ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220868324

Activity Diagram Patterns for Modeling Quality Constraints in Business Processes

Conference Paper *in* Lecture Notes in Computer Science · October 2005

DOI: 10.1007/11557432_2 · Source: DBLP

CITATIONS
38

READS
1,072

3 authors, including:

Gregor Engels
Universität Paderborn
417 PUBLICATIONS **6,602** CITATIONS
SEE PROFILE

Tim Schattkowsky
Hochschule Hamm-Lippstadt
40 PUBLICATIONS **432** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Linking Services to Websites by Leveraging Semantic Data View project

Project Model-Driven Engineering of Adaptive UIs View project

All content following this page was uploaded by Tim Schattkowsky on 05 June 2014.

The user has requested enhancement of the downloaded file.

Activity Diagram Patterns for Modeling Quality Constraints in Business Processes

Alexander Foerster, Gregor Engels, Tim Schattkowsky

University of Paderborn, Germany
{alfo|engels|timschat}@uni-paderborn.de

Abstract. Quality management is an important aspect of business processes. Organizations must implement quality requirements, e.g., according to standards like ISO 9001. Existing approaches on business process modeling provide no explicit means to enforce such requirements. UML Activity Diagrams are a well recognized way of representing those business processes. In this paper, we present an approach for enforcing quality requirements in such business processes through the application of process quality patterns to Activity Diagrams. These patterns are defined using a pattern description language, being a light-weight extension of UML Activity Diagrams. Accordingly, such patterns can be used in forward-engineering of business processes that incorporate quality constraints right from the beginning.

Keywords: UML Activity Diagrams, Business Process, Process Quality, ISO 9001

1. Introduction

Total Quality Management (TQM) is a management concept which is an increasingly hot issue in most organizations. Businesses have become more and more competitive and the concentration on customer satisfaction is a key trait. Due to the increasing competitiveness of the markets, the customers formulate high expectations on products and services like user friendliness, reliability, security etc. The implementation of a TQM system in an organization is one answer to this situation since it puts the customer's demands in the first place and incorporates profound means of customer satisfaction.

The ISO 9001 standard [11] is one of the most popular TQM systems world-wide. It is in itself very process oriented and includes many quality related demands on business processes. In an organization with TQM, modeling business processes means that the process developer has to consider the quality requirements of the TQM system. We will present an approach how such quality requirements can be easily and thoroughly formulated and enforced. The approach is based on deriving business process patterns from the text of the ISO 9001 standard and applying these patterns to existing business processes.

A pattern based approach has many advantages. Using patterns is a common technology in software engineering. With design patterns, proven concepts and solutions can be easily described and communicated. Their application can significantly improve the quality of software models. Complex models can be easier

understood and handled if the included patterns are known. These are only some advantages of the general concept of design patterns that we want to transfer to the world of business processes.

In our approach, we make use of UML Activity Diagrams as modeling language of choice for business processes. Therefore, the formulation of quality patterns will also be based on UML Activity Diagrams as modeling paradigm. Since they were not intentionally designed especially for the formulation of quality patterns, some aspects of the quality patterns cannot be satisfactorily modeled by them, as we will show further below. As a solution, we developed a pattern description language that is a light-weight extension of UML Activity Diagrams.

Fig. 1 shows an abstract model of a business process. If we assume that this process was created by the application of a quality management pattern to an original core business process, the resulting process contains Actions that have a different origin or belong to different aspects of the business process. Some Actions belong to the original core business process whereas others may fulfill only technical purposes or are part of the quality management system. These Actions are weaved together and can interleave each other in the final business process as shown by the different textures in Fig. 1. This means that the Actions and control flows of the original business process and the pattern have to be mixed together. Enforcing quality requirements with quality management patterns requires defined rules on how exactly a pattern must be applied to an existing business process. In addition to that, the pattern has to be a most exact formulation of the quality requirements. The formulation of quality patterns and the pattern application process is unfortunately neither easy nor straight-forward; we will provide an impression of that in the next example.

