Semantics-Based Change Impact Analysis for Heterogeneous Collections of Documents *

Serge Autexier
Safe and Secure Cognitive Systems
German Research Center for Artificial Intelligence (DFKI), Bremen, Germany
serge.autexier@dfki.de

Normen Müller
Comp. Science, Jacobs University, Bremen, DE
n.mueller@jacobs-university.de
BSgroup Technology Innovation, Zürich, CH
normen.mueller@bsgroup.ch

ABSTRACT

An overwhelming amount of documents is produced and changed every day in most areas of our everyday life, such as, for instance, business, education, research or administration. The documents are seldom isolated artifacts but are related to other documents. Therefore changing one document possibly requires adaptations to other documents. Although dedicated tools may provide some assistance when changing documents, they often ignore other documents or documents of a different type. To resolve that discontinuity, we present a framework that embraces existing document types and supports the declarative specification of semantic annotation and propagation rules inside and across documents of different types, and on which basis we define change impact analysis for heterogeneous collections of documents. The framework is implemented in the tool GMoC which can be used to semantically annotate collections of documents and to analyze the impacts of changes made in different documents of a collection.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Document Management*

General Terms

Theory,Management,Algorithms

Keywords

Document Collections, Document Management, Change Impact Analysis, Semantics, Graph Rewriting

*This work was funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grants Hu737/3-1 and KO2428/8-1 (project OMoC)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng2010, September 21–24, 2010, Manchester, United Kingdom.
Copyright 2010 ACM 978-1-4503-0231-9/10/09 ...$10.00.

1. INTRODUCTION

Recent analyses of the Association for Information and Image Management, the international authority on enterprise content management, indicate that our globalized information society produces, maintains, and publishes about 5 petabytes of documents a year. Thus, an overwhelming amount of documents is produced and changed every day in nearly all areas of our every day life, such as, for instance, in business, in education, in research or in administration. A non-exhaustive list of examples are filled and signed forms in administration, research reports, test documents, or lecture notes, slides and exercise sheets in education, as well as requirements, documentations and software artifacts in development processes. The documents are seldom isolated artifacts but are intentionally related and intertwined with other documents. Thus changing a document may require adaptations to other documents and due to the huge amount of documents, there is a need for a reliable and efficient system support. While dedicated authoring and maintenance tools ranging from simple text-editors to integrated development environments may provide some assistance when changing documents, they typically are restricted to single documents or documents of a specific type. To resolve that discontinuity, we present a framework that embraces existing document types, allows for the declarative specification of semantic annotation and propagation rules inside and across documents of different types, and on that basis define change impact analysis for heterogeneous collections of documents.

The enabling idea is to represent in a single graph all related structured documents together with that part of their intentional semantics necessary to analyze specific semantic properties of the documents, for instance consistency checks, as well as to analyze the impact of changes on these semantic properties. The whole framework is based on *document models* defining the syntax, semantics and impacts description languages for specific document types as well as graph transformations to obtain the semantic annotation and to propagate the effect of changes for documents of this type. For the interaction between documents of different types it builds on *interaction models* specifying graph transformations propagating semantic information over document boundaries. The framework is implemented in the prototype tool GMoC built on top of the graph rewriting tool GrGen, where annotation and propagation rules can be specified in the declarative GrGen syntax and used to semantically annotate collections of documents and to analyze the impact of changes on the whole the collection.

```
<guests>
  <person confirmed="true">
    <firstName>Serge</firstName>
    <lastName>Autexier</lastName>
    <email>serge.autexier@dfki.de</email>
  </person>
  <person confirmed="true">
    <firstName>Normen</firstName>
    <lastName>Müller</lastName>
    <email type="prv">normen.mueller@gmail.com</email>
  </person></guests>
```

```
<guests>
  <person confirmed="true">
    <firstName>Normen</firstName>
    <lastName>Müller</lastName>
    <email type="prv">normen.mueller@gmail.com</email>
  </person>
  <person confirmed="false">
    <firstName>Serge</firstName>
    <lastName>Autexier</lastName>
    <birthday>1/23/45</birthday>
    <email type="bus">serge.autexier@dfki.de</email>
  </person></guests>
```

Figure 1: Semantically Equivalent Guest Lists.

The paper is organized as follows: In Sec. 2 we define the central *semantic document impact (SDI) graphs* together with the document models and interaction models to define the semantics-based analysis framework. In the realization part in Sec. 3 we describe the GMoC tool implementing that framework and discuss applications in Sec. 4. Related work is reviewed in Sec. 5 before concluding in Sec. 6.

As a running example for this paper we consider a wedding planning scenario, where two types of documents occur. The first document represents the guest list and the second one the seating arrangement. The latter depends on the former with the condition of male and female guests being paired. Using this simple example, we explain the complex practice of verification of consistency and identification of ripple effects: if one guest cancels the invitation the respective change in the guest list ripples through to the seating arrangement breaking the consistency condition of guests being rotationally arranged by gender.

2. SEMANTICS-BASED ANALYSIS

The semantics document impact (SDI) graphs are designed around the following idea: An SDI graph comprises both the syntactic parts of a document, such as the actual files containing the guest list and the seating arrangement, which are, for instance, given in some XML format as shown in Fig. 1. Additionally, the graph contains a separated explicit representation of the intentional semantics of the file content, which can be similar to the actual syntax but also quite different; in general, this will be a qualitative representation of the semantic entities of these documents and the relationship among them. For instance, these are the guests from the guest list, links indicating which guests belong together, the seats and their arrangement and which guest is assigned to which seat. The idea of the semantic entities is that the semantic entity for a specific guest remains the same, even if its syntactic structure may change. For instance, the semantic entity of guest `Serge` remains the same even though the syntactic subtree is changed by updating the `confirmed` status.

That way each SDI graph by design has interesting properties, which can be methodologically exploited for the semantic analysis and change propagation: indeed, it can contain parts in the document subgraph, for which there exists no semantic counter-part, which can be exploited during the analysis to distinguish added from old parts. Conversely, the semantic graph may contain parts, which have no syntactic origin, that is they are *dangling semantics*. This can be exploited during the analysis to distinguish deleted parts of the semantics from preserved parts. The information about added and deleted parts in an SDI graph is the basis for propagating the effect of changes throughout the semantic graph structure. This exemplifies the benefit of the dual representation of the documents with their syntactic structure and their intentional semantics in separate subgraphs, and we call this the *explicit semantics method*.

An SDI graph thus will consist of the documents, the semantic information computed from the documents, and in addition of impact information on parts of the documents in a serializable format. In our running example, this can be inconsistency information, such as that a seat is occupied by a guest, who has declined the invitation. To distinguish the different parts we define SDI graphs as instances of *typed graphs*, which consist of nodes n which have exactly one *node type* from some given set of node types and of a set of partially ordered links l which have exactly one *link type* from some given set of link types.

