Chapter 6

APPLYING GRAPH TRANSFORMATIONS TO DATABASE RE-ENGINEERING

J. H. JAHNKE A. ZÜNDORF

Universität Paderborn, FB 17, Softwaretechnik, 33095 Paderborn, Germany
E-mail: {jahnke,zuendorf}@uni-paderborn.de

This chapter reports on our approach to apply graph transformations (GTs) in the domain of *database re-engineering* (DRE). The described approach has been implemented and evaluated in the *Varlet* DRE environment. The *Varlet* environment supports analysis of legacy relational database systems, translation of the relational schema into an object-oriented conceptual schema and interactive enhancement of the resulting conceptual schema. We employ *Triple Graph Grammars* (TGG) to generate a customizable translation mechanism between legacy data models and the conceptual data model. Moreover, we specify and implement *schema redesign transformation (SRT) rules* which allow the enhancement of the target schema by fully exploiting higher-level, object-oriented concepts. At this, we benefit from GT-theory to prove major properties of SRT rules (e.g., losslessness).

Contents

6.1 Introduction

A crucial factor for the competitiveness of today's companies is an efficient infrastructure for information management. Such an infrastructure often demands to merge business data with engineering data from application domains like computer-aided design, computer-integrated manufacturing, or software engineering. Complex dependencies between data stored in different systems (network, hierarchical, relational, object-oriented [3]) might exist and should be maintained. There are various techniques to achieve the desired integration, e.g., gateways, federated database systems, database migration, etc. [22].

A prerequisite to use any of these techniques is to have a complete technical documentation of the information systems which are to be integrated. Unfortunately, this is missing in many industrial database applications that have evolved over several generations of programmers. Such so-called *legacy database applications* (LDA) impede the establishment of an integrated information management. Database re-engineering (DRE) aims to recover an up-to-date conceptual design of an LDA's static data structure. This can be a delicate problem, because the schema catalogs of older database systems only provide information about the physical (low-level) representation of data. For example, older relational database management systems (RDBMS) do not support the means to explicitly represent relationships between database tables (foreign-key constraints)[3] in their schema definition. Abstract modeling concepts like inheritance, aggregation, and n-ary associations cannot be expressed in the relational data model. Additionally, many physical schemas of LDAs comprise optimizations that make it even harder to grasp the real semantics of the data structure.

Hence, the first activity in the DRE process is to *analyze* all available sources of information about the LDA, in order to yield a semantically annotated database schema. The resulting annotated schema is translated into an equivalent object-oriented conceptual schema. Subsequently, the conceptual schema might be extended or used as the basis for further re-engineering activities. Both tasks, legacy schema analysis and conceptual translation, are considered to be *human-intensive* and *iterative* [1,5], i.e., they cannot be perfomed in a fully-automatic, batch-oriented process. The reason for this is that legacy systems vary with respect to many technical and non-technical parameters: they use various hard- and software platforms and comprise arcane coding concepts (*idiosyncracies* [6]).

Computer aided DRE tools have a great potential to reduce the complexity (and risk) of re-engineering large LDAs that comprise several hundred thousand

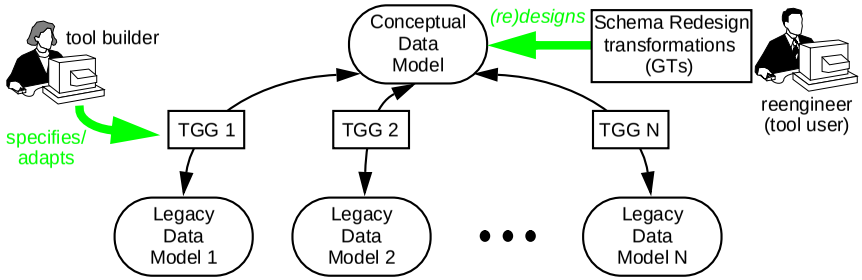

Figure 6.1: The *Varlet* Approach: Adaptable tool support for DRE

lines of code and maintain a vast amount of data. They can shorten project durations and achieve document consistency by unburdening re-engineers from laborious and error-prone tasks. However, the development of environments that support the described process is a demanding task. Currently available tools are insufficient because of the following two main reasons. (1) They lack *adaptability* [18]: in most DRE tools, analysis and translation are hard-coded in a general-purpose programming language and can hardly be customized due to specific characteristics of LDAs. (2) They impose phase-oriented, strictly *waterfall-like processes*. In our research, we aim to overcome these two problems. However, due to space limitations, this chapter focuses on the first problem, only. We refer the interested reader to [16] for more information on how we tackle the second problem (iteration).

Our approach is mainly based on concepts proposed in [20], i.e., we use *abstract syntax graphs* (ASG) and structure oriented editors specified with *graph transformations* (GTs) for building legacy and conceptual schema editors. In more detail, we employ *Triple Graph Grammars* (TGG) invented by [17] to map legacy data models to a canonical *conceptual data model* CDM and to keep these models consistent during incremental editing. In [11], we have developed a generator that creates a dedicated mapping tool based on a given TGG specification. This enables the adaption of our DRE environment to a specific legacy data model by simply customizing the TGG mapping description. After an initial (automatic) translation of the legacy schema into the CDM, the reengineer can use a predefined set of conceptual *schema redesign transformation (SRT) rules* to interactively enhance and extend the resulting conceptual schema. This approach is illustrated in Figure 6.1.

Major benefits of using GT rules in *Varlet* are their high level of abstraction and their rich theory. The following section introduces a small DRE sample scenario

that deals with a relational LDA, in order to illustrate the desired functionality of the tool. In Section 6.3, we outline the usage of TGG to specify mapping descriptions between legacy data models and the CDM. In Section 6.4, the sample scenario is revisited to introduce the idea of using SRT rules to further enhance and redesign the resulting conceptual schema. Subsequently, we show that SRT rules can formally be defined as parallel GT rules. Based on this formal definition, we employ GT theory to determine major properties of SRT rules, e.g, their impact on the information capacity of the transformed schema (Section 6.5). Finally, Section 6.6 summarizes our results and compares our approach with related work.