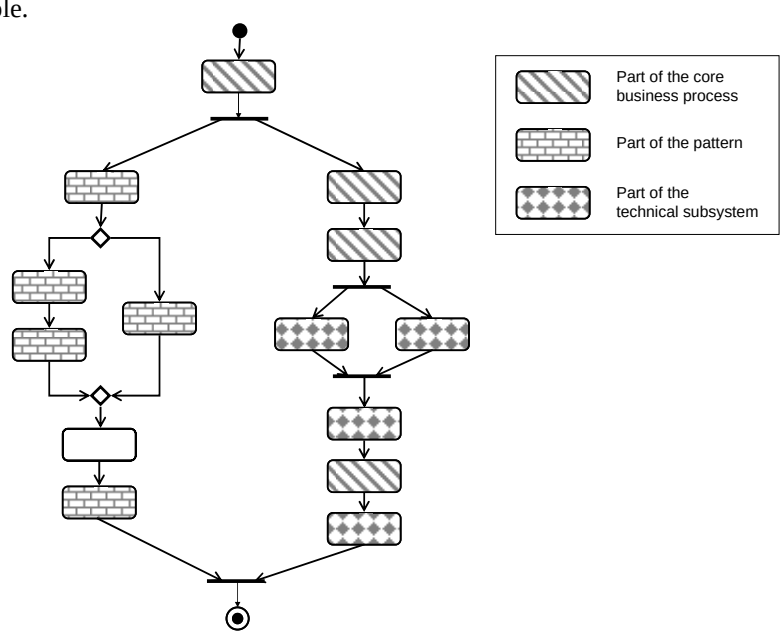

Fig. 1. Example behavior model including Actions that belong to different aspects of a business process

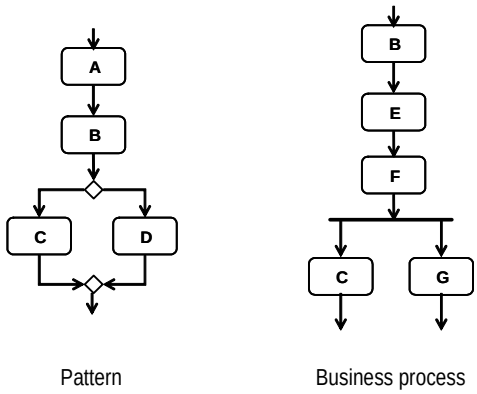

Fig. 2. Simple example of a pattern and a business process

Fig. 2 shows two simple workflows, the left shall be regarded as the pattern; the right shall be regarded as an excerpt of an existing business process in an organization. Let us assume the process modeler wants to apply the pattern on the left to the process on the right. The question is how exactly that can be done. Obviously, there are some distinct connection points which are the similar Actions in both processes, namely "B" and "C". Assuming that pattern application means to build one single united process and therefore one single united control flow from both Activities, we have to determine if the "B"s and "C"s of both processes have to be mapped to one single Action "B" and "C" in the resulting process. Another conceivable possibility would be that they represent the same *kind of* Action but different Action executions. If we assume "B" and "C" are the same Action executions in both processes, we make the following observation: in the pattern process, "C" follows "B" directly, apart from the decision node. In the business process, there are two other Actions in the control flow between "B" and "C". In an effort to create a joint process out of the pattern and the existing business process, how does the modeler know if inserting other Actions in a control flow is allowed or forbidden by the intention of the pattern or the original process? To solve these problems, there has to be additional information laid down in the model of the pattern. This can be achieved by the light-weight Activity Diagram extension that we are going to present in this paper.

The next section gives a brief overview of the related work in this area. Section 3 starts with examining the quality requirements that we want to be able to describe. We develop general requirements on the application of quality management patterns. After that, we transfer these general requirements to pattern application rules related to the particular model elements of quality process patterns. In that context, we define extensions to UML Activity Diagrams based on stereotypes that form the language to describe quality process patterns. In section 4 we show how the use of this language improves the way how a quality pattern can be described and applied to a concrete business process, before section 5 closes with a conclusion and future work.

2. Related Work

The pattern concept in computer science is well known when it comes to design patterns as presented in [9] and [10]. These design patterns are intended to describe good practice and proven solutions for common problems in object-oriented software design. These behaviors take place in a different context and are situated on a different level of abstraction than the business processes and patterns here.

In recent years, investigation of the pattern approach for application to the design of processes has begun. Especially workflows as automated processes controlled by computerized workflow management systems have been in the focus of interest for process patterns. Van der Aalst et al. provide an extensive coverage of workflow patterns in [1]. Their intention is mainly to demonstrate the expressiveness and capabilities of existing workflow management systems and their process description languages. Unlike our approach, their process patterns cover mainly technical concepts like all kinds of different basic and complex control flows and they are focused on Petri-nets.

In other works, Basten and van der Aalst aim at providing a theoretical foundation for the definition of inheritance of dynamic behavior of objects based on object life cycles [5]. There, four different inheritance rules based on hiding and blocking of transitions in transition systems are defined. The paper imparts a theoretical background for a possible way of defining the semantics of behavior inheritance. It is focused on behavior in form of object life cycles and studies inheritance in a process-algebraic setting and in a Petri-net framework. In [2], the theoretical framework of [5] is transferred to UML behavior models like sequence diagrams, state charts and activity diagrams. As in [5], this paper also focuses on modeling object life cycles and omits the particularities of business processes. However, the application of these inheritance concepts in the context of actual scenarios, e.g. to model patterns or quality constraints, is not clarified.

