Object Versioning Compliance Analysis Plug-in in ProM 6.0

Zhaoxia Wang\textsuperscript{1,2}, Arthur H.M. ter Hofstede\textsuperscript{3,4⋆}, Chun Ouyang\textsuperscript{3}, Moe T. Wynn\textsuperscript{3}, Jianmin Wang\textsuperscript{1}, and Xiaochen Zhu\textsuperscript{1,2}

\textsuperscript{1} School of Software, Tsinghua University, Beijing, China
\{wang-cx06,zhu-xc10\}@mails.thu.edu.cn, jinwang@thu.edu.cn
\textsuperscript{2} Dep. of Computer Science and Technology, Tsinghua University, Beijing, China
\textsuperscript{3} Queensland University of Technology, Brisbane, Australia
\{a.terhofstede,c.ouyang,m.wynn\}@qut.edu.au
\textsuperscript{4} Eindhoven University of Technology, Eindhoven, The Netherlands

1 Introduction

This document shows how to use the object versioning compliance analysis plug-in (also called the VWF-net analysis plug-in) in ProM 6.0.

This analysis plug-in is dedicated to the verification of versioning compliance between object versioning lifecycles and process models in PLM systems. Contemporary PLM systems typically use workflow technology to provide support for process management. Many common business processes in the manufacturing industry, e.g. in areas such as accounting, engineering design, product release, process planning, and production control, involve the use of object version operations. The use of these operations in the context of tasks is subject to access control restrictions. In addition to that, the order in which these operations may be used may be governed by a lifecycle model wherein it can be specified that certain operations can only be applied when the object is in a certain state. As ordering relations between version operations are also implicitly enforced by the ordering relations between tasks in the workflow model, compliance issues may rear their head. Specifically, on the process side, one can specify for tasks which version operations are permitted during the execution of these tasks and how they may progress the state of objects, while on the access control side the access privileges of users are stored and control is maintained over the order in which object version operations may be applied. The order of tasks may thus contradict the order in which version operations may be applied. Determining whether this is the case is nontrivial, as task ordering relations can be complex.

The object versioning compliance analysis plug-in implements versioning compliance checking of process models in PLM systems. Its main objective is to provide a versioning-annotated workflow net (VWF-net) viewer that possesses both syntactical compatibility checking and behavioural compatibility checking capabilities. A VWF-net includes three parts: a WF-net, an object versioning lifecycle and task versioning annotation information (including the versioning

⋆ Senior Visiting Scholar of Tsinghua University.
assignment information of tasks and the object-state pairs of tasks. A VWF-net is generated using the information from PLM systems regarding a business process, the object versioning lifecycles of the objects used in the process as well as task versioning annotations. This analysis tool has been integrated into ProM 6.0. ProM 6.0 can be downloaded from http://prom.win.tue.nl/research/wiki/prom/downloads.

2 Input Format

Since each PLM system has its own format to describe processes, the object lifecycles and task annotations, a generic XML format for VWF-nets was developed which is platform-independent and which can be used to convert relevant information from different PLM systems into a uniform format (see Fig. 1). The root node of a VWF-net is labelled as VWF-net, representing a workflow net with versioning annotations. Each VWF-net contains a WF-net, an ObjectLifecycle and an arbitrary number of VersioningAnnotation elements as child elements. The WF-net element represents a workflow process modelled as a Petri net with the notions of transitions, places and arcs. The format of a WF-net and its sub-elements are in accordance with the workflow net standard provided in [1]. The ObjectLifecycle element contains an Object element and a Lifecycle element, which describe an object handled in a workflow and the object’s versioning lifecycle respectively. For the purpose of extensibility, we adopted the XML format of Java Finite State Machine framework\(^5\) for the Lifecycle element. We made use of State and StateTransition elements (with attributes name, sourceRef and

