Using Model Comparison to Maintain Model-to-Standard Compliance

Martín Soto
Fraunhofer Institute for
Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
+49 (631) 6800 2214
Martin.Soto@iese.fraunhofer.de

Jürgen Münch
Fraunhofer Institute for
Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
+49 (631) 6800 1300
Juergen.Muench@iese.fraunhofer.de

ABSTRACT
Keeping an artifact, such as a an industrial design document or a process description, compliant with a relevant standard or norm is a challenging task. A wide variety of organizations, ranging from software houses to defense contractors, are concerned with this problem, since norm compliance (or lack thereof) can have important technical and business related consequences for them. In this paper, we discuss a compliance maintenance approach for complex structured documents that we are currently developing at Fraunhofer IESE. At the heart of this approach lies our *Evolyzer* model comparison tool and its underlying model comparison approach, *DeltaProcess*.

Categories and Subject Descriptors
D.2.0 [**General**]: Standards

General Terms
Algorithms, Management, Documentation, Standardization.

Keywords
Compliance, Compliance Maintenance, Model Comparison, Software Processes, *Evolyzer*, *DeltaProcess*

1. Introduction
Since the introduction of the `diff` utility in the early 1970s, the ability to compare program files has been strongly linked with one single application: software versioning. Consequently, it is not surprising that researchers working on model comparison often see model versioning and collaborative work on models as their only target applications. However, our experience in working with complex software process models at Fraunhofer IESE shows us that other applications of model comparison may be equally interesting, from both a practical and a scientific point of view.

In this paper, we discuss one such application, namely, compliance maintenance. In general, the problem of compliance maintenance can be described as guaranteeing that an evolving artifact (e.g., as in our case, a software process model) complies with the requirements and restrictions posed by a second artifact (usually a published standard or norm). People in a wide variety of organizations and areas of endeavor are faced with this problem on a daily basis. For example, almost every industrial product must comply with published industrial norms and regulations, which means that the corresponding specification and design artifacts must often be verified for compliance and kept in a compliant state through their evolution. Procedure manuals and process descriptions of different types are also often subject to a similar treatment.

Verifying compliance is usually a laborious manual process that, for cost reasons, cannot be performed for every version of an artifact. Consequently, having the ability to compare artifact versions can play a key role in reducing costs, since compliance verification can be restricted to the changed areas of the artifact.

In what follows, we describe our approach for compliance maintenance based on looking at different types of artifacts as models, and performing model comparison on them. Section 2 describes the compliance maintenance problem using software process models as an example. Section 3 presents our general approach to compliance management, whereas Sections 4 and 5 concentrate on the technical details of our current model comparison approach, and our planned implementation of compliance maintenance. We close the paper with a short discussion of related work, and a summary.

2. An Example of Compliance Maintenance: Software Process Models
In order to illustrate the compliance maintenance problem, we will rely on an example taken from our direct experience: maintaining complex software process descriptions compliant with software process standards.

2.1 Processes and Process Models
Informally, software processes can be defined as the set of activities performed by a group or organization to develop software systems. This comprises both technical and managerial activities, and involves a potentially large number of organizational and technical roles. During the last 25 years, the notion of software process has become increasingly important, since many modern approaches to software quality rely explicitly or implicitly on the assumption that the quality of software products is directly related to the quality of the processes used to develop them.

According to this view, in order to be able to predictably deliver high-quality products, an organization must reach a certain level of *process maturity*. Among other aspects, maturity involves a *defined process*, namely, a process that is documented and can be followed in a consistent, repeatable fashion for every software de-

Conference'04, Month 1–2, 2008, City, State, Country.

velopment project. Defined processes are not only important for making software projects predictable, but, above all, for supporting improvement. The usual improvement cycle involves analyzing a defined process to look for improvement opportunities, changing the process accordingly, comparing the performance of the new process with that of its predecessor in order to determine if the changes were effective, and taking corrective actions in case they were not.