Some approaches consider the application of process patterns to software development processes. In [6], Coplien defines a pattern language for this kind of processes. Riehle gives the reader some advice on how to use such software process patterns in [13]. Finally, Ambler presents a broad collection of useful patterns for software design in [3] and [4]. However, these cover only specific aspects and usually omit the discussion of the actual application to existing processes.

An interesting discussion about analysis patterns and business objects can be found in [8] which claims that in the first category are patterns with a suggestive character that just give the idea but the designer is free to tweak the pattern any way he wants when he applies the pattern. In contrast to that, business objects are considered to have a prescriptive character, so they cannot be easily altered by the developer. Possible changes are based on what the writer of the business objects allows the designer to alter. This notion of a pattern is much closer to our notion of quality patterns since we want to be able to enforce properties of the resulting process instead of suggesting them.

3. Describing Quality Constraints with Process Patterns

Our approach is based on a visual language for describing quality process patterns based on UML Activity Diagrams. Such Activity Diagrams are well suited to express for example detailed process models that are ready for execution. In contrast to that, patterns describe processes that contain merely action roles and incomplete or loose temporal/logical relationships between Actions and high-level constraints. These are going to be substituted by concrete process elements from the application domain when the pattern is applied.

When modeling patterns, we want to follow the *principle of least constraint*; this means to describe everything that is elementary for the pattern without risking to lay down properties of the resulting process that are not elementary parts of the pattern. This principle of least constraint is a necessary prerequisite to enable the highest possible flexibility when applying the pattern.

In this section, we will show how quality requirements can be directly derived from standards like ISO 9001 and used as a basis for the definition of quality patterns using our pattern description language.

3.1 Deriving Patterns from Quality Requirements of the ISO 9001

A central part of a TQM system is quality control. This means that the result of a production process has to be compared to predefined quality objectives. The textual formulation of the requirements of quality control in the ISO 9001 standard is composed of different parts of the standard's text:

> "The organization shall monitor and measure the characteristics of the product to verify that product requirements have been met." [11]

> "7.1 Planning of product realization […]
> In planning product realization, the organization shall determine the following, as appropriate:
>
> a) quality objectives and requirements for the product;
> b) the need to establish processes, documents, and provide resources specific to the product;
> c) required verification, validation, monitoring, inspection and test activities specific to the product and the criteria for product acceptance;
> d) records needed to provide evidence that the realization processes and resulting product meet requirements (see 4.2.4)." [11]

> "The organization shall ensure that product which does not conform to product requirements is identified and controlled to prevent its unintended use or delivery. […] The organization shall deal with nonconforming product by one or more of the following ways
> a) by taking action to eliminate the detected nonconformity;
> […]" [11]

This description of quality requirements is still rather informal. At first we will state more precisely in natural language what steps have to be performed to fulfill the requirements of the given problem:

- Quality tests have to be performed in the processes in which products for external customers are produced.
- Quality objectives have to be defined before the production process starts.
- The quality objectives have to be communicated to the persons conducting the production process before the production process starts. *(This requirement results partly from other parts of the ISO 9001)*
- The quality of the product has to be measured and compared to the quality objectives.
- In case that the quality objectives are not met the detected nonconformities have to be eliminated. *(This requirement results partly from other parts of the ISO 9001)*
- Quality related information has to be collected and sent to the quality management department for statistical analysis, systemic improvements etc. *(This requirement results partly from other parts of the ISO 9001)*

In our approach, such requirements are the basis for the definition of quality patterns for processes to enforce these requirements. The application of such a pattern to an existing business process means merging two control flow structures into one, in other words weaving them together, leaving both processes "intact".

It must be ensured that the original behavior is preserved. All Actions of the pattern as well as all Actions of the business process have to be preserved when applying the pattern, although they might coincide in the resulting process. Furthermore, the partial order defined for Actions by both the pattern and the original business process has to be preserved when applying the pattern. However, the actual interleaving of Actions from the pattern and the original business process may vary.

3.2 Pattern Application and Activity Diagram Extensions

In order to explain the application of the model elements of quality patterns to an existing business process, we will see many diagrams in this section in which we have to distinguish between the pattern Activity Diagram and the model Activity Diagram, which contains the applied pattern. We will depict this in the following figures by adding the labels "Pattern:" and "Model:" to the Activity Diagrams.

Applying a pattern means that there exists a mapping between the pattern model elements and the model elements of the resulting process in which the pattern can be found again. We will depict this mapping using arrows with solid arrowheads. Fig. 3 shows an example of such a diagram in which Actions "A" and "B" have been specialized by Actions "A1" and "B2" and the relative order relationship between "A" and "B" (which we will explain further below) has become an ActivityEdge at the model level after applying the pattern.