DEFINITION 1 (TYPED (PRE-)GRAPHS). *Let $\mathbb{N}$ be a set of* **node types** *and $\mathbb{L}$ a set of* **link types**. *A $(\mathbb{N},\mathbb{L})$-***typed pre-graph** *is a triple $(N, L, \prec)$ where $N = \uplus_{\nu\in\mathbb{N}} N_\nu$ is the disjoint union of sets of nodes of type ν, $L = \uplus_{\lambda\in\mathbb{L}} L_\lambda$ the disjoint union of sets of links of link type λ and $\prec$ a partial order on L. A typed pre-graph is a* **typed graph** *if for each $n \to n' \in L_\lambda$ it holds $n, n' \in N$. The class of all $(\mathbb{N},\mathbb{L})$-typed pre-graphs (resp. graphs) is denoted by $\mathcal{G}^{\mathbb{L}}_{\mathbb{N}}$ (resp. $\hat{\mathcal{G}}^{\mathbb{L}}_{\mathbb{N}}$) or simply $\mathcal{G}$ (resp. $\hat{\mathcal{G}}$) if $\mathbb{N}$ and $\mathbb{L}$ are clear from the context. Let $\mathbb{N}' \subseteq \mathbb{N}$ and $\mathbb{L}' \subseteq \mathbb{L}$ and $g = (N, L, \prec)$ be a $(\mathbb{N},\mathbb{L})$-typed pre-graph. Then the projection g on $\mathbb{N}'$ and $\mathbb{L}'$ is defined by*

$$g_{|\mathbb{N}',\mathbb{L}'} = (N, L, \prec)_{|\mathbb{N}',\mathbb{L}'} := (\uplus_{\nu\in\mathbb{N}'} N_\nu, \uplus_{\lambda\in\mathbb{L}'} L_\lambda, \prec_{|\uplus_{\lambda\in\mathbb{L}'} L_\lambda})$$

where $\prec_{|E} := \{l \prec l' \mid l, l' \in E\}$, for every set of links E. We denote empty typed (pre-)graphs by $\emptyset$. A path in a pre-graph is a non-empty list of links $n_0 \to n_1 \to \ldots \to n_k$ from L.

In the following we deal with collections of documents which are part of a whole graph, but that are actually collections of trees; for this we introduce the notion of collections of typed trees.

DEFINITION 2 (TYPED TREES). *A typed graph $(N, L, \prec)$ is a* **typed tree** *if there exists a unique node r such that for every node n different from r there exists exactly one path from r to n, no path from r to r, and $\prec_{|\{n\to n'\in L\}}$ is a total order, for every node $n \in N$. We denote r as the* **root node** *of the typed tree. A typed graph g is a* **collection of typed trees** *if there exists a partitioning of g into non-overlapping typed graphs $g_1, \ldots, g_n$ (i.e. $g = g_1 \uplus \ldots \uplus g_n$) and each g_i is a typed tree. The root nodes of a collection of typed trees is the set of root nodes of each typed tree g_i.*

Now we can distinguish the three parts using specific node and link types for each type of subgraph: We introduce *syntactic node and link types* $(\mathbb{N}_{syn}, \mathbb{L}_{syn})$ for the document

parts and *semantic node and link types* $(\mathbb{N}_{sem}, \mathbb{L}_{sem})$ for the semantic parts. For the impact information parts we use *impact node and link types* $(\mathbb{N}_{imp}, \mathbb{L}_{imp})$.

An entire *semantic document impact graph (SDI)* g is a typed graph with respect to all these node types and link types: its *document subgraph* is the projection of g to the syntactic node and link types and must be a typed graph; the *semantics subgraph* is the projection of g to the semantic node and link types and must also be a typed graph; finally, the *impacts subgraph* is obtained by projecting g on the impact node and link types which must not form a typed graph on its own but all links must connect one node from the impact graph to a node from the document graph.

Definition 3 (SDI Graph). *Let* $\mathbb{N} = \mathbb{N}_{syn} \uplus \mathbb{N}_{sem} \uplus \mathbb{N}_{imp}$ *be the disjoint union of syntactic, semantic and impact node types and* $\mathbb{L} = \mathbb{L}_{syn} \uplus \mathbb{L}_{sem} \uplus \mathbb{L}_{imp}$ *the disjoint union of syntactic, semantic and impact link types. A* $(\mathbb{N}, \mathbb{L})$*-typed document graph* $g = (N, L, \prec)$ *is a* **semantic document impact graph** *if, and only if,*

(1) the **document subgraph** $g_{|\mathbb{N}_{syn}, \mathbb{L}_{syn}}$ *is a collection of* $(\mathbb{N}_{syn}, \mathbb{L}_{syn})$*-typed trees;*

(2) the **semantics subgraph** $g_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}}$ *is* $(\mathbb{N}_{sem}, \mathbb{L}_{sem})$*-typed graph;*

(3) the **impacts subgraph** $g_{|\mathbb{N}_{imp}, \mathbb{L}_{imp}}$ *is* $(\mathbb{N}_{imp}, \mathbb{L}_{imp})$*-typed pre-graph and for all* $n \to n' \in L_{\mathbb{L}_{imp}}$ *it holds:* $n \in N_{\mathbb{N}_{imp}}$ *and* $n' \in N_{\mathbb{N}_{syn}}$ *(or vice-versa)*

We agree on the following notation for SDI graphs that makes the three parts document graph D, semantic graph S and impact graph I explicit: $g = \langle D, S, I \rangle$.

In our running example, a document subgraph represents the tree-structured syntactical content of the documents. The semantics graph holds information, for example, of the individual guests and tables and if they have been added, deleted or maintained. Moreover, it links guests and tables and left and right neighbors of guests. The impacts subgraph marks the ripple effects resulting from modifications, for example, on a guest's status.

Graph Transformations: Given a graph, we want to denote transformations of this graph into a new graph. In general, given a typed graph g a graph transformation results in a new typed graph and given some node and link types $\mathbb{N}$ and $\mathbb{L}$, a graph transformation τ is a total function $\mathcal{G}^{\mathbb{L}}_{\mathbb{N}} \to \mathcal{G}^{\mathbb{L}}_{\mathbb{N}}$. However, we want to distinguish different kinds of graph transformation with respect to how they affect the document graph, the semantics graph and the impacts graph of an SDI graph. We consider the following two basic graph transformations:

Semantic Annotation is a graph transformation starting with an SDI graph with empty impacts subgraph and returns a graph with an updated semantics subgraph, possibly some updated document graph and an impacts subgraph. We will also write 'annotation' instead of 'semantic annotation', if the context is clear.

Change Impact Analysis is a graph transformation starting with an SDI graph with empty impact subgraph, applies a *patch graph transformation* which only affects the document graph, followed by a semantic annotation.

In order to meet the real world, we have to refine these two basic graph transformations for the application context, where we want to treat collections of documents, which are each of a specific document type. For the refinement we have to answer the following questions:
(1) Where does a patch transformation come from?
(2) Where does the annotation transformation come from?
For the patches we will rely on a generic tree patch mechanism and tree difference analysis algorithm, that can take some semantic properties of the documents into account. For example, a stronger notion of equality leads to more compact, less intrusive edit scripts[1] which allows to ignore semantically insignificant differences. The semantic properties are specific to *document types* which will be described as *equivalence specifications* in *document models*. For a given collection of document trees we assign specific document models to each document tree. In order to determine the changes between two versions of collections of document trees we use the individual equivalence specifications to determine the changes between associated document trees.

Definition 4. (Document Types and Equivalence Specifications) *A* **document type specification** $\mathbb{D}$ *is a triple* $(\mathbb{N}_{syn}, \mathbb{L}_{syn}, P)$*, where* P *is a predicate on* $\hat{\mathcal{G}}^{\mathbb{L}_{syn}}_{\mathbb{N}_{syn}}$ *which specifies the set of syntactically valid documents*

$$\mathcal{D}_{\mathbb{D}} := \{d \in \hat{\mathcal{G}}^{\mathbb{L}_{syn}}_{\mathbb{N}_{syn}} \mid d \text{ is a tree and } P(d) \text{ holds}\}$$

Two document type specifications are **disjoint** *if their respective node and link types are pairwise disjoint. An* **equivalence specification** $\approx_{\mathbb{D}}$ *for* $\mathbb{D}$ *is a congruence on* $\mathcal{D}_{\mathbb{D}}$*.*

A document type specification characterizes the syntactically correct documents and the equivalence specification defines equivalence classes on their subparts.