Since processes are intended to be followed by human beings, process descriptions are usually natural language documents. This does not mean, however, that they are simple. Indeed, software processes are typically very complex, and this necessarily calls for long and quite complex process descriptions, which must be highly structured in order to be useful. Also, the fact that they are used as reference documents requires the inclusion of a network of internal and external cross-references.

Since this complex internal structure is difficult to create and modify using a standard text editing program, some approaches to process definition use specialized tools to formalize a description's underlying structure. The formalized structure (e.g., activity and role hierarchies, relationships showing which roles are involved in which activities) is now used as a scaffold to hold (potentially large) text descriptions intended for human consumption. The advantages of such an approach are twofold. On the one hand, the use of natural language keeps the process description accessible to a wide audience, including users without a technical background. On the other hand, the formalized structure can be used to perform a number of automated and semiautomated checks that help guarantee internal and external consistency. We call a process description with an underlying formalized structure a *process model.*

2.2 Process Standards

Large software purchasers (governments, large companies) are often faced with the problem of guaranteeing the quality of the software they purchase. Low-quality software may have a significant negative impact on the performance of an organization, with the negative consequences becoming catastrophic if the software in question is mission- or life-critical.

One possible way to increase an organization's confidence in the quality of its software purchases is to ensure that software contractors follow sound software development practices. This can be done by defining a standard that states a set of minimum requirements that a sound software process should fulfill. A number of software process standards are currently available to organizations interested in improving the quality (maturity) of their processes, or evaluating potential software providers. Examples of widespread, general-purpose process standards are CMMI [1] and SPICE [2]. Many other standards exist that cover software development for specific areas of application.

Since the inception of the first general-purpose process standards at the beginning of the 1990s their use has been steadily increasing, to the point that many large companies and governmental institutions now require their software contractors to be compliant with specific standards. For this reason, companies offering software development services are often concerned with the problem of guaranteeing that their processes comply with the requirements imposed by one or more of these standards.

In order for a company to show that its processes are compliant, a so-called *process assessment* must be performed, where the process is thoroughly checked in order to determine the extent to which it fulfills the stated requirements. This is usually quite an involved procedure that must be performed by trained experts, and that implies significant time and highly-qualified labor investments for both the assessing team and the assessed software contractor. As expensive as it may be, this procedure is often necessary for a contractor to be able to do business with its potential customers.

2.3 Model Evolution and Compliance Maintenance

As long as a company's process descriptions are not changed, the results of any assessment based on them remain valid. Changes to the process descriptions, on the other hand, always involve the risk of breaking standard compliance. For this reason, companies that have already performed an expensive process assessment often try to refrain from changing their processes in the future.

However, no process is perfect, and keeping a process unchanged in order to keep it compliant actually defeats the main purpose of documenting it, namely, improvement. In order to improve at all, a process must necessarily change. Companies are then faced with a difficult problem: either they keep their processes (or, at least, process descriptions) static, and miss any improvement opportunities, or they change their processes and incur the high costs of frequently assessing them for compliance. Clearly, none of these alternatives is entirely satisfactory. In practice, organizations often end up letting their actual processes deviate from the certified, written definition, a situation that leads to a host of problems of its own.

A similar problem arises when standards change. As soon as a new version of a standard is available, compliance to the new version must be reassessed. This often involves costs a software development company may not be able to avoid, because its clients will eventually start requiring compliance with the new version of the standard.

3. An Approach to Compliance Maintenance

The high costs associated with process assessments cannot be avoided completely. Since both process descriptions and process standards usually contain large amounts of natural language text, the assessment procedure absolutely requires trained human beings to interpret the meaning of these texts. For the foreseeable future, process assessments will remain manual- and labor-intensive.