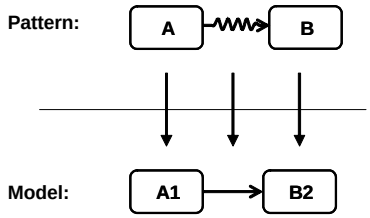

Fig. 3. Mapping between Pattern Level and Model Level

An Action in an Activity Diagram is usually notated as a rounded rectangle containing a label. Since "Action" is an abstract class in the UML 2.0 metamodel, we assume that Actions in a business process normally refer to "CallBehaviorActions". The Behavior itself is not further specified. In a business process, an Action is often something like "Send Invoice" or "Assemble Product", which refers to a Behavior that is complex but does not need further explanation in the context of the actual business process. So the text that is written into the rectangle symbolizing an Action is in fact the *name* of a Behavior.

When a pattern is applied, the pattern Actions are mapped to Actions of the resulting model where the pattern can be found again expressing that the two Actions refer to the same Behavior. There is a problem since the pattern and the model Activity Diagram have most likely been devised by different persons with different "Behavior" namespaces. To make a mapping possible, the process developer is responsible for determining similar Behaviors from both namespaces, so without loss of generality we assume that the "Behavior" namespaces are already synchronized.

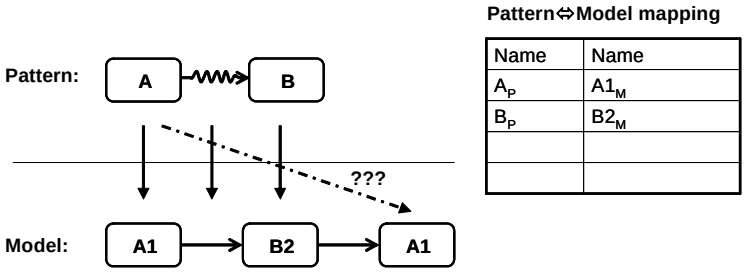

Fig. 4. Wrong pattern/model mapping

Actions referring to the same Behavior can occur multiple times in the same Activity Diagram. For example one Action at the pattern level is mapped to one Action at the model level which occurs multiple times but we want to express that only one of them is referred to in the pattern. This means that the mapping between the pattern Actions and the model Actions as in Fig. 4 can not simply be done by creating a relation

(Pattern Action name) ⇔ (Model Action name)

but the mapping has to be done based on individual Actions.

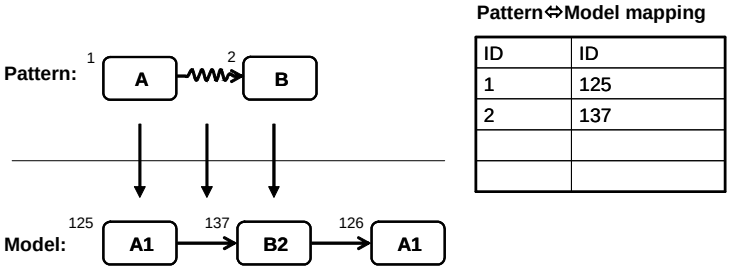

Fig. 5. Correct pattern/model mapping

As solution, the individual occurrences of the Actions have to be considered. The mapping can for example be defined upon individual Action IDs. In Fig. 5 the Action IDs are depicted as small numbers close to the upper left corner of the Action symbol. In our example these IDs are numeric, but other representations are also possible as long as they are unique. Now the correct mapping between pattern and model Actions has to be made using the IDs as shown in Fig. 5.

Up to now, we have mapped Actions that refer to the exact same Behavior. Since patterns usually describe more generic processes, the pattern Actions and the Behaviors they refer to are also usually more generic than those at the model level. Therefore, pattern Actions can, when the pattern is applied to a concrete business process, be replaced by model Actions referring to a specialized Behavior or sub-processes. Activity diagrams at the model level describe rather fixed temporal and logical relationships between Actions. In pattern Activity Diagrams the designer of the pattern wants to have the possibility to describe flexible relationships between Actions in some situations and strict relationships between Actions in other situations.

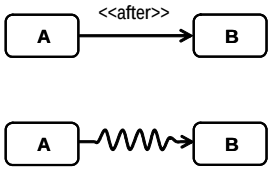

Fig. 6. Presentation option for the <<after>> stereotype

In the last subsection we have pointed out that pattern workflows have to be adapted to the actual business process when they are applied to it; especially sometimes Actions have to be inserted into the control flow of each original Activity Diagram. In other situations the designer might also want to express that Actions have to directly follow each other. A normal ActivityEdge in a pattern Activity Diagram means that Actions are tightly connected and there may no other Actions or control flows be inserted between them. For the flexible alternative, we introduce a stereotype of an ActivityEdge called <<after>>. This stereotype expresses a kind of temporal/logical order relationship or in other words a control flow path, which means that there may be other Actions in between. We also suggest a visualization option as shown in Fig. 6. A visual interpretation of the semantics of the <<after>> stereotype can be found in Fig. 7.