The intentional semantics of documents of a specific type is represented explicitly in semantics graphs with specific node and link types and defined in *semantic models*. Furthermore, it defines the *impact* nodes and links to annotate the syntactic parts of a semantic document impact graph.

Definition 5 (Semantic Model). *Let* $\mathbb{D} = (\mathbb{N}_{syn}, \mathbb{L}_{syn}, P)$ *be a document type specification, then* $\mathbb{S} = (\mathbb{D}, (\mathbb{N}_{sem}, \mathbb{L}_{sem}), (\mathbb{N}_{imp}, \mathbb{L}_{imp}))$ *is a* **semantic model for** $\mathbb{D}$ *if* $\mathbb{N}_{syn}$*,* $\mathbb{N}_{sem}$ *and* $\mathbb{N}_{imp}$ *are pairwise disjoint node types and* $\mathbb{L}_{syn}$*,* $\mathbb{L}_{sem}$*, and* $\mathbb{L}_{imp}$ *are pairwise disjoint link types. Two semantic models are* **disjoint** *if their document type specifications, semantic node types and link types as well as impact node types and link types are pairwise disjoint.*

For our running example, the semantic model for guest lists contains a node type for individual persons, the model for the seating contains node types for individual tables and seats and link types for identification of which seat belongs to which table and the neighborhood relationship of seats.

To compute the semantic annotation we use graph rewriting rules that operate on SDI graphs. Like the document type specific equivalence specifications these rewriting rules are document type specific and are also part of the document model. The overall semantic annotation mechanism is entirely parametric in these document type specific graph rewriting rules. In our running example, these are, for instance, rules that lift syntactic entries in the guest list and seating to semantic objects in the semantic model and stores the origin of semantic nodes. However, we cannot yet check

[1] An edit script between a tree T and a tree T' is a sequence of edit operations turning T into T'.

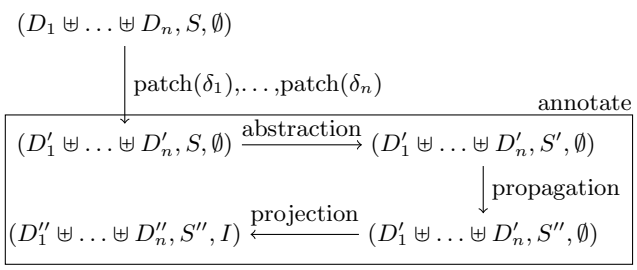

Figure 2: Change Impact Analysis in Detail

the seating arrangement, as we have no links from persons to seats in the semantic graph. Indeed, so far, semantic models only allow us to semantically annotate single documents, but not to semantically annotate across document boundaries. For this we introduce *interaction models* for a set of document models that specify annotation systems that operate between such documents. In our running example, these are graph rewrite rules that lift the declarative assignment of guests to seats given in the documents to semantic links between the respective individual persons and seats obtained before by the document model annotation rules and checks consistency of the seating arrangements. This in turn makes it unclear how to combine the different annotation graph transformations: indeed, some information propagated across document boundaries needs to be propagate further inside these documents, whereas some information needs to be first propagated inside the documents before the information can be propagate further to other documents using the propagation information from an interaction model. Thus, the introduction of cross-document interaction models requires the specification how the different graph transformations must be orchestrated.

In order to do this, we have a closer look at how semantic annotation actually proceeds: First, it gets a syntactic document and synchronizes it with a corresponding semantic graph, on which the impact analysis is then performed. During the analysis phase, the syntax does not change and until the analysis is completed on the semantic graph, it does not make sense to annotate the impacts on the syntax. This observation also allows for a better orchestration of annotation models and interaction models defined in a modular manner for different document kinds: First, *all* syntactic documents are analyzed in parallel to synchronize with the semantic graphs. Then the impact analysis is applied for *all* document and interaction models exhaustively and exclusively on the semantic graphs. Only then the documents are syntactically annotated with the impact information.

We now turn this observation into a methodology and subdivide the annotation graph transformations methodologically into three phases, namely (i) an *abstraction* phase which synchronizes the semantic graph with the (new) document trees, (ii) a *propagation* phase which propagates the information inside the semantic graph only, and (iii) a *projection* phase which dumps the information from the semantic graph into the document trees and the impact graph. The overall change impact analysis can be depicted as shown in Fig. 2, where the δ_i are the patches of the individual document trees.

To describe the three phases of the annotation process we introduce the concept of *annotation models* as follows:

Definition 6 (Annotation Model). *Let* $\mathbb{S} = (\mathbb{D}, (\mathbb{N}_{sem}, \mathbb{L}_{sem}), (\mathbb{N}_{imp}, \mathbb{L}_{imp}))$ *be a semantic model, then an* **annotation model** $\mathbb{R}$ **for** $\mathbb{S}$ *is a triple* $(\alpha_{\mathbb{S}}, \pi_{\mathbb{S}}, \iota_{\mathbb{S}})$ *of graph transformations of the following form:*

Abstraction: $\alpha_{\mathbb{S}} : \mathcal{P}(\mathcal{D}_{\mathbb{D}}) \times \hat{\mathcal{G}}^{\mathbb{L}_{sem}}_{\mathbb{N}_{sem}} \to \hat{\mathcal{G}}^{\mathbb{L}_{sem}}_{\mathbb{N}_{sem}}$ *is a mapping synchronizing the semantic graph with the document graph. Its homomorphic extension* $\alpha^{\#}_{\mathbb{S}}$ *to SDI graphs is only applicable on semantic document impact graphs with empty impact graph and defined by:* $\alpha^{\#}_{\mathbb{S}}(D_1 \uplus \ldots \uplus D_n \uplus D', S, \emptyset) = (D_1 \uplus \ldots \uplus D_n \uplus D', S', \emptyset)$ *where* D' *contains no documents of type* $\mathbb{D}$ *(i.e.,* $D' \cap \mathcal{D}_{\mathbb{D}} = \emptyset$*) and* $S' := (S \setminus S_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}}) \cup \alpha_{\mathbb{S}}(D_1 \uplus \ldots \uplus D_n, S_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}})$ *and* $D_i \in \mathcal{D}_{\mathbb{D}}$.

