Connecting the Rationale for Changes to the Evolution of a Process

Alexis Ocampo and Martín Soto

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663, Kaiserslautern and Germany
{ocampo,soto}@iese.fraunhofer.de

Abstract. In dynamic and constantly changing business environments, the need to rapidly modify and extend the software process arises as an important issue. Reasons include redistribution of tasks, technology changes, or required adherence to new standards. Changing processes ad-hoc without considering the underlying rationale of the process design can lead to various risks. Therefore, software organizations need suitable techniques and tools for storing and visualizing the rationale behind process model design decisions in order to optimally introduce future changes into their processes. We have developed a technique that support us in systematically identifying the differences between versions of a process model, and in connecting the rationale that motivated such differences. This results in a comprehensive process evolution repository that can be used, for instance, to support process compliance management, to learn from process evolution, or to identify and understand process variations in different development environments. In this article, we explain the underlying concepts of the technique, describe a supporting tool, and discuss our initial validation in the context of the German V-Modell XT process standard. We close the paper with related work and directions for future research.

1 Introduction

The field of software process modeling has become established within the software engineering community. An explicit process model is a key requirement for high productivity and software quality. The process description content might be collected in several ways, for example by observing real projects, describing intended activities, studying the literature and industry reports, or interviewing people involved in a project [10]. Usually, considerable effort is invested into the definition of such processes for an organization. Once the process is defined and institutionalized, modifying it further becomes unavoidable due to various reasons, such as the introduction of a new software development technology in a development team (e.g., new testing support tools and techniques), a new/updated process engineering technology (e.g., a new process modeling technique), new/updated standards/guidelines for software development or process engineering, new/updated regulatory constraints, or new/updated standards/guidelines for software development or process engineering. Such changes must be reflected accordingly in the corresponding process models. Achiev-

J. Münch and P. Abrahamsson (Eds.): PROFES 2007, LNCS 4589, pp. 160–174, 2007.

ing a compromise that satisfies such a challenge usually depends on the information available for rapidly judging if a change is consistent and can be easily adopted by practitioners.

Having information about the reasons for process changes (i.e., the *rationale*) at hand can be of great help to process engineers for facing the previously mentioned challenges. Currently, the common situation is that there is a lack of support for systematically evolving process models. Combined with other facts such as budget and time pressure, process engineers often take shortcuts and therefore introduce unsuitable or inconsistent changes or go through a long, painful update process. In many cases, precipitous and arbitrary decisions are taken, and process models are changed without storing or keeping track of the rationale behind such changes.

According to our experience, systematically describing the relationships between an existing process and its previous version(s) is very helpful for efficient software process model evolution [2]. Such relationships should denote differences between versions due to distinguishable modifications. One can identify the purpose of such modifications if one can understand the rationale behind them. Rationale is defined as the justification of decisions [8]. Rationale models represent the reasoning that led to the system or, in our case, to the process in their current form. Historically, much research about rationale has focused on software/product design. By making rationale information explicit, decision elements such as issues, alternatives, criteria, and arguments can improve the quality of software development decisions. Additionally, once new functionality is added to a system, the rationale models enable developers to track those decisions that should be revisited and those alternatives that have already been evaluated.

We are currently working on transferring rationale concepts into the process modeling domain. We do this based on the assumption that the rationale for process changes can be used for understanding the history of such changes, for comprehensive learning, and for supporting the systematic evolution of software processes. We are looking at the possibilities that can be used for documenting changes and connecting them to their corresponding rationale.

This article presents a technique for comparing process models, recognizing a set of standard changes, and connecting them to their respective rationale as follows: In Section 2, we present the conceptual model for capturing rationale; In Section 3, we present the technique for identifying changes. In Section 4, we illustrate the connection of changes to rationale; In Section 5, we briefly discuss our implementation of this technique, as well as our experience in applying it to the German V-Modell XT [33] process standard. In Section 6, we present a short description of related approaches for comparing models and for capturing rationale; and finally, in section 7, we provide a summary and future research questions to be resolved.

2 Process Rationale

The following is a conceptual model that can be considered a second version of our attempt to understand the information needs for capturing the rationale behind process changes (see Fig 1). The results of our first attempt have been documented in [21]. In

order to complement our previous work, we decided to take the IBIS [14], QOC [18], and DRL [15] approaches as the basis. These approaches are called *argumentation-based* because they focus on the activity of reasoning about a design problem and its solution [8]. Based on our previous experience in collecting rationale [21], we assessed these approaches and defined a small subset of entities (see shadowed classes in Fig 1) that suited our goal, namely capturing rationale in a more structured way, while being careful of keeping the involved costs under control especially because this issue has been highly criticized by rationale experts. We connected these basic entities to three additional entities relevant for us, namely, event, changes, and process entities (the non-shadowed classes in Fig. 1).