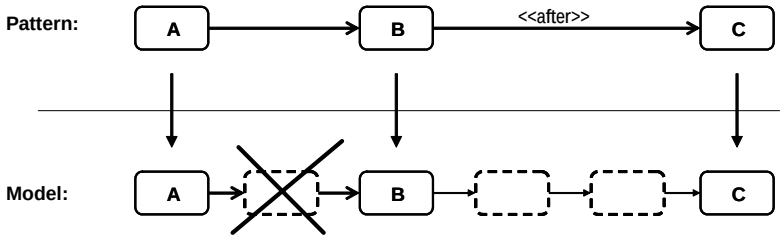

Fig. 7. Visualization of the semantics of the <<after>> stereotype

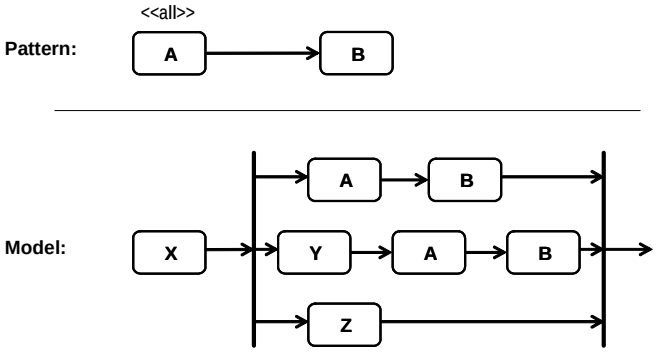

Fig. 8. Referring to all Actions referencing the same Behavior

Fig. 9. Presentation option for the <<all>> stereotype

As we have already seen, Actions referencing the same Behavior can occur multiple times in a process. If all occurrences of an Action referencing the same Behavior shall be referred to in the pattern at the same time, this can be expressed using the stereotype <<all>> as in Fig. 8 or writing a multi-action as in Fig. 9. For example, Action "A" of the pattern in Fig. 8 refers to all Actions "A" at the model level as far as the application of the pattern goes, that way all execution instances of "A" in the model have to be directly followed by Action "B". So the model instance shown in Fig. 8 is a correct application of the (very academic and simplified) "pattern".

Now we can take a look on some special cases that occur in connection with control nodes, i.e. parallel split/parallel join and decision/merge. Fig. 10 shows abstractly how the sequence "A"→"B" (with <<after>> stereotype) could be applied to become a parallel join (Model 3) or a parallel split (Model 4). Model 3 is a correct specialization of the pattern since "B" still has to be executed after "A". The fact that the execution has to wait between "A" and "B" until the synchronization with the other control flow takes place makes no difference for the order relationship between "A" and "B". Model 4 is also a correct specialization of the pattern since the fact that

another control flow is forked between “A” and “B” also makes no difference to the fact that “B” is executed after “A”.

Fig. 11 shows how decision and merge control nodes are treated in the pattern application process. The problem with the pattern application in Model 5 is that we cannot guarantee that “B” is executed after “A” or even at all. But since “B” is definitely executed in the pattern after “A”, we regard this as a wrong application. The same is true for Model 6 where we cannot guarantee that Action “A” is executed before “B” or at all. So neither Model 5 nor Model 6 is a correct application of the pattern. Thus, if a conditional control flow is desired in the resulting process model after the pattern application, there has to be a conditional construction (split/merge/condition) in the pattern process, too.