Propagation: $\pi_{\mathbb{S}} : \hat{\mathcal{G}}^{\mathbb{L}_{sem}}_{\mathbb{N}_{sem}} \to \hat{\mathcal{G}}^{\mathbb{L}_{sem}}_{\mathbb{N}_{sem}}$ *propagates semantic information in the semantic graph. Its homomorphic extension* $\pi^{\#}_{\mathbb{S}}$ *to semantic document impact graphs is only applicable on semantic document impact graphs with empty impact graph and defined by:* $\pi^{\#}_{\mathbb{S}}(D, S, \emptyset) = (D, S', \emptyset)$ *where* $S' := (S \setminus S_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}}) \cup \pi_{\mathbb{S}}(S_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}})$

Projection: $\iota_{\mathbb{S}} : \mathcal{P}(\mathcal{D}_{\mathbb{D}}) \times \hat{\mathcal{G}}^{\mathbb{L}_{sem}}_{\mathbb{N}_{sem}} \to \mathcal{P}(\mathcal{D}_{\mathbb{D}} \times \mathcal{G}^{\mathbb{L}_{imp}}_{\mathbb{N}_{imp}})$ *propagates semantic information back into the document graph and builds up the impact graph for this document, that is if* $\iota_{\mathbb{S}}(D_1 \uplus \ldots \uplus D_n, S) = \{(D'_1, I'_1), \ldots, (D'_n, I'_n)\}$*, then for all* $1 \leq i \leq n$ *holds that* $D'_i \uplus I'_i$ *are typed graphs (not a pre-graph). Its homomorphic extension* $\iota^{\#}_{\mathbb{S}}$ *to semantic document impact graphs is defined by:* $\iota^{\#}_{\mathbb{S}}(D_1 \uplus \ldots \uplus D_n \uplus D', S, I) = (D'_1 \uplus \ldots \uplus D'_n \uplus D', S, I \uplus I'_1 \uplus \ldots \uplus I'_n)$ *where* $D_i \in \mathcal{D}_{\mathbb{D}}$, $D' \cap \mathcal{D}_{\mathbb{D}} = \emptyset$ *and* $\{(D'_1, I'_1), \ldots, (D'_n, I'_n)\} = \iota_{\mathbb{S}}(D_1 \uplus \ldots \uplus D_n, S_{|\mathbb{N}_{sem}, \mathbb{L}_{sem}})$.

We agree to denote by $\mathbb{R}_{\mathbb{S}}$ that the annotation model belongs to the semantic model $\mathbb{S}$. Furthermore, we will not distinguish the mappings of the annotation models and their homomorphic extensions.

Definition 7 (Document Model). *Let* $\mathbb{D}$ *be a document type specification,* $\approx_{\mathbb{D}}$ *an equivalence specification for* $\mathbb{D}$*,* $\mathbb{S}$ *a semantic model for* $\mathbb{D}$*, and* $\mathbb{R}$ *an annotation model for* $\mathbb{S}$*. Then* $\mathbb{M} := (\approx_{\mathbb{D}}, \mathbb{S}, \mathbb{R})$ *is a* **document model for** $\mathbb{D}$*. Two document models are* **disjoint** *if their respective semantic models are disjoint.*

We agree to denote by $\mathbb{M}_{\mathbb{D}}$ that the document model belongs to the document type specification $\mathbb{D}$.

2.1 Interaction Models

An interaction model intentionally defines the propagation of semantic information between documents of different types. Thus, an interaction model presupposes a set of document models and extends it by an additional propagation graph transformation between the different semantic graphs of the given document models, for instance, the creation of links assigning persons to seats and checking the seating arrangement in our running example. To do so, it may require additional types of semantic nodes and links, which are also specified in the interaction model.

Definition 8 (Interaction Model). *Let* $\mathbb{M}_1, \ldots, \mathbb{M}_n$ *be disjoint document models with respective semantic node and link types* $\mathbb{N}^i_{sem}$ *and* $\mathbb{L}^i_{sem}$*,* $1 \leq i \leq n$ *and let* $\mathbb{N}^0_{sem}$ *and* $\mathbb{L}^0_{sem}$ *be node types and link types such that* $\mathbb{N}^0_{sem} \cap \mathbb{N}^i_{sem} = \mathbb{L}^0_{sem} \cap \mathbb{L}^i_{sem} = \emptyset$*,* $1 \leq i \leq n$ *and let* $\mathbb{N}^*_{sem} = \bigcup_{i=0}^{n} \mathbb{N}^i_{sem}$ *and* $\mathbb{L}^*_{sem} = \bigcup_{i=0}^{n} \mathbb{L}^i_{sem}$*. Then an* **interaction model** $\mathbb{I}$ **between** $\mathbb{M}_1, \ldots, \mathbb{M}_n$ *is a pair* $((\mathbb{N}_{sem}, \mathbb{L}_{sem}), \pi)$ *where*

Propagation: *$\pi : \mathcal{G}^{\mathbb{N}^*_{sem}}_{\mathbb{L}^*_{sem}} \to \mathcal{G}^{\mathbb{N}^*_{sem}}_{\mathbb{L}^*_{sem}}$ propagates semantic information in the joint semantic graph of the different document models. Its homomorphic extension $\pi^\#$ to semantic document impact graphs is only applicable on semantic document impact graphs with empty impact graph and defined by $\pi^\#(D, S, \emptyset) = (D, S', \emptyset)$ where $S' := (S \setminus S_{|\mathbb{N}^*_{sem}, \mathbb{L}^*_{sem}}) \cup \pi(S_{|\mathbb{N}^*_{sem}, \mathbb{L}^*_{sem}})$*

As for the graph transformations of annotation models, we will not distinguish the propagation graph transformations of interaction models and their homomorphic extensions.

2.2 Combined Models

We now consider semantic document impact graphs composed from documents of different types. We assume one document model for each document type is given as well as one interaction model that specifies the interaction between these different document types.

DEFINITION 9. *Let $\mathbb{M}_i = (\approx_{\mathbb{D}_i}, \mathbb{S}_i, \mathbb{R}_i)$ be document models for $\mathbb{D}_i$, $1 \leq i \leq n$ and $\mathbb{I}$ be an interaction model for the $\mathbb{M}_i$. An* SDI graph *g is* compatible with the *$\mathbb{M}_i, 1 \leq i \leq n$* and the *$\mathbb{I}$ if it contains only node types and link types from the document models and the interaction model.*

The combined annotation graph transformation for the whole graph consist of (1) the abstraction graph transformations from each document model, followed by (2) an exhaustive application of the propagation graph transformations from the document models *and* the interaction model, and (3) a final phase where all projection functions from the document models are applied.

DEFINITION 10 ($\mathbb{M}$ AND $\mathbb{I}$ COMBINED). *Let $\mathbb{M}_i = (\approx_{\mathbb{D}_i}, \mathbb{S}_i, (\alpha_{\mathbb{S}_i}, \pi_{\mathbb{S}_i}, \iota_{\mathbb{S}_i})$ be document models for $\mathbb{D}_i$, $1 \leq i \leq n$ and $\mathbb{I} = ((\mathbb{N}_{sem}, \mathbb{L}_{sem}), \pi_{\mathbb{I}})$ be an interaction model for the $\mathbb{M}_i$ and let g be a compatible semantic document impact graph. Then the combined graph transformations are defined by:*

Abstraction: *The combined abstraction of some g is the application of the combined abstraction $\alpha := \alpha^\#_{\mathbb{S}_n} \circ \ldots \circ \alpha^\#_{\mathbb{S}_1}$.*

Propagation: *The combined propagation on some g is the exhaustive application of the intermediate combined propagation $\pi := \pi_{\mathbb{I}} \circ \pi^\#_{\mathbb{S}_n} \circ \ldots \circ \pi^\#_{\mathbb{S}_1}$ on g. I.e., we apply π on g until we reach a fix-point which is easily expressed using a fix point combinator Fix defined by $(Fix\ F) = F(Fix\ F)$ on $F = \lambda f.\lambda g.(if\ (g = \pi(g))\ g\ else\ f(\pi(g))$.*[2]

Projection: *The combined projection of some g is the application of the combined projection $\iota := \iota^\#_{\mathbb{S}_n} \circ \ldots \circ \iota^\#_{\mathbb{S}_1}(g)$.*

From the disjointness of the document models it follows that the order of the combinations of the abstraction and projections in the definition of the combined abstraction and projections is irrelevant.

