flows. When a control token is received by the DecisionNode, it presents it to all the outgoing flows. The evaluation of the *guards* determine the selected outgoing flow for the DecisionNode. Only one of the outgoing edges is enabled by the DecisionNode.

Figure 30: Pattern 4 - Exclusive choice controlflow pattern

Enactment Analysis This pattern can be implemented in PRODAN for enactment, as shown in figure **??**. Six tasks (A, B, C, D, E and F) are modeled such that B and C are diverged after A. Tasks D and E are modeled to study the post-conditions of B and C. Task F is not part of the pattern and is just modeled for the sound completion of model. Execution of the process starts through the *initial node* and then A is proposed for the execution. After the execution of A, it is the choice of the user to perform one of the tasks amongst B and C. The tool does not offer any guard. After the execution of B or C, its subsequent activity is proposed for execution.

Deviation Analysis The tool responds to the deviations, depending upon the different type of deviation in the process.

1. *Skip A instead execute B:* the tool issues a warning of deviation where two violated constraints are shown to the user. c1) A must be the initial activity and c2) B should be performed after A. Proposed activities for further execution are A & D. [Same situation arises when C is executed instead of A.]

2. *Skip A instead execute B & C:* the tool issues a warning of deviation where two violated constraints are shown to the user for B (as in

previous case) and for C only one constraint is shown c1) C should be performed after A. Proposed activities for further execution A, D and E.

3. *After the execution of A & B, execute C:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) one in B and C should be executed after A. Proposed activity for further execution are D and E, even though only one of them should be followed according to the original plan. This is the propagation of deviation.

4. *Skip B & C instead execute D:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) B should be performed before D. Constraint for the execution of B or C after A is not displayed to the user. Proposed activities for further execution are B, C and F. A better prediction mechanism can question the execution of C as a proposed task.

8.5 Pattern 5: Simple Merge

The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch. In UML this pattern is implemented through the use of MergeNode. It is depicted by a diamond node with multiple incoming edges and a single outgoing edge. If token is received on any of the incoming branches, it is passed on to the outgoing branch. If token is received on multiple incoming branches it is passed on to the outgoing edge multiple times. In simple merge pattern, we do not take into account the situation where tokens can arrive on multiple incoming edges.

Figure 31: Pattern 5 - Simple merge controlflow pattern

Enactment Analysis: This pattern can be implemented in PRODAN for enactment, as shown in figure **??**. Five tasks (A, B, C, D, E) are modeled such that B and C are converged to D. Tasks E is modeled to study the post-conditions of D. Task A is not part of the (active) pattern and is just modeled for the sound completion of model. Execution of the process starts through the *initial node* and then A is proposed for the execution. After the completion of A, the next proposed tasks are B and C.

Deviation Analysis: The tool responds to the deviations, depending upon the different type of deviation in the process.

1. *Skip B & C instead execute D:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be executed after B & C. Proposed activities for further execution B, C and E.

2. *Skip B, C & D instead execute E:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be performed before E. [Constraint for the execution of B & C after A is not displayed]. B and C are still proposed for further execution, after the execution of E.

3. *Execute B, Skip D instead execute E:* the tool issues a warning of deviation where a violated constraint is shown to the user. c1) D should be performed before E. D is proposed for further execution after the execution of E.