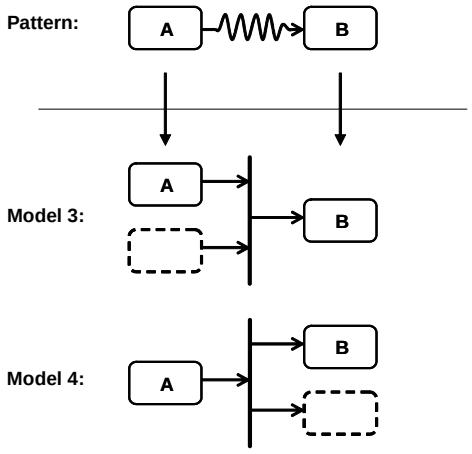

Fig. 10. Pattern application and parallel control flows

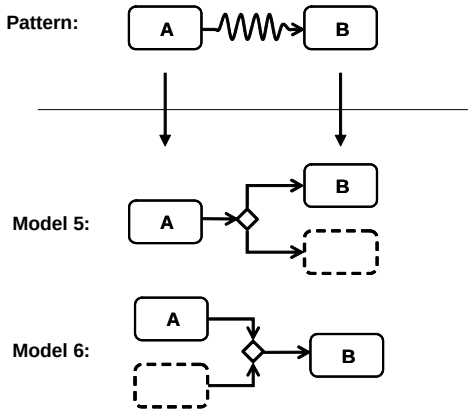

Fig. 11. Pattern application and conditional constructs

Parallel constructions in the pattern should generally be sustained when applying the pattern. Parallelism in pattern processes can however be just an expression of the fact that the order in which Actions are executed is irrelevant. According to the UML 2.0 specification [11], no real concurrency is enforced. This means for the application of patterns containing parallel control flows that they can potentially be serialized when applying the pattern, if needed. Fig. 12 shows an example; both Model 7 and Model 8 are valid instances of the pattern. Parallel control flows can also be partly serialized as shown in Fig. 13. Note that this may become problematic if parallel Actions have side effects that mutually affect each other.

This concludes the analysis of the modeling elements of quality pattern Activity Diagrams. In the next section we will apply our findings to a concrete example from the ISO 9001.

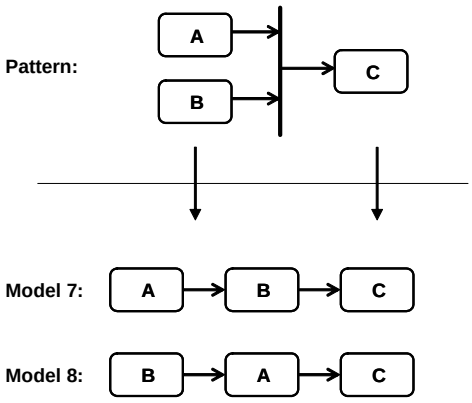

Fig. 12. Parallel constructs in the pattern being serialized

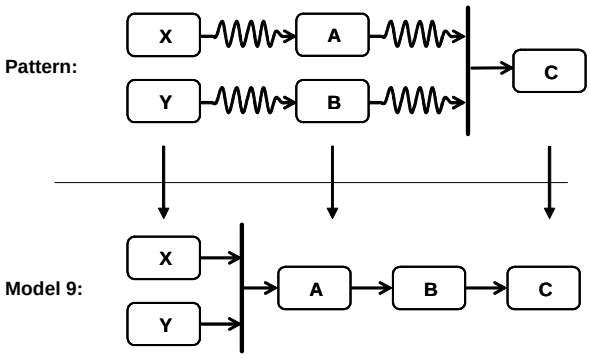

Fig. 13. Parallel constructs in the pattern partly serialized

4. Example

In the last section we made rather theoretical observations about process patterns and their application. In section 3.1 we have already presented some excerpts from the ISO 9001 standard's text. We can now make use of the new stereotypes to reflect the quality requirements stated in the standard's text.

From the ISO excerpts we can derive the pattern "Quality control" which is shown in Fig. 14. As the next step, we are going to apply this pattern to a concrete business process which is in our case a production process. We chose a heavily simplified model of an "Incremental software development" process as example of a business process, as shown in Fig. 15.

Without the concept of path-like ActivityEdges with the <<after>> stereotype, only one very simple way of applying the pattern to the business process would be possible: the "Incremental software development process" as a whole could be viewed as a specialization of the Action "Execute Production Process" of the pattern. That would mean that the whole existing business process would become a sub-activity of "Execute Production Process". This is certainly a possible application of the pattern, but it is questionable if this was intended by the ISO 9001 standard.