Nonetheless, the fact that we cannot dispense of human labor altogether does not mean that the current situation cannot be improved. In particular, when an already compliant process model is changed, we can assume that only the changed portions of the

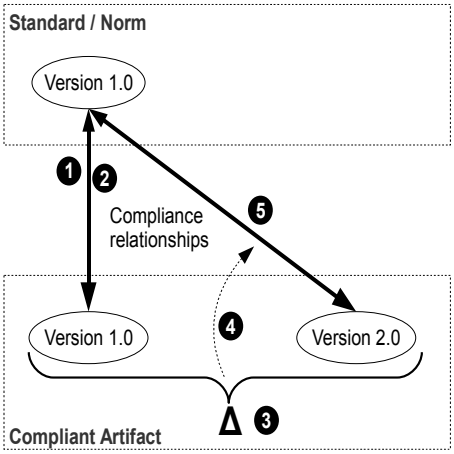


Figure 1: Approach to compliance maintenance

model may have possibly broken compliance. By limiting (re)assessment to those changed portions, it is possible to achieve significant cost savings.

The previous idea leads us to the compliance management approach described below and sketched in Figure 1 (where the numbers enclosed in black circles correspond to the steps below):

1. Perform a manual compliance assessment.
2. Express the results of the assessment as a set of relationships connecting both models.
3. When a new version of the process description is available, compare it with the previous version.
4. Use the resulting set of differences (or delta, shown in the figure using the Δ symbol) to determine which of the compliance relationships (produced in step 2) may be affected by the changes.
5. Based on the results of the previous step, update the process model and/or the compliance relationships as necessary.

The viability of these steps deserves some discussion. Step 1 is a standard process assessment that must be performed manually as explained in the previous section. For step 2, the authors already have experience in formalizing compliance relationships for a large process model (as reported in [3]), so there are good reasons to expect this to be reasonably possible for most process standards.[1] For the remaining steps, we expect our *Evolyzer* model comparison system to be able to support them to a large extent. The following sections cover this last point in more detail.

4. *Evolyzer*: A Tool For Model Comparison

This section describes the *Evolyzer* model comparison system, as well as its underlying *DeltaProcess* approach [4, 5] to model comparison. The goals of *DeltaProcess* can be summarized as follows:

- Operate on a variety of model schemata. New schemata can be supported with relatively little effort. Here, we understand schema as the set of entity and relationship types used by a model, as well as the set of restrictions that applies to them.
- Be flexible about the types of changes that are recognized and how they are displayed.
- Allow for easily specifying change types that are specific to a particular schema or to a particular application.
- Be tolerant of schema evolution, by allowing the comparison of model instances that correspond to different versions of a schema (this sort of comparison requires additional effort, though.)

These characteristics make the approach especially suitable for compliance maintenance. In particular, as discussed in Section 5, support for multiple schemata, and flexibility regarding the types of changes that are recognized play a key role when reestablishing compliance.

4.1 Comparison Approach

The *DeltaProcess* comparison approach that constitutes the basis of the *Evolyzer* tool can be decomposed into the following steps:

1. Convert the model instances into a normalized triple-based representation. We use the Resource Description Framework (RDF) for this purpose.
2. Perform an identity-based comparison of the resulting RDF models, to produce a so-called *comparison model.* This model corresponds roughly to the results of a simple comparison based on unique identifiers.
3. Find relevant changes by recognizing the corresponding change patterns in the comparison model.

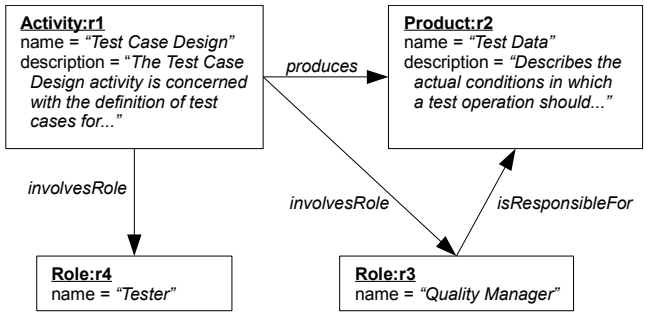
4.2 The Resource Description Framework

DeltaProcess' step 1 consists of expressing the models using the Resource Description Framework (RDF). RDF [6] was originally defined as a notation for describing resources in the World Wide Web. In actuality, however, it can be seen as a highly abstract metamodel for representing arbitrary sets of objects and their relationships.

