Achieving Consistency and Completeness of Business Process Models throughout the Lifecycle

Marta S. Tabares¹, Fernando Arango²

¹ Programa de Informática, Escuela de Ingeniería de Antioquia, Envigado, Colombia
pfmstabare@eia.edu.co

² Escuela de Sistemas, Universidad Nacional de Colombia, Medellín, Colombia
farango@unalmed.edu.co

Abstract. Business processes of an organization are sources of information to develop software applications; nevertheless, they change during the development process and finally disagree with the software products that support them, or vice versa. This paper describes a Traceability Pattern that facilitates the verification of both consistency and completeness of business processes throughout the lifecycle. Specifically, we treat with the business process from the UML 2.0 Activity diagram to support the trace with other UML models.

Keywords: Traceability pattern, Model-Driven Development, Business Process Modeling, UML modeling, model transformations, tracing model, tracing link, transformation rules, WorkFlow, MDD.

1 Introduction

How can development teams support the consistency and completeness of business processes with software models throughout the lifecycle? This question is made by the software industry many times because the business processes change during the software development process and finally disagree with the software products that support them, or vice versa. Although this is not new, currently there is a major interest in providing techniques that help solve that issue.

Business process can be described formally by UML 2.x Activity Diagrams (UML 2.0 ADs) which facilitates the modeling of subprocesses, activities, actions, and other elements¹ [1]. Thus, business process models can be traced and transformed to maintain the consistency and completeness with software development models through the lifecycle.

This paper presents a Traceability Pattern that provides a mechanism to control model transformations and support the consistency and completeness of models.

¹ UML 2.0 Activity Diagram is defined according to BPMI techniques (Business Process Management Initiative: http://www.bpmi.org) which is part of the OMG (www.omg.org) and formalized by the Business Process Management Notation v1.0 (2006): http://www.bpmn.org/Documents/.
during the software development process. Specifically, we treat with business processes from the UML 2.0 ADs and its interaction with other UML models.

The structure of this paper is as follows. Section 2 presents a general description of the case study by means of which this approach is explained. Section 3 defines the traceability pattern that supports this approach. Section 4 presents the treatment of the consistency and completeness. Section 5 presents related works. Finally, Section 6 concludes and shows directions for future work.

2 Case Study

The Auction System is an online system that allows people to negotiate the buying and selling of goods in the form of English-style auctions [2]. This system is supported by three main business processes: Buy Goods, Sell Goods, and Credit Management. In order to apply the traceability pattern, we consider the Buy goods process which is represented in a UML 2.0 ADs (see Figure 1). In this, two actors take part: the Buyer which is a customer or user of the system, and the Administrator/System which makes internal functions in the system. This process allows the customer (Buyer) to place bids at different auctions. Procedure: when the buyer desires to buy, he/she must select an active auction. Once a buyer has joined the auction, he/she may make a bid. The Buyer could increase his/her credit in parallel or in the event that this is not sufficient.

![Figure 1. A business process model for the Buy Goods Process.](image-url)
3 The Traceability Pattern

In order to support the consistency\(^2\) and completeness\(^3\) of business processes during the development process, the development team must define which are the model elements or software artifacts involved in the verification. In this approach, the definition is carried out by means of a tracing model where a business process can be defined as a traceable element and verified in these quality attributes.

A Tracing Model is composed of traceable elements and tracing links and it is supported by a metamodel that facilitates the model transformations controlling consistency and completeness, and change propagation [3]. A Tracing Model is a pattern defined in the model-driven development context [4].

A traceable element can be defined as an extension of the Element metaclass in the UML metamodel or in other MOF metamodels [5]. It is identified by a role of tracing that assigns responsibilities to elements in services provided by the pattern. A role uses one of the following stereotypes: <<axisTracing>> (axis of tracing), <<predecessor>>, or <<successor>>. A tracing link is a UML abstraction relationship (<<trace>>, <<realize>>, and <<refine>>) related to one or more transformation rules in charge of transforming source into target traceable elements. Rules can be separated in groups according to different modeling alternatives allowing the development team to take decisions without worrying about the consistency and completeness. Transformation rules might be implemented in a transformation language such as QVT [6], ATL [7], etc.

Instances of tracing models are versioning views of development models generated by the tracingModel-Engine (defined at the metamodel level [3]) during the transformation process for a particular software project.

4 Consistency and Completeness

When the development team is modeling the business process by means of the UML 2.0 ADs must identify and understand the use of each business model element so as to define an appropriate tracing model. For example, a function of a process (represented by an action node) could be traced to several requirements or basic flow steps in a use case, an object action could be traced to classes in the domain model, and a subprocess (represented by a CallBehaviorAction) could be traced to a base use case or others such as included/extend use cases, etc. Therefore, the operationalizable functions of the business process in the requirement level can be traced to activities diagrams or other model elements in the solution space.

Knowing these traces, the development team can define or use a tracing model based on cases of uses (see Figure 2) as pattern of traceability, whose axes of tracing are UseCase, Activity, Class, and InteractionFragment metaclasses. Here, the UseCase

\(^2\) Consistency refers to giving support to the meaning of requirements throughout models that represent them.