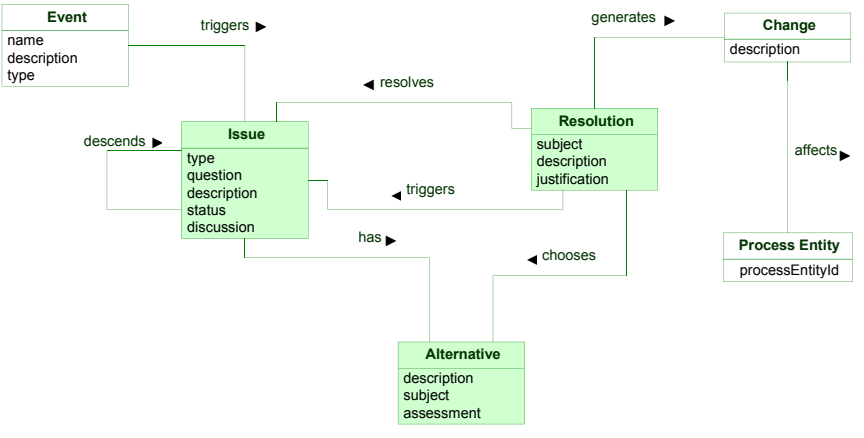

Fig. 1. Rationale Model (UML static structure diagram)

An *event* is considered to be the trigger of issues. Events can be *external* or *internal*. Examples are:

- External: new/updated process engineering technology (e.g., a new process modeling technique); new/updated regulatory constraints
- Internal: responses to failures to pass internal or external appraisals, assessments or reviews (e.g., changes needed to address a failure in passing an FDA audit); new/updated best practices emerging from "lessons learned" in just-completed projects (e.g., a new "best practice" approach to handling design reviews).

Issues can be problems or improvement proposals that are related to a (part of a) process and that need to be addressed. Issues are stated usually as questions in product-oriented approaches. In this work, the question has the purpose of forcing process engineers to reason about the situation they are facing. Additionally, an issue also contains a long description, a status (open, closed) and a discussion. The discussion is intended for capturing the emails, memos, letters, etc. where the issue was treated by process engineers. Additionally, an issue can be categorized by a type. This type can be selected from a classification of issues that needs to be developed or customized for an organization. It is possible to start with a preexisting classification (see for example [21]) as a basis, which can be refined based on experience gained from process evolution projects.

Alternatives are proposals for resolving the issue. Alternatives can be captured with subject (short description) or long descriptions. Alternatives are evaluated and assessed by process engineers regarding their impact and viability.

A *resolution* chooses an alternative that gets implemented by changing the process model. At the same time, one resolution could lead to opening up more issues. Note that a resolution has a subject (short description), a long description, and a justification. The justification is intended for capturing a summary of the analysis of the different alternatives, the final decision, and the proposed changes. *Changes* are the result of the decision captured in the resolution. They are performed on process entities. Some examples of changes performed to process entities are: activity *x* has been inserted; artifact *y* has been deleted; activity *x* has been moved to be a sub-activity of activity *z*.

Identifying which changes affect which process element is not an easy task. So far, we have developed a mechanism that allows us to document the rationale information proposed in Fig. 1 directly in the process model being altered [22]. The actual rationale information can be documented in special tables at the end of the process model description. The process engineer can then introduce the rationale information, perform the changes to the respective process entities, and then establish a reference to the corresponding rationale element. Then with the support of a special tool, the process evolution repository is updated. Although the approach proves to be suitable, we saw the need to provide more flexibility during the identification of process entities being changed and the documentation of the rationale behind such changes. In the following section, we present the technique we developed for that purpose.

3 Pattern-Matching-Based Change Identification

Our change identification technique is based on the Delta-P approach for process evolution analysis [28, 29]. This technique makes it possible to handle a wide variety of types of changes in a completely uniform way, to flexibly define the types of changes that are considered interesting or useful (this can be based on the structure and semantics of the process' metamodel), and to restrict the results to only certain types of changes, or even to certain interesting portions of a model.

3.1 A Normalized Representation for Process Models and Their Comparisons

Our first step consists of representing models (and later their differences) in such a way that a wide range of change types can be described using the same basic formalism. The representation we have chosen is based on that used by RDF [19] and similar description or metadata notations. For our purposes, this notation has a number of advantages over other generic notations:

- Being a generic notation for graph-like structures, it is a natural representation for a wide variety of process model types.
- It has a solid, standardized formal foundation.
- As shown below, the uniformity of the notation, which does not differentiate between relations and attributes, makes it possible to describe a wide range of changes with a straightforward pattern-matching notation.

– Also as shown below, the fact that many model versions can be easily put together into a single model makes it possible to use the same pattern-matching notation for single model versions and for comparisons.

Fig. 2 shows an example of this representation. The graph contains only two types of nodes, which we will call *entity* nodes (ovals in the figure) and *value* nodes (boxes in the figure). Entity nodes have arbitrary identifiers as labels. Value nodes are labeled by the value they represent, which can belong to a basic type (string, integer, boolean, etc.)