2.3 Semantic Analysis

Based on the notion of combined document models, we can now precisely define the semantic annotation and change impact analysis for a heterogeneous collection of documents.

Assume a set of document models with associated interaction model and a collection of documents $D_1, \ldots, D_n$ each belonging to one of the document models. Let further be (α, π, ι) the respective graph transformations of the combined document and interaction model. The *semantic annotation of the document collection* consists of applying the three graph transformations on the SDI graph $(D_1 \uplus \ldots \uplus D_n, \emptyset, \emptyset)$ for the given collection of documents with empty semantics graph and empty impacts graph:

$$(D'_1 \uplus \ldots \uplus D'_n, S, I_1 \uplus \ldots \uplus I_n) := \iota \circ \pi \circ \alpha(D_1 \uplus \ldots \uplus D_n, \emptyset, \emptyset)$$

Here D'_i is the new version of document D_i and I_i contains further annotations for parts of this document, such as, for instance, consistency information, errors, and others.

Analogously, the *change impact analysis of the document collection* consists of applying the patches for each individual document, and followed by the application of the three graph transformations of the combined document and interaction model. Again, in the resulting graph $(D'_1 \uplus \ldots \uplus D'_n, S, I_1 \uplus \ldots \uplus I_n)$ the D'_i is the new version of document D_i after patch application and semantic annotation. The impacts graph I_i contains the annotations for parts of this document, such as, for instance, consistency information, errors, etc. obtained by ripple effects from *all* patches via the combined document and interaction model graph transformations.

3. REALIZATION

The semantics-based analysis framework has been realized in the prototype tool GMoC [1] that annotates and analyses the change impacts on collections of XML-documents. The tool is parametrized over document models and interaction models specified in a declarative syntax (cf. Sec. 3.3).

For the difference analysis we use a generic tree difference analysis algorithm (cf. Sec. 3.1) which is parametrized over an equivalence specification for XML-trees. For the graph transformations we use the graph rewriting tool GrGen [8], which has a declarative syntax to specify typed graphs as well as graph rewriting rules and strategies (cf. Sec. 3.2).

3.1 Semantic Difference Analysis

Before we can analyze impacts of changes, we have to identify them — just because everything is different, does not mean anything has changed. Most previous work in change detection has focused on computing differences between text-based (*aka.* flat) files [10] which cannot be generalized to handle XML because these methods do not understand the hierarchical structure information contained in such data sets. Fortunately, the increasing use of XML over the last years has motivated the development of many differencing tools capable of handling tree-structure documents. However, none of these tools considers *semantics* of XML documents and either work with completely ordered trees, e.g. [24, 25, 9, 30, 6, 4], or completely unordered trees, e.g. [31, 28]. A comprehensive analysis and evaluation of theses methods can be found in [23]. In addition, due to the neglect of the semantics, none of these are capable of efficiently calculating changes within XML documents by comparing two XML documents for *semantic equivalence*. An example scenario, where this is important is a web service that serves results of queries, and wants to cache query results so that duplicate queries use previously cached results instead of always accessing the underlying database. The senders of those queries may potentially be using a variety of tools to generate the queries, though, these tools may introduce trivial differences into the XML. The intent of the

[2]To wit: $(Fix\ F)\ g \to (F\ ((Fix\ F))\ g \to (if\ (g = \pi(g))\ g\ else\ (Fix\ F)(\pi(g))$

```
equivspec E for 𝔻 {
 element guests { constituents { unordered { person } }}}
 unordered element person { constituents { alternative {
   {unordered {firstName, lastName, email}}
   {unordered {email, birthday? }}}}
 element firstName { constituents { unordered { <TEXT> }}}
 element lastName { constituents { unordered { <TEXT> }}}
 unordered element email{constituents{unordered{<TEXT>}}}
 element birthday { constituents { unordered { <TEXT> }}}}
```

Figure 3: An Equiv. Spec. for the Guest List in Fig. 1.

queries may be identical, but the standard DOM equality [7] returns false even if the XML trees compared contain semantically equivalent, but trivially different queries. It is this class of equality that this finding addresses: given two distinct XML structures, can we decide if they convey "the same information".

Example 1. *To sharpen our intuition on equivalent XML fragments, let us assume the two* `guests` *records depicted in Fig. 1. Our focus in this scenario is on the identification of the invited guests. A guest is represented by an unordered* `person` *with the addition of whether this is committed or canceled (*`confirmed` *attribute). Furthermore, information such as first name (*`firstName`*), surname (*`lastName`*), and e-mail address is stored. For the latter the distinction is between private address (*`prv`*) and business address (*`bus`*) encoded within a (*`type`*) attribute whereas the type is defaulted to be private. Optionally the date of birth is represented in a* `birthday` *element. Over time, the entries in the guest list change. Thus, for example, in the bottom half of Fig. 1 the order of guests is permuted and for one* `person` *element the status of commitment and the e-mail address type changed. In addition, the date of birth has been registered. To identify guests we are not interested in the status regarding commitment or cancellation and we do not care if we send the invitation to the business address or home address. Therefore, we define the primary key of a guest as a combination of first name, last name and e-mail address, i.e. when comparing two* `person` *elements, changes in confirmed status or e-mail address type are negligible as well as the existence of birthday information. Existing XML differencing tools, however, would even with an adequate normalization consider the two guest records to be different. The change of the confirmation status makes the two XML fragments unequal although regarding our primary key definition for* `person` *elements those two are equal.*

With an equivalence specification $\approx$ at hand, however, we can determine if two XML documents are equivalent and decide if they convey "the same information", in particular: two XML documents are considered as "semantically" equal with respect to an equivalence specification $\approx$ if and only if both respective documents are in the same equivalence class. Architecturally, an equivalence specification splits up the vocabulary of an XML document into similarity groups with respect to specific constraints.

The EQ Syntax: Here we illustrate our declarative EQ syntax to formalize equivalence specifications. A detailed description of the syntax and the semantics can be found in [19]. An implementation is available at [20]. In general, an equivalence specification as depicted in Fig. 3 comprises a set of element specifications which altogether define the congruence relation $\approx$. An element is declared by the `element` keyword and identified by name. With respect to the XML specification all elements are ordered *per se*. By prefixing an element specification with the keyword `unordered` one can break through this property. The body of an element specification comprises either alternative constraints or a singleton constraint. An alternative is a sequence of disjoint constraints. With both one determines when two XML fragments are considered equivalent: two XML fragments are considered to be equivalent *iff* the specified singleton constraint holds or, in case of alternatives, if at least one constraint within the sequence holds. A constraint consists of optional restrictions on attributes or constituents. Both sets specify the element's attributes/children to consider during a comparison. Constituents can be distinguished to be considered order dependent or order independent. For the ordered case the specified list of constituents has to be pairwise equal. For the unordered case the set of constituents has to be equal. For both cases equality is with respect to the individual element constraints. Items marked with a question mark (`?`) are optional.