\(^3\) Completeness refers to giving support to the knowledge about which elements from the source models (in the problem) will have to be included in the target models (in the solution).
leads the transformation models with verification responsibilities. Both predecessors and successors have metaclasses such as Requirement, and Collaboration respectively leading the trace with the UseCase.

Figure 2. Tracing Model Based on Use Cases.

The consistency is guaranteed by means of the <<trace>> tracing links (see Table 1), and the completeness is guaranteed by means of <<realize>>/<<refine>> tracing links (see Table 2); both can be verified by means of trace sequences. These tracing links provide the information needs to carry out the verification when a change is made.

Table 1. Some <<trace>> Tracing Link to guarantee the Consistency.

<table>
<thead>
<tr>
<th>Name</th>
<th>sourceTracingLink</th>
<th>targetTracingLink</th>
<th>TransformationRule</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;trace&gt;&gt;</td>
<td>&lt;&lt;axisTracing&gt;&gt; Activity</td>
<td>&lt;&lt;axisTracing&gt;&gt; UseCase</td>
<td>UseCase2Activity</td>
</tr>
<tr>
<td>&lt;&lt;trace&gt;&gt;</td>
<td>&lt;&lt;axisTracing&gt;&gt; Class</td>
<td>&lt;&lt;axisTracing&gt;&gt; UseCase</td>
<td>UseCase2Class</td>
</tr>
<tr>
<td>&lt;&lt;trace&gt;&gt;</td>
<td>&lt;&lt;axisTracing&gt;&gt; InteractionFragment</td>
<td>&lt;&lt;axisTracing&gt;&gt; Class</td>
<td>Class2InteractionFragment</td>
</tr>
<tr>
<td>&lt;&lt;trace&gt;&gt;</td>
<td>&lt;&lt;predecessor&gt;&gt; Requirement</td>
<td>&lt;&lt;predecessor&gt;&gt; BusinessProcess</td>
<td>BusinessProcess2Requirement</td>
</tr>
</tbody>
</table>

Table 2. <<realize>>/<<refine>> Tracing Link to guarantee the Completeness.

<table>
<thead>
<tr>
<th>Name</th>
<th>sourceTracingLink</th>
<th>targetTracingLink</th>
<th>TransformationRule</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;realize&gt;&gt;</td>
<td>&lt;&lt;axisTracing&gt;&gt; UseCase</td>
<td>&lt;&lt;predecessor&gt;&gt; Requirement</td>
<td>Requirement2UseCase</td>
</tr>
<tr>
<td>&lt;&lt;realize&gt;&gt;</td>
<td>&lt;&lt;successor&gt;&gt; Collaboration</td>
<td>&lt;&lt;axisTracing&gt;&gt; UseCase</td>
<td>UseCase2Collaboration</td>
</tr>
</tbody>
</table>

Figure 3 shows an instance of the tracing model based on use cases for the AuctionSystem v.1.0.0 from the Buy Goods process. This shows the main model elements (in the model level) involved in the trace and tracing links that control the completeness and consistency defined by the pattern.
Verification of consistency and completeness is made by means of trace sequences guaranteeing transitivity among different traceable elements, and facilitating the chaining of transformation rules. Figure 4 shows an example of two sequences: (1) the Activity traces the UseCase and this realizes the Business Process, then, the Activity realizes the Business Process; (2) this sequence uses the association law to achieve a direct trace between Class and Activity.

\[
\begin{align*}
\text{Activity}::\text{BuyGoods} \leftrightarrow \text{UseCase}::\text{BuyGoods} & \Rightarrow \text{Business Process}::\text{BuyGoods} \\
\text{(Class}::\text{BuyGoods} \leftrightarrow \text{Business Process}) \land \\
\text{(Activity}::\text{BuyGoods} \leftrightarrow \text{UseCase}::\text{BuyGoods}) & \Rightarrow \text{Class}::\text{BuyGoods} \leftrightarrow \text{Activity}::\text{BuyGoods}
\end{align*}
\]

Figure 4. Trace Sequences to verify the Consistency and Completeness.

5 Related Works

In the last years, Model-Driven Development approaches propose models and metamodels of traceability as mechanisms to register model transformations, solve issues of interoperability during the transformations, provide matrices between requirements and models, and define tracing links that support transformations, etc [8, 9]. Furthermore, other approaches provide model traceability guaranteeing the consistency and completeness from transformation models [10, 11]. Sousa et al., in a recent model-driven approach, provides a mechanism to make “the traceability from the business process until the user interface to help business analysts in predicting the impact of process changes on the user interaction and propose changes in the processes when the user interaction is improved” [12].
6 Conclusions and Future Work

The work presented in this paper applies directly to the traceability practice and the model-driven development. This approach provides a mechanism to support the consistency and completeness of development models through a Traceability Pattern that controls the model transformations from the business process definition to the design or implementation stages.

A short term objective is the implantation of the Traceability Pattern in a software company. Right now, the research team is implementing the pattern for modeling tools, by means of a UML profile that can be used to verify the completeness and consistency of development models and its transformation in different abstraction levels. Besides, the transformations are supported by XMI files that allow the verification of these quality attributes according to tracing models created by the development team.

References