6.2 A Database Re-Engineering (DRE) Sample Scenario

The following sample scenario deals with a legacy product and document information system based on a RDBMS[1]. The analysis of the LDA starts with the extraction of its schema catalog. This reveals at least all existing tables together with their columns. Schema catalogs of newer RDBMS applications may additionally contain information on candidate keys and referential integrity constraints. Figure 6.2 shows a screenshot of the *Varlet Analysis Tool*, that displays a detail of the analyzed legacy schema. In this view, foreign keys are represented as directed edges between tables and attributes that belong to candidate keys are set in bold face.

As stated in the introduction, a fully-automatic schema extraction often fails to detect all relevant information and to recover high-level design concepts of a schema. Thus, the reengineer can add further domain knowledge and assumptions about the LDA. Based on this information, the analysis tool provides a means to semi-automatically retrieve further semantic information, e.g., by inspecting the application code, stored procedures, event handling procedures and by inspecting the available data itself. The analysis results are used to annotate the relational schema, e.g., the equal sign "=" in the triangles of the foreign key from table *Version* to *ProductSpec* (cf. Figure 6.2) denotes the information that the reengineer assumes an *inclusion dependency* (IND) over columns *doc_id* and *id* in both directions. Such a foreign key with an inverse IND imposes an additional cardinality constraint that requires that for every *id* in *ResearchReport* there exists a tuple in *Version* with the same value in column *docid* and vice-versa. Furthermore, in our example, the analysis reveals that there are two different variants of tuples in table *Version*, and that

[1] We presume that the reader is familiar with the basic terminology of relational databases [3]

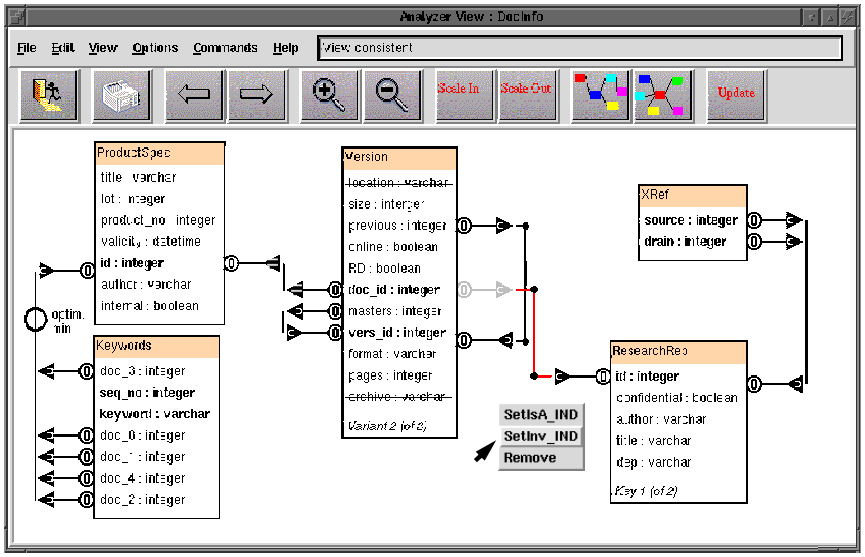

Figure 6.2: *Varlet* Environment, Relational Analysis Tool

the legacy schema comprises an optimization structure [6] for a many-to-many relationship between tables *Keywords* and *ProductSpec* that consists of five foreign keys (*doc0-4*). A detailed description on how this information is extracted is beyond the scope of this paper and can be found in other contributions [12, 14,7,2,6,21,26].

Once the relational schema is retrieved and annotated by our analysis tool, it is translated into an equivalent conceptual schema. This translation exploits all semantic annotations and employs the additional expressive power of the CDM as far as possible. Figure 6.3 shows a screenshot of the *Varlet (Re)Design Tool*, which displays the initial conceptual translation of our sample schema. The knowledge about the different variants of tuples in table *Version* was used to create an inheritance hierarchy for class *Version* with two new subclasses (*ArchiveVersion* and *OnlineVersion*[2]). The two occurring relational implementations of many-to-many relationships, i.e., the join table *Xref* and the optimization structure between tables *Keywords* and *ProductSpec* are mapped onto associations between the corresponding classes. Furthermore, knowledge about *not-null* constraints and inverse INDs in the relational schema is used to determine cardinality constraints of associations.

[2]Of course, reasonable names have to be chosen interactively.

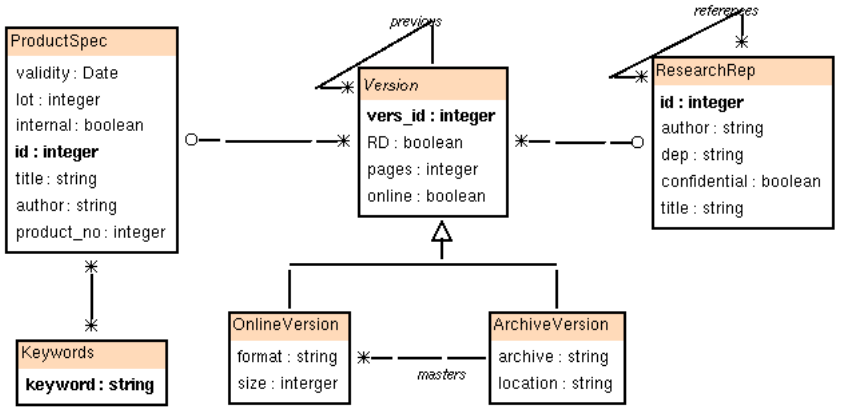

Figure 6.3: *Varlet* Environment, Conceptual (Re)Designer View

6.3 Mapping Data Models with Triple Graph Grammars

Translating legacy schemas to conceptual schemas is a central task in DRE. Within this task one frequently faces unforeseen situations like idiosyncracies and various kinds of optimization structures. Most currently existing DRE tools are not able to deal with such situations because their translation process is hard-coded using general-purpose programming languages.