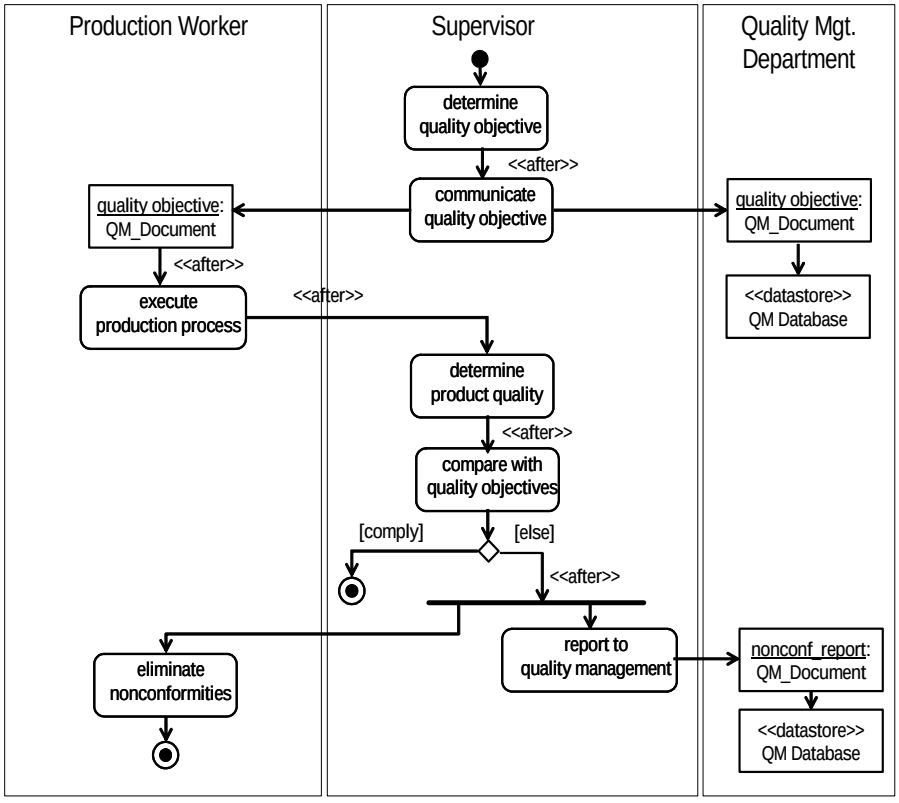

Fig. 14. Example pattern "Quality Control"

With the new stereotypes included at many places in the processes of Fig. 14, there open up completely new possibilities of pattern application. A comparision with the actual process example in Fig. 15 shows a number of similarities between the concrete business process and the pattern. Some Actions of the business process can be seen as specialization of actions of the pattern and the control structures have distinct similarities, too.

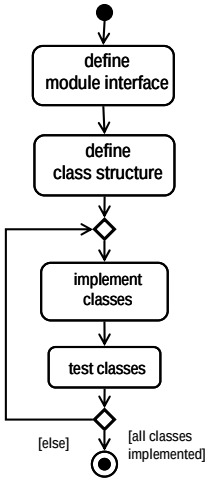

Fig. 15: Concrete business process (simplified incremental software development)

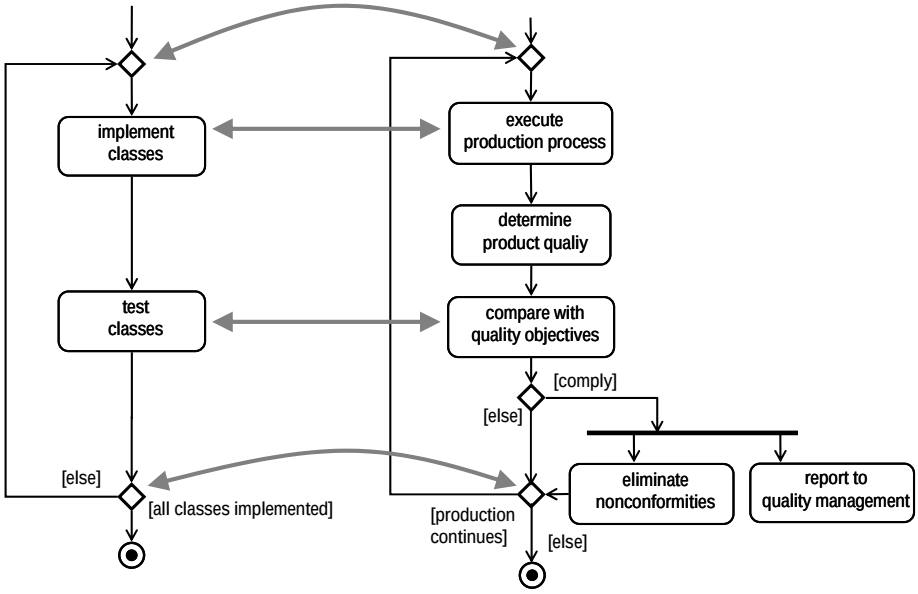

Fig. 16: Similar control structures in business process (left) and pattern (right)

In both processes,

- The actual “production” takes place in a loop.
- The Action “Implement classes” is main “production” task and could therefore be viewed as a specialization of the Action “Execute production process” of the pattern.
- The Action “Test classes” can be seen as a specialization of “Determine product quality” of the pattern.

Fig. 16 depicts these similarities between the pattern process and the original business process graphically. Now, having the stereotyped ActivityEdges and the possibility to add Actions into the control flows of both processes, we can weave both processes much more tightly together as it would be possible without these concepts as shown in Fig. 17.