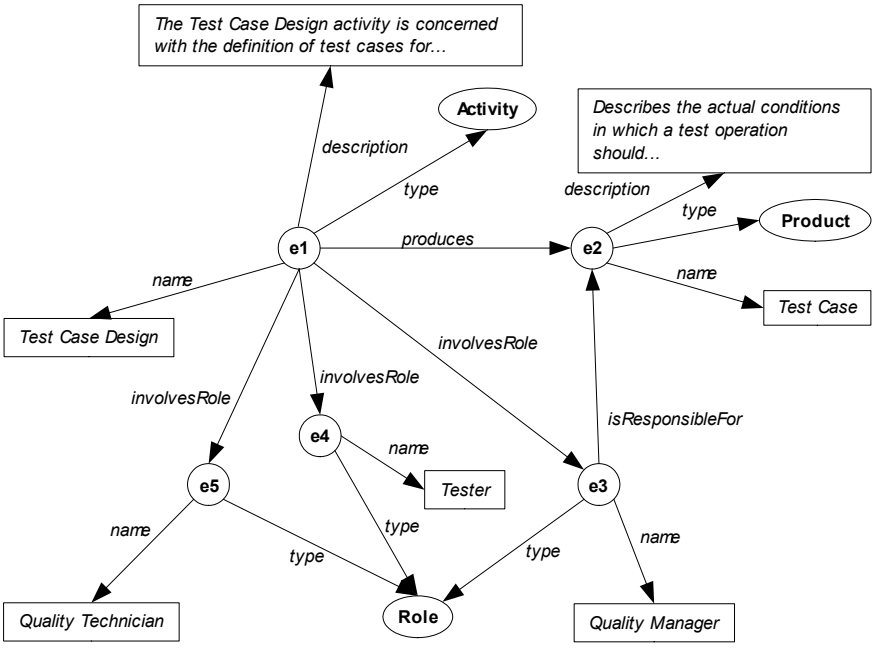

Fig. 2. A process model in normalized form

Arrows represent typed directed relations (type is given by their labels). Relations may connect two entity nodes, or may go from an entity node to a value node. It is not allowed for a relation to leave a value node. It is also not allowed for a node to exist in isolation. All nodes must be either the start or the end point of *at least* one relation. It follows that the graph is characterized completely by the set of the relations (edges) present in it, since the set of nodes is exactly the set of all nodes that are the start or the end of an edge.

The correspondence between attributed graphs and this normalized form is straightforward:

– *Entities and types correspond to entity nodes.* For each entity instance and entity type in the original graph, there is an entity node in the normalized graph. There is also a *type* relation between each node representing an entity and the node representing its type.
– *Attributes correspond to entity-value relations.* For each entity attribute in the orig-

inal graph, there is a relation labeled with the attribute name that connects the entity with the attribute value (that is, attributes in the original metamodel are converted into relation types). The value is a separate (value) node.

– *Entity relations correspond to entity-entity relations.* For each relation connecting two entities in the original graph, a relation connecting their corresponding entity nodes is present in the normalized graph.[1]

Fig. 3 shows an evolution of the model presented in Fig. 2, using the normalized notation, with changes also highlighted. Formally, this graph respects exactly the same restrictions as the normalized model representation. The only addition is that edges are *decorated* (using interrupted and bolder lines) to state the fact that they were deleted, added, or simply not touched. This leads us to the concept of a *comparison graph* or *comparison model.* The comparison model of two normalized models A and B contains all edges present in either A or B, or in both, and only those edges. Edges are marked (decorated) to indicate that the edge is only in A, only in B, or in both A and B.

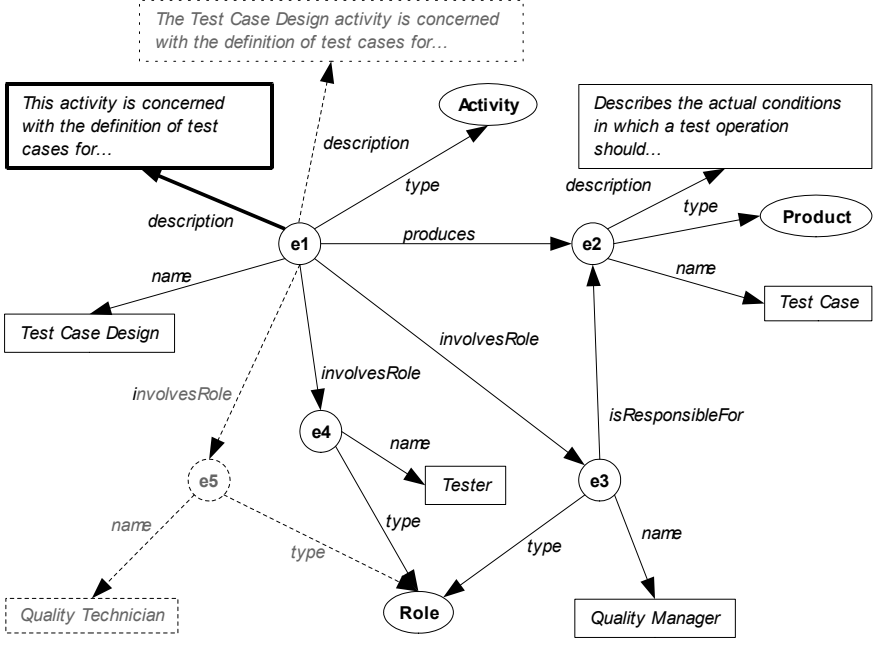

Fig. 3. A process model comparison in normalized form

The main aspect to emphasize here is the fact that all changes are actually reduced to additions and deletions of relations between nodes. This results in part from the fact that attributes are represented as relations, but also from the fact that nodes cannot exist in isolation. It is possible (and safe) to identify entity additions and deletions by looking for additions and deletions of *type* relations in the model.