\(^5\) http://unimod.sourceforge.net/fsm-framework.html
targetRef) to store the information about an object versioning lifecycle. A VersioningAnnotation element describes the object’s versioning assignment for a task. The StatePairs element captures the pre-states and post-states of objects when tasks are executed. Fig. 2 describes an excerpt of an XML document.

```
<VWF-net ProcessRef="String" ProcessName="String">  
  <ObjectLifecycle>
    <Lifecycle name="lifecycle">  
      <State name="s1" type="INITIAL"/>
      <State name="s2" type="PREMOL"/>
      <State name="s3" type="FINAL"/>
      <Transition from="s1" to="s2" operation="operation"/>
      <Transition from="s2" to="s3" operation="operation"/>
      <Transition from="s3" to="s1" operation="operation"/>
    </Lifecycle>
    <VersioningAnnotation>
      <OperationRef name="operation1"/>
      <ObjectRef name="object1"/>
      <StatePair from="s1" to="s2"/>
      <StatePair from="s2" to="s3"/>
      <StatePair from="s3" to="s1"/>
    </VersioningAnnotation>
  </ObjectLifecycle>
</VWF-net>
```

Fig. 2. An excerpt of the XML document for an VWF-net example.

3 How to Use the Plug-in

3.1 Step 1: syntactical compatibility checking

This step checks whether the VWF-net satisfies certain syntactical requirements. There are six syntax rules for VWF-nets.

1. The empty annotation consistency rule focuses on checking the syntactical consistency for tasks between assigned version operations and object-state pairs. That is, the rule checks two conditions: one to check whether a task has an empty annotation of version operations while it has an object-state pairs annotation or vice versa.
2. The version operation assignment completeness rule considers whether all version operations in the object versioning lifecycle are assigned to tasks in the process model.
3. The local object path existence rule describes that a task, by performing the assigned version operations, is able to move an object from its pre-state to post-state in all the assigned object-state pairs.
4. The no locally assigned dead version operation rule captures whether a task is able to perform, at least once, every assigned version operation.
5. The no dead object state transition rule is concerned with the fact whether any transition in the object versioning lifecycle is possible in the context of a
certain task. It is possible that multiple state transitions in a lifecycle model are labelled by one version operation. The difference between the fifth rule and the fourth rule is that the fifth rule concentrates on judging whether all transitions can potentially happen or not while the fourth rule concentrates on whether all assigned operations can be used at all.

6. The global object path existence rule focuses on whether there exists a path from the initial state to a final state among all assigned object-state pairs in the process model, that is, the tasks altogether should be able to move the object from its initial state to one of its final states.

The first rule represents the preconditions needed for a valid VWF-net. If there are errors related to the first rule, the VWF-net should be corrected first before any subsequent syntax checks are considered.

3.2 Step 2: behavioural compatibility checking

This step is aimed at carrying out behavioural compliance checking between business process models and business object versioning lifecycles. In this step, a VWF-net is first transformed to a WF-net and then the soundness property of this transformed WF-net is checked by the Woflan plug-in [2]. Finally, the results provided by this plug-in are translated to meaningful error messages in terms of the original VWF-net.

1. Step 2.1: Woflan soundness check. A VWF-net is transformed into a WF-net so that behavioural compliance checking can be achieved through application of the Woflan plug-in.

2. Step 2.2: behavioural compatibility check. The behavioural compliance properties of the original VWF-net (e.g., option to complete, no dead tasks and no unused versioning annotations) can be interpreted from the soundness results of the transformed WF-net.

4 Example

This section illustrates the concrete use of the VWF-net analysis through a VWF-net example. This example comes from our paper [3] and the corresponding XML file of this VWF-net can be downloaded from http://www.yawlfoundation.org/research/compliance.

4.1 Importing VWF-net

At first, a VWF-net is imported into ProM 6.0. The import method is similar to other import methods present in ProM 6.0. Subsequently, the VWF-net analysis plug-in is selected as shown in Fig. 3. Once the button start (in the middle part of the bottom of Fig. 3) is pressed, the current interface shifts to the VWF-net viewer interface (as shown in Fig. 4). Fig. 4 shows the imported VWF-net which includes three parts: the process model (Fig. 4 (a)), the versioning lifecycle of business object $o_1$ (Fig. 4 (b)) and task versioning annotation information (Fig. 4 (c)).
4.2 Syntactical compatibility checking

After the button Check Syntactical Compatibility (which is not shown in the screenshots of Fig. 4, but which appears in the middle part of the bottom of
the VWF-net viewer interface) is pressed, syntactical compatibility checking is performed and the result is shown in Fig. 5.