RDF models are sets of so-called *statements,* each of which makes an assertion about the world. Statements are composed of three parts, called the *subject*, the *predicate*, and the *object*. The subject is an arbitrary, unique identifier of an entity. The predicate identifies an attribute of this entity or a relation the entity has with another entity. Depending on the predicate, the object contains either the value of the attribute, or the identifier of the entity pointed to by the relation. To illustrate this type of model representation, Figure 2 shows a small process model sample (top) and its corresponding RDF representation (bottom).

Our choice of RDF as the base notation for comparison may be controversial. It could be argued, for example, that the Meta Object Facility (MOF) would be a more appropriate choice. Our reasons for choosing RDF can be summarized as follows:

1. As already explained, RDF is a highly normalized notation, where attributes and relations are handled uniformly. This makes it easier to store and process models.




```
(r1, type, Activity)
(r1, name, "Test Case Design")
(r1, description, "The Test Case Design activity is
 concerned with the definition of test cases
 for...")
(r2, type, Product)
(r2, name, "Test Data")
(r2, description, "Describes the actual conditions
 in which a test operation should...")
(r1, produces, r2)
(r4, type, Role)
(r4, name, "Tester")
(r1, involvesRole, r4)
(r3, type, Role)
(r3, name, "Quality Manager")
(r1, involvesRole, r3)
(r3, isResponsibleFor, r2)
```

Figure 2: A process model sample and a possible RDF representation for it.

[1] Since compliance is defined by each process standard in different terms, more experience is required in this area.

2. RDF is schema agnostic. Models with arbitrary sets of entity and relationship types can be expressed in RDF without restrictions.
3. It is generally inexpensive and straightforward to convert models to the notation.
4. Models do not lose their "personality" when moved to the notation. Once converted, model elements are still easy for human beings to recognize.
5. The results of a basic, unique-identifier-based comparison can be expressed in the same notation. That is, comparisons are models, too. Additionally, elements remain easy for human beings to identify even within the comparison.
6. We already have a standardized notation (SPARQL) [7] to express patterns in RDF graphs. With minimal adaptations, this notation is adequate for specifying interesting types of changes in a generic way. Our *Evolyzer* system provides an efficient implementation of a subset of SPARQL that is adequate for this purpose.
7. The tabular structure of RDF makes it possible to rely on standard (mature, inexpensive) relational database technology for storage and retrieval.

4.3 The Comparison Model

The second step in *DeltaProcess* corresponds to the construction of the so-called comparison model. Using the fact that RDF statements refer to the modeled entities by using unique identifiers, two versions of a model expressed as RDF can be efficiently compared to determine which statements are common to both versions, and which are exclusive to only one of them. For the comparison of two model versions A and B, we express the results of this comparison by constructing the union set of A and B, and *marking* the statements in it as belonging to one of three subsets: statements *only in A*, statements *only in B*, and statements *common to A and B*. The resulting set, with the statements marked this way, is called the *comparison model* of A and B.

One central aspect of the comparison model is that it is also a valid RDF model. The theoretical device that makes this possible is called *RDF reification,* and is defined formally in the RDF specification. The main purpose of RDF reification is to allow for statements to speak about other statements. This way, it is possible to add assertions about the original model statements, telling which of the groups above they belong to.

```
select ?activity
where {
  graph comp:B {?activity rdf:type proc:Activity}
}

select ?product ?oldName ?newName
where {
  graph comp:AB {?product rdf:type proc:Product}
  graph comp:A {?product proc:name ?oldName}
  graph comp:B {?product proc:name ?newName}
}

select ?activity ?role ?roleName
where {
  graph comp:AB {?activity rdf:type proc:Activity}
  graph comp:B {?role rdf:type proc:Role .
                ?activity proc:involvesRole ?role .
                ?role proc:name ?roleName}
}
```

Figure 3: SPARQL queries describing some basic change types in process models.