Recalling Example 1, the equivalence specification in Fig. 3 precisely denotes the desired primary key for identification of guests. However, in contrast to the introductory example, we now even specify two alternative primary keys for identification: a `person` element is either identified by its first name, last name and e-mail address, *or* the e-mail address and, if available, the date of birth. All remaining elements are considered to be equivalent *iff* their nested text fragments are equal. Note that the order of the constituents to take into account in the equivalence check of `person` elements does not matter. However, in the individual equivalence checks, for example, of `firstName` elements and `lastName` elements, respectively, we declare that the position matters. One may think of a compound primary key for `person` elements whereby in this case the key is interpreted as a set. Comparing the sequences of children of two `person` elements, however, the order of `firstName`, `lastName`, and `birthday` elements matters to `E`. Exactly this flexibility is covered by an equivalence specification. Hence, we achieved to identify both lists of persons to be considered equivalent, i.e. we expressed the fact that both sequences convey the same information modulo the equivalence specification `E`.

sdiff – A Semantic Differ: We employ the semantic-tree difference analysis algorithm *sdiff* defined in [19]. The complete code is available at [21]. The difference to standard tree difference analysis algorithms is, that it relies on an equivalence specification for subtrees to identify corresponding subtrees that need not be syntactically equal. The equivalence specification is a parameter to the algorithm. The advantage is that one can indicate on which basis syntactically different subtrees are identified, such as indicating primary and secondary key attributes or sub-elements. This allows for a more fine-tuned control compared to the heuristic approaches of standard tree difference analysis algorithms that rely on some built-in syntactic metric to measure the similarity of subtrees. A detailed comparison and a discussion on the complexity of *sdiff* can be found in [26].

At this point, we are interested not in the actual algorithm, but only the basic mode of operation. In general, the *sdiff* algorithm is a function $\delta_{\mathsf{XML}} : \approx \times \mathsf{XML} \times \mathsf{XML} \mapsto \Delta$ parametrized by an equivalence specification, a *source* XML document and a *target* XML document (modified version). The algorithm compares the contents of the two files and

```
enum Effect {none, local, implicit}
abstract node class Item {
 url          : string;
 effect       : Effect = Effect::none;
 changeTypes  : set<string>;
 activator    : string;  }
node class Elem extends Item { path : string; }
abstract directed edge class Dependency connect Item → Item;
edge class childOf extends Dependency connect Item → Item[+];
edge class dependsOn extends Dependency
 connect Item[+] → Item[+];
```

Figure 4: A Graph Meta Model `W`

```
using W;
rule markModified(v: Item) { modify { eval {
 v.effect =Effect::local;
 v.changeTypes=v.changeTypes | {"modified"}; } } }
rule rippleEffects {
 pattern { dep:Item −:dependsOn−> prov:Elem; }
 if {(dep.effect ==Effect::none && prov.effect==Effect::local);}
  modify { eval {
   dep.effect      = Effect::implicit;
   dep.changeTypes = prov.changeTypes;
   dep.activator   = prov.url; } } }
```

Figure 5: Graph Rewrite Rules for `W`.

produces a delta[3], a sequence of edit operations turning *source* into *target*. Applied on the two guest lists depicted in Fig. 1 and the corresponding equivalence specification illustrated in Fig. 3 the computed delta is:

```
sdiff(new EquivSpec(spec), source, target) == (
  update("/*[1]/*[1]/@confirmed", "false") ::
  update("/*[1]/*[1]/*[3]/ @type", "bus") ::
  append("/*[1]/*[1]", <birthday>1/23/45</birthday>)::Nil)
```

The *sdiff* differencing algorithm equipped with an identification of syntactically different but semantically equal sequences clearly generates less intrusive edit scripts, which lessens the rippling of effects and simplifies the tracking and understanding of changes.

3.2 Graph Transformation

Over the last few years, graph rewriting theory gained more and more of importance. There are currently various tools available all being theoretically sound, fast and easy to use. We chose GrGen due to its reliability and sophisticated declarative syntax for specifications of graph meta models, rewrite rules, and rule application tactics. In this section, we present the most important features of GrGen utilized by GMoC to annotate and analyze the change impacts on collections of XML-documents. A comprehensive description of all GrGen features and grammars can be found at [2].

Graph Meta Models. A GrGen meta model defines typed and directed multi-graphs with multiple inheritance on node and edge types. In addition these types can be equipped with typed attributes and in order to restrict the set of well-formed graphs, the user can give so called connection assertions. These key features of a meta model are exemplary shown in Fig. 4. We created a text file `W.gm` that contains the graph meta model for our guest list scenario. Therein we defined an `Effect` enumeration type comprising the enumeration items `none, local`, and `implicit`. Furthermore we declared two base types, `Item` for general XML items and `Dependency` for connecting XML elements. For the `Elem` type we use inheritance which works like inheritance in object oriented languages. Accordingly the `abstract` modifier on, for example, `Item` means that one cannot create a node of that precise type. The fact that certain edge types can only connect specific nodes is stated by respective connection assertions (`connect`). In particular these assertions allow for constraints on the number of outgoing and incoming edges.

Graph Rewrite Rules: A GrGen rule file refers to graph models (`using`) and specifies a set of rewrite rules (`rule`). The rule language covers `pattern`, `replace`, and `modify` specifications. The pattern matcher is able to perform plain isomorphic subgraph matching as well as homomorphic matching for a selectable set of nodes and edges. Matches can further be restricted by arithmetic and logical conditions on the attributes and types. Nested negative application conditions are supported, too. Attributes of graph elements can be re-evaluated (`eval`) during an application of a rule. We present two example graph rewrite rules with the names `markModified` and `rippleEffects` in Fig. 5.

Rule Application. The GrGen graph rewrite language offers an extensive set of useful graph operations, including recalculation of node and edge attributes and retyping of nodes and edges. The rules can be composed with logical (e.g. `&`) and iterative sequence control (e.g. `*`) to an extended graph rewrite sequences (XGRS) like, for example, **xgrs** <markModified(p) & rippleEffects*>. Those sequences can be executed within `replace`/ `modify` specifications as well as in the script language of GrGen.

Because GrGen works on pure text files, handling of large meta models and rule sets is easy. In addition GrGen enables powerful interactive and batched execution of the graph rewrite functionality. This includes commands for creation, deletion, input, and output of graphs, nodes, and edges as well as application of rewrite rules. These features made the integration with GMoC straightforward.

3.3 The GMoC Tool

The GMoC-tool employs the described semantic difference analysis from Sec. 3.1 and the GrGen-tool (Sec. 3.2) to implement the semantic annotation and change impact analysis for heterogeneous collections of documents as described in Sec. 2. The tool is available from [1] and implemented in Java and utilizing GrGen for graph rewriting as a subsystem via an internal interface. GMoC is based on declarations of document types along with document models as well as interaction models. For this we devised an XML language to specify *document models* by: (1) the name of the file containing *equivalence specification* (2) the file name of the GrGen graph meta model containing the node and link types of the semantic graph and the impact graph for documents of this type, (3) the file names of the GrGen rules for abstraction, propagation, and projection for this model, (4) the names of the main rules for abstraction, propagation and projection, (5) a symbolic name used to reference it, and (6) a list of file extensions that can be used to recognize files of that type. As an example consider the document model specification for our `guests` domain:

```
<DocumentModel name="Guests">
 <suffix name="gxml"/> <equivspec filepath="Guests.eq"/>
 <graphmodel filepath="Guests.gm"/>
 <rulesystems>
```

[3] *sdiff* has a generic output format but close to XUpdate [15].

```
  <abstraction top="guestAbs" filepath="GuestsAbstr.gri"/>
  <propagation top="guestProp" filepath="GuestsProp.gri"/>
  <projection top="guestProj" filepath="GuestsProj.gri"/>
 </rulesystems></DocumentModel>
```

The XML syntax to specify *interaction models* consists of a symbolic name for that model as well as: (1) the names of the document models ("partners") this interaction model is for, (2) the file name of the GrGen graph meta model containing the node and link types of the semantic graph used to represent the semantic relationships between the semantic graphs of the respective document models, (3) the name of GrGen rule files containing the propagation file for this interaction model, and (4) the name of the main propagation rule. As an example consider the interaction model specification for our `guests` and `seats` domain:

```
<InteractionModel name="GuestAndSeatingInteractionModel1">
 <partner name="Guests"/><partner name="Seats"/>
 <graphmodel filepath="Guests2Seats.gm"/>
 <rulesystems>
 <propagation top="gsProp" filepath="GuestsSeatsProp.gri"/>
</rulesystems></InteractionModel>
```

The XML document type definitions for document and interaction models are available at [1].