In *Varlet*, the mapping between a legacy data model and the CDM is defined by a set of *schema mapping rules* defined by a TGG. This approach facilitates easy customization of our environment according to different application contexts. In general, a TGG specifies a bidirectional mapping between two document types (based on their ASG representation) (cf. [17]). Consequently, this enables us to generate schema mapping tools that support reverse engineering as well as forward engineering of database schemas [16]. In this chapter, we focus on the reverse engineering aspect, only.

Figure 6.4 presents a simple example of a TGG schema mapping rule that defines the correspondency between relational columns to attributes in the CDM. The solid parts (i.e., nodes *L1*, *M1*, *R1*, and the attached *map* edges) describe the so-called *context-part*, while the dashed parts (i.e., nodes *L2*, *M2*, *R2*, and the attached edges) describe the so-called *addition part*. A TGG rule represents a concise notation for a number of GTs that can be derived from it. For example, we can derive a GT gt_r that translates a relational column into an

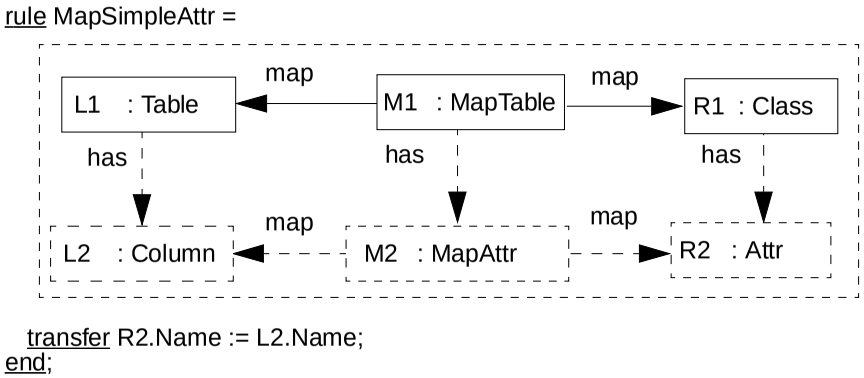

Figure 6.4: Sample TGG mapping rule *MapSimpleAttr*

attribute in the CDM, by taking the context-part and the *relational-part* (node *L2* and the attached *has* edge) as the left-hand side and the entire rule as the right-hand side of gt_r. Analogously, we can also derive a *forward engineering* GT gt_f that translates simple attributes to columns. We refer to [17,13,11] for a more detailed description of TGGs and their application to DRE.

6.4 Schema Redesign Transformation Rules

Due to the semantic gap between most legacy data models and our object-oriented CDM, the result of the initial conceptual schema translation often has a structure similar to the legacy schema. For instance, in the case of our legacy relational schema, this means that basically, tables become classes, columns become class attributes, and foreign keys are translated into associations. In order to exploit additional concepts of the CDM like aggregation (i.e., complex attributes) and inheritance, the initial conceptual translation of the legacy schema can interactively be restructured by using the *Varlet (Re)Design tool*. To achieve this, *Varlet* provides a predefined set of SRT rules.

We revisit our sample scenario to illustrate the application of SRT rules: Figure 6.5 shows a second (manually annotated) screenshot of the *Varlet (Re)Design tool* that displays a restructured version of the initial conceptual schema shown in Figure 6.3: It contains a new generalization *Document* for classes *ProductSpec* and *ResearchReport*, class *ProductSpec* has been split in two associated classes *ProductSpec* and *Product*, class *Version* has been aggregated into class

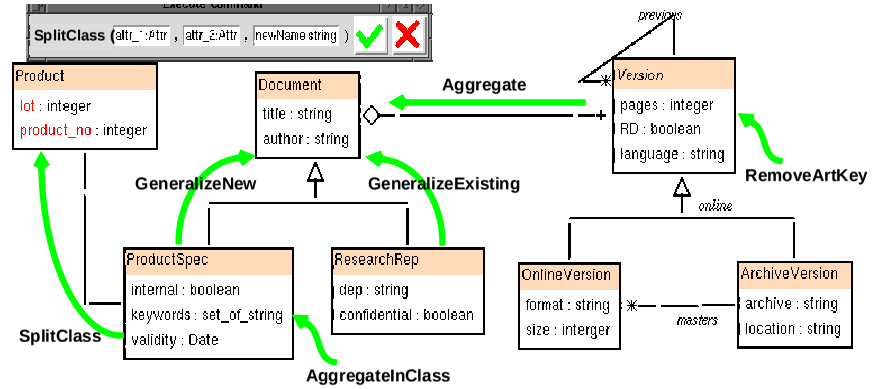

Figure 6.5: *Varlet* Environment, Redesigned Conceptual Schema

Document, class *Keywords* has been transformed into a complex attribute of class *ProductSpec*, and artificial keys have been removed.