4.4 Identification of Specific Changes

In most practical situations, comparison models are too large and detailed to be useful for direct human analysis. The main problem lies in the fact that most changes that human beings perceive as a unit potentially correspond to many added and/or deleted statements. For example, the change "product `r2` was renamed from 'Test Data' to 'Test Case'" corresponds in RDF to removing the statement `(r2, name, "Test Data")` and adding the statement `(r2, name, "Test Case")`. More complex changes, such as adding a new process entity, may involve a larger number of statements, and, for this reason, may be much more difficult to identify and interpret.

For this reason, a mechanism that makes it possible to automatically identify changes at a higher level of abstraction becomes necessary. In our approach, this mechanism is provided by the SPARQL RDF query language [7]. SPARQL queries are made up of a number of patterns that are intended to be matched to the statements in a model. When a pattern matches a statement, variables in the pattern are instantiated with the corresponding values from the matching statement. A result is returned for every possible combination of statements from the model that matches all patterns in the query.

Although a detailed description of SPARQL is outside the scope of this paper, Figure 3 shows three queries that are capable of identifying basic changes in a process model comparison (using a schema similar to that illustrated in Figure 2.) The first query identifies added activities, the second finds renamed products and returns their old and new names, and the third identifies activities with new involved roles, and displays the role names. These examples can give the reader an idea of the level of conciseness with which different change types can be described.

4.5 Technical Aspects

Our current implementation of model comparison was originally designed to work on large software process models. We have already accumulated extensive experience in working with the German V-Modell XT [8] and its variants. For example, [9] shows the results of a study we performed using our tools that analyzes the changes that happened to the V-Modell XT over a history of 600 versions. Converted to RDF, a recent version of the V-Modell contains over 13.000 statements that describe over 2000 different entities and their relationships. Many useful comparison queries on models of this size (identification of specific changes as illustrated above) run in under 5 seconds on a modern PC.

The current system is written in the Python programming language, and relies on the MySQL database management system for storage and retrieval. Queries in the SPARQL language are translated into SQL queries (based on an ad-hoc database schema that stores RDF triples in "triplestore" style) and executed in the database. The results from the database are then postprocessed to produce SPARQL results.

The database format is specially optimized for comparison. When initially read into the system, models are indexed in such a way that comparisons are more efficient in the future. Also, statements common to many versions are stored only once for space efficiency. Comparison models are then directly calculated by the database and cached for future use, if necessary. We have worked with databases containing relatively large numbers of model versions (over 600 for the case presented in [9]) with reasonable time and space performance.

4.6 Limitations

In order to operate properly, the *Evolyzer* system requires model entities to have *unique* and *stable* identifiers. Identifiers are unique when, given an identifier, there is at most one entity in the model that has it. They are stable when entities are guaranteed to keep their identifiers if the model is changed (this could also be said to be a property of the editing tools and not of the model itself.) The lack of any of these properties leads to difficulties when converting models to RDF. In particular, the lack of stability would make the results of a statement-based comparison worthless, since it would be impossible to reliably identify the statements that speak about the same entity in two versions of a model.

We have found this limitation to be less problematic in practice than it may sound in theory. Since identifier uniqueness and stability are useful for a number of purposes lying beyond model comparison, many common model formats and their corresponding modeling tools actually support them. Currently, we have no experience with models for which this is not the case, but we think it would be possible to handle many of these cases by combining our approach with one that matches entities using heuristics, such as SiDiff [11].