For the practical application the idea is that document models with the equivalence specifications and graph models and rules are written once for a class of documents, and similarly for the interaction models. When using GMoC for change impact analysis the author of the analyzed documents never has to look at these document model definition files. Moreover, we envision that in a company, a change impact analysis system engineer is responsible to develop and maintain the document models and interaction models for the documents occurring in the company. The other employees get the result of the analysis in form of annotations to their documents, which are part of *document plans* which specify document collections as a list of documents indexed by a symbolic name for referencing purpose. The individual documents are given in an XML format and are either referenced by their filename or can be in-lined. For each document we can specify its type using the symbolic name of the respective document model. If no type is specified, the suffix of the document filename is used to determine its document type. Upon loading a document plan and having determined the document types of the documents, the GMoC tool will select known interaction models between these document types. For fine tuning, a document plan can include directives to exclude specific interaction models using their symbolic names. This is only for convenience but useful, for instance, to experiment with different models or use different models in different application contexts. An example document plan is

```
<DocumentPlan>
 <Document id="guests" filename="guests.gxml"/>
 <Document id="seating" documentmodel="Seats"
           filename="seating.sxml"/>
 <exclude model="GuestAndSeatingInteractionModel1"/>
</DocumentPlan>
```

Since document plans also represent the output of GMoC, a document plan can either contain the new document versions or the individual patch descriptions in the XUpdate-format. In addition, we can specify impact information for each document in form of a list of annotations for parts of the documents referenced by XPaths. The complete DTD for document plans is also available at [1].

The *semantic annotation* procedure is implemented as follows: Given a collection of documents, these are all parsed into the one SDI graph, hence forming a disjoint set of *document subgraphs*. The link types here are simple `childOf` and `isAttribute` links, whereas the node types are the element types (labels) of the XML language (for instance, `guests`, `firstName`, ...). These are assembled along with the GrGen graph meta model declarations from the document model specifications to one large GrGen graph meta model file. Furthermore, from the document models and interaction models we assemble all GrGen rules into a single large rule file by memorizing all main rules for abstraction, propagation and projection declared in the document and interaction model specifications. From these we create the combined abstraction, propagation and projection rules using the XGRS syntax and exactly following the construction given in Def. 10. For the semantic annotation we then generate from the SDI graph a respective GrGen graph instance and include the combined XGR sequence. Finally, we add the GrGen command to emit the resulting graph to some temporary file. GrGen is invoked on that file and upon completion we incrementally adjust the SDI graph by reading the new graph from the temporary file. The single graph now contains the new document graphs, the corresponding impact graphs and the semantic graphs. Finally, the new document plan and the impact information is generated in the document plan format described above.

The *change impact analysis* procedure is implemented as follows: Given an old and a new document plan and having checked they are compatible to each other, we first semantically annotate the old document plan applying the above procedure. The internal SDI graph now contains the semantically annotated documents. The impact information is skipped, since we are interested only in the ripple effects when going from the old documents to the new documents. Next we compute for each pair of document from the old and the new document plan the semantic differences instantiated with the equivalence specification obtained from the document model. The resulting list of patches are applied to the document subgraphs of the current SDI graph. The resulting SDI graph has – like any SDI graph – the interesting properties resulting from the *explicit semantics method* (see Sec. 2). Thus, in the resulting SDI graph the abstraction rules can statically determine for the semantics subgraph which parts have been added and which parts have been deleted. This information is used by the propagation rules to ripple the effects of these semantic changes throughout the semantic subgraph. Finally, the computed ripple effects are projected into the documents and the impact subgraphs using the projection rules. Technically this is realized by calling the GrGen-tool on the patched SDI graph exactly as for the semantic annotation. Afterwards we generate the resulting document plan with the new documents, which may have been further changed by the graph rewriting rules, and the impact information.

Based on the implementation of these basic routines, the GMoC-tool provides the following high-level services: (i) computation of semantic differences between two document plans (ii) semantic annotation, (iii) change impact analysis, and (iv) semantic management of change:

Semantic management of change is a special form of change impact analysis. Tools with no change impact analysis support may utilize GMoC as a subsystem without hav-

ing to implement the actual analysis. Indeed, assume some e-teaching environment where students can submit their solution, say S, to exercise sheets. These are marked by the teacher in an extended internal format, say S_A, of these documents and the teacher may request revisions from the student. The revised solution S', which does not contain the teacher's marking and remarks made for himself, must be integrated into the e-teaching environment while preserving the still valid teacher's markings and remarks. In principle, this is a special form of a three-way difference problem, where we have an inclusion relation between S and S_A, and an edit script from S to S' which we want to apply to S_A, i.e. compute the push-out in the category theory sense. This is supported by the GMoC tool by semantically annotating the extended document (S_A). For the change impact analysis, the semantic differences between S and S' is computed and the known inclusion relationship between S and S_A is exploited to apply the changes on S_A and to obtain S'_A. Next the standard abstraction, propagation and projection loop on S'_A computes the effect of the changes while preserving as much of the original information. The resulting change impact information can then simply be read-out by the e-teaching environment to update its internal representation. Of course, this requires the e-teaching environment to fully trust the GMoC-tool, but avoids having to implement the change impact analysis procedures from scratch.

4. APPLICATIONS

Authoring. We have applied our framework on the *Subversion* book [5] ($\sim$ 7 MB in total). The manual is split into 17 documents represented in the DocBook format [27] which gives us a sophisticated structural markup of content. Equipped with a DocBook singleton interaction model we could identify inconsistencies as well as ripple effects long before the next revision. For example, we identified — with respect to the level of markup granularity — all the occurrences regarding the syntactical extension of externals definitions from version 1.4 to 1.5.

Formal Methods. We currently extend the existing tool Hets [18] for formal software verification with change impact analysis support. The Hets tool maintains the status of formal software specifications along with information about how required global properties have been decomposed into local conjectures and the status of the proofs for these conjectures. Thus we are in the change management setting described at the end of Sec. 3.3: the user specifies and changes the software specification (which corresponds to the solutions written by the student in our example) and the internal representation in Hets extends that with information about proof decomposition and proof status (which corresponds to the representation on which the teacher worked on in our example). Here GMoC is used to add change impact analysis to Hets exactly in the same style as described above. In the same area we have applied the GMoC to the change impact analysis for annotated C programs in the SAMS verification framework [17], and used it with source code developed in the SAMS project. Only a few simple rules where necessary to obtain the desired result, which confirmed that the principles of the explicit semantic method are well suited for this kind of operation and the division of analysis rules into document models and interaction models allows for a modular and compositional development of change impact analysis. On a more technical note, it also showed that the current prototype implementation of GMoC does not scale well for large files, which is due to the fact, that the interaction between GMoC, implemented in Java, and GrGen, based on .NET, is currently based on exchanging files. Thus, future work will consist of moving to an API based communication between GrGen and GMoC.