In our approach, SRT rules are formally defined by GT rules based on the *double-pushout* theory [24]. That means, we employ *edge-object graphs* consisting of sets of node and edge identifiers, source and target functions for edges, and node and edge labeling functions. In addition, our nodes are attributed. Within this theory, we represent databases as graphs consisting of a database schema representation (upper part of Figure 6.6) and a representation of the database extension (lower part of Figure 6.6). We use nodes of type *Class* to represent classes. Associations are represented by two association stub nodes mutually connected via *assoc* edges. An association stub node represents the role that the corresponding class has in that association. We use nodes of type *1-RefT* to represent *exactly-one* roles, nodes of type *0-1-RefT* to represent *at-most* one roles, and nodes of type *SetRefT* to represent *many* roles. *AttrT* nodes represent the attributes of classes. Within our graph schema, we use type *ClassPropT* as super type for all kinds of class properties. All class properties are connected to their class via a *has* edge. All database schema nodes carry a *name* attribute. Nodes of type *Obj* represent objects in database extensions. Object properties are represented via *1-Ref* nodes (exactly-one links), *SetRef* nodes (many-links), etc. Each node in the database extension has exactly one *inst_of* edge connecting it to the corresponding schema element. For database graphs a large number of consistency constraints must hold that are formally defined in our graph schema. As one example, we demand that a *1-Ref* node has exactly one outgoing link edge connecting it to another link stub node.

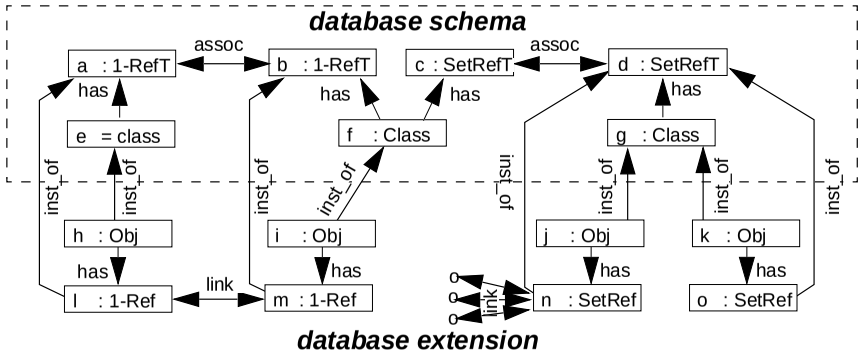

Figure 6.6: Graph Representation of a Database

(Note, *SetRef* nodes may have no or multiple outgoing link edges.) A formal definition of all constraints is given in [25].

Definition 6.4.1 (Partial Schema Redesign Transformation Rules)
A partial SRT rule $\bar{T}$ is defined by a pair of (double-pushout) GT rules $\bar{T} = (t, i)$ where t is called the *structure transformation rule* and i is the *instance mapping rule* of $\bar{T}$. The structure transformation rule represents a function $t : S^{\bar{T}} \longrightarrow S$ that is defined on the subset $S^{\bar{T}} \subset S$ of all possible schemas S, where each $s \in S^{\bar{T}}$ has to satisfy the precondition of $\bar{T}$, i.e., each $s \in S^{\bar{T}}$ has to contain a valid match for the left-hand side of t.

According to a given application $s' = t(s)$ of the structure transformation rule, the instance mapping rule $i : \mu(s) \longrightarrow \mu(s')$ converts database extensions of s to extensions of the target schema s'. At this, $\mu(x)$ denotes the *information capacity* of a schema x which is defined as the set of all valid database states (or instances) of x.

Remark:
Obviously, t must be contained in i, i.e., there exists the inclusion morphism $id : t \longrightarrow i$. Hence, we can define t as a subrule of i. Later on, t will serve as a subrule for the amalgamation of i to a parallel graph rewriting rule (cf. [27]) that transforms a graph schema together with its extension(s). In the following, we make use of this property by defining only one GT rule (the instance mapping rule i) and marking the contained structure transformation rule t. □

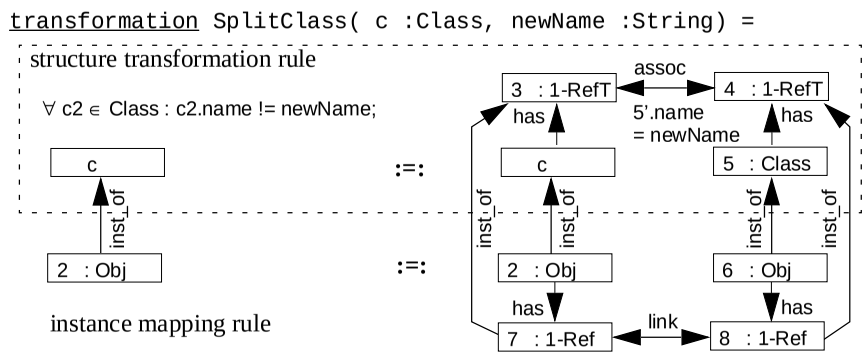

Figure 6.7: Schema Redesign Transformation *SplitClass*

Figure 6.7 shows an example SRT rule, which consists of the structure transformation rule (upper part of Figure 6.7) and the instance mapping rule (whole Figure 6.7). Note, we use a *Progres*-like notation (cf. [24]) to represent SRT rules, even though we employ the double-pushout approach. Thus, the figure shows only the left graph L and the right graph R of the represented graph rewriting rule. The glueing graph K consists just of the common parts of L and R. The morphisms $l : K \longrightarrow L$ and $r : K \longrightarrow R$ are just the obvious inclusions.

Definition 6.4.2 (Execution of Redesign Transformation Rules)
The parallel execution of a redesign transformation rule is defined by fully synchronized partial coverings [27], where the instance mapping rules are fully synchronized by the (contained) structure transformation rule. In addition we consider only injective matches for the structure transformation rule and each application of the instance mapping rule.

Informally, an SRT rule is executed by first finding a match for (the left-hand side) of its structure transformation rule. Subsequently, we look for all possible extensions of this structure rule match to matches for the instance mapping rule. Thereby, we construct an amalgamated graph rewriting rule. The execution of the resulting amalgamated graph rewriting rule executes the structure transformation only once, while the *instance mapping rule* is executed repeatedly for all instances of the matched schema elements.