![Fig. 5. The interface of syntactical compatibility checking.](image)

The result reveals that the VWF-net example is not compliant with the syntax requirements. One solution to fix this incompatibility is to remove state transition \((s_1, v_5, s_4)\) from the lifecycle model of object \(o_1\).

### 4.3 Behavioral compatibility checking

After all errors of a VWF-net related to syntax rules are fixed, behavioural compatibility checking of a VWF-net can be started. This consists of two steps: the first step is soundness checking by Woflan (a verification tool developed for workflow nets), while the second step is behavioural compatibility checking.

1. **Woflan soundness check.** Once the button Check Behavioural Compatibility is pressed (which is not shown in the screenshot of Fig. 5, but which appears at the bottom of the interface of the syntactical compatibility result), the transformed WF-net is shown in the upper part of Fig. 6 (a) while the Woflan diagnosis result is illustrated in the lower part of Fig. 6 (a). The Woflan diagnosis result tells us that there are dead tasks for the transformed WF-net: \(X(s_2, T_6)\), \(Y(s_2, T_6)\), \(Z(T_6, o_1, [s_2, v_3, s_1])\) and \(Z(T_6, o_1, [s_1, v_2, s_2])\). And it also tells us that whether the option to complete property holds is unknown.

   It should be noted that Woflan checks soundness of WF-nets in the following order: first the proper completion property is checked, then the no dead tasks property and finally the option to complete property. If all steps succeed, then the net is sound. If one of the steps fails, the net is not sound and the verification process ends. In that case, the results of the subsequent properties will be unknown. Thus, if the net is not sound and the no dead tasks property does not hold, the option to complete property is not checked and whether this property holds or not is not known. Hence, first the issue
of the presence of dead tasks should be solved before one can look into the property of option to complete.

(a) The soundness results of the transformed WF-net

(b) Behavioural compliance checking results of VWF-net

Fig. 6. Behavioural result screenshot for the revised VWF-net shown in Fig. 4 (without state transition \((s_1, v_5, s_4)\) in the lifecycle of \(o_1\)).

2. Behavioural compatibility check. After the button Translate the result (which is not shown in the screenshot of Fig. 6(a), but which appears at the bottom of the Woflan Soundness interface) is pressed, the behavioral compatibility result of the original VWF-net, interpreted from Woflan’s results, is shown in Fig. 6(b).

Based on the Woflan diagnosis result, the behavioural compatibility result of the original VWF-net can be interpreted as follows: There is a dead task \((T_6)\) and there are unused versioning annotations \(((s_1, v_2, s_2), (s_2, v_3, s_1))\) assigned to a task \((T_6)\). The result of the option to complete property is unknown because, as mentioned before, Woflan diagnosis stops when errors
related to the no dead tasks property are encountered in the transformed WF-net.

To correct these errors, a possible solution is to change the control flow relationship between tasks $T_2$ and $T_6$ from an alternative relationship to a sequential relationship. In addition, the versioning assignment of tasks $T_2$ and $T_6$ and the task object-state pairs can also be modified.

After the errors related to the no dead tasks property and the no unused versioning annotations property are fixed, the syntactical compatibility checking and the behavioural compatibility checking can be executed again, and this process of checking and correcting can be repeated till no more errors are found.

The above example is simple and is used to illustrate the analysis plug-in. More complex examples, from practice, can be downloaded from http://www.yawlfoundation.org/research/compliance. For these VWF-nets, domain experts were consulted in relation to the assignment of state-pairs. This sometimes resulted in errors and these were fixed in an incremental manner. For more details about this plug-in and the experiments conducted with it, please consult our paper [3].

References