5. RELATED WORK

Change Impact Analysis: Modeling data, control, and component dependency relationships are useful ways to determine software change impacts within the set of source code. The basic impact analysis techniques to support these kinds of dependencies are data flow analysis [29], data dependency analysis [13], control flow analysis [16], program slicing [14], cross referencing, and browsing [3], and logic-based defects detection and reverse engineering algorithms [11]. All these approaches only consider documents referring to the same document type, but changing one document (e.g. a functional system specification) possibly requires adaptations to other documents of different type (e.g. the documentation).

Requirements Traceability [12] is concerned with tracing requirements over different levels of refinement. It is a key technology for software maintenance to determine which artifacts need to be adapted when requirements change. There is strong tool support for this, such as the DOORS system [22]. However, these systems have the limitation, that they only support their own document types. Any change in a software artifact referenced from DOORS cannot be traced automatically. Hence, there is also no support to synchronize the version control of the software artifacts and the versioning in DOORS (Baselines). While some of the support offered by DOORS could probably be mimicked with GMoC, we envision the actual contribution of GMoC to bridge the gap between DOORS and other tools used in the software development process, such as version control systems, software documentation tools, but also issue tracking systems.

6. CONCLUSION

We have presented a framework to model the annotation of semantic properties for heterogeneous collections of documents and to design change impact analysis procedures in a user-friendly declarative style. The key ingredients are (i) changes are determined using a generic semantic tree difference analysis parametrized over document type specific equivalence specifications, (ii) explicit representation of both the syntactic documents and their intentional semantics in a single graph (explicit semantics method), (iii) view of the semantic annotation process as a specific graph transformation process and its decomposition into the three phases abstraction, propagation and projection which allows one to combine different document types via interaction models. The framework has been implemented in the GMoC tool exactly following the principles of the framework, based on the graph rewriting tool GrGen. The primary application scenario for GMoC is to bridge the gap between existing tools supporting document type specific change impact analysis. However, it can also be applied to add change impact analysis support to existing systems. First experiments provide evidence that GMoC can indeed significantly help to identify and manage effects of changes in an environment of heterogeneous documents. Although an evaluation regarding scalability for

large collections of heterogeneous documents is missing, the possibility to parse only semantically relevant parts of the documents into the graph makes us confident that the approach scales.

The framework and its implementation provide good foundation for future research: First, to develop methods to prove properties about semantic annotation and interactions, either on paper, but ideally using automated proof support based on formal languages. Second, even if we assume individual graph rewriting rule systems do terminate, we are only sure that the combined abstraction and projections terminate as well. However, the combined propagation rule systems may well be non-terminating due to some ping-pong ripple effects. For this it would be highly desirable to have some automated termination analysis, which is only known for very restricted graph rewriting classes.

7. REFERENCES

[1] S. Autexier. The GMoC Tool for Generic Management of Change, seen April 2010. `http://www.informatik.uni-bremen.de/dfki-sks/omoc/gmoc.html`.

[2] J. Blomer and R. Geiß. The GrGen.NET User Manual. Technical report, Universität Karlsruhe (TH), Institut für Programmstrukturen und Datenorganisation, June 2008.

[3] S. A. Bohner. *A graph traceability approach for software change impact analysis*. PhD thesis, George Mason University, Fairfax, VA, USA, 1995.

[4] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, pages 41–52. IEEE Computer Society, 2002.

[5] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. Version Control with Subversion. `http://svnbook.red-bean.com/`.

[6] F. Curbera and D. Epstein. Fast difference and update of XML documents. In *XTech*, San Jose, 1999.

[7] Document object model, 2006. seen June.

[8] R. Geiß, G. V. Batz, D. Grund, S. Hack, and A. Szalkowski. GrGen: A fast SPO-based graph rewriting tool. In *Graph Transformations - ICGT 2006. Lecture Notes In Computer Science*, pages 383–397. Springer, 2006.

[9] C. Hoffmann and M. O'Donnell. Pattern Matching in Trees. *Journal of the ACM*, 29(1):68–95, 1982.

[10] J. W. Hunt and M. D. McIlroy. An Algorithm for Differential File Comparison. Technical Report CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.

[11] Y.-F. Hwang. *Detecting faults in chained-inference rules in information distribution systems*. PhD thesis, George Mason University, Fairfax, VA, USA, 1998.

[12] M. Jarke. Requirements tracing. *Communication of the ACM*, 41(12), 1998.

[13] J. Keables, K. Roberson, and A. von Mayrhauser. Data Flow Analysis and its Application to Software Maintenance. In *Proceedings of the Conference on Software Maintenance*, pages 335–347, Los Alamitos, CA., October 1988. IEEE CS Press.

[14] B. Korel and J. Laski. Dynamic slicing of computer programs. *The Journal of Systems and Software*, 13(3):187–195, 1990.

[15] A. Laux and L. Martin. XUpdate - XML Update Language. `http://xmldb-org.sourceforge.net`.

[16] J. P. Loyall and S. A. Mathisen. Using Dependence Analysis to Support the Software Maintenance Process. In *ICSM '93: Proceedings of the Conference on Software Maintenance*, pages 282–291, Washington, DC, USA, 1993. IEEE Computer Society.

[17] C. Lüth. Safety Componenet for Autonomous Mobile Robots, seen April 2010. `http://www.informatik.uni-bremen.de/dfki-sks/sams/index.en.html`.

[18] T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set Hets. In O. Grumberg and M. Huth, editors, *13th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume LNCS 4424, pages 519–522. Springer, 2007.

[19] N. Müller. *Change Management on Semi-Structured Documents*. PhD thesis, School of Engineering & Science, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany, 2010.

[20] N. Müller. SCAla UP!, seen February 2010. `http://code.google.com/p/scaup`.

[21] N. Müller. Scala XML Extensions, seen February 2010. `http://code.google.com/p/scalaxx`.

[22] rcm2 Ltd. DOORS - Dynamic Object-Oriented Requirements System. `http://www.rcm2.co.uk`.

[23] S. Rönnau, G. Philipp, and U. M. Borghoff. Efficient change control of xml documents. In U. M. Borghoff and B. Chidlovskii, editors, *ACM Symposium on Document Engineering*, pages 3–12. ACM, 2009.

[24] S. M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 6(6):184–186, 1977.

[25] K.-C. Tai. The Tree-to-Tree Correction Problem. *Journal of the ACM*, 26(3):422–433, 1979.

[26] M. Wagner. *Change-Oriented Architecture for Mathematical Authoring Assistance*. PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, 2010.

[27] N. Walsh. The DocBook Schema Version 5.0. `http://www.docbook.org/`.

[28] Y. Wang, D. J. DeWitt, and J. yi Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In U. Dayal, K. Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 519–530. IEEE Computer Society, 2003.

[29] L. J. White. A Firewall Concept for both Control-Flow and Data-Flow in Regression Integration Testing. *IEEE Trans. on Software Engineering*, pages 171–262, 1992.

[30] K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[31] K. Zhang, R. Statman, and D. Shasha. On the Editing Distance Between Unordered Labeled Trees. *Information Processing Letters*, 42(3):133–139, 1992.