For example the SRT rule *SplitClass* (Figure 6.7) is applied by first looking for a match for class c (which is actually passed as parameter). Subsequently, the

instance mapping matches all objects in the current database extension that comply to node 2, i.e., that are connected to the match of c via *inst_of* edges. The resulting amalgamated rule generates a new class 5 with name *newName* and with a 1:1 association to the original class. In addition, all instances of the original class are transformed to linked pairs of objects that comply to the new schema.

In order to be applicable, our sample redesign transformation rule *SplitClass* (Figure 6.7) has to fulfill the *parallel gluing condition* ([27], Def. 4.2.10). The parallel gluing condition requires that the amalgamated rule fulfills the *dangling edge* and the *identification condition*. At this, the dangling edge condition requires that the deletion of nodes must not create dangling edges and the identification condition demands that edges or nodes that match the same host graph element have to be preserved.

A redesign transformation rule aims to specify a modification of database schemas including the migration of its current extension. However, we want to provide general redesign transformations that apply to a database schema independent of its current extension. Thus, whenever the structure transformation rule part is applicable, we require that the instance mapping rule has to be able to migrate the data. Therefore, we introduce the following definition:

Definition 6.4.3 (General Schema Redesign Transformation Rules)
For a valid SRT rule T we require that for any database schema s and any of its extensions $\mu(s)$ the amalgamated rule must fulfill the parallel gluing condition. We call this condition the *general parallel gluing condition*. A redesign transformation rule that fulfills the general parallel glueing condition is called a *general schema redesign transformation rule*.

Remark:
The supposition that the parallel glueing condition has to be fulfilled for *all* possible amalgamations of a general SRT rule (Definition 6.4.3) ensures that the instance mapping is able to migrate the database extension, whenever the structure transformation is applicable. □

SplitClass is a general schema redesign transformation rule (according to Definition 6.4.3) since the rule does not delete any element and thus its amalgameted applications do not delete anything, too, and thereby they fulfill the required general parallel glueing condition, trivially.

Figure 6.8 shows another SRT rule: *MoveProperty* is used to transfer a given class property *prop* (attribute or reference) from class 3 to class 4 via a total 1:1 association. *MoveProperty* fulfills the general parallel glueing condition: each

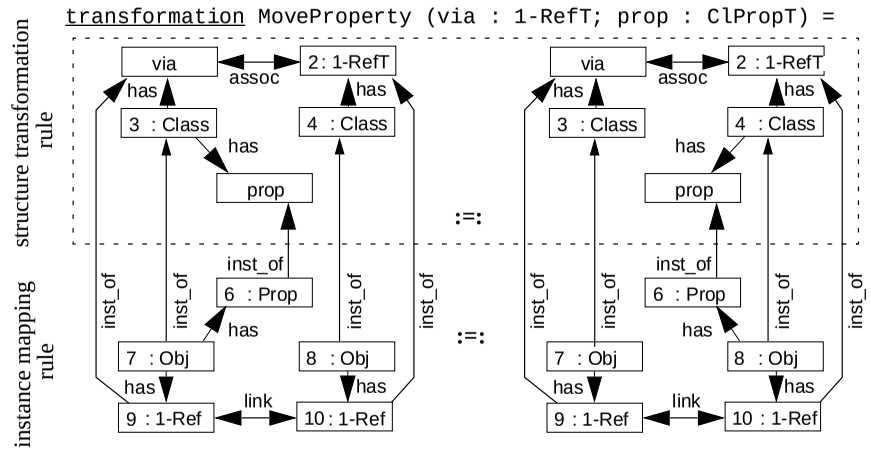

Figure 6.8: Preserving Redesign Transformation *MoveProperty*

match of its instance part applies to a different pair of database objects that are connected via a 1-1 link where one of the objects holds the property to be moved to the other. There is no way to overlap two different matches in their instance part. Thus, different matches do not interfere and the parallel glueing condition holds for all possible amalgamations.

6.5 Properties of Schema Redesign Transformation Rules

DRE aims on recovering conceptual designs that reflect the real semantics of LDAs. Thus, a major issue is to prove equivalency of the legacy schema with its resulting counterpart. One possibility to achieve this is to produce the resulting schema by applying a sequence of *information-preserving* transformations to the initial schema. The notion of information-preservation can be defined based on the notion of *losslessness* of SRT rules, cf. [9,3]. Being inspired by [10], we employ static criteria based on GT theory to prove losslessness of SRT rules. We need the following definitions and proprositions:

Definition 6.5.1 (Losslessness of SRT rules)
Let $T = (t, i)$ be a general SRT rule and $T^{-1} = (t^{-1}, i^{-1})$ be constructed by swapping the left- and right-hand sides of t and i.
T is called *lossless* (reversible) iff $\forall H, G$ with $H = T(G)$ holds $G = T^{-1}(H)$, i.e.,

T^{-1} is a total function on $T(\mu(s))$
$T^{-1}(T) = id_{\mu(S)}$

Remark:
The above definition requires that any extension $\delta \in \mu(s)$ of a schema s that has been transformed by T is built back by applying T^{-1} to the original instance. □

Definition 6.5.2 (Properties of SRT rules)
An SRT rule $T = (t, i)$ is called
information-augmenting if T is lossless,
information-reducing, if its inverse SRT rule T^{-1} is lossless,
information-preserving if T is symmetrically-lossless, i.e., if T and its inverse SRT rule T^{-1} are lossless.