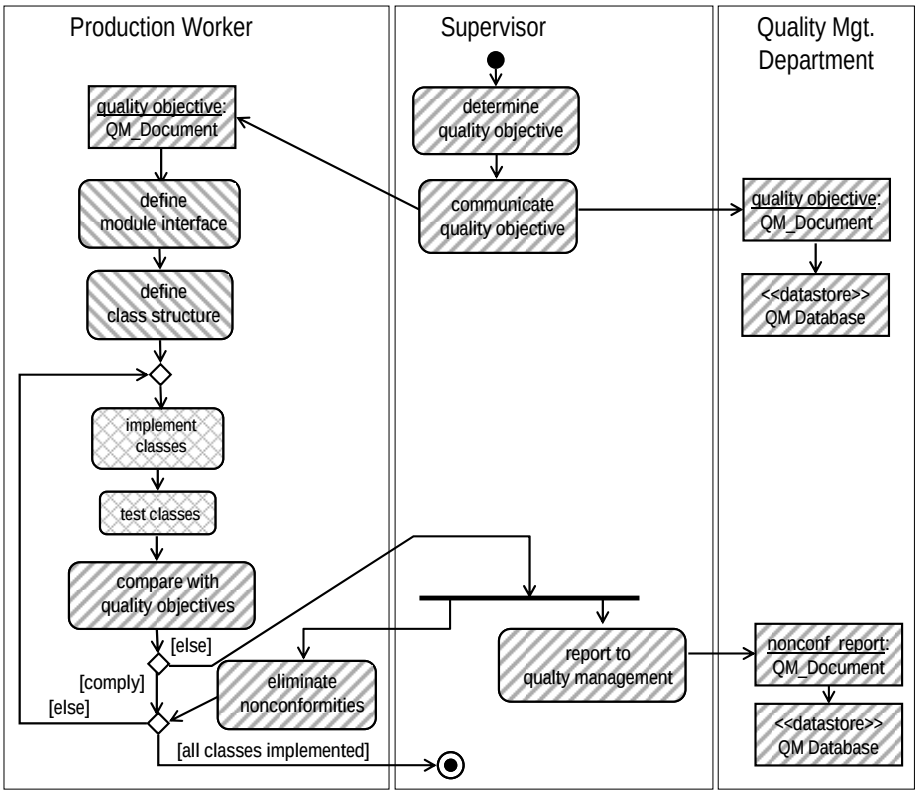

Fig. 17: More sophisticated application of the pattern

5. Conclusion and Future Work

Standards like ISO 9001 can be used as a source for deriving quality patterns. However, we have shown that Activity Diagrams cannot be applied directly to capture such patterns. Thus, we have introduced light-weight extensions captured by stereotypes to overcome the shortcomings on expressiveness for this particular application. We elucidated rules and properties that have to be taken care of when applying a quality management process pattern in general. Finally, we have formulated an example that demonstrates the new possibilities that arise from the application of these extensions.

Further research will include the application of our approach to other TQM systems and an evaluation in a real word process context. Furthermore, in this paper we have seen the concept of quality patterns rather in a forward-engineering perspective. Forthcoming works will also be more focused on quality pattern matching and recognition in existing business processes. Thus, we finally aim at providing a complete formal notion for the application of quality constraints to business processes as we did in previous work [7] for the refinement of state machines.

6. References

[1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros: Workflow Patterns. Distributed and Parallel Databases. 14(3), pages 5-51, July 2003.

[2] W.M.P. van der Aalst: Inheritance of Dynamic Behavor in UML. In D. Moldt, editor, Proceedings of the Second Workshop on Modelling of Objects, Components and Agents (MOCA 2002), volume 561 of DAIMI, pages 105-120, Aarhus, Denmark, August 2002.

[3] S. W. Ambler: Process Patterns - Building Large-Scale Systems Using Object Technology. SIGS Books/Cambridge University Press, Cambridge 1998.

[4] S. W. Ambler: More Process Patterns - Delivering Large-Scale Systems Using Object Technology. SIGS Books/Cambridge University Press, Cambridge 1999.

[5] T. Basten, W.M.P. van der Aalst: Inheritance of Behavior. Journal of Logic and Algebraic Programming, 47(2):47-145, 2001.

[6] J. Coplien: A Generative Development-Process Pattern Language. In Coplien & Schmidt 1995, pp. 183-238, 1995.

[7] J. Ebert, G. Engels: Specialization of Object Life Cycle Definitions. Fachberichte Informatik Nr. 19/95, Universität Koblenz-Landau, 1997.

[8] M. Fowler: Analysis Patterns. Addison Wesley, Menlo Park, California, 1997.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, 1995.

[10] D. Gross and E.S.K. Yu: From Non-Functional Requirements to Design through Patterns. Requirements Engineering, 6(1):18-36, 2001.

[11] ISO 9001:2001: Quality Management Systems – Requirements. ISO International Organization for Standardization. 2001.

[12] Object Management Group, The: UML 2.0 Superstructure, 2005. Version 2.0. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02. Last visited: 03-23-05.

[13] D. Riehle, H. Zullighoven: Understanding and Using Patterns. Software Development. Theory and Practice of Object Systems. 2(1):3-13, 1996.

View publication stats