Another limitation is related to the fact that certain types of changes may correspond to patterns that cannot be expressed as SPARQL queries (e.g., they have a recursive structure) or that, even if they can be expressed, lead to low-level queries that are very hard to optimize. Since we are dealing here with the intrinsic complexity of models and model changes, this is a general problem that will probably affect every single model comparison approach in one way or another. Proper optimization of model data retrieval remains an open research problem.

5. Towards a Compliance Management Implementation

We believe that the *Evolyzer* system, as described in the previous section, can be effectively used to support the compliance maintenance approach outlined in Section 3. In the following, we go through the steps of the compliance maintenance approach and explain why we think that they are properly covered.

The first two steps (manual compliance assessment and formalization of the compliance relationships) can be seen as a single one for support purposes. We envision a user interface (based on the *Evolyzer* system) that makes it possible for assessors to browse both the standard and the assessed artifact, and add compliance relationships connecting them as necessary. In order for this interface to be viable, it must be possible to store the standard, the assessed model, and their compliance relationships within *Evolyzer*. Although most of our work so far has concentrated on the V-Modell XT, the fact that *Evolyzer* uses RDF as its internal representation makes it schema-neutral (see Section 4.2). *Evolyzer* has been used successfully to store models based on a variety of schemata, ranging from UML class diagrams to industrial models for the car manufacturing industry. This leads us to believe that we can handle many practical standards and artifacts successfully. Furthermore, the compliance relationships can also be modeled naturally using RDF, and stored in the model database together with the models they refer to.[2]

Step 3 (comparison of two model versions) corresponds to the main purpose of the *Evolyzer* system. One potential problem in this step would be the lack of unique or stable entity identifiers in the models. As suggested in Section 4.6, this problem could be handled by matching nodes through other techniques, such as applying matching heuristics.

Comparison queries can be used to support step 4 (identify compliance relationships affected by changes). Concretely, queries can be written to identify changes that are directly connected to compliance relationships, for instance, "find all process products whose description was changed and that correspond to an artifact required by the standard" or "determine if any activities were deleted that used to fulfill a requirement from the standard". Notice that these queries involve elements from the concrete model, the standard, and compliance relationships.

It is also possible to use comparison queries to exclude changes that do not affect compliance. For example, some areas of a process model might not be related to compliance with a particular standard, and they could be excluded when maintaining compliance with that standard. This can be achieved by relying on existing compliance relationships: areas of the process (or particular changes) that cannot be tied to a compliance relationship can be safely excluded from analysis. It is important to point out that this type of identification of potential non-compliant regions also has some risks. In particular, assessors may "overtrust" the system, and fail to notice problems because they are not immediately evident from direct change analysis. We must evaluate our approach empirically to find practical ways to mitigate this risk.

The final step (updating the model and/or compliance relationships to reinstate compliance) is mainly a manual step. Still, model comparison can be used to make sure that no changes fall outside the scope of the compliance related activities.

We are currently working on an initial implementation of these ideas, and expect to have a prototype available for demonstration at the time this workshop takes place.

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. Most of them concentrate on UML models representing diverse aspects of software systems. Coral [10], SiDiff [11], UMLDiff [12] and the approach discussed in [13], deal with the comparison of UML models. Although their basic comparison algorithms are applicable to our work, they are not concerned with providing analysis or visualization for specific uses. Additionally, FUJABA [14] manages model versions by logging the changes made to a model during editing, but is not able to compare arbitrary model instances. Models must also be edited with the FUJABA tool in order to obtain any useful change information.

Mens [15] 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 analysis. The Delta Ontology [16] provides a set of basic formal definitions related to the comparison of RDF graphs. SemVersion [17] and the approach discussed by [18] are 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 (such as process improvement).

[2] For our experience described in [3], we used a standard relational schema to describe compliance relationships, but RDF would be equally adequate for this task.