Proposition 6.5.3 (Losslessness Criterion)
Let $T = (t, i)$ be an SRT.
Let sg be a subgraph of the right-hand side of i,
Let $T^{-1} = (t^{-1}, i^{-1})$ be constructed by swapping the left- and right-hand sides of t and i.
if $\forall H, G$ with $H = T(G)$ implies
1. H contains exactly one match for t^{-1},
2. $\forall$ matches m_1, m_2 of i^{-1} in H holds $m_1|_{sg} = m_2|_{sg} \Longrightarrow m_1 = m_2$,
3. $\forall$ matches m_{sg} of sg in H $\exists$ an "original" match m' of i^{-1} in H such that $m'|_{sg} = m_{sg}$
$\Longrightarrow$ T is lossless.

Proof
Figure 6.9 outlines an application of an SRT rule T to a graph G. t is matched to the subgraph m_t of G. There are two extensions $m1_i$ and $m2_i$ of m_t to matches for for i. The execution of T transforms G into graph H, where m_t is replaced by $m_{t^{-1}}$ and $m1_i$ is replaced by $m1_{i^{-1}}$ and $m2_i$ is replaced by $m2_{i^{-1}}$.

An amalgamated graph rewriting rule is a double-pushout rule, too. Thus, the amalgamated rule is reversible, i.e., the amalgamated rule applied to the match it created during the forward execution reconstructs the original graph, cf. [27].

In Figure 6.9 this is depicted by the three morphism arrows highlighting that $m_{t^{-1}}$ is a valid match for t^{-1} and that $m1_{i^{-1}}$ and $m2_{i^{-1}}$ are valid ("original") matches for i^{-1}. Applying T^{-1} to H at these matches would result exactly in the original Graph G.

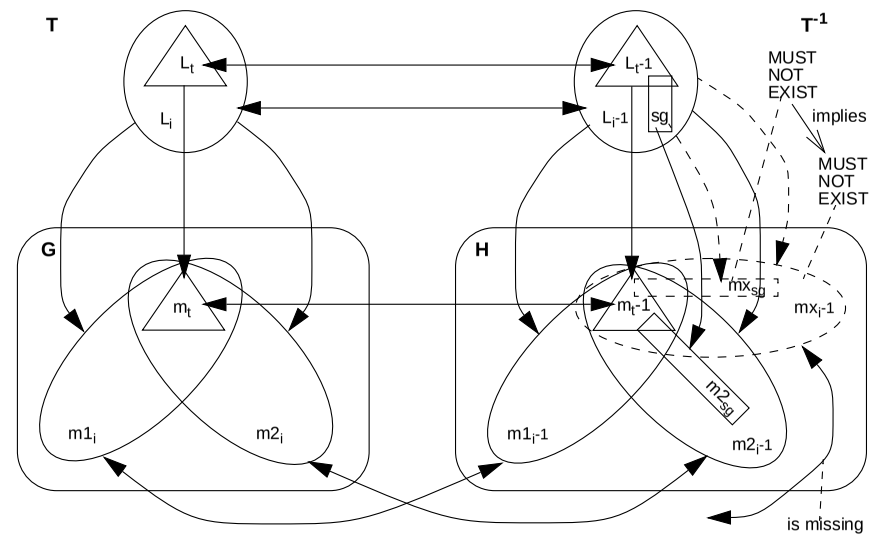

Figure 6.9: Losslessness Criterion motivation

However, applying T^{-1} to H we have to construct the amalgamation anew. During the (forward) application of T on G each excution of i has created an "original" match for i^{-1} (e.g. $m1_{i^{-1}}$ and $m2_{i^{-1}}$). Each of these original matches provides the "original" match for t^{-1} (i.e., $m_{t^{-1}}$ in our example). If our graph contains no other match $mx_{t^{-1}}$ for t^{-1} and no "foreign" match $mx_{i^{-1}}$ for i^{-1} the application of T^{-1} results in exactly the same amalgamation as the one used for the forward transformation.

Within Proposition 6.5.3 we employ a certain subgraph sg of i^{-1}. Condition 2 states that this sg determines any match of i^{-1}, uniquely. Thus, a match $m2_{sg}$ for sg identifies the corresponding match $m2_{i^{-1}}$ for i^{-1}. Provided with such a subgraph sg, Condition 3 requires that there is no match mx_{sg} for sg other than an *"original"* one.

This suffices to prove that there exists no "foreign" match $mx_{i^{-1}}$ for i^{-1}: Assume conditions 1 to 3 of Proposition 6.5.3 are fulfilled but there exists a "foreign" match $mx_{i^{-1}}$ for i^{-1}. Then $mx_{i^{-1}}$ contains a match mx_{sg} for sg, since sg is contained in i^{-1}. According to Condition 3, there exists an "original" match m' of i^{-1} that is determined by sg. According to Condition 2, m' equals $mx_{i^{-1}}$ which is a contradiction to the assumption that $mx_{i^{-1}}$ is a "foreign" match.

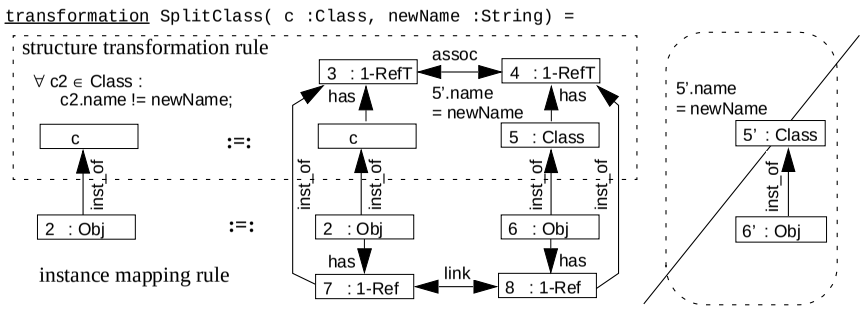

Figure 6.10: Losslessness post condition for *SplitClass*

Thus, the absence of a "foreign" match for sg suffices to prove the absence of a "foreign" match for i^{-1}. Together with Condition 1 this guarantees that the application of T^{-1} to H creates exactly the original amalgamation and its application yields the original graph G. This implies that T^{-1} is a total function on $T(\mu(S))$ and $T^{-1}(T) = id_{\mu(S)}$. □