[1] Relations with attributes can be modeled by introducing entity nodes that represent them, but the details are beyond the scope of this paper.

The fact that the normalized representation reduces all changes to sets of relation additions and deletions is useful because it permits describing many types of changes uniformly. On the other hand, an adequate formalism to describe changes must have clear, unambiguous semantics, and must be, at the same time, accessible to users. The following sections discuss with the help of examples, the mechanism that we have chosen for this task: a graphical pattern-matching language.

3.2 Example 1: Additions and Deletions

Our first example is related to one of the simplest possible model changes: adding or deleting process entities. Fig. 4 shows four patterns that identify changes of this type with different levels of generality. The pattern in Fig. 4a) matches all additions of process activities, and for each match, sets the *?id* variable with the identifier of the new activity. In a similar way, the pattern in Fig. 4b) matches all deletions of process products. These patterns can be generalized to identify arbitrary additions and deletions: the pattern in Fig. 4c) identifies all entity additions, and instances an additional variable with the type of the added entity. Finally, Fig. 4d) shows a pattern that not only finds new activities, but sets a variable with the corresponding name, a useful feature if the results of matching the pattern are used, for example, to produce a report.

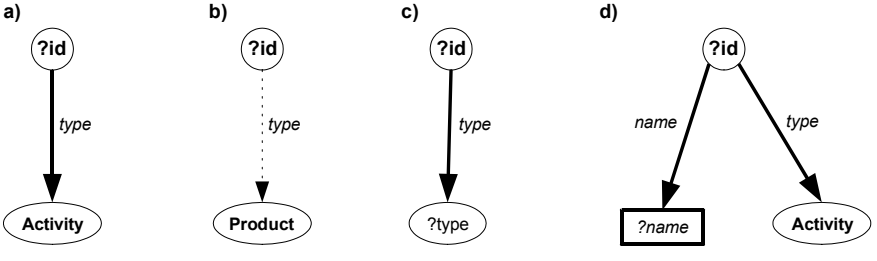

Fig. 4. Patterns for identifying entity additions and deletions

3.3 Example 2: Changes in Attribute Values

Just as important as identifying entity additions and deletions is finding entities whose attributes were changed. Fig. 5 shows three patterns that describe changes in attribute values. Fig. 5a) is basically an excerpt from the comparison graph in Fig. 3, which captures the fact that an attribute *description* was changed. This pattern, however, matches only the particular change shown in the example. The pattern in Fig. 5b) is a generalized version of the first one. By using variables for the entity identifier, as well as for the old and new property values, this pattern matches all cases where the *description* attribute of an arbitrary entity was changed. Note that each match sets the value of the *?id* variable to the identifier of the affected entity, and the values of *?oldValue* and *?newValue* to the corresponding old and new values of the *description* property. The pattern in Fig. 5c) goes one step further and uses a variable for the attribute labels as well, which means it matches all attribute value changes. Note that

these patterns match once for *each* changed property in *each* object. Finally, the pattern in Fig. 5d) constitutes a specialization of its peer in Fig. 5c): it is restricted to all attribute value changes affecting process activities.

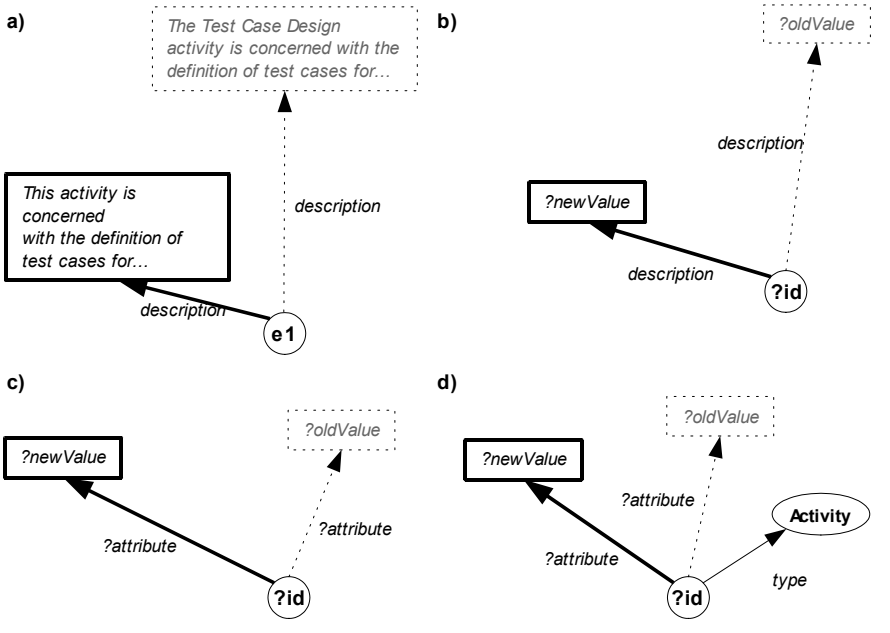

Fig. 5. Four patterns for identifying attribute value changes

Changes identified in this way can be fed into additional algorithms that perform attribute-specific comparisons, such as, for example identifying added or deleted individual words or characters in text-based attributes. This way, potentially expensive specific comparison algorithms are only applied to relevant items.

4 Connecting Rationale to Process Changes

RDF has been designed with the intention of supporting the interchangeability of separate packages of metadata defined by different resource description communities We have defined a separate RDF model containing the rationale concepts described in Fig. 1. This allow us to reference the comparison model. Fig. 6 elaborates on the previous example and illustrates how we connect the comparison model to the rationale model. The figure can be interpreted as having two separated RDF models, one for the comparison of processes and the other one for the rationale for changes. Let us assume that a review board met, discussed, and documented issue (i1) concerning the expensive performance of the activity Test Case Design (e1) by more than one role, i.e, Quality Manager (e3), Tester (e4), and Quality Technician (e5). The review board analyzed two different alternatives (a1) and (a2), resolving to reduce costs (r1) by removing the Quality Technician (e5) from the list of roles responsible for the Test Case Design (e1). Afterwards, appropriate changes were performed to the process.

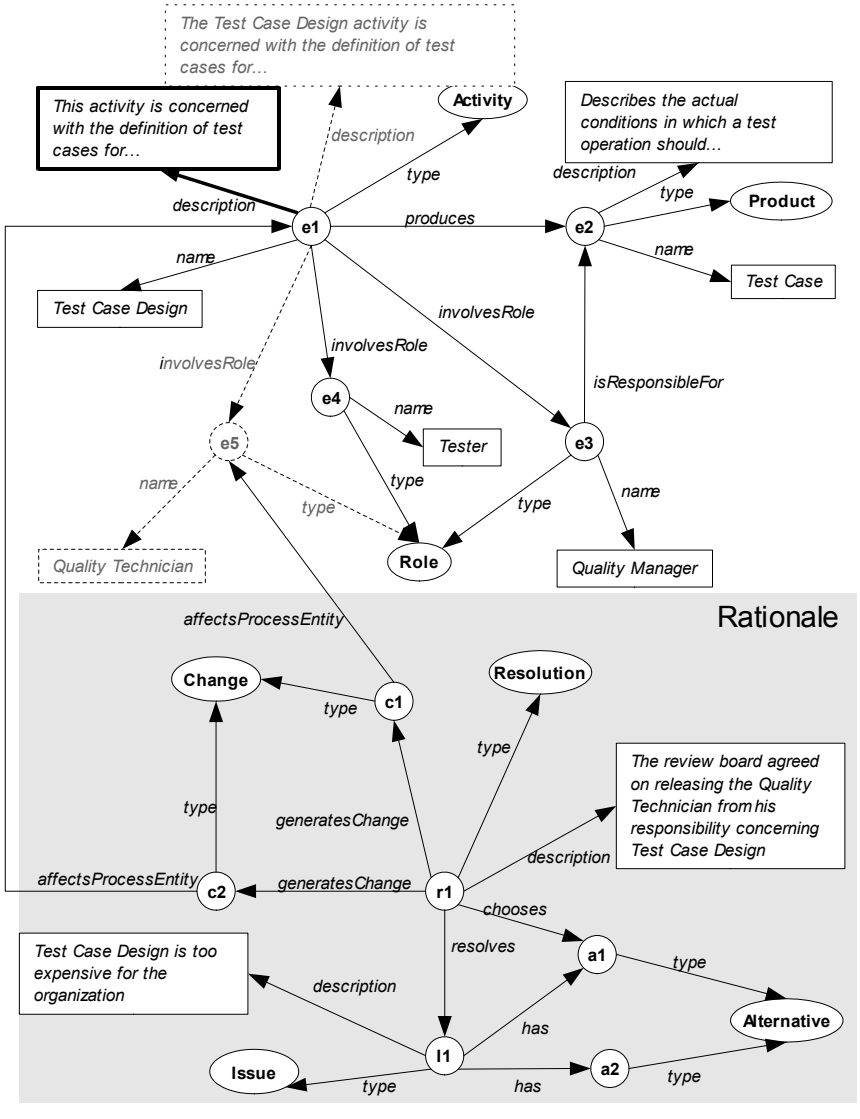

Fig. 6. Connecting the rationale model to the comparison model

Using the pattern for identifying entity additions and deletions (see Fig. 4), we can detect that entity e5 has been deleted. This can be counted as a change and labeled as c1. We can then connect change c1 to the process entity e5 through the property *affectsProcessEntity*. Also, by using the pattern for identifying attribute value changes, we can detect that the description has been modified. This change can be counted and labeled as c2. As in the previous case, we use the property *affectsProcessEntity* to connect c2 to e1. Both changes were generated by a single decision, which in this case corresponds to r1. We can then connect r1 to c1 and c2 through the *generatesChange* property. The resolution r1 is also connected to the other elements of the rationale

model, namely alternatives and issues. This way we can go through comparison models identifying changes and connecting them to their corresponding rationale model.

5 Implementation and Validation

An implementation of the pattern-matching based change identification technique presented in the previous sections is available as part of the *Evolyzer* tool [29]. We have tested our approach and tools by applying them to the various official releases of the V-Modell XT [33], a large prescriptive process model intended originally for use in German public institutions, but finding ever increasing acceptance from the German private sector. As of this writing, the *Evolyzer* tool still lacks a graphical editor for change patterns. However, patterns can be expressed as textual queries using a syntax that basically follows that of the emerging SPARQL [25] query language for RDF. Expressed as queries, patterns can be executed to find all their occurrences in a model. The V-Modell XT constitutes an excellent testbed for our approach and implementation. Converted to the normalized representation defined in Section 3.1, the latest public version at the time of this writing (1.2) produces a graph with over 13,000 edges. This makes it a non-trivial case, where tool support is actually necessary to perform an effective analysis of the differences between versions. Our first trial with the V-Modell XT consisted of analyzing the evolution of the V-Modell XT itself, where we compared 559 model versions that were produced in 20 months of actual model development. First, we normalized each release by using a parser we implemented in the interpreted, object-oriented Python programming language [17] which is able to navigate through the XML-specific version of the V-Modell XT, identifying the entities and properties, and moving this information to a process evolution repository. The rationale model was obtained from processing the information stored in the bug tracking system used for the continuous improvement of the V-Modell XT. This system supports the change management process designed for the model. Users can report problems and provide improvement suggestions in the form of change requests. Such change requests are processed by the team responsible for the model. The resulting analysis and decisions are also documented in the tracking system. Once the approved changes are finished (using specialized model editing tools) a new version of the V-Modell XT is stored in a configuration management system. In order to keep track of the decisions implemented, the V-Modell XT team member provides a short description of the change with a reference to the respective change request. We processed the information contained in the bug tracking system as well as in the configuration management system to create a rationale RDF model. With this rationale model as a basis, we proceeded to calculate changes between versions, connect them to the rationale model (as explained in the previous section), and store them in our process evolution repository.

By using our patterns, we found 19104 changes between version 1 and version 559. 30% corresponded to entity additions, 27% to entity deletions, and 63% to attribute modifications.

Fig. 7 shows the results of querying the rationale for process evolution repository using SPARQL queries via the *Evolyzer* tool. The query shows for each change the

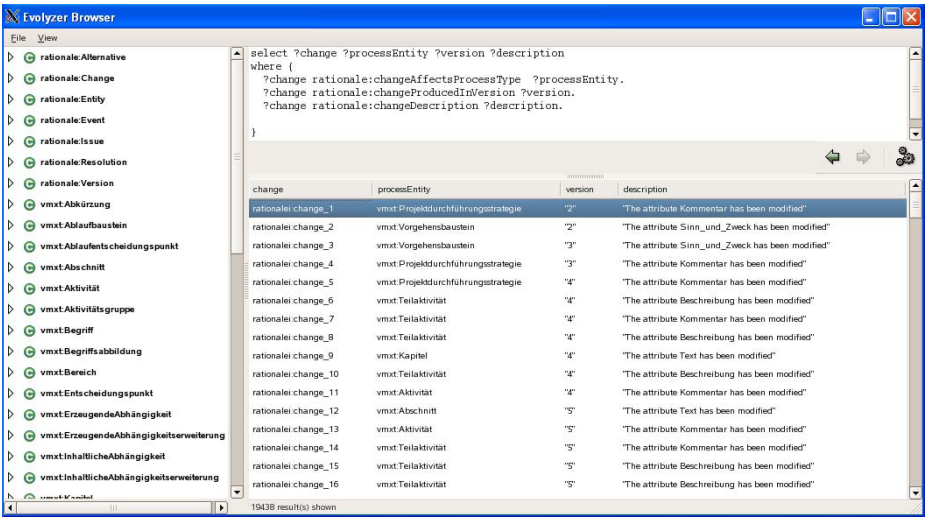

Fig. 7. Querying the Rationale for the V-Modell XT via the *Evolyzer* tool

process entity that affected, the version where this occurred and the description of such a change.

6 Related Work

Several other research efforts are concerned in one way or another with comparing model variants syntactically and providing an adequate representation for the resulting differences.

[1] and [16] deal with the comparison of UML models representing diverse aspects of software systems. These works are oriented towards supporting software development in the context of the Model Driven Architecture. Although their basic comparison algorithms are applicable to our work, they are not concerned with providing analysis or visualization for specific users.

[20] presents an extensive survey of approaches for software merging, many of which involve comparison of program versions. The surveyed works mainly concentrate on automatically merging program variants without introducing inconsistencies, but not, as in our case, on identifying differences for user analysis.

[3] provides an ontology and a set of basic formal definitions related to the comparison of RDF graphs. [32] and [11] describe two systems currently under development that allow for efficiently storing a potentially large number of versions of an RDF model by using a compact representation of the raw changes between them. These works concentrate on space-efficient storage and transmission of change sets, but do not go into depth regarding how to use them to support higher-level tasks (like process improvement).

An extensive base of theoretical work is available from generic graph comparison research (see [13]), an area that is concerned with finding isomorphisms (or correspondences that approach isomorphisms according to some metric) between arbitrary

graphs whose nodes and edges cannot be directly matched by name. This problem is analogous in many ways to the problem that interests us, but applies to a separate range of practical situations. In our case, we analyze the differences (and, of course, the similarities) between graphs whose nodes can be reliably matched in a computationally inexpensive way.

Concerning rationale, Dutoit et al. [8] introduce the term *software engineering rationale*, claiming that this term is more useful for discussing rationale management in software engineering. They emphasize that the software development life cycle contains several activities where important decisions are taken, and where rationale plays an important role. In software engineering, most approaches have contributed to the rationale domain by providing new ideas and mechanisms to reduce the risk associated with rationale capture. Such approaches were conceived having in mind the goal of providing short-term incentives for those stakeholders who create and use the rationale. For example, SCRAM [30], an approach for requirements elicitation, integrates rationale into fictitious scenarios that are presented to users or customers so that they understand the reason for them and provide extra information. It is expected that they can see the use and benefit of rationale. Something similar happens in the inquiry cycle [24], which is an iterative process whose goal is to allow stakeholders and developers to work together towards a comprehensive set of requirements.

Most of the approaches developed for software engineering rationale offer tool support provided as either adaptations or extensions of specific requirements and development tools, e.g., SEURAT [6], Sysiphus [9], DRIMER [23], or the Win-Win Negotiation Tool [34], REMAP [26], and C-ReCS [12].

Little work has been done in other areas apart from design and requirements. One of them is the process modeling area. Here, the need and value have been identified, and a couple of research initiatives have been followed with the goal of generating rationale information from project-specific process models. One approach developed by Dellen et al. [7] is Como-Kit. Como-Kit allows automatically deducing causal dependencies from specified process models. Such dependencies could be used for assessing process model changes. Additionally, Como-Kit provides a mechanism for adding justifications to a change. The Como-Kit system consists of a modeling component and a process engine. Como-Kit was later integrated with the MVP approach [5]. The result of this integration effort was the Minimally Invasive Long-Term Organizational Support platform (MILOS) [31]. Sauer presented a procedure for extracting information from the MILOS project log and for justifying project development decisions [27]. According to Sauer, rationale information could be semi-automatically generated. However, the approach does not capture information about alternatives that were taken into account for a decision.

7 Conclusions and Outlook

Due to factors like model size and metamodel differences, the general problem of identifying and characterizing changes in process models is not trivial. By expressing models in a normalized representation, we are able to characterize interesting changes using a graphical pattern matching language. Graphical patterns provide a well-de-

fined, unambiguous and, arguably, intuitive way to characterize common process model changes, as our examples show.

Our implementation of pattern queries in the *Evolyzer* system demonstrates that our pattern-based change identification technique can be used in practical situations involving very large process models like the V-Modell XT. It is important to stress, however, that the technique requires the process entities in compared models to have stable identifiers that are used consistently across versions. This is normally the case when comparing versions of the same model, but not when comparing models that were created independently from each other.

Using RDF allowed us to connect two different data sets and create an even more comprehensive one that makes it easier process engineers or stakeholders to analyze and understand the evolution of a process. The information that we processed from the V-Modell XT bug tracking system and stored in the process evolution repository as a rationale model allowed us to answer questions that we would otherwise have to guess, such as: Which process elements were affected by a change? Which issue had the largest impact on a process? , Which type of issues demand the highest number of changes? A remaining important research question deals with the visualization of the large amount of information stored in a process evolution repository like the one of the V-Modell XT. We are currently investigating mechanisms that facilitate visualization, e.g., we are trying to identify a set of "most wanted queries" based on the special interests of organizations interested in managing process evolution. Such queries can be deduced from the goals of the organization and reduce the scope of the information to be analyzed.

Acknowledgements. We would like to thank Sonnhild Namingha from Fraunhofer IESE for proofreading this paper, and the members of the V-Modell XT team for providing us with the information needed to perform this work. This work was supported in part by the German Federal Ministry of Education and Research (V-Bench Project, No. 01| SE 11 A).

References

1. Alanen, M., Porres, I.: Difference and Union of Models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003 - The Unified Modeling Language. Modeling Languages and Applications. LNCS, vol. 2863, pp. 2–17. Springer, Heidelberg (2003)
2. Armbrust, O., Ocampo, A., Soto, M.: Tracing Process Model Evolution: A Semi-Formal Process Modeling Approach. In: Oldevik, Jon. (ed.) u.a.: ECMDA Traceability Workshop (ECMDA-TW) 2005- Proceedings. Trondheim, pp. 57-66 (2005)
3. Berners-Lee, T., Connolly D.: Delta: An Ontology for the Distribution of Differences Between RDF Graphs. MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) (last checked 2006-03-30) Online publication http://www.w3.org/DesignIssues/Diff
4. Bohem, B., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.: Using the WinWin Spiral Model: A Case Study. IEEE Computer 31(7), 33–44 (1998)
5. Bröckers, A., Lott, C.M., Rombach, H.D., Verlage, M.: MVP-L Language Report Version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, Germany (1995)

6. Burge, J., Brown, D.C.: An Integrated Approach for Software Design Checking Using Rationale. In: Gero, J. (ed.) Design Computing and Cognition '04, pp. 557–576. Kluwer Academic Publishers, Netherlands (2004)
7. Dellen, B., Kohler, K., Maurer, F.: Integrating Software Process Models and Design Rationales. In. Proceedings of 11th Knowledge-Based Software Engineering Conference (KBSE '96), Syracuse, NY, pp. 84-93 (1996)
8. Dutoit, A., McCall, R., Mistrík, I., Paech, B. (eds.): Rationale Management in Software Engineering. Springer, Berlin (2006)
9. Dutoit, A., Paech, B.: Rationale-Based Use Case Specification. Requirements Engineering Journal. 7(1), 3–19 (2002)
10. Heidrich, J., Münch, J., Riddle, W.E., Rombach, H.D.: People-oriented Capture, Display, and Use of Process Information. In: Acuña, Silvia T (ed.) u.a.: New Trends in Software Process Modeling. Singapore: World Scientific, Series on Software Engineering and Knowledge Engineering, vol. 18, pp. 121–179 (2006)
11. Kiryakov, A., Ognyanov, D.: Tracking Changes in RDF(S) Repositories. In: Proceedings of the Workshop on Knowledge Transformation for the Semantic Web, KTSW 2002. Lyon, France (2002)
12. Klein, M.: An Exception Handling Approach to Enhancing Consistency, Completeness, and Correctness in Collaborative Requirements Capture. Concurrent Engineering Research and Applications 5(1), 37–46 (1997)
13. Kobler, J., Schöning, U., Toran, J.: The Graph Isomorphism Problem: Its Structural Complexity. Birkhäuser (1993)
14. Kunz, W., Rittel, H.: Issues as Elements of Information Systems. Working Paper No. 131, Institut für Grundlagen der Plannung, Universität Stuttgart, Germany (1970)
15. Lee, J.: A Qualitative Decision Management System. In: Winston, P.H., Shellard, S. (eds.) Artificial Intelligence at MIT: Expanding Frontiers, vol. 1, pp. 104–133. MIT Press, Cambridge, MA (1990)
16. Lin, Y., Zhang, J., Gray, J.: Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development. In: OOPSLA Workshop on Best Practices for Model-Driven Software Development, Vancouver (2004)
17. Lutz, M.: Programming Python, 2nd edn. O'Reilly & Associates, Sebastopol, California (2001)
18. MacLean, A., Young, R.M., Belloti, V., Moran, T.: Questions, Options, and Criteria: Elements of Design Space Analysis. Human-Computer Interaction 6, 201–250 (1991)
19. Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation (2004) (last checked 2006-03-22) available from http://www.w3.org/TR/rdf-primer/
20. Mens, T.: A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering 28(5) (2002)
21. Ocampo, A., Münch, J.: Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) Software Process Change. LNCS, vol. 3966, pp. 334–334. Springer, Heidelberg (2006)
22. Ocampo, A., Münch, J.: The REMIS Approach for Rationale-driven Process Model Evolution. Submitted to ICSP 2007 (submitted, 2007)
23. Pena-Mora, F., Vadhavkar, S.: Augmenting Design Patterns with Design Rationale. Artificial Intelligence for Engineering Desgin, Analysis, and Manufacturing 11, 93–108 (1996)
24. Potts, C., Bruns, G.: Recording the Reasons for Design Decisions. In: Proceedings of the 10th International Conference on Software Engineering (ICSE'10). Los Alamitos, CA, pp. 418–427 (1988)
25. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Working Draft (2006) (last checked 2006-10-22) available from http://www.w3.org/TR/rdf-sparql-query/

26. Ramesh, B., Dhar, V.: Supporting Systems Development by Capturing Deliberations During Requirements Engineering. IEEE Transactions on Software Engineering 18(6), 498–510 (1992)
27. Sauer, T.: Project History and Decision Dependencies. Diploma Thesis. University of Kaiserslautern (2002)
28. Soto, M., Münch, J.: Process Model Difference Analysis for Supporting Process Evolution. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) EuroSPI 2006. LNCS, vol. 4257, Springer, Heidelberg (2006)
29. Soto, M., Münch, J.: The DeltaProcess Approach for Analyzing Process Differences and Evolution. Internal report No. 164.06/E, Fraunhofer Institute for Experimental Software Engineering (IESE) Kaiserslautern, Germany (2006)
30. Sutcliffe, A., Ryan, M.: Experience with SCRAM, a Scenario Requirements Analysis Method. In: Proceedings of the 3rd International Conference on Requirements Engineering, 1988, Colorado Springs, CO, pp. 164–173 (1998)
31. Verlage, M., Dellen, B., Maurer, F., Münch, J.: A Synthesis of Two Process Support Approaches. In: Proceed-ings of the 8th International Conference on Software Engineering and Knowledge Engineering (SEKE'96), June 10-12, 1996, Lake Tahoe, Nevada, USA, pp. 59–68 (1996)
32. Völkel, M., Enguix, C.F., Ryszard-Kruk, S., Zhdanova, A.V., Stevens, R., Sure, Y.: Sem-Version - Versioning RDF and Ontologies. Technical Report, University of Karlsruhe (2005)
33. V-Modell XT (last checked 2006-03-31) Available from http://www.v-modell.iabg.de/
34. WinWin. The Win Win Spiral Model. Center for Software Engineering University of Southern California http://sunset.usc.edu/research/WINWIN/winwinspiral.html