Regarding compliance, a number of existing commercial tools support traceability in the software realm, particularly with respect to requirements. One example is the well-known Doors tool [19] for requirements management. We are not aware of any approaches that combine model comparison and traceability in the way we are proposing in this paper.

7. Summary

Showing that artifacts (such as software processes) comply with a given standard is an expensive, labor-intensive task. For this reason, companies often tend to keep already compliant process descriptions unchanged, missing valuable improvement opportunities that could reduce risks and costs and significantly increase software product quality. In this paper, we proposed an approach that may strongly reduce the amount of work—and, consequently, the cost—necessary to keep a model compliant. This approach is based on comparing model versions in order to isolate changes, and then focuses on how these changes may have broken compliance. Since, in most cases, the size of changes will be significantly smaller than the size of the complete model, we think that this approach has the potential of saving a significant amount of work, and thus radically reducing compliance maintenance costs.

In order to test these assumptions, we are now making progress towards a practical implementation of the ideas exposed here. We are currently building a working prototype that we plan to present at the CVSM08 workshop.

8. Acknowledgments

We sincerely thank our anonymous reviewers for their valuable, constructive criticism. We did our best to address the issues they pointed out. We would also like to thank Sonnhild Namingha from Fraunhofer IESE for the final proofreading of this paper.

This work was supported in part by SoftDiff, a project financed by the Fraunhofer Challenge Program.

9. References

[1] Software Engineering Institute (SEI): Capability Maturity Model Integration (CMMI). Available from http://www.-sei.cmu.edu/cmmi/general/index.html (2006) (last checked 2008-01-24)

[2] International Standards Organization (ISO): ISO/IEC 15504-1: Information technology—Process assessment—Part 1: Concepts and vocabulary (2004)

[3] 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, Norway (2005)

[4] Soto, M., Münch, J.: Process Model Difference Analysis for Supporting Process Evolution. In: Proceedings of the 13th European Conference in Software Process Improvement, EuroSPI 2006. Springer LNCS 4257 (2006)

[5] Soto, M., Münch, J.: Focused Identification of Process Model Changes. In: Proceedings of the International Conference on Software Process (ICSP 2007), Minneapolis, MN, USA, May 19-20, 2007. Springer-Verlag (2007).

[6] Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation, available from http://www.w3.org/TR/rdf-primer/ (2004) (last checked 2007-12-20)

[7] Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Working Draft, available from http://www.w3.org/TR/rdf-sparql-query/ (2006) (last checked 2006-10-22)

[8] V-Modell XT. Available from http://www.v-modell.iabg.de/ (last checked 2007-12-20).

[9] Soto, M., Ocampo, A., Münch, J.: The Secret Life of a Process Description: A Look Into the Evolution of a Large Process Model. Accepted for publication in the International Conference on Software Process (ICSP 2008), Leipzig, Germany (2008)

[10] Alanen, M., Porres, I.: Difference and Union of Models. In: Proceedings of the UML Conference, LNCS 2863 Produktlinien. Springer-Verlag (2003) 2-17.

[11] Kelter, Udo., Wehren, J., Niere, J.: A Generic Difference Algorithm for UML Models. German Software Engineering Conference 2005 (SE2005). (2005)

[12] Xing, Z. Stroulia, E.: UMLDiff: an algorithm for object-oriented design differencing. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA (2005).

[13] 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).

[14] The Fujaba Manual, available from http://wwwcs.uni-paderborn.de/cs/fujaba/. (last checked 2007-09-06)

[15] Mens, T.: A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering, Vol. 28, No. 5, (2002).

[16] Berners-Lee, T., Connolly D.: Delta: An Ontology for the Distribution of Differences Between RDF Graphs. MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). Online publication http://www.w3.org/DesignIssues/Diff (last checked 2006-03-30).

[17] 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).

[18] 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).

[19] Telelogic Doors. Information available from http://www.telelogic.com/Products/doors/doors/index.cfm. (last checked 2008-01-24)