Example 6.1 (Losslessness of *SplitClass*)

In the following example, we will exemplify the application of the losslessness citerion with the previously defined SRT rule *SplitClass*. As subgraph sg we choose nodes 5 and 6 of the right-hand side of *SplitClass* together with the connecting *inst_of* edge. This subgraph sg will serve as a *negative post-condition* for the amalgamated rule, cf. Figure 6.10, [10]. This negative post-condition requires:
after the execution of T it must not be possible to *extend* the resulting "original" match for T^{-1} by a match for the negative post-condition.

We will show that the chosen subgraph sg fulfills the three conditions of the criterion (and thus the negative post-condition is never violated):

Condition 1: Node 5 contains a name that identifies the match for node 5, uniquely. Since the corresponding class is just created, it can have only one property that is matched by node 4. Since 4 is a *1-RefT* node, its match has exactly one outgoing *assoc* edge determining the match for node 3, which in turn, belongs to exactly one *Class* nocd c. Thus, H contains only one match for t^{-1}.

Condition 2: Assume we have a match m_{sg} for sg. The match for node 5 is the new class that has been created by applying *SplitClass* to the original database. This just created class contains only one property, namely the corresponding

1-RefT association stub. This *1-RefT* node serves as the only possible match for node 4 of the right-hand side of *SplitClass* and the connecting has edge. In turn, the match for node 4 determines the match for the partner association stub 3 and the connecting *assoc* edges, uniquely. Similarily, the object matched by node 6 has only a single link stub attached that determines the match for node 8 and the connecting *has* edge. The match for node 8 determines the match for node 7, the connecting *link* edges, and the *inst_of* edges from 7 and 8 to 3 and 4, respectively. Thus, the match for node 5 and 6 determines the whole match.

Condition 3: Let us assume there exists a match m_{sg} for sg that has not been created by *SplitClass*. Class c_5 matched by m_{sg} has a unique name that did not exist before the execution of *SplitClass*, (cf. the precondition of *SplitClass*). Since c_5 is just created, the match for the *inst_of* edge connecting node 6 and node 5 must have been created by the same rule application. The match for node 6 must also have been created by this rule application, because *SplitClass* connects only just created objects to the new class c_5. Thus, the match for sg is part of an original match m', which contradicts to our assumption. Consequently, Condition 3 holds. Together, the three conditions prove that *SplitClass* is lossless. □

Example 6.2 (Losslessness of *SplitClass*$^{-1}$)

First we have to show, that *SplitClass*$^{-1}$ is a general SRT rule, i.e., that it fulfills the general parallel glueing condition (Definition 6.4.3). The structure rule t^{-1} of *SplitClass*$^{-1}$ is applicable only, if the match for node 5 contains no other property than the one matched by node 4. Otherwise, t^{-1} would violate the dangling edge condition. Therefore, any object o matched by node 6 has only a single property which is matched by node 6. Thus, i^{-1} does not violate the dangling edge condition. Different matches for i^{-1} can not overlap in nodes 6 or 8 since a match for nodes 6 or 8 determines the entire match (hence, all matches overlapping in 6 or 8 are equal). Thus, the identification condition can not be violated, either. The general parallel glueing condition holds for *SplitClass*$^{-1}$.

Condition 1 is fulfilled trivially, as the match for s is exactly determined by the parameter c.

Condition 2 We choose node 2 and node c as our subgraph sg. It represents the entire right-hand side of *SplitClass*$^{-1}$ and, thus, fulfills this conditions, trivially.

Condition 3 Any "foreign" match for sg would match node c to the passed parameter. Assume node 2 is matched to a "foreign" object o' (not covered by

the execution of SplitClass^{-1}). Then o' has been connected to class c before the application of SplitClass^{-1}), already. To be a valid instance of class c, object o' must have had an associated neighbour object as described by nodes 6, 7, and 8. Thus o' would have been covered by the execution of SplitClass^{-1}. This contradicts our assumption and thus proves that the condition holds. □

Consequently, SRT rule *SplitClass* can be classified to be information-preserving, because it is symetrically-lossless (cf. Definition 6.5.2). In [15], we show that the property of information-preservation can be proved analogously for SRT rule *MoveProperty*. This reference also discusses other kinds of SRT rules, e.g., information-augmenting SRT rules. A comprehensive catalog and classification of SRT rules is elaborated in [25].

The two sample SRT rules presented so far are so-called *primitive* SRT rules. The user can define *complex* SRT rules by specifying macros for concatenations of primitive SRT rules. An example for such a complex SRT rule is *SplitAndMove* (cf. Figure 6.5), which is a concatenation of *SplitClass* and several times *MoveProperty* (for each moved property). *SplitAndMove* is information-preserving, because both constituting primitive SRT rules are information-preserving.

6.6 Conclusion and Related Work

Most existing DRE tools are based on a fixed set of automatically applied schema translation rules, e.g., [4,23,8,7,19]. These tools lack adaptability as they do not consider the great variety of idiosyncrasies in existing LDAs. Moreover, they lack flexibility, as they generate a canonical translation of the LDA schema with little possibility for human design decisions. In order to overcome these problems, we follow a hybrid approach: we combine *automatic* conceptual schema translation with *interactive* SRT rules. At this, our environment exploits the power of GT rules in several ways: (1) we are able to customize our DRE environment by generating schema translation tools based on an adaptable set of schema mapping rules defined as a TGG, (2) by employing parallel GT systems we are able to define the semantics of SRT rules in a formal, yet understandable way, (3) the richness of GT theory enables us to reason about major properties of SRT rules.

The *Varlet* environment comprises about 235 000 lines of (C, C++, and TCL/TK) code (LOC). We have developed the 195 000 LOC core parts of *Varlet* using *Progres* [24]. The *Progres* specification consists of about 18 000 LOC.

Acknowledgements Many thanks to Gabi Taentzer, Annika Wagner, and especially to Reiko Heckel for their private lessons in double-pushout theory, proof readings, discussions and recommendations. We would also like to thank Heather Steel for editing the final version of this chapter.

References

1. F. Abbattista, F. Lanubile, and G. Visaggio. Recovering conceptual data models is human-intensive. In *Proc. of 5th Intl. Conf. on Software Engineering and Knowledge Engineering, San Francisco, California, USA*, pages 534–543, 1993.
2. M. Andersson. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In *Proc. of the 13th Int. Conference of the Entity Relationship Approach, Manchester*, pages 403–419. Springer, 1994.
3. C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database design*. Benjamin/Cummings, 1992.
4. Andreas Behm, Andreas Geppert, and Klaus R. Dittrich. On the migration of relational schemas and data to object-oriented database systems. In *Proc. 5th International Conference on Re-Technologies for Information Systems, Klagenfurt, Austria*, December 1997.
5. Ted J. Biggerstaff. Human-oriented conceptual abstractions in the reengineering of software. In *Proceedings of the 12th International Conference on Software Engineering*, page 120, March 1990.
6. M. Blaha and W. Premerlani. Observed idiosyncracies of relational database designs. In *Second Working Conference on Reverse Engineering*. IEEE, 1995.
7. C. Fahrner and G. Vossen. Transforming Relational Database Schemas into Object-Oriented Schemas according to ODMG-93. In *Proc. of the 4th Int. Conf. of on Deductive and Object-Oriented Databases 1995*, 1995.
8. J. Fong. Converting relational to object-oriented databases. *ACM SIGMOD Record*, 26(1), March 1997.
9. J-L. Hainaut. Entity-generating schema transformations for entity-relationship models. In *Proc. of the 10th Entity-Relationship Conference, San Mateo*, 1991.
10. R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars - a constructive approach. In *Proc. of SEGRAGRA '95: Graph Rewriting and Computation, Electronic Notes of TCS, http://www.elsevier.nl/locate/entcs/volume2.html*, volume 2, 1995.

11. Jens Holle. Ein generator für integrierte werkzeuge am beispiel der objekt-relationalen datenbankschemamigration. Master's thesis, Universität-GH Paderborn, Fachbereich 17, Paderborn, Germany, 1997.
12. J. H. Jahnke and M. Heitbreder. Design recovery of legacy database applications based on possibilistic reasoning. In *Proceedings of 7th IEEE Int. Conf. of Fuzzy Systems (FUZZ'98). Anchorage, USA*. IEEE Computer Society, May 1998.
13. J. H. Jahnke, W. Schäfer, and A. Zündorf. A design environment for migrating relational to object oriented database systems. In *Proc. of the 1996 Int. Conference on Software Maintenance (ICSM'96)*. IEEE Computer Society, 1996.
14. J. H. Jahnke, W. Schäfer, and A. Zündorf. Generic fuzzy reasoning nets as a basis for reverse engineering relational database applications. In *Proc. of European Software Engineering Conference (ESEC/FSE)*, number 1302 in LNCS. Springer, September 1997.
15. J. H. Jahnke and A. Zündorf. Using graph grammars for building the varlet database reverse engineering environment. In *Proc. of Theory and Application of Graph Transformations, Paderborn, Germany*, LNCS. Springer Verlag, Berlin, November 1998. to appear.
16. Jens H. Jahnke and Jörg Wadsack. Integration of analysis and redesign activities in information system reengineering. In *Proc. of Intl. Conf. on Software Maintenance and Reengineering (CSMR'99)*. IEEE CS, March 1999. to appear.
17. M. Lefering and A. Schürr. Specification of integration tools. In M. Nagl, editor, *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, chapter 3.4. Springer, Berlin (LNCS 1170), 1996.
18. L. Markosian, p. Newcomb, R. Brand, S. Burson, and T. Kitzmiller. Using an enabling technology to reengineer legacy systems. *Communications of the ACM*, 37(5):58–70, May 1994.
19. P. Martin, J. R. Cordy, and R. Abu-Hamdeh. Information capacity preserving of relational schemas using structural transformation. Technical Report ISSN 0836-0227-95-392, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, November 1995.
20. M. Nagl, editor. *The IPSEN Book*, volume 1170 of *LNCS*. Springer, Berlin, Germany, 1996.
21. J-M. Petit, J. Kouloumdjian, J-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 369–386. Springer, 1994.
22. Elke Radeke. *Federation and Migration among Database Systems*. PhD thesis, University of Paderborn - Dept. of Mathematics and Computer

Science, 1995.
23. S. Ramanathan and J. Hodges. Extraction of object-oriented structures from existing relational databases. *ACM SIGMOD Record*, 26(1), March 1997.
24. Grzegorz Rosenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume I. World Scientific, Singapore, 1997.
25. Christian Rummel. Ein transformationsbasierter ansatz zur migration von relationalen zu objektorientierten datenbanken. Master's thesis, Univeristät-GH Paderborn, Mathematik-Informatik, 33095 Paderborn, Germany, November 1998.
26. O. Signore, M. Loffredo, M. Gregori, and M. Cima. Reconstruction of er schema from database applications: a cognitive approach. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 387–402. Springer, 1994.
27. Gabriele Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, Technische Universität Berlin, Fachbereich